Eurographics Symposium on Geometry Processing 2017
J. A. Bærentzen and K. Hildebrandt
(Guest Editors)

*Volume 36 (2017), Number 5*

# Restricting Voronoi diagrams to meshes using corner validation

paper1052

**Abstract**

*Restricted Voronoi diagrams are a fundamental geometric structure used in many applications such as surface reconstruction from point sets or optimal transport. Given a set of sites* $\mathbf{V} = \{\mathbf{v}_k\}_{k=1}^n \subset \mathbb{R}^d$ *and a mesh* $\mathbf{X}$ *with vertices in* $\mathbb{R}^d$ *connected by triangles, the restricted Voronoi diagram partitions* $\mathbf{X}$ *by computing for each site the portion of* $\mathbf{X}$ *for which the site is the nearest. The restricted Voronoi diagram is the intersection between the regular Voronoi diagram and the mesh. Depending on the site distribution or the ambient space dimension computing the regular Voronoi diagram may not be feasible using classical algorithms. In this paper, we extend Lévy and Bonneel's approach [LB12] based on nearest neighbor queries. We show that their method is limited when the sites are not located on* $\mathbf{X}$*. We propose a new algorithm for computing restricted Voronoi which reduces the number of sites considered for each triangle of the mesh and scales smoothly when the sites are far from the surface.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

## 1. Introduction

Voronoi diagrams are fundamental tools used in many applications. A Voronoi diagram is a decomposition of space induced by a finite set of points called *sites*. Each cell in the decomposition corresponds to one site, and consists of all the points in space closer to that site than to any other sites. In many practical situation, we only need to decompose a domain, not the whole space. In other words, for each cell in the Voronoi diagram, we are only interested in the intersection of that cell with the domain. The collection of all the non-empty intersections form the *Voronoi diagram restricted* to the domain. Our goal in this paper is to provide an efficient algorithm for computing restricted Voronoi diagrams when domains are finite unions of triangles, for instance triangulated surfaces. We put no restrictions on the dimension of the ambient space.

### 1.1. Contribution

Restricted Voronoi diagrams have already been studied and algorithms exist to compute them. Our algorithm outperforms state of the art methods when:

- sites are not distributed on the domain;
- many sites do not contribute to the restricted Voronoi diagram;
- the embedding dimension is higher than 3;
- the number of vertices in the Voronoi diagram is much bigger than the number of vertices in the restricted Voronoi diagram; see Figure 1 for an example.

Figure 1 depicts a case that is pathological for all previous methods but ours. In this case, sites are not located on the domain, the

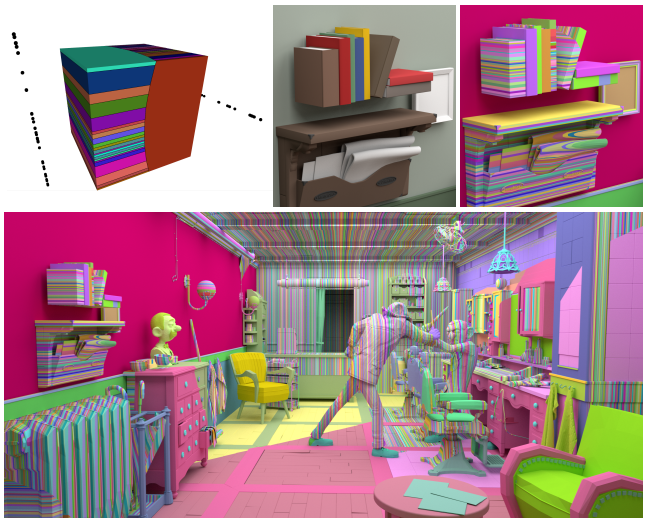


Figure 1: Pathological configuration for state of the art methods. Sites are distributed on axis aligned lines far from the domain (top left) which makes the full diagram complex. On a 6D (x,y,z,r,g,b) scene with 2M faces and using 100k sites on 20 lines (bottom) we compute the restricted diagram in 43s while other methods cannot compute it in a reasonable time. Note how regions with different input colors (middle top) are clipped by different sites (top right). Agent 327 scene courtesy of Francesco Siddi, Blender Cloud.

number of vertices in the Voronoi diagram is quadratic, whereas the number of vertices in the restricted Voronoi diagram is only linear. In this paper, we focus on domains that are finite unions of triangles, typically triangulated surfaces. However, it should be notice that our approach generalizes to higher-dimensional domains, that is, finite unions of $k$-dimensional simplices for $k \geq 2$, but at the price of computing $k$-dimensional weighted Voronoi diagrams.

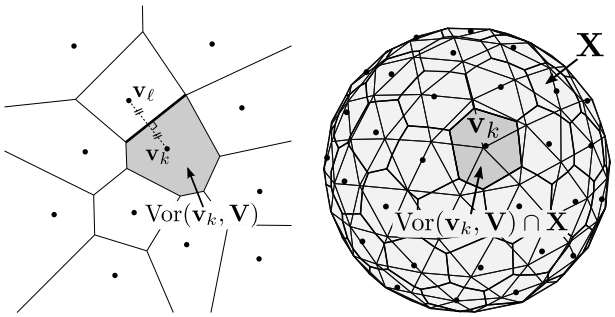
### 1.2. Application

The main application we target is the computation of an optimal transport map between two surfaces. Using a semi-discrete setting, one may assume that one of the surface, say **A**, has been sampled by a finite set of sites, and the second surface, say **B**, is represented by a triangulation. One way to transport surface **A** onto surface **B** consists in computing the Voronoi diagram of the sites that sample **A** restricted to the triangulation of **B** and sending each restricted Voronoi cell onto its associated site. This gives a transport map. To get the optimal one, we associate each site with a weight and adjust weights iteratively, minimizing an objective function; see [Mér11, Lév15] for the details. Computing an optimal transport thus requires the computation of many weighted restricted Voronoi diagrams. Notice that, for this particular application, sites do not lie necessarily on the domain. This is especially true when weights increase. Furthermore, every site contributes to the diagram and it is not possible to gain performance by ignoring some of them. Unlike existing methods, our method behaves well in this situation.

Optimal transport maps are useful for establishing correspondences between surfaces. Such correspondences are needed in many applications (registration, fitting or interpolation). Assuming the cost of moving point $a$ to point $b$ is $\|a-b\|^p$, the cost of an optimal transport map is a distance called the *Wasserstein distance of order p*. Such a distance can be used to evaluate methods that modify meshes (remeshing, simplification or compression), by giving a measure of the difference between input and output. Optimal transport algorithms and the related Wasserstein distance are currently gaining popularity due to recent advances making the computation of optimal transport maps tractable [FCRT14, SdGP*15, BPC16]. Our contribution is one more step in making such computations affordable.

### 1.3. Method

Our algorithm is based on nearest neighbor queries. Such an approach has already been used successfully to accelerate Voronoi diagram computations [LB12, Ryc09]. Data structures exist to perform these queries efficiently even for high dimensional data [FLA15, MA97], using hierarchical space partitioning. To ensure that the final restricted Voronoi diagram is computed correctly, the centers of the nearest neighbor queries have to be carefully chosen, along with a criterion ensuring that sufficient information was gathered to validate the computed Voronoi cells. We improve the existing methods by changing the location of the queries and the validation criterion.




