Master of Science in Informatics at Grenoble
Master Mathématiques Informatique - spécialité Informatique
option *GVR*

Adaptive point cloud denoising

MEYRON Jocelyn

26th June 2015

Research project performed at GIPSA-lab

Under the supervision of:
ATTALI Dominique, GIPSA-lab
MÉRIGOT Quentin, CEREMADE, Université Paris-Dauphine

Defended before a jury composed of:
Mr CROWLEY James
Mr BOYER Edmond
Mr LAYAIDA Nabil
Mr VAUFREYDAZ Dominique
Mr FRANCO Jean-Sébastien
Mr HÉTROY-WHEELER Franck

June 2015

Abstract

During this internship, we were interested in the denoising, smoothing of point clouds sampled on an unknown surface. In order to do that, we will use a mean curvature flow approach: points are moved while minimizing an energy, this energy being the area of the underlying surface. But, we do not know the underlying surface, only points sampled on it. To approximate the area, we will use the volume of a union of balls centered on the point cloud and a gradient descent algorithm. We will also be interested in another kind of flow: an anisotropic one. For this particular flow, the idea is to replace the union of balls by a union of convex polyhedra. The magnitude of the smoothing will be more important in some directions dictated by the choice of a convex polyhedron.

This report is divided as follows. Firstly, we will give a detailed introduction about point cloud smoothing: the existing techniques and why using a mean curvature flow can be interesting. Then, we will study the two dimensional case. We will show that this technique can be used to simulate a discrete mean curvature flow and to smooth point clouds. We will also see that this technique can be used to estimate the mean curvature. Many examples will be studied. Secondly, we will look at the anisotropic flow in 3D. Several theoretical and practical issues will arise. We will explain the choices that were mode to solve these issues. Finally, we will look at the theory behind our approach. We will prove that, under some conditions, the area of a surface can be well approximated by the volume of a union of balls. We will also prove that our algorithm is, indeed, a discretization of the continuous mean curvature flow.

Résumé

Durant ce stage, nous nous sommes intéressés au débruitage, ou lissage, de nuages de points échantillonnés sur une surface inconnue. Pour faire cela, nous avons utilisé une approche basée sur le flot de courbure moyenne : nous faisons évoluer le nuage de points tout en minimisant une énergie, l'aire de la surface sous-jacente. Mais, dans notre cas, la surface est inconnue, nous ne connaissons que des points échantillonnés sur celle-ci. Pour approcher l'aire de la surface, nous avons utilisé le volume de l'union des boules centrées sur le nuage de point et un algorithme basé sur une descente de gradient. Nous nous sommes aussi intéressés à un autre type de flot : un flot anisotropique. Pour ce flot, l'idée est de remplacer l'union des boules par une union de polyèdres convexes. La magnitude du flot sera plus importante dans certaines directions privilégiées, dictées par le choix du polyèdre.

Ce rapport est structuré de la manière suivante : tout d'abord, nous donnerons une introduction détaillée sur les techniques existantes de lissage de nuages de points et pourquoi il est intéressant d'avoir choisi une approche basée sur le flot de courbure moyenne. Ensuite, nous étudierons le cas bidimensionnel. Nous montrerons que cette technique peut être utilisée pour simuler un flot de courbure moyenne discret et pour lisser des nuages de points. Nous verrons aussi que cette technique peut être utilisée pour estimer la courbure moyenne. Beaucoup d'exemples illustreront ces résultats. Ensuite, nous nous intéresserons au cas anisotropique en 3D. Nous serons confrontés à des difficultés aussi bien théoriques que pratiques. Nous expliquerons alors les choix qui ont été faits pour résoudre les problèmes rencontrés. Enfin, nous nous intéresserons à la théorie utilisée derrière notre approche. Nous montrerons que, sous certaines conditions, l'aire de la surface peut être approchée par le volume d'une union de boules. Nous prouverons aussi que notre algorithme est bien une discrétisation du flot de courbure moyenne continu.

Contents

— 1 —
Introduction

A 3D scanner is a commonly used device that measures 3D points on the surface of a physical object. It produces point clouds that can then be used to create more complex and complete digitalized models such as meshes. Several methods exist to build a mesh from a point cloud such that the one described in [1]. But an important problem may arise when processing point clouds: since the 3D scanner is a physically-based device, error can be made during measurements: the resulting point cloud will be noisy. In this work, we are interested in removing the noise from point clouds: this technique is called *denoising* or *smoothing*.

