Restricting Voronoi diagrams to meshes using corner validation
Maxime Sainlot, Vincent Nivoliers, Dominique Attali

To cite this version:

HAL Id: hal-01626140
https://hal.science/hal-01626140
Submitted on 30 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Restricting Voronoi diagrams to meshes using corner validation

M. Sainlot\textsuperscript{1,2}, V. Nivoliers\textsuperscript{1}, D. Attali\textsuperscript{2}

\textsuperscript{1}Université de Lyon, LIRIS UMR 5205, France
\textsuperscript{2}Université Grenoble-Alpes, CNRS, GIPSA-lab UMR 5216, France

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 $V = \{v_i\}_{i=1}^{n} \subset \mathbb{R}^d$ and a mesh $X$ with vertices in $\mathbb{R}^d$ connected by triangles, the restricted Voronoi diagram partitions $X$ by computing for each site the portion of $X$ 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 \cite{LB12} based on nearest neighbor queries. We show that their method is limited when the sites are not located on $X$. 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. Two approaches currently exist: computing the full diagram before intersecting or using nearest neighbors. Nearest neighbor approaches (like ours) are to be used when any of the following is true:

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

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.
In any case, our algorithm has similar performances than the existing nearest neighbor based algorithm (security radius), and solves the issues of the latter when the sites are not located on the restriction domain.

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 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 noticed 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 $A$ onto $B$ 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 on the set $X$ but ours. In this case, sites are not located on the domain, the issues of the latter when the sites are not located on the restriction domain.

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.

We did not implement robust exact predicates in our implementation, as compared to [LB12]. We detail in Section 3.5 the behavior of our algorithm when floating point errors arise.

2. Background

2.1. Voronoi diagrams

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

$$\text{Vor}(v_k, V) = \left\{ x \in \mathbb{R}^d, \|v_k - x\| \leq \|v_\ell - x\| \text{ for all } \ell \neq k \right\}.$$

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

The Voronoi diagram $\text{Vor}(V)$ is the collection of all Voronoi cells, $\text{Vor}(V) = \{\text{Vor}(v, V), \forall v \in V\}$ (see Figure 2a).

Voronoi diagrams and their dual, Delaunay complexes have been studied a lot [OBS00] 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 [McM07] 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 $X \subseteq \mathbb{R}^d$, the restricted Voronoi cell of $v_k$ is defined as the intersection:

$$\text{Rvd}(v_k, X) = \left\{ x \in X, \|v_k - x\| \leq \|v_\ell - x\| \text{ for all } \ell \neq k \right\}.$$

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

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

- an intersection between $T$ and the bisector of $(v_k, v_i)$ triggers the computation of $\text{Vor}(v_k, V) \cap T$;
- a piece of an edge between $T$ and another triangle $T'$ triggers the computation of $\text{Vor}(v_k, V) \cap T'$.

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

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 $v_k$.

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 lower 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 $v_k$, a triangle $T$ and a candidate polygon $P \subset T$ for the restricted cell $\text{Vor}(v_k, V) \cap T$, let us consider a minimal ball centered on $v_k$ and containing $P$. Let $r$ be the radius of this ball. Since the ball is centered on $v_k$, if another
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 T rather than the site v_k. Therefore, by definition of the Voronoi diagram, the result of a query performed from 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.

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 S ⊂ R^d be a convex set. An extreme point of S is a point s ∈ S such that no segment entirely in S contains s in its interior.

Now let Ex(S) denote the set of extreme points of a convex set and Conv(S) denote the convex hull of a set S, and let us define the notion of witness set for a restricted Voronoi diagram, named after the work of De Silva [De 08].

Definition 2 (Witness set) Let V ⊂ R^d be a set of sites and let Vor(v, V) denote the Voronoi cell of a site v ∈ V. Given a set X ⊂ R^d, the witness set of V with respect to X is defined as

W(V, X) = \bigcup_{v ∈ V} \{ Ex(Conv(Vor(v, V) ∩ 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 V ⊂ R^d be a set of sites and U ⊂ V be a subset of the sites. Let X ⊂ R^d be a compact set. If for all w ∈ W(U, X), w ∈ Vor(u, V) for some site u ∈ U then for all u ∈ U, Vor(u, U) ∩ X = Vor(u, V) ∩ X.