(a) classical : $\mathrm{Vor}(\mathbf{V})$ (b) restricted : $\mathrm{Rvd}(\mathbf{V}, \mathbf{X})$

Figure 2: Voronoi diagrams for a set of sites $\mathbf{V} = \{\mathbf{v}_k\}_{k=1}^n$

## 2. Background

### 2.1. Voronoi diagrams

Given a set of sites, $\mathbf{V} = \{\mathbf{v}_k\}_{k=1}^n \subset \mathbb{R}^d$, the Voronoi cell of site $\mathbf{v}_k \in \mathbf{V}$ is defined as:

$$\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) = \Big\{ \mathbf{x} \in \mathbb{R}^d, \|\mathbf{v}_k - \mathbf{x}\| \leq \|\mathbf{v}_\ell - \mathbf{x}\| \text{ for all } \ell \neq k \Big\}.$$

The Voronoi neighbors of site $\mathbf{v}_k$ are defined as the sites $\mathbf{v}_\ell \in \mathbf{V}$ such that $\mathrm{Vor}(\mathbf{v}_\ell, \mathbf{V}) \cap \mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \neq \emptyset$. Given a pair of sites $(\mathbf{v}_k, \mathbf{v}_\ell) \in \mathbf{V}^2$, the set of $\mathbf{x} \in \mathbb{R}^d$ such that $\|\mathbf{v}_k - \mathbf{x}\| = \|\mathbf{v}_\ell - \mathbf{x}\|$ is called the bisector of $\mathbf{v}_k$ and $\mathbf{v}_\ell$. The bisector bounds two half spaces. Let $H_{k,l}$ be the one that contains $\mathbf{v}_k$ and excludes $\mathbf{v}_\ell$. It is not difficult to see that the Voronoi cell $\mathrm{Vor}(\mathbf{v}_k, \mathbf{V})$ is the intersection of all half spaces $H_{k,l}$ defined by bisectors between site $\mathbf{v}_k$ and its Voronoi neighbors $\mathbf{v}_\ell$. The Voronoi diagram $\mathrm{Vor}(\mathbf{V})$ is the collection of all Voronoi cells, $\mathrm{Vor}(\mathbf{V}) = \big\{\mathrm{Vor}(\mathbf{v}, \mathbf{V}), \forall \mathbf{v} \in \mathbf{V}\big\}$ (see Figure 2a).

Voronoi diagrams and their dual, Delaunay complexes have been studied a lot [OBSS00] and robust algorithms exist for their computation [CGA16, BDH96, She96]. State of the art algorithms generally implement an incremental construction of the Delaunay complex: the sites are inserted one by one, checking for the validity of the existing simplices, and remeshing the area covered by invalid simplices. When the set of sites is known in advance, the insertion order is generally optimized using spatial sorting techniques to improve data locality [ACR03]. These algorithms perform very well in two or three dimensions. But, as the dimension of the ambient space increases, it is well-known that the size of the Delaunay complex explodes [McM70] in the worst-case. These worst-cases are however unlikely, and the major problem with the dimension is usually the storage and traversal of the Delaunay complex itself. Algorithms exist to compute Delaunay complexes in arbitrary dimension [BDH09, CGA16], which only store the Delaunay graph of the sites. This approach remains viable in medium dimensions, but does not scale very well for higher dimensions. Another approach is that of Rycroft [Ryc09], who uses nearest neighbor queries to compute Voronoi cells one by one. Nearest neighbor queries stay efficient even for high dimensions.

### 2.2. Restricted Voronoi diagrams

Given a domain $\mathbf{X} \subseteq \mathbb{R}^d$, the *restricted Voronoi cell* of $\mathbf{v}_k$ is defined as the intersection:

$$\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{X} = \left\{ \mathbf{x} \in \mathbf{X}, \|\mathbf{v}_k - \mathbf{x}\| \leq \|\mathbf{v}_\ell - \mathbf{x}\| \text{ for all } \ell \neq k \right\}$$

By definition $\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{X} \subset \mathrm{Vor}(\mathbf{v}_k, \mathbf{V})$. The restricted Voronoi diagram of $\mathbf{V}$ to the domain $\mathbf{X}$ is the set of all the non-empty restricted cells :

$$\mathrm{Rvd}(\mathbf{V}, \mathbf{X}) = \left\{ \mathrm{Vor}(\mathbf{v}, \mathbf{V}) \cap \mathbf{X}, \forall \mathbf{v} \in \mathbf{V}, \mathrm{Vor}(\mathbf{v}, \mathbf{V}) \cap \mathbf{X} \neq \emptyset \right\}$$

For the computation of restricted Voronoi diagrams, we are not interested in the whole diagram. Computing the full diagram is therefore a waste of time, in particular when the dimension of the restricting domain is lower than that of the ambient space, and small. The idea is not new and was already applied to large datasets [DGH01]. Efficient restricted Voronoi diagram algorithms mainly emerged for applications in surface remeshing [YLL*09, YWLL13]. First algorithms were computing the full Delaunay skeleton and using it to clip the domain by the Voronoi cells. Such approaches do not scale to higher dimensions since the cost of computing the Delaunay complex becomes prohibitive. For anisotropic remeshing purposes and optimal transport, Lévy and Bonneel [LB12, Lév15] therefore developed a new algorithm based on nearest neighbor queries with a criterion similar to that of [Ryc09] to ensure that the final diagram is correct. For remeshing purposes, the sites are usually located on the surface to be remeshed. As we will show in § 2.3.2.3, when this hypothesis is no longer valid and the sites may contribute to the restricted diagram while being far from the surface, this approach can degenerate and use lots of sites for each Voronoi cell. Such a situation is not an abstract pathological case. Restricted Voronoi diagrams are for instance used to compute distances between surfaces [NYL14] or semi-discrete optimal transport [Mér11]. In such cases, the sites are located on another surface, and can therefore be far away.

### 2.3. Algorithms

#### 2.3.1. Classical algorithm

**2.3.1.1. Principle** The straightforward way to compute a restricted Voronoi diagram $\mathrm{Rvd}(\mathbf{V}, \mathbf{X})$ is by computing the Voronoi diagram $\mathrm{Vor}(\mathbf{V})$ in the ambient space and then computing its intersection with $\mathbf{X}$. When $\mathbf{X}$ is a finite union of triangles, this means partitioning every triangle $\mathbf{T}$ in the union using the Voronoi cells. Given a site, its restricted Voronoi cell is computed by clipping the domain $\mathbf{X}$ using the Voronoi neighbors of the site. This approach has been used for remeshing purposes [YLL*09], where the restriction domain $\mathbf{X}$ is the remeshed mesh. In this context, for a triangle $\mathbf{T} \subset \mathbf{X}$, the intersection is computed using reentrant polygon clipping [SH74]. Once $\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{T}$ is computed, the computation is propagated using its boundary :

- an intersection between $\mathbf{T}$ and the bisector of $(\mathbf{v}_k, \mathbf{v}_\ell)$ triggers the computation of $\mathrm{Vor}(\mathbf{v}_\ell, \mathbf{V}) \cap \mathbf{T}$ ;
- a piece of an edge between $\mathbf{T}$ and another triangle $\mathbf{T}'$ triggers the computation of $\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{T}'$.

For every connected component of $\mathbf{X}$, the propagation is initiated using a nearest neighbor query from a triangle.

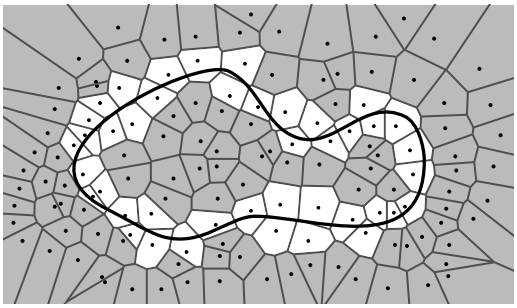


Figure 3: Contributing sites: sites in white cells are sites that contribute to the Voronoi diagram restricted to the target region (the solid black line).

**2.3.1.2. Limitations** This method is relevant in low dimensions since there exists very efficient algorithms to compute Voronoi diagrams in 2D and 3D [CGA16, BDH96, She96]. When only a few sites contribute to the restricted diagram (see Figure 3) or when the complexity of the full diagram is pathological in areas useless for the restricted diagram (see Figure 1), we shall see that nearest neighbor algorithms may be faster.

#### 2.3.2. Nearest neighbor algorithms

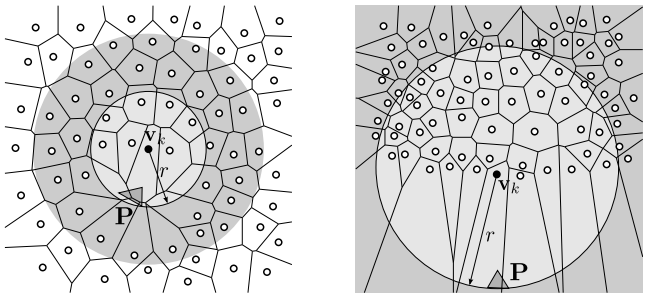
**2.3.2.1. Principle** Because of the above limitations (§ 2.3.1.2), Lévy and Bonneel [LB12] developed a new algorithm for computing restricted Voronoi diagrams using nearest neighbor requests. The idea is to replace the Voronoi neighbors with the sites nearest to the site $\mathbf{v}_k$.