There is another issue: because we only deal with points, we cannot use all the techniques developed in image processing like the Fourier transform, wavelets... A common way to remove the noise from an image is to take its Fourier transform and filter the high frequencies. In order to apply a filter, we need a notion of neighbours and thus some kind of parametrization. An image can be easily parametrized using a grid of pixels. When we have a mesh, we also have a kind of parametrization but when we only have points, we do not have such parametrization.

Several methods already exist to remove noise and get smooth point clouds. They come from different fields: computer vision, computational geometry...

- *Gaussian / Laplacian smoothing*: each point is replaced by another one computed from the nearest neighbours of the initial point. In the Laplacian case, the new point is a convex combination of the neighbours (see [16]).

- *Jet fitting* (see [5]): a jet is a truncated Taylor expansion. Such jets are fitted around points. Jet smoothing operates by projecting the input points on an estimated smooth parametric surface (the so-called jet surface). Jets are good because they intrinsically contain differential information such as normal, curvature...

- *Bilateral smoothing* (see [10]): the algorithm works by first resampling away from the edges i.e. smooth the point cloud around the edges. The second phase of the algorithm consists in inserting points by projecting each point onto the implicit surface patch fitted over its k nearest neighbours. This smoothing algorithm takes into account the sharp edges present in the point set.

During this internship, we will use the following idea: suppose we want to smooth a surface. To do so, one could move each point in the direction of the normal to the surface by a quantity related to the mean curvature at that point. The sign of the mean curvature only depends on the choice of the orientation of the normal. The more curved the surface is, the more important the displacement will be. For an initial surface S_0 and $t \geq 0$, let $S_t = X_t(S_0)$ where $X_t : S_0 \to \mathbb{R}^3$. The evolution of the surface thus produced can be modelled with the following partial differential equation:

$$\frac{\partial X_t(x)}{\partial t} = \vec{H}_{S_t}(x)$$

where $\vec{H}_{S_t}(x)$ is the mean curvature vector of S_t at x (see Definition 7 for a more complete definition). A family of surfaces S_t that satisfies the above equation is said to evolve by mean curvature flow (MCF).

This partial differential equation can be seen as the geometric equivalent of the heat equation where the Laplacian is replaced by the mean curvature vector. Both of the Laplacian and the mean curvature vector can be interpreted as sums of eigenvalues of some matrices: the Hessian for the first quantity and the Shape operator for the second one.

The MCF is well-known for its smoothing properties in the computer graphics community (see [9]) as well as in the image processing one (see [7]). For example, Figure 11 is an illustration of the mean curvature flow on a noisy surface (extracted from [8]).

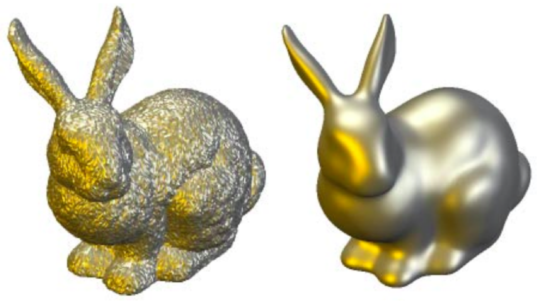


Figure 11 – Noisy surface (on the left) and after evolving by MCF (on the right)

We will show in the last chapter of this report that this flow is related to the variation of the area of the surface. If we try to transform the surface while minimizing the area, we will obtain the same evolution as the one given by the MCF. But let us keep in mind that we do not know the surface, only a finite set of points that sample the surface. So, we will need a way to compute the area without knowing the original surface. A common way of doing so is to consider the volume of a union of balls centered on the points of the cloud. We will show that this quantity is proportional, under some conditions, to the area of the underlying surface. Thus, it suffices to minimize the volume of this union of balls to minimize the area of the underlying surface. We will call this volume the energy $A(P)$ associated to the point cloud $P = \{p_1, \dots, p_N\}$.

Now that we have our energy, we need to minimize it. If we are in three dimensions then we can see the energy A as a function $A : \mathbb{R}^{3N} \to \mathbb{R}$. The main technique used for minimizing a function are based on the computation of its gradient. So, we will need a way to compute the gradient of A. There are many possibilities for doing this: one can use exact formulae if they exist, or finite differences...