Proof We will prove the result by mutual inclusion. By definition of the Voronoi diagram we have that Vor(u, V) ⊂
Vor(u, U) since U is a subset of V. Therefore Vor(u, V) ∩ X ⊂ Vor(u, U) ∩ X.

The convex hull of a set S is the union of all the convex combinations of points in S. A point in Conv(S) that is not in S is therefore a convex combination of two or more points in S and is therefore in the interior of a segment contained in S. The extreme points of the convex hull of S are therefore in S.

For every extreme point w of Ex(Conv(Vor(u, U) ∩ X)), our hypothesis state that there exists a site u ∈ U such that w ∈ Vor(u′, V). Since u′ ∈ U and w ∈ Vor(u, U), w is therefore equidistant to u′ and u and w ∈ Vor(u, V). This leads to Ex(Conv(Vor(u, U) ∩ X)) ⊂ Vor(u, V) and since Voronoi cells are convex we finally have that

\[ \text{Conv}(\text{Ex}(\text{Conv}(\text{Vor}(u, U) \cap X))) \subset \text{Vor}(u, V). \]

X is compact and Vor(u, U) is closed. Vor(u, U) ∩ X is therefore compact, and as a corollary of Carathéodory’s Convexity Theorem [AB06, Corollary 5.33] Conv(Vor(u, U) ∩ X) is compact as well. Using the Krein-Milnner Theorem, we obtain

\[ \text{Conv}(\text{Ex}(\text{Conv}(\text{Vor}(u, U) \cap X))) = \text{Conv}(\text{Vor}(u, U) \cap X), \]

and therefore

\[ \text{Vor}(u, U) \cap X \subset \text{Conv}(\text{Vor}(u, U) \cap X) \subset \text{Vor}(u, V). \]

Finally

\[ \text{Vor}(u, U) \cap X \subset \text{Vor}(u, U) \cap X. \]

\[
\Box
\]

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 spatial 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 performed 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 [De 08] 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 through our iterations, update it after site insertions and mark them as valid. We do so using a power diagram of the dimension of X. In our case, X is a collection of triangles and on each triangle, we use a 2D power diagram in the support plane of the triangle. When introducing new sites to be considered for a triangle, we project them to the support plane of the triangle and their weights in the power diagram account for the information lost in the projection.

3.3.1. Power diagram and regular triangulation

The power diagram is a generalization of the Voronoi diagram in which each site \( v_k \) is given a weight \( w_k \). Voronoi diagrams are a special case of power diagrams where all sites \( v_k \) have a weight \( w_k = 0 \). The regular triangulation is the dual of the power diagram.

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

\[ \text{Pow}(v_k, V) = \left\{ x \in \mathbb{R}^d, \|v_k - x\| - w_k \leq \|v_\ell - x\| - w_\ell, \forall \ell \neq k \right\} \]

3.3.2. Support frame of a triangle

Given a triangle \( T \) of vertices \( x_1, x_2, x_3 \in \mathbb{R}^d \), we can compute a 2D orthonormal frame \((e_1, e_2)\) centered on one of the vertices of \( T \) (i.e. \( x_1 \)) as follows:

\[ e_1 = \frac{x_2 - x_1}{\|x_2 - x_1\|}, \quad e_2 = \frac{(x_3 - x_1) - (x_2 - x_1).e_1}{\|x_3 - x_1 - (x_2 - x_1).e_1\|} \]

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

\[ x = x^t + x^\perp \]

\[ x^t = \|x - x_1\|.e_1 + \|x - x_2\|.e_2 \]

\[ x^\perp = x - x^t \]