Efficient data structures based on Kd-trees have been developed [MA97, FLA15] to provide these nearest neighbors even if the dimension is high. Classical applications for high dimensional requests are for instance computer vision or machine learning where the query points are feature sets. The space complexity of such data structures grows linearly with the number of points and the dimension, and is therefore much better than storing a full Delaunay complex.

For restricted Voronoi diagrams, using too many sites for the clipping is not a problem: some bisectors may not contribute to the final restricted Voronoi cell, but the result is the same. The problem is therefore to ensure that *sufficient* sites have been considered.

**2.3.2.2. Security radius** The security radius is a technique used by both [Ryc09] and [LB12] to ensure that sufficient sites have been used. It works as follows: given a site $\mathbf{v}_k$, a triangle $\mathbf{T}$ and a candidate polygon $\mathbf{P} \subset \mathbf{T}$ for the restricted cell $\mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{T}$, let us consider a minimal ball centered on $\mathbf{v}_k$ and containing $\mathbf{P}$. Let $r$ be the radius of this ball. Since the ball is centered on $\mathbf{v}_k$, if another site $\mathbf{v}_\ell$ is further from $\mathbf{v}_k$ than $2r$, then the bisector of $(\mathbf{v}_k, \mathbf{v}_\ell)$ cannot clip the ball, and therefore $\mathbf{P}$.

When using nearest neighbor queries, one can check whether the furthest site considered is further from $\mathbf{v}_k$ than $2r$ or not. If so, every other unconsidered site is further as well and therefore $\mathbf{P} = \mathrm{Vor}(\mathbf{v}_k, \mathbf{V}) \cap \mathbf{T}$ is valid. If not, additional nearest neighbors are requested until the condition is fulfilled.




(a) usual case (b) pathological case

Figure 4: Limitations of the security radius approach: when the sites are far from the surface, the radius increases and may force the algorithm to consider every site in the computation of every restricted Voronoi cell.

**2.3.2.3. Limitations** The above method works very well for sites located *on* the triangle $\mathbf{T}$. In such a situation, the ball radius is small, and few sites are considered. However, whenever a site contributes to the partitioning of a triangle while being far away from that triangle, the radius of the ball grows, and so does the number of sites considered. In a worst case scenario, the loop asking for additional sites goes on until every other site has been considered (see Figure 4).

**2.3.2.4. Discussion** For the above criterion to hold, the nearest neighbor requests need to be *exact*. Data structures for such requests are generally not implemented using exact predicates, and are usually much more efficient when a small tolerance is allowed. Algorithms based on the security radius therefore do not fully benefit from their efficiency. This is a limitation our method shares.

## 3. Contribution

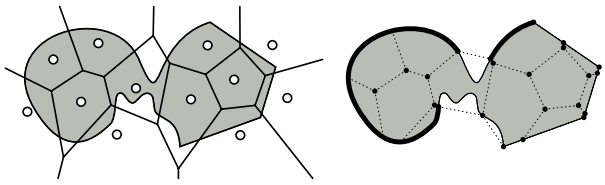
### 3.1. Idea

We define a new criterion guaranteeing a correct diagram while limiting the number of sites to consider. Our algorithm is based on that of Lévy and Bonneel [LB12] summarized in § 2.3.2.2. Similarly, it relies on nearest neighbor queries. However, those queries are centered on the triangle $\mathbf{T}$ rather than the site $\mathbf{v}_k$. Therefore, by definition of the Voronoi diagram, the result of a query performed from $\mathbf{T}$ is a site whose Voronoi cell intersects the triangle. We therefore only consider sites that contribute to the restricted Voronoi diagram (see Figure 3).

### 3.2. Corner validation

Our construction of the restricted Voronoi diagram is an iterative process. Every iteration introduces new sites into the restricted diagram. These site insertions generate new cells (and alter old ones) that have to be validated. A validated cell is a cell that is final in the restricted Voronoi diagram : it cannot be altered by the insertion of new sites.

Our *corner validation* criterion validates the restricted cells using nearest neighbor queries centered on their *corners*. In the specific case of a diagram restricted to a triangle, a restricted cell is a polygon. The corners are the vertices of the polygon. The general idea is that when a nearest neighbor query centered on the corner returns a site already inserted, the corner is guaranteed to exist in the final diagram. When all the corners of a restricted cell are valid, the cell can no longer be altered and the restricted cell itself is valid.



(a) diagram (b) witness set

Figure 5: On the left, a Voronoi diagram for a set of sites, and a restriction set. On the right, the convex hulls of the restricted cells (dotted lines) and the witness set (thick lines and black points).

#### 3.2.1. Formulation

We provide here a general formulation and proof of our validity criterion, not limited to restrictions to 2D triangulations. In such a setting, the *corners* we mentioned can be expressed in terms of *extreme points*.

**Definition 1 (Extreme point)** Let $\mathbf{S} \subset \mathbb{R}^d$ be a convex set. An extreme point of $\mathbf{S}$ is a point $\mathbf{s} \in \mathbf{S}$ such that no segment entirely in $\mathbf{S}$ contains $\mathbf{s}$ in its interior.

Now let $\mathrm{Ex}(\mathbf{S})$ denote the set of extreme points of a convex set and $\mathrm{Conv}(\mathbf{S})$ denote the convex hull of a set $\mathbf{S}$, and let us define the notion of *witness set* for a restricted Voronoi diagram, named after the work of De Silva [Sil08].

**Definition 2 (Witness set)** Let $\mathbf{V} \subset \mathbb{R}^d$ be a set of sites and let $\mathrm{Vor}(\mathbf{v}, \mathbf{V})$ denote the Voronoi cell of a site $\mathbf{v} \in \mathbf{V}$. Given a set $\mathbf{X} \subset \mathbb{R}^d$, the witness set of $\mathbf{V}$ with respect to $\mathbf{X}$ is defined as

$$\mathrm{W}(\mathbf{V}, \mathbf{X}) = \bigcup_{\mathbf{v} \in \mathbf{V}} \{\mathrm{Ex}(\mathrm{Conv}(\mathrm{Vor}(\mathbf{v}, \mathbf{V}) \cap \mathbf{X}))\}.$$

This witness set is the set of corners we use to validate the restricted Voronoi diagram. The following theorem states that checking the nearest neighbors of the points of a witness set is sufficient to validate a restricted Voronoi diagram.

**Theorem 1 (Corner validation)** Let $\mathbf{V} \subset \mathbb{R}^d$ be a set of sites and $\mathbf{U} \subset \mathbf{V}$ be a subset of the sites. Let $\mathbf{X} \subset \mathbb{R}^d$ be a compact set. If for all $\mathbf{w} \in \mathrm{W}(\mathbf{U}, \mathbf{X})$, $\mathbf{w} \in \mathrm{Vor}(\mathbf{u}, \mathbf{V})$ for some site $\mathbf{u} \in \mathbf{U}$ then for all $\mathbf{u} \in \mathbf{U}$, $\mathrm{Vor}(\mathbf{u}, \mathbf{U}) \cap \mathbf{X} = \mathrm{Vor}(\mathbf{u}, \mathbf{V}) \cap \mathbf{X}$.

*Proof* We will prove the result by mutual inclusion.
By definition of the Voronoi diagram we have that $\mathrm{Vor}(\mathbf{u}, \mathbf{V}) \subset \mathrm{Vor}(\mathbf{u}, \mathbf{U})$ since $\mathbf{U}$ is a subset of $\mathbf{V}$. Therefore $\mathrm{Vor}(\mathbf{u}, \mathbf{V}) \cap \mathbf{X} \subset \mathrm{Vor}(\mathbf{u}, \mathbf{U}) \cap \mathbf{X}$.
The convex hull of a set $\mathbf{S}$ is the union of all the convex combinations of points in $\mathbf{S}$. A point in $\mathrm{Conv}(\mathbf{S})$ that is not in $\mathbf{S}$ is therefore a convex combination of two or more points in $\mathbf{S}$ and is therefore in the interior of a segment contained in $\mathbf{S}$. The extreme points of the convex hull of $\mathbf{S}$ are therefore in $\mathbf{S}$.