In this report, we will use another technique called *Automatic Differentiation*. This technique is described in more details in Appendix *Automatic Differentiation*. In short, it is a technique allowing us to compute the derivatives of any quantity evaluated by a computer program. It exploits the fact that any program can be thought of as a sequence of basic arithmetic operations and elementary functions. Now, using the chain rule and overloading the number type we can compute derivatives without changing too much the cost of the original function. This will allow us to only worry about the computation of $A(P)$ and not of its gradient. This will also allow us to consider different kind of energies: the volume of the union of balls, the area of the boundary of the union and also anisotropic ones. Let us detail the latter point.

Anisotropic smoothing is a type of smoothing where the magnitude will depend on the point and the topology of its neighbourhood. In image processing, this technique is called anisotropic diffusion (see [17]): it reduces the image noise without removing significant parts of the image such as sharp edges, lines...

During this internship, we were interested in a kind of anisotropic evolution based on [6] where the smoothing will be stronger in some privileged directions. These directions will be given by the user.

Anisotropic flow may have applications that we did not have time to explore during this internship. It could be used, for example, in physically based simulations like crystal growth. Another example would be when we process a 3D scan of an object: we want to be able to smooth the point cloud while preserving certain details (such as the signature of the author on a sculpture).

This report is structured as follows: first, we study the two dimensional case. We show that our discrete MCF is able to smooth a point cloud and can be used to estimate the mean curvature vector. Then, we consider the 3D case and study an anisotropic flow. The final part will be dedicated to the theory: we will show that one step of the constructed discrete MCF is an approximation of one Euler step for the continuous MCF.

All the code developed during this internship is available at the following address: `http://tinyurl.com/p5nwrsz`.

— 2 —
2D case

2.1 Introduction

In this part, we focus on point clouds in 2D which sample a planar curve. We will develop algorithms to smooth them by minimizing different energies. First, we will consider for the energy the area of a union of disks centered on the point cloud. Then, we will test other types of energies such as the perimeter of the boundary, a weighted area and a weighted perimeter of the boundary.

Firstly, we will explain how to compute the area of a union of disks. Then, we will run different experiments to check that the gradient of a union of disks is proportional to the mean curvature vector of the underlying curve. Finally, we will run our flows on different point clouds with different parameters to show that it can simulate discrete mean curvature flows and indeed smooth point clouds.

2.2 Area of a union of balls

Let P be a finite set of points in 2D. We call the union of disks with radius r centered on P, the r-offset and we will denote it by P^r. We will first need the definition of the Voronoi diagram of set of points:

Definition 1. *Given a set of points $P \subseteq \mathbb{R}^2$ also called sites, we define the Voronoi cell of $p \in P$ by:*

$$V(p,P) = \{x \in \mathbb{R}^2,\ \forall p' \in P,\ ||p-x|| \leq ||p'-x||\}$$

In other words, the Voronoi cell of p is composed of all the points which are closer to p than to any other points in P. Then, the Voronoi diagram of P is composed of all the Voronoi cells of $p \in P$. It defines a partition of the plane $\mathbb{R}^2$.

For example, Figure 21 provides an example of Voronoi diagram. Now, we have, because the Voronoi diagram defines a partition (see Figure 22 for an illustration):

$$Area(P^r) = Area\left(\bigcup_p B(p,r)\right) = \sum_p Area(V(p,P) \cap P^r) \overset{(\star)}{=} \sum_p Area(V(p,P) \cap B(p,r)) \quad (2.1)$$

where $(\star)$ is the following inclusion:

$$\forall p, q \in P,\ B(q,r) \cap V(p,P) \subseteq B(p,r) \cap V(p,P)$$

Indeed:

$$\begin{aligned} x \in B(q,r) \cap V(p,P) &\implies \|x-q\| \le r \text{ and } \forall p' \in P,\ \|x-p\| \le \|x-p'\| \\ &\implies x \in V(p,P) \text{ and } \|x-p\| \le \|x-q\| \le r \text{ with } p' = q \\ &\implies x \in B(p,r) \cap V(p,P) \end{aligned}$$