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 \( V \subset \mathbb{R}^d \) restricted to a hyperplane is known [Aur87, AK96, CGA16] to be equivalent to a power diagram in the hyperplane. This relation was extended by Boissonnat and Gosh [BG14, Lemma 2.2] to the following:

**Proposition 1** Let \( T \) be an affine subspace of \( \mathbb{R}^d \). Let \( V = \{v_k\}_{k=1}^n \subset \mathbb{R}^d \) be a set of sites, \( V^t = \{v_k^t\}_{k=1}^n \) their projections on \( T \), \( V^\perp = \{v_k^\perp\}_{k=1}^n \) their normal components. The Voronoi diagram of \( V \) restricted to \( T \) is the power diagram of \( V^t \) in \( T \) with weights \( W = \{w_k\}_{k=1}^n = \{-v_k^t\}_{k=1}^n \).
As in [BG14], we reduce a dD Voronoi diagram restricted to a 2D plane to a 2D power diagram on T using V⊥ as positions for the sites and weights derived from V⊥. 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.

### 3.4. Algorithm

Algorithm 1 and Figure 6 present our algorithm. The set of Voronoi corners is initialized to the 3 vertices of T. At every iteration, we check the nearest site of each new Voronoi corner (NN(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 \( w_2 = -v_2^2 \) (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 T is complete.

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

Most of the time, a Voronoi corner is shared among 2 or 3 restricted cells. This means that the Algorithm 1 would naively 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.

### 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 realize many nearest site searches from new corners to find new sites (\( o \leftarrow \text{NN}(x) \)). This is achieved 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 \( \mathbb{R}^2 \) (change of basis). That way, the global structure is only built once and can be queried from any triangle.

![Figure 6: Iterations of Algorithm 1](image)

**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 \( o \) 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).

### Algorithm 1: Compute Rvd(V, T) for a single triangle T

**Input:** T: a triangle \( T \in X \)

**Data:** new_sites: a table of untreated new sites

new_corners: a table of new Voronoi corners

PD: the power diagram

**Result:** Rvd(V, T): the Voronoi diagram of X restricted to T

```plaintext
function compute_rvd(T)
  /* Initialization */
  compute tangent plane /* Section 3.3.2 */
  foreach vertex xi of triangle T do
    add NN(xi) to new_sites
  while new_sites is not empty do
    /* Insert contributing sites in the power diagram */
    foreach site in new_sites do
      project site in tangent plane
      add site to PD
      extract new_corners
      empty new_sites
      /* Validate the corners, search for new sites */
      foreach xi in new_corners do
        if NN(xi) is not already in PD then
          add NN(xi) to new_sites
      end if
    end foreach
  end while
  return PD \( \cap T \)
```

**Algorithm 1:** Compute Rvd(V, T) for a single triangle T
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 queries become more and more subject to errors. Our computations make 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 computing nearest neighbor requests.

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 of radius 1. 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.

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 [Vil08] 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 [SdGP15].

We applied our algorithm to the problem of \( L^2 \) semi-discrete optimal transport [Mér11]. Similarly, [Lév16] realize semi-discrete optimal transport between tetrahedral meshes in 3D.

Semi discrete optimal transport  Given two meshes \( A \) (the source) and \( B \) (the target), we discretize the source (see Figure 8a) by distributing a set of sites \( V \) over the mesh \( A \). Sites can be distributed homogeneously using Lloyd relaxation [Llo82] or Newton-type methods for faster convergence [LWL’09]. Every site \( v_k \) 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 \( A \). The target mesh \( B \) has a measure \( \mu \) with density. We aim to transport the mass contained on sites to the target location on \( B \). That is to say, obtain a mapping between \( V \) and portions of \( B \) 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 \( V = \{v_k\}_{k=1}^n \) with masses \( \lambda_k \), finding an optimal transport map is finding the weight vector \( w = \{w_k\}_{k=1}^n \) such that the power cells \( \text{Pow}(v_k, V) \cap B \) respect the capacity constraints given by \( A \). These weights are a global minimizer of the convex
function $\Phi(w)$ [Mér11].

$$\Phi(w) = \sum_{v_i \in V} \left( -\lambda_k w_i - \int_{\text{Pow}(v_i, V) \cap B} \left( \|v_i - x\|^2 - w_k \right) d\mu(x) \right)$$

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

$$\frac{\partial \Phi}{\partial w_k} = \int_{\text{Pow}(v_i, V) \cap B} d\mu(x) - \lambda_k$$

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 $\|\nabla \Phi(w)\|_\infty \leq 10^{-6}$.