For every extreme point $\mathbf{w}$ of $\mathrm{Ex}(\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}))$, our hypothesis state that there exists a site $\mathbf{u}'\in\mathbf{U}$ such that $\mathbf{w}\in\mathrm{Vor}(\mathbf{u}',\mathbf{V})$. Since $\mathbf{u}'\in\mathbf{U}$ and $\mathbf{w}\in\mathrm{Vor}(\mathbf{u},\mathbf{U})$, $\mathbf{w}$ is therefore equidistant to $\mathbf{u}'$ and $\mathbf{u}$ and $\mathbf{w}\in\mathrm{Vor}(\mathbf{u},\mathbf{V})$. This leads to $\mathrm{Ex}(\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}))\subset\mathrm{Vor}(\mathbf{u},\mathbf{V})$ and since Voronoi cells are convex we finally have that

$$\mathrm{Conv}(\mathrm{Ex}(\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X})))\subset\mathrm{Vor}(\mathbf{u},\mathbf{V}).$$

$\mathbf{X}$ is compact and $\mathrm{Vor}(\mathbf{u},\mathbf{U})$ is closed. $\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}$ is therefore compact, and as a corollary of Carathéodory's Convexity Theorem [AB06, Corollary 5.33] $\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X})$ is compact as well. Using the Krein-Millner Theorem, we obtain

$$\mathrm{Conv}(\mathrm{Ex}(\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X})))=\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}),$$

and therefore

$$\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}\subset\mathrm{Conv}(\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X})\subset\mathrm{Vor}(\mathbf{u},\mathbf{V}).$$

Finally

$$\mathrm{Vor}(\mathbf{u},\mathbf{U})\cap\mathbf{X}\subset\mathrm{Vor}(\mathbf{u},\mathbf{V})\cap\mathbf{X}.$$

□

#### 3.2.2. Discussion

Voronoi diagrams are dual to Delaunay complexes. In such a complex, the Voronoi sites are the vertices, and every simplex is such that the interior of its circum-hypersphere contains no site. In terms of nearest neighbors, this means that for every simplex, the nearest neighbors of its circum-center are the vertices of the simplex. These circum-centers are our Voronoi vertices.

In terms of construction, Delaunay complexes are generally built by incrementally adding new vertices. Each vertex insertion yields a conflict area where simplices are destroyed and rebuilt to integrate the new vertex. This procedure is made efficient by using spacial sorting algorithms. In contrast, our approach chooses the vertices to add by checking for every simplex if at least one vertex not currently in the complex is in conflict with the simplex. This test is done using a nearest neighbor query. Our approach differs in that for *restricted* Voronoi diagrams, all the sites do not necessarily contribute to the diagram, and we only store restricted Delaunay simplices to validate the corresponding corners for the restriction domain. Another related incremental Delaunay construction algorithm is Delaunay refinement [She02]. In such a context however, sites are introduced in specific positions to improve the quality of the Delaunay simplices, whereas in our case, our refinement criterion is the validity of the diagram itself and we have no choice for the position of the sites.

De Silva [Sil08] already formulated the Delaunay complex in terms of nearest neighbors. In his formulation, our construction algorithm is just an iterative process to discover strong witnesses for every restricted Delaunay simplex.

### 3.3. Corner management

In the incremental construction, we need a data structure to manage the set of corners, update it after site insertions and mark them as valid. We do so using a power diagram of the dimension of $\mathbf{X}$. In our case, $\mathbf{X}$ is a collection of triangles and on each triangle, we use a 2D power diagram in the support plane of the triangle.

#### 3.3.1. Power diagram and regular triangulation

The power diagram is a generalization of the Voronoi diagram in which, each site $\mathbf{v}_k$ is given a weight $w_k$. Voronoi diagrams are a special case of power diagrams where all sites $\mathbf{v}_k$ have a weight $w_k=0$. The regular triangulation is the dual of the power diagram.

**Definition 3 (Power cells)** Let $\mathbf{V}=\{\mathbf{v}_k\}_{k=1}^n\subset\mathbb{R}^d$ be a set of sites. Given a set of weights $\mathbf{W}=\{w_k\}_k$ for each site, let us define $\hat{\mathbf{v}}_k=(\mathbf{v}_k,w_k)$ a weighted site and $\hat{\mathbf{V}}=\{\hat{\mathbf{v}}_k\}_k$ the set of weighted sites. The Power cell $\mathrm{Pow}(\hat{\mathbf{v}}_k,\hat{\mathbf{V}})$ of a weighted site is defined as:

$$\mathrm{Pow}(\hat{\mathbf{v}}_k,\hat{\mathbf{V}})=\left\{\mathbf{x}\in\mathbb{R}^d,\|\mathbf{v}_k-\mathbf{x}\|-w_k\leq\|\mathbf{v}_\ell-\mathbf{x}\|-w_\ell,\forall\ell\neq k\right\}$$

#### 3.3.2. Support frame of a triangle

Given a triangle $\mathbf{T}$ of $\mathbf{X}$ with vertices $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3\in\mathbb{R}^d$, we can compute a 2D orthonormal frame $(\mathbf{e}_1,\mathbf{e}_2)$ centered on one of the vertices of $\mathbf{T}$ (i.e. $\mathbf{x}_1$) as follows:

$$\mathbf{e}_1=\frac{\mathbf{x}_2-\mathbf{x}_1}{\|\mathbf{x}_2-\mathbf{x}_1\|}\qquad\mathbf{e}_2=\frac{(\mathbf{x}_3-\mathbf{x}_1)-(\mathbf{x}_3-\mathbf{x}_1).\mathbf{e}_1}{\|\mathbf{x}_3-\mathbf{x}_1)-(\mathbf{x}_3-\mathbf{x}_1).\mathbf{e}_1\|}$$

Every point $\mathbf{x}\in\mathbb{R}^d$ can be expressed as a combination of a tangent component $\mathbf{x}^{/\!/}$ (included in the plane spanned by $\mathbf{T}$) and a normal component $\mathbf{x}^\perp$:

$$\begin{aligned}\mathbf{x}&=\mathbf{x}^{/\!/}+\mathbf{x}^\perp\\ \mathbf{x}^{/\!/}&=[(\mathbf{x}-\mathbf{x}_1).\mathbf{e}_1]\mathbf{e}_1+[(\mathbf{x}-\mathbf{x}_1).\mathbf{e}_2]\mathbf{e}_2\\ \mathbf{x}^\perp&=\mathbf{x}-\mathbf{x}^{/\!/}\end{aligned}$$

With these notations, we can now provide the relation between a power diagram and a restricted Voronoi diagram.

#### 3.3.3. Expressing a restricted diagram as a power diagram

The Voronoi diagram of a set of sites $\mathbf{V}\subset\mathbb{R}^d$ restricted to a hyperplane is known [Aur87, AK96, CGA16] to be equivalent to a power diagram in the hyperplane. This relation was extended by Boissonat and Gosh [BG14, Lemma 2.2] to the following:

**Proposition 1** Let $\mathbf{X}$ be an affine subspace of $\mathbb{R}^d$ of dimension $a$. Let $\mathbf{V}=\{\mathbf{v}_k\}_{k=1}^n\subset\mathbb{R}^d$ be a set of sites, $\mathbf{V}^{/\!/}=\{\mathbf{v}_k^{/\!/}\}_k$ their projections on $\mathbf{X}$ and $\mathbf{V}^\perp=\{\mathbf{v}_k^\perp\}_k$ their normal components. The Voronoi diagram of $\mathbf{V}$ restricted to $\mathbf{X}$ is the power diagram of $\mathbf{V}^{/\!/}=\{\mathbf{v}_k^{/\!/}\}_{k=1}^n$ in $\mathbf{X}$ with the weights $\mathbf{W}=\{w_k\}_{k=1}^n=\{-\mathbf{v}_k^{\perp^2}\}_{k=1}^n$.

The computation of a $d$D Voronoi diagram restricted to a 2D plane can therefore be made through a power diagram on $\mathbf{T}$ using $\mathbf{V}^{/\!/}$ as positions for the sites and weights derived from $\mathbf{V}^\perp$. Note that we use a different power diagram on each triangle since the tangent and normal components of the sites depend on the support frame of the triangle. The goal of our algorithm is therefore to choose the sites to introduce in the diagram and determine when sufficient sites have been considered for the restricted diagram to be valid using our corner validation mechanism.

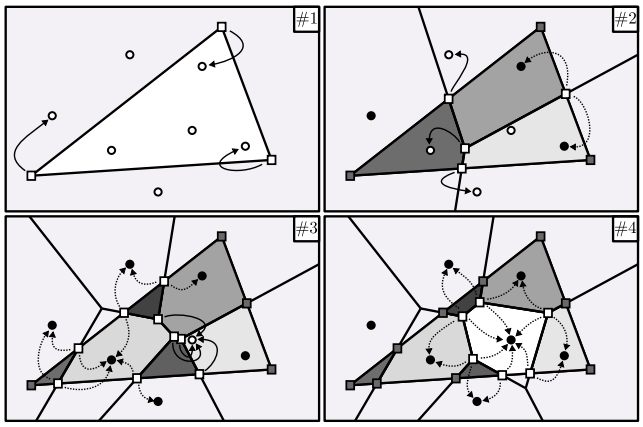



Figure 6: Iterations of Algorithm 1
□: a new corner
■: a validated corner
●: a site already found and inserted in the regular triangulation
○: a site not found yet
At every iteration, every new corner (□) is tested with a nearest site request (represented by arrows). Dashed arrows mean that the corner tested is equidistant to 2 sites and therefore a nearest site request can return any of the two sites. New sites found this way are inserted in the diagram, leading to the emergence of new corners to test, until there is no more new sites to insert (#4).

### 3.4. Algorithm

Algorithm 1 and Figure 6 present our algorithm. The set of Voronoi corners is initialized to the 3 vertices of $\mathbf{T}$. At every iteration, we check the nearest site of each new Voronoi corner (NN($\mathbf{x}$) in Algorithm 1). All newfound sites are projected in the triangle frame (Section 3.3.2) and inserted in the power diagram (PD) using their weight $w_k = -\mathbf{v}_k^{\perp^2}$ (Proposition 1). Every insertion of a new site in PD will cause the emergence of new corners to test. Every test of a new corner will lead to either the validation of this corner (if the nearest site has already been inserted in PD) or the discovery of a new contributing site (that will be added to new_sites before being inserted in PD). At every iteration, we solely extract the new corners from the power diagram (Section 3.5). When all corners are validated, no more site is to be inserted, and the restricted Voronoi diagram of $\mathbf{T}$ is complete.

Since the Algorithm 1 works independently on each triangle, the computation on every triangle $\mathbf{T} \in \mathbf{X}$ can be parallelized, as exposed in Algorithm 2. At the end of this algorithm, when all $\mathrm{Rvd}(\mathbf{V},\mathbf{T})$ are computed, we gather them together in order to obtain one final Voronoi diagram restricted to $\mathbf{X}$.

Most of the time, a Voronoi corner is shared among 2 or 3 restricted cells. This means that the Algorithm 1 would naïvely use 2 or 3 times the number of required nearest site requests. In Section 3.5 we present some of our implementation details to handle this issue.

**Input**: $\mathbf{T}$ : a triangle $\in \mathbf{X}$
**Data**: new_sites: a table of untreated new sites
new_corners: a table of new Voronoi corners
PD: the power diagram
**Result**: $\mathrm{Rvd}(\mathbf{V},\mathbf{T})$ : the Voronoi diagram of $\mathbf{X}$ restricted to $\mathbf{T}$

```
1  function compute_rvd(T)
       /* Initialization                          */
2      compute tangent plane    /* Section 3.3.2 */
3      foreach vertex x_i of triangle T do
4          add NN(x_i) to new_sites
5      while new_sites is not empty do
           /* Insert contributing sites in
              the power diagram                   */
6          foreach site in new_sites do
7              project site in tangent plane
8              add site to PD
9          extract new_corners
10         empty new_sites
           /* Validate the corners, search
              for new sites                       */
11         foreach x_i in new_corners do
12             if NN(x_i) is not already in PD then
13                 add NN(x_i) to new_sites
14     return PD ∩ T
```

**Algorithm 1:** Compute $\mathrm{Rvd}(\mathbf{V},\mathbf{T})$ for a single triangle $\mathbf{T}$

**Input**: $\mathbf{X}$ : a triangulated mesh
$\mathbf{V}$ : a set of sites
**Result**: $\mathrm{Rvd}(\mathbf{V},\mathbf{X})$ : the Voronoi diagram of $\mathbf{V}$ restricted to $\mathbf{X}$

```
1  begin
2      forall triangle T ∈ X do in parallel
3          Rvd(V,T) ← compute_rvd(T)
4      Rvd(V,X) ← merge all Rvd(V,T) together
```

**Algorithm 2:** Compute $\mathrm{Rvd}(\mathbf{V},\mathbf{X})$

### 3.5. Implementation

**Regular triangulation using CGAL** [CGA16] To compute the 2D power diagram on every triangle, we use a regular triangulation structure from CGAL, configured using the Epic kernel. Given a newly inserted site, the new corners due to the insertion are obtained as the weighted circum-centers of the simplices sharing the new site.

**Nearest site requests using FLANN** [FLA15] In Algorithm 1 we use a lot of nearest site searches from new corners to find new sites (○ ← NN(□) ). This is done using a global FLANN index, which is thread-safe. Even though FLANN is designed for finding *approximate* nearest neighbors, it remains possible to ask for an exact request causing, however, a loss of efficiency.

Despite the fact that we are working in a local frame for each triangle, a single global nearest neighbor structure is shared. This global structure is built in $\mathbb{R}^d$ space and does not depend on any

triangle. Therefore, when working on a triangle, making nearest site requests from a Voronoi corner requires to previously express it in $\mathbb{R}^d$ (change of basis). That way, the global structure is only built once and can be queried from any triangle.

**Avoiding infinite triangles in the regular triangulation** Corners can be vertices of the triangle, intersections between edges of the triangles and bisectors, or restricted Voronoi vertices inside the triangle. To ease the management of these corners, we compute the diagram in two steps. First, we compute the diagram restricted to the edges of the triangle. Then, we build the regular triangulations from the resulting sites, and go on with the corner validation and insertions. In the second step, since the Voronoi diagram is valid on the edges, all the new restricted cells created will be entirely in the triangle. We therefore only need to deal with corners inside the triangle, and avoid handling infinite triangles in the regular triangulation or intersections with the edges of the triangle.

**Robustness** We do not use exact predicates in our implementation. The first reason is that the nearest neighbor library we use [FLA15] plays a central role in the algorithm, and does not use exact predicates itself. Even if it did, compared to Lévy and Bonneel [LB12], our nearest neighbor request centers are not the sites but the Voronoi vertices and therefore the result of a calculation. This makes exact nearest neighbor requests more difficult: we would require a predicate to determine among two sites the one nearest to a corner, which depends on both sites and mesh vertices. Another potential source of numerical errors is the projection we use before inserting sites in the regular Delaunay triangulation. The regular triangulation itself is robust since we rely on CGAL [CGA16] for this part with Epic kernel. Since our final RVD is built from this triangulation, we are sure that whatever degeneracy appears because of the initial data or our computations, CGAL will handle it properly and provide us with a coherent triangulation.

To validate our method experimentally, we compared the combinatorics of our result with that of Geogram [Lév16] with exact predicates enabled. To do so, we distributed 2k sites on a sphere of radius 1, and restricted the diagram to a sphere with identical center and radius $\frac{1}{2^d}$ with increasing $d$. In such a setup, nearest neighbor queries become more and more subject to errors. Our computations are performed using doubles, and CGAL Epic kernel. As shown on Figure 7, our resulting diagram stays valid up to $d = 43$ meaning an inner sphere of radius $2.10^{-13}$.

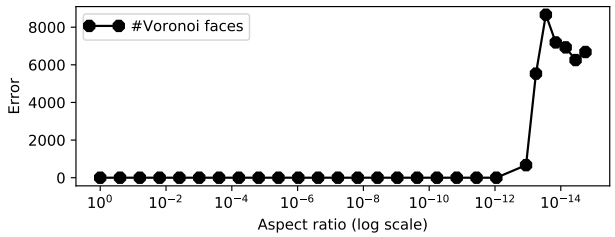
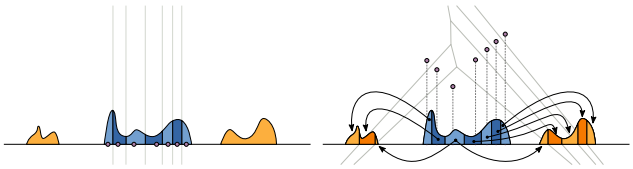



Figure 7: Precision of our diagrams. 2k sites are distributed on a sphere of radius 1, and the diagram is restricted to a sphere of decreasing radius (abscissae). The curve shows the error in terms of difference in the number of computed Voronoi faces w.r.t. the reference computed with Geogram using exact predicates [Lév16].



(a) Initialization
Discrete source (blue, center)
Target (orange, left and right)

(b) Transport map
Result after #n iterations

Figure 8: First and last step of optimal transport computation. (a) Sites are distributed on the source, each one is given a mass.

## 4. Application to Optimal transport

In 1781, Gaspard Monge [Mon81] described what will later become the field of optimal transport. The problem was the following : how to move a pile of sand into another while minimizing the displacement cost $c(x,y)$ of every sand particles from $x$ to $y$. Nowadays, it has been widely studied [Vil09] and used in many applications such as image retrieval using the Earth Mover's Distance (defined by optimal transport) [RTG00], histogram regression [BPC16] or shape interpolation [SdGP*15].

We applied our algorithm to the problem of $L^2$ semi-discrete optimal transport [Mér11]. Similarly, [Lév15] realize semi-discrete optimal transport between tetrahedral meshes in 3D.

**Semi discrete optimal transport** Given two meshes $\mathbf{A}$ (the source) and $\mathbf{B}$ (the target), we discretize the source (see Figure 8a) by distributing a set of sites $\mathbf{V}$ over the mesh $\mathbf{A}$. Sites can be distributed homogeneously using Lloyd relaxation [Llo82] or Newton-type methods for faster convergence [LWL*09]. Every site $\mathbf{v}_k$ is assigned with a mass $\lambda_k$. In our specific case, the Dirac masses used are the areas of the restricted Voronoi cells of the sites on mesh $\mathbf{A}$. The target mesh $\mathbf{B}$ has a measure $\mu$ with density. We aim to transport the mass contained on sites to the target location on $\mathbf{B}$. That is to say, obtain a mapping between $\mathbf{V}$ and portions of $\mathbf{B}$ that is optimal in terms of displacement cost while satisfying the constraints (see Figure 8b).

It is known that any least squares assignment problem subject to capacity constraints can be realized by a power diagram [AHA98, Theorem 1]. Given a source discrete measure, supported on a set of sites $\mathbf{V} = \{\mathbf{v}_k\}_{k=1}^n$ with masses $\mathbf{\Lambda} = \{\lambda_k\}_{k=1}^n$, finding an optimal transport map is finding the weight vector $\mathbf{w} = \{w_k\}_{k=1}^n$ such that the power cells $\mathrm{Pow}(\hat{\mathbf{v}}_k, \hat{\mathbf{V}}) \cap \mathbf{B}$ respect the capacity constraints given by $\mathbf{\Lambda}$. These weights are a global minimizer of the convex function $\Phi(\mathbf{w})$ [Mér11].

$$\Phi(\mathbf{w}) = \sum_{\hat{\mathbf{v}}_k \in \hat{\mathbf{V}}} \left( -\lambda_k w_k - \int_{\mathrm{Pow}(\hat{\mathbf{v}}_k, \hat{\mathbf{V}}) \cap \mathbf{B}} (\|\mathbf{v}_k - \mathbf{x}\|^2 - w_k) d\mu(\mathbf{x}) \right)$$

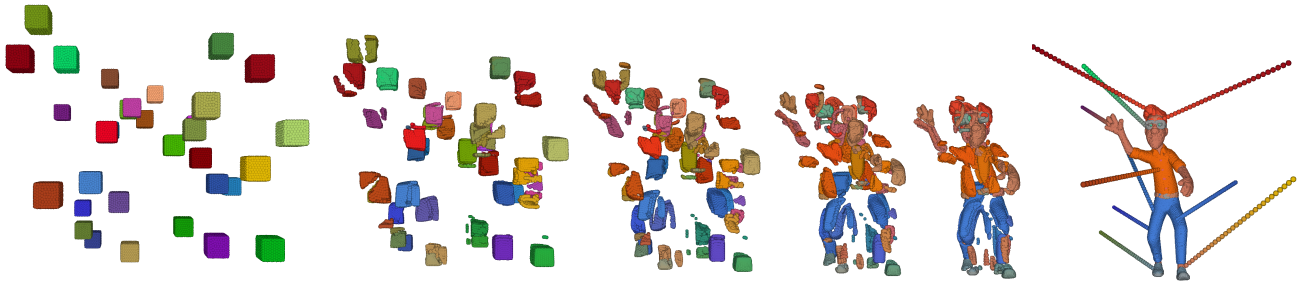


Figure 9: Linear interpolation of the transport map obtained using optimal transport between 6D colored meshes. Right figure also depicts some trajectories of this interpolation for some selected sites. Vincent model courtesy of Andy Goralczyk, Blender cloud.
Conditions : 15k sites distributed on a source mesh (left) and transported to a target mesh with 30k faces (right). Computation terminated using 7079 iterations in 5h54min that is 1 iteration every 3 seconds.

The gradient of $\Phi(\mathbf{w})$ is as follows :

$$\frac{\partial \Phi}{\partial w_k}(\mathbf{w}) = \int_{\mathrm{Pow}(\hat{\mathbf{v}}_k, \hat{\mathbf{V}}) \cap \mathbf{B}} d\mu(\mathbf{x}) - \lambda_k$$

**Implementation details** Minimizing this convex function using L-BFGS requires computing both the value of the function and its gradient. In our runs, we used the same stopping condition as [Mér11], namely $\|\nabla\Phi(\mathbf{w})\|_\infty \leq 10^{-6}$.

The evaluation of $\Phi(\mathbf{w})$ requires the computation of the power diagram which is equivalent to a restricted diagram (Proposition 1). As for the gradient, it is a vector of the size of the number of sites. For a given $w_k$ at iteration $i$, the value of the gradient is the difference between the area of the power cell restricted to target mesh **B** and the area $\lambda_k$ of the Voronoi cell restricted to source mesh **A** at iteration 0 (when weights were equals to 0). One can note that when the areas of the restricted power cells on mesh **B** are equal to $\lambda_k$, the gradient vanishes, the transportation is complete. The integral part of $\Phi(\mathbf{w})$, can be computed as in [LL10, Equation 4].

**Discussion** Optimal transport requires to compute iteratively thousands of restricted Voronoi diagrams before reaching an optimal set of weights. Optimal transport is a typical case of failure for *Security Radius* since a set of sites (and their associated mass) distributed on mesh **A** needs to be used to compute a Voronoi diagram restricted to a mesh **B**. Therefore, sites are likely to be far from the restricted domain. Even in case of overlapping between **A** and **B** (limited distance between the sites and the target mesh **B**), the weights of the power diagram add an additional dimension to the sites, increasing all the distances. In optimal transport, sites will *always* lie far from the surface. For those reasons, the *Security radius* algorithm is not suitable for an optimal transport application. This is experimentally confirmed in Table 1, which describes the timings for solving optimal transport between two copies of the same mesh translated with an offset going from full overlap to no intersection at all.

To achieve optimal transport in reasonable time [Mér11, Lév15] used *classical algorithm* (Section 2.3.1). This approach is only efficient in 2D and 3D but blows in higher dimensions (Figure 10). Using our algorithm as a replacement to compute power diagrams makes it possible to realize optimal transport between featured meshes which neither *Security radius* nor *classical algorithm* can do. In Figure 9 we present an optimal transport computation realized in 6D between colored meshes (x,y,z,r,g,b). From the output of the computation (the transport map), we realized a linear interpolation between the initial position of sites and their corresponding barycenters on the power cells restricted to **B** at last iteration. Looking at the trajectories, one can notice the trade-off made between mapping a similar color and the closest point in a 3D sense.

| Target mesh offset | 0 | 0.5 | 1 |
|---|---|---|---|
| Security Radius | 0,3s | 2h 40m | 3h 31m |
| Corner Validation (mono) | 0,05s | 37s | 48s |
| Corner Validation (parallel) | 0,04s | 5,3s | 7,7s |

Table 1: Optimal transport between 2 identical meshes normalized in [0,1] and translated between 0 and 1 using an offset.
Conditions : 1k sites generated on the surface of the mesh **A** of 3k faces, transported to a copy of **A** translated by the offset.

## 5. Results

**Setup** In this section, we present tests and particular configurations with the relative performance of the algorithms used. We compare our algorithm with 3 others, namely Security radius, GEOGRAM [LB12] and DT$d$D (CGAL) [BDH09]. "Security radius" refers to our implementation of the Lévy and Bonneel algorithm that they implemented in GEOGRAM. We use our implementation of their algorithm only for tests on high dimensional meshes to demonstrate that the concept works well for high dimensions. Although, our algorithm and GEOGRAM are parallel, we used a single-threaded version of these algorithms to compare with the others. Our timings are an average over 10 experiments for more stability. All tests are ran on a 2×8 hyper-threaded cores architecture (two CPUs Xeon E5-2640 v3 of 8 hyper-threaded cores each).

In Section 5.1 we illustrate the superiority of the nearest neighbor approach when the dimension of the ambient space increases. In Section 5.2 and 5.3 we show how our algorithm behaves according to the distribution of the sites. In particular, we show that it

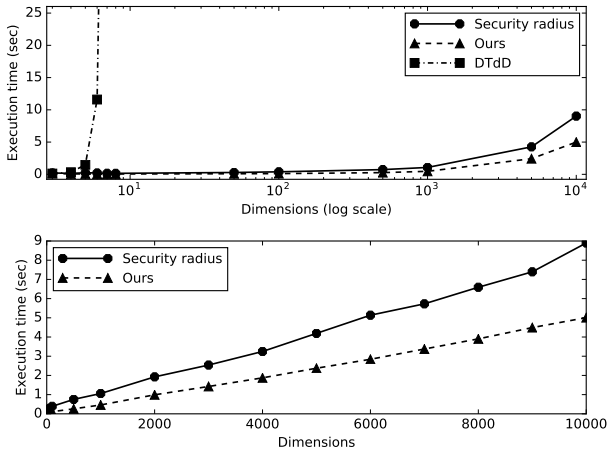
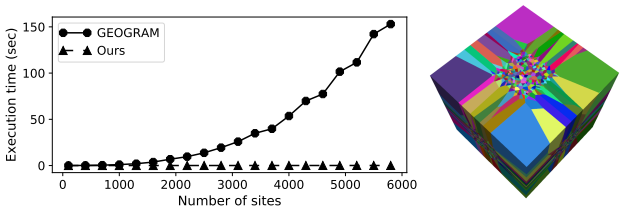



Figure 10: Comparison of the behavior of the classical approach 2.3.1 (DT$d$D) and the nearest neighbors approach 2.3.2 (Security radius and ours) when the dimension of the ambient space increases.
Conditions : 3k sites generated on the surface of the $d$D meshes. The $d$D meshes are randomly generated as a soup of 65 2-simplex with vertices in $[0,1]^d$. DT$d$D has only been ran until dimension 8 due to the large increase in execution time beyond.




(a) Performance for sites far form the domain (b) Related output

Figure 11: (a) Performances of our algorithm compared to GEOGRAM configured with security radius [LB12,Lév16]. Their performance for sites far from the domain drops while our algorithm scales smoothly (see § 2.3.2.3 and Section 5.3).
Conditions : (b) Restricted Voronoi diagram of a cube with sites generated on an inner smaller cube.

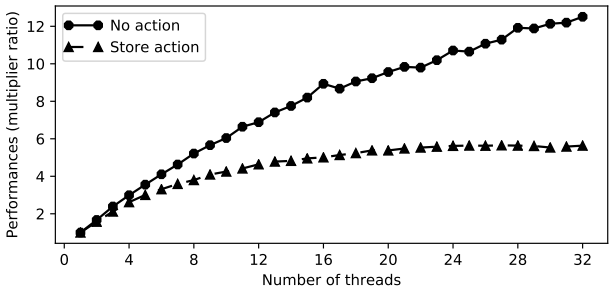



Figure 12: Parallel performances of our algorithm versus the number of threads depending on the action.
Conditions : 100k sites generated on the surface of a 2k faces mesh.

remains efficient when the sites move away from the surface of the mesh. Finally, in Section 5.4 we presents the performances of our parallel implementation.

### 5.1. Sites and mesh in high dimension

In Figure 10 we present the evolution of the execution time when increasing the dimension of the ambient space. Every triangle of this $d$D mesh is a 2-simplex with vertices in $\mathbb{R}^d$. The DT$d$D algorithm [BDH09] (for Delaunay complexes in $d$-Dimensions) is computed using CGAL. Computing DT$d$D is the mandatory first step when using the classical approach 2.3.1. The execution time of the computation of a $d$-dimensional restricted Voronoi diagram using this approach can therefore only be higher because it requires additional steps. This figure clearly shows that this approach is prohibitive when dimensions get bigger.

### 5.2. Sites on the surface of the mesh and complexity

The bottom of the Figure 10 shows the same test as the above part without the log scale in x and the DT$d$D execution time curve. This bottom part is there to demonstrate the good behavior of the nearest neighbor approach, and that our algorithm shows comparable performance in the case when the sites are located on the surface.

### 5.3. Sites far from the surface of the mesh

The security radius used in Lévy and Bonneel [LB12] and Voro++ [Ryc09] algorithms degenerates when the sites are far from the surface (§ 2.3.2.3). In Figure 11a, we illustrate this limitation. For that we used a cube as input mesh and generated sites inside it on a smaller cube (with a scale ratio of 0.25), see Figure 11b. We find that the GEOGRAM algorithm is not designed for this type of use. In contrast, our algorithm still behaves nicely.

### 5.4. Parallel performance

The Figure 12 shows the evolution of the performance according to the number of threads used. The multiplier ratio is computed by dividing the execution time at the current number of threads by the execution time of the single thread run. We describe as *action* the treatment(s) realized during the RVD computation with a portion of restricted cell that has just been computed. The Store Action reconstructs and reconnects the diagram cells. This action induces a bottleneck due to concurrent writing, requiring a "one-thread-at-a-time" merging. However, in other cases such as computing the area and barycenter of the cells leads to a thread safe action, resulting in much better performance. This is, for example, the case for optimal transport [Mér11].

### 5.5. Influence of the site distribution

We used different methods for generating sites around a mesh. With the same amount of 100k sites, and the same input mesh, depending

on the sites distribution we can have different behaviors. In Table 2, we present the execution time of three different random site distributions :

**Surface** sites are generated on the surface of the input mesh
**Uniform** sites are generated in the bounding box of the input mesh
**Lines** sites are generated in the bounding box of the input mesh on lines aligned on the 3D axis

This table also presents the number of nearest neighbor requests that have been made and the number of polygons that have been used for the restricted Voronoi diagram. The polygons does not exactly correspond to the number of restricted cells since a restricted cell is made of as many polygons as the triangles it intersects. Still, this figure provides insight on the number of sites that contribute to the diagram.

When sites are distributed on the surface the number of polygons is almost equivalent to the number of sites since all sites contribute. Considering the uniform site distribution, the number of sites contributing to the diagram is much lower since all sites that are far from the surface are hidden by closer ones. Obtaining the diagram for this distribution is, therefore, faster. This table confirms the idea illustrated in Figure 3 about the fact that our algorithm will only consider sites that contribute to the diagrams.

| Sites generation | Surface | Uniform | Lines |
|---|---|---|---|
| Time | 1395 ms | 30 ms | 6164 ms |
| # NN requests | 344k | 6.4k | 73k |
| # Polygons | 103k | 1.9k | 22k |

Table 2: Sites generation methods influence for 100k sites

We already saw in Section 5.2 and 5.3 that our algorithm was scaling nicely for case 1 and 2. However, generating sites on lines could lead to pathological case where the complexity of the diagram becomes quadratic in the number of sites. In such cases, we find that our algorithm behaves well and manages to get the diagram in a reasonable time.

## 6. Conclusion & Future work

We presented a new algorithm for computing Voronoi diagrams of sites in $\mathbb{R}^d$ restricted to a 2D triangulated mesh. Compared to the existing algorithms, ours shows comparable performance when the others perform well. When the sites are far from the surface, the state of the art algorithms become prohibitive while ours continues to show good behavior. To do so, we derived a new search mechanism to find the set of sites contributing to the restricted Voronoi diagram, along with a new criterion to ensure that the diagram we computed is correct. We believe our corner validation criterion can be easily integrated into existing software and could therefore benefit to many applications, in particular for mesh generation and surface reconstruction.

For future work, our algorithm could be improved in several ways. First, in his work, de Silva [Sil08] studies the notion of approximated witnesses. A potential application in our case would be to derive a new criterion based on *approximate* nearest neighbors, thus improving the performance of the queries. We would also like to build exact predicates to make our algorithm more robust to degenerate cases and provide guarantees of correctness of the result. This could be possible using the Predicate Construction Kit [Lév16]. Finally a more open problem would be to restrict the diagram to a wider class of subsets. In particular, for surface reconstruction, such a set could be the $\alpha$-offset of a point cloud or cocones [ACDL00].

## References


[AB06] ALIPRANTIS C. D., BORDER K. C.: *Infinite dimensional analysis: a hitchhiker's guide*. Springer Verlag, 2006. 5

[ACDL00] AMENTA N., CHOI S., DEY T. K., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry* (2000), ACM, pp. 213–222. 10

[ACR03] AMENTA N., CHOI S., ROTE G.: Incremental constructions con brio. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 2003), SCG '03, ACM, pp. 211–219. 2

[AHA98] AURENHAMMER F., HOFFMANN F., ARONOV B.: Minkowski-Type Theorems and Least-Squares Clustering. *Algorithmica 20*, 1 (jan 1998), 61–76. 7

[AK96] AURENHAMMER F., KLEIN R.: *Voronoi Diagrams : Handbook of Computational Geometry, Chapter V*. Elsevier Science Publishing, 1996. 5

[Aur87] AURENHAMMER F.: Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing 16*, 1 (1987). 5

[BDH96] BARBER C., DOBKIN D., HUHDANPAA H.: The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS) 22*, 4 (1996), 469–483. 2, 3

[BDH09] BOISSONNAT J.-D., DEVILLERS O., HORNUS S.: Incremental construction of the Delaunay graph in medium dimension. In *Annual Symposium on Computational Geometry* (Aarhus, Denmark, June 2009), pp. 208–216. 2, 8, 9

[BG14] BOISSONNAT J.-D., GHOSH A.: Manifold reconstruction using tangential delaunay complexes. *Discrete and Computational Geometry 51*, 1 (2014), 221–267. 5

[BPC16] BONNEEL N., PEYRÉ G., CUTURI M.: Wasserstein Barycentric Coordinates: Histogram Regression Using Optimal Transport. *ACM Transactions on Graphics 35*, 4 (2016). 2, 7

[CGA16] CGAL : Computational Geometry Algorithms Library. http://www.cgal.org/, 1995-2016. 2, 3, 5, 6, 7

[DGH01] DEY T. K., GIESEN J., HUDSON J.: Delaunay based shape reconstruction from large data. In *Parallel and Large-Data Visualization and Graphics, 2001. Proceedings. IEEE 2001 Symposium on* (2001), IEEE, pp. 19–146. 3

[FCRT14] FLAMARY R., COURTY N., RAKOTOMAMONJY A., TUIA D.: Optimal transport with Laplacian regularization. *NIPS* (2014). 2

[FLA15] FLANN : Fast Library for Approximate Nearest Neighbors. http://www.cs.ubc.ca/research/flann/, 2015. 2, 3, 6, 7

[LB12] LÉVY B., BONNEEL N.: Variational anisotropic surface meshing with voronoi parallel linear enumeration. *21st International Meshing Roundtable* (2012). 1, 2, 3, 4, 7, 8, 9

[Lév15] LÉVY B.: A Numerical Algorithm for L 2 Semi-Discrete Optimal Transport in 3D. *ESAIM: Mathematical Modelling and Numerical Analysis 49*, 6 (nov 2015), 1693–1715. 2, 3, 7, 8

[Lév16] LÉVY B.: Robustness and efficiency of geometric programs: The predicate construction kit (pck). *Computer-Aided Design 72* (2016), 3–12. 7, 9, 10

[LL10] LÉVY B., LIU Y.: Lp Centroidal Voronoi Tesselation and its applications. *ACM Trans-actions on Graphics 29*, 4 (2010). 8

[Llo82] LLOYD S. P.: Least Squares Quantization in PCM. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 2 (1982). 7

[LWL*09] LIU Y., WANG W., LÉVY B., SUN F., YAN D.-M., LU L., YANG C.: On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Transactions on Graphics 28*, 4 (aug 2009), 1–17. 7

[MA97] MOUNT D. M., ARYA S.: ANN: A library for approximate nearest neighbor searching. In *CGC Workshop on Computational Geometry* (1997), pp. 33–40. 2, 3

[McM70] MCMULLEN P.: The maximum numbers of faces of a convex polytope. *Mathematika 17*, 02 (1970), 179–184. 2

[Mér11] MÉRIGOT Q.: A multiscale approach to optimal transport. *Computer Graphics Forum 30*, 5 (2011), 1583–1592. 2, 3, 7, 8, 9

[Mon81] MONGE G.: Mémoire sur la théorie des déblais et des remblais. *His- toire de l'Académie Royale des Sciences de Paris* (1781), 666–704. 7

[NYL14] NIVOLIERS V., YAN D.-M., LÉVY B.: Fitting polynomial surfaces to triangular meshes with voronoi squared distance minimization. *Engineering with Computers 30*, 3 (2014), 289–300. 3

[OBSS00] OKABE A., BOOTS B., SUGIHARA K., S.N.CHIU: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, 2000. 2

[RTG00] RUBNER Y., TOMASI C., GUIBAS L.: The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision 40*, 2 (2000), 99–121. 7

[Ryc09] RYCROFT C.: Voro++: A three-dimensional voronoi cell library in c++. *Lawrence Berkeley National Laboratory* (2009). 2, 3, 9

[SdGP*15] SOLOMON J., DE GOES F., PEYRÉ G., CUTURI M., BUTSCHER A., NGUYEN A., DU T., GUIBAS L.: Convolutional wasserstein distances. *ACM Transactions on Graphics 34*, 4 (jul 2015), 66:1–66. 2, 7

[SH74] SUTHERLAND I. E., HODGMAN G. W.: Reentrant polygon clipping. *Commun. ACM 17*, 1 (Jan. 1974), 32–42. 3

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222. 2, 3

[She02] SHEWCHUK J. R.: Delaunay refinement algorithms for triangular mesh generation. *Computational geometry 22*, 1-3 (2002), 21–74. 5

[Sil08] SILVA V. D.: A weak characterisation of the delaunay triangulation. *Geom Dedicata* (2008). 4, 5, 10

[Vil09] VILLANI C.: Optimal transport. Old and new. xxii, 973 p. 7

[YLL*09] YAN D. M., LÉVY B., LIU Y., SUN F., WANG W.: Isotropic remeshing with fast and exact computation of restricted voronoi diagram. *Eurographics Symposium on Geometry Processing 28*, 5 (2009). 3

[YWLL13] YAN D.-M., WANG W., LÉVY B., LIU Y.: Efficient computation of clipped voronoi diagram for mesh generation. *Computer-Aided Design 45*, 4 (2013), 843–852. 3