This imply:

$$V(p,P) \cap P^r = V(p,P) \cap \bigcup_q B(q,r) = \bigcup_q (V(p,P) \cap B(q,r)) = V(p,P) \cap B(p,r)$$

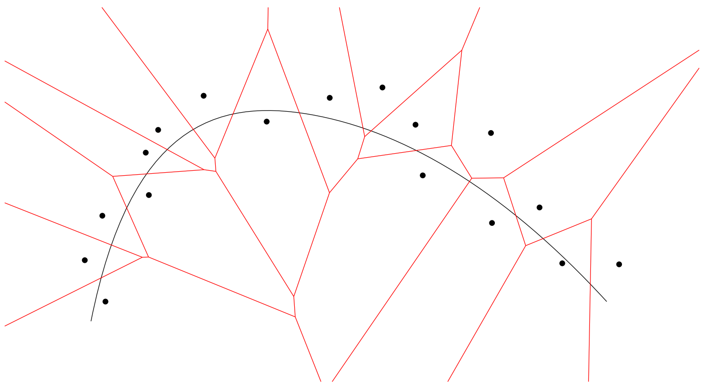


Figure 21 – Points are in black and the boundary of the Voronoi cells are in red

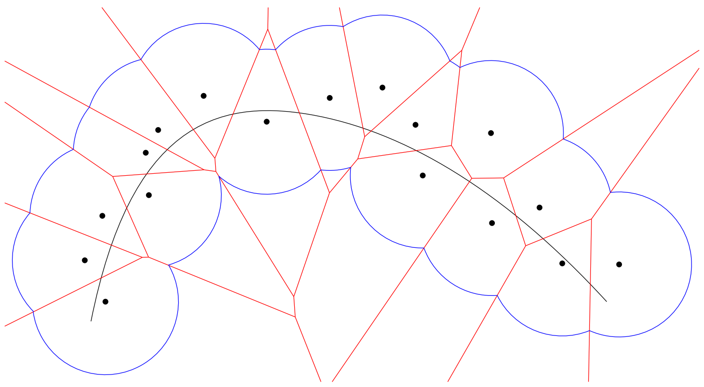
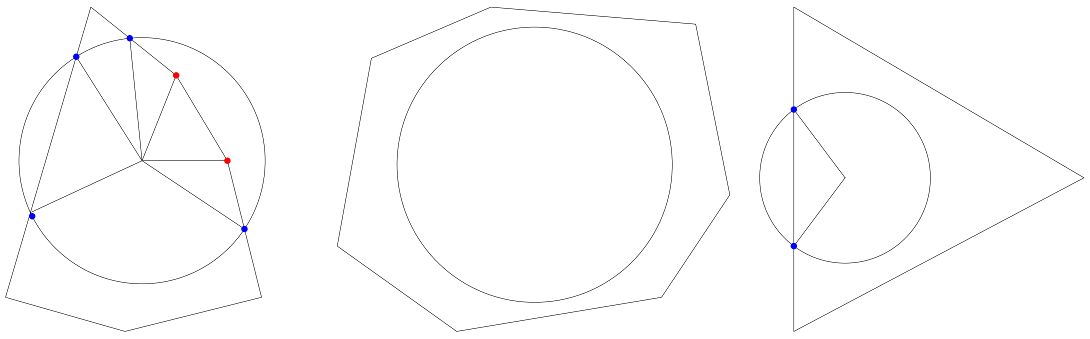


Figure 22 – Points are in black, boundary of the Voronoi cells in red and the r-offset in blue

Algorithm In order to compute the volume of the union, we need to know how to estimate the area of the intersection of a ball and a Voronoi cell. Figure 23 illustrates different cases we may encounter when intersecting a Voronoi cell and a ball in 2D.

We use `CGAL` to compute the Delaunay triangulation of our point set: it is the dual of the Voronoi diagram. A triangle in the Delaunay triangulation corresponds to a vertex in the Voronoi diagram, an edge corresponds to an edge, a site corresponds to a face...



(a) General case (b) No intersections (c) 2 intersections

Figure 23 – Different cases for the intersection between a Voronoi cell and a disk

Given this triangulation, we can compute the vertices composing the boundary of the Voronoi cell of a point by doing the following:

1. Access the neighbouring faces of a site using the `incident_faces` method.

2. Compute the Voronoi vertices (circumcenters) of these faces using the `dual` method.

We are now ready to compute the intersection of the Voronoi cell of p and a disk $B(p,r)$. The boundary of this intersection is composed of segments and circular arcs. There are two types of points connecting those elements: some are Voronoi vertices (in red in Figure 23) and some are intersections of Voronoi edges and the disk $B(p,r)$ (in blue). We will refer to the red points as the interior points and to the blue ones as intersection points.

First, we construct an array of all the points composing the boundary of the intersection. In order to do this, we loop over the vertices composing the Voronoi cell: for any Voronoi vertex v, we have access to the next one in the counterclockwise order and so we construct the corresponding edge. We check whether this edge intersects the ball or not and we construct the intersection points if necessary. In parallel, we maintain a boolean indicating whether the current vertex is inside or not the disk $B(p,r)$. By doing that, we can construct an ordered list of all blue and red points present on the boundary of the intersection.

Then, we loop over this array. For any two consecutive points u and v, we can construct the edge $e = uv$. Depending on whether the points u and v are interior or not, the contribution of e to the global area will be different:

- if u and v are interior points, we add the area of triangle upv.

- if u or v is interior point, we add the area of triangle upv.

- if u and v are intersection points, then if they belong to the same Voronoi edge, we add the area of triangle upv. If not, we add the area of angular sector $(\vec{pu}, \vec{pv})$.

Some special cases need to be handled:

- the boundary of the Voronoi cell lies entirely outside the ball, then we add πr^2 to the area of the union (see Figure 23b).

- the boundary consists of two intersection points p and q, then we add the triangle pvq and the angular sector $\vec{vp}, \vec{vq}$ (see Figure 23c).

- there is only one point on the boundary (can happen if adjacent balls are tangential), then we add πr^2.

This algorithm allows us to compute exactly the area of a union of disks.

We use the same technique for computing the contribution of each disk $B(p,r)$ to the perimeter of the boundary of the union of disks except that if $B(p,r)$ is contained in $V(p,P)$ then the contribution to the perimeter is $2\pi r$ and instead of adding triangles areas or angular sectors, we only add length of circular arcs.

Implementation details For the implementation, we used different libraries: `CGAL` which is a C++ library that offers all the basic geometric types (point, vector, line, plane..), data structures and algorithms (triangulations...) we need. We also used `Eigen` which is a C++ linear algebra library which provides types such as vector, matrix and manipulation operations. We used `Qt` for the GUI programming.

We used the automatic differentiation technique (see Appendix *Automatic Differentiation* and Section 3.4): it allows us to only worry about the computation of the area and not of its gradient.

2.3 Experiments

2.3.1 Gradient

In this section, we will illustrate on some examples what gives the computation of the gradient of the area of union of balls and the gradient of the perimeter of the boundary. The perimeter of the boundary is defined in the same fashion as the area (see Equation (2.1)):

$$Length(\partial P^r) = \sum_p Length(V(p,P) \cap \partial B(p,r)) \tag{2.2}$$

We will use the automatic differentiation technique described in Appendix *Automatic Differentiation*.

See Figures 24 and 25 for a few examples of such gradients for different input point sets. The colour of the vector depend on the norm of the gradient: the bigger the gradient is, the redder the vector will be.

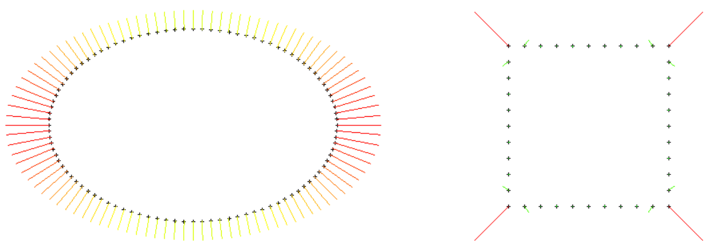


Figure 24 – Gradients of the area for points on an ellipse (left) and on a square (right)

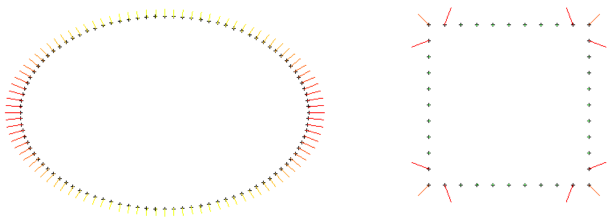


Figure 25 – Gradients of the perimeter for points on an ellipse (left) and on a square (right)

In the previous screenshots, we observe that the gradients are in the same direction as the normals to the underlying curve. This is the case if the radii of the balls are big enough and if the sampling is sufficiently uniform. Also, the norm of these gradients is proportional to the mean curvature of the underlying curve (see Proposition 6). Gradients of the area seem to have a bigger magnitude than the ones of of the perimeter of the boundary.

2.3.2 Mean curvature estimation

In this section, we experimentally check in a particular case that we can estimate the mean curvature on a point set using the norm of the gradients of the area and of the perimeter of the boundary of a union of disks. The theoretical justification, for the area of the union, is given in chapter 4.

We run our tests on a sampled ellipse whose major and minor axes are a and b. We can parametrize this ellipse by :

$$\begin{cases} x(t) & = a\cos(t) \\ y(t) & = b\sin(t) \end{cases} \text{for t } \in [0, 2\pi]$$

We recall the formula for computing the curvature of a parametrized curve:

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{(x'^2(t) + y'^2(t))^{\frac{3}{2}}} \tag{2.3}$$

For an ellipse, we obtain:

$$\kappa(t) = \frac{ab}{(a^2\cos^2(t) + b^2\sin^2(t))^{\frac{3}{2}}}$$

We compare this for $t = \frac{2k\pi}{N}$ for $k = 0 \dots N-1$ with the computed ones where N is the number of samples.

Figure 26 represents the computed gradients, mapped to a colour ramp (from green to red), for an ellipse defined by $a = 180$, $b = 120$. For a matter of visibility, gradients of the perimeter were rescaled. We observe that the gradients are more important where the curvature is higher and are collinear to the outward normals of the underlying curve.

Now, we study the absolute difference between the computed and expected curvatures for different choices of gradients (area, perimeter of the boundary...), see Figure 27.

The error seems to be less important for the area. But curvatures computed with the perimeter of the boundary have less oscillations.

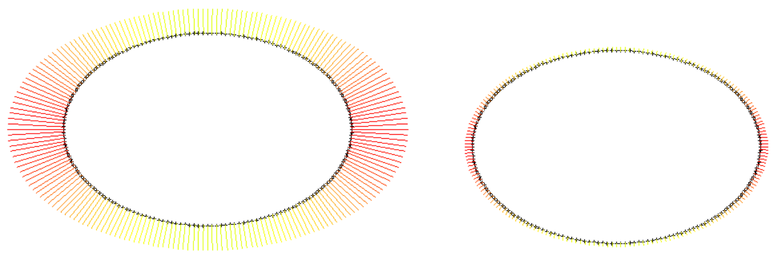


Figure 26 – Computed curvatures on an ellipse using gradients of the area / perimeter with $r = 15$

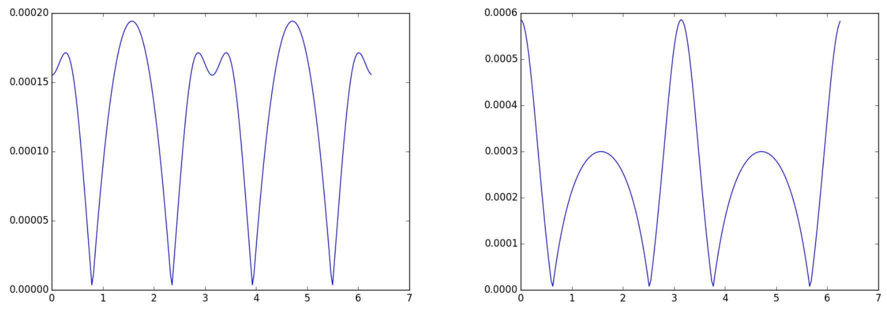


Figure 27 – Error between the curvatures on an ellipse using gradients of the area / perimeter

2.3.3 Discrete Mean Curvature Flow

Now, we will be interested in approximating the continuous mean curvature flow by applying a gradient descent algorithm in order to minimize a functional A. This gradient descent will be done using a constant timestep (Euler explicit scheme). More precisely, given a point cloud $P = (p_1, \ldots, p_N) \in