The evaluation of $\Phi(w)$ 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 equal 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 $\Phi(w)$, 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 11). 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.

As a proof of concept, Figure 10 shows matchings obtained using optimal transport in a 2D dimensional feature space. We extended the 3D coordinates of the mesh vertices by computing their heat kernel signatures [SOG09]. The transport plan obtained in this way is more likely to result in matching similar portions of the meshes. Although our results in this field are nowhere near state of the art standards, our performances show that semi-discrete optimal transport can be available in such spaces using our algorithm.

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 (CGAL) [BDH09]. “Security radius” refers to
Figure 10: Matching in 20 dimensional feature space between meshes (using HKS [SOG09]) from Tosca dataset [BBK08]. Top: cat0 and cat8, Middle: michael0 and gorilla1, Bottom: horse10 and dog2. All matching are achieved in less than 50sec in a 4 cores architecture using 200 sites. Matching limitations are visible on the right front leg of the dog-horse figure or in the hip/chest matching from the human-gorilla figure.

![Figure 10: Matching in 20 dimensional feature space between meshes (using HKS [SOG09]) from Tosca dataset [BBK08].](image1)

Figure 11: Comparison of the behavior of the classical approach 2.3.1 (DTdD) 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 dD meshes. The dD meshes are randomly generated as a soup of 65 2-simplex with vertices in \([0, 1]^d\). DTdD has only been ran until dimension 8 due to the large increase in execution time beyond.

![Figure 11: Comparison of the behavior of the classical approach 2.3.1 (DTdD) and the nearest neighbors approach 2.3.2 (Security radius and ours) when the dimension of the ambient space increases.](image2)

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 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 11 we present the evolution of the execution time when increasing the dimension of the ambient space. Every triangle of this dD mesh is a 2-simplex with vertices in \(\mathbb{R}^d\). The DTdD algorithm [BDH09] (for Delaunay complexes in \(d\)-Dimensions) is computed using CGAL. Computing DTdD 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 gets larger.

5.2. Sites on the surface of the mesh and complexity

The bottom of the Figure 11 shows the same test as the above part without the log scale in x and the DTdD 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 12a, we illustrate this limitation. For that we used a cube as input mesh and generated sites inside it on a
Performance for sites far from the domain.

**Figure 12:** (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 13:** 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. The Figure 13 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].

5.4. Parallel performance

The Figure 13 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 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.

### Table 2: Sites generation methods influence for 100k sites

<table>
<thead>
<tr>
<th>Sites generation</th>
<th>Surface</th>
<th>Uniform</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1395 ms</td>
<td>30 ms</td>
<td>6164 ms</td>
</tr>
<tr>
<td># NN requests</td>
<td>344k</td>
<td>6.4k</td>
<td>73k</td>
</tr>
<tr>
<td># Polygons</td>
<td>103k</td>
<td>1.9k</td>
<td>22k</td>
</tr>
</tbody>
</table>

6. Conclusion & Future work

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.

For future work, our algorithm could be improved in several ways. First, in his work, de Silva [De 08] 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 α-offset of a point cloud or cocones [ACDL00].

Acknowledgments

This work was funded by a PhD grant from the Rhône-Alpes region and two ANR projects: TopData ANR-13-BS01-0008 and MAGA ANR-16-CE40-0014.

References

[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, 10
[She02] Shewchuk J. R.: Delaunay refinement algorithms for triangular mesh generation. Computational geometry 22, 1-3 (2002), 21–74. 5

submitted to Eurographics Symposium on Geometry Processing (2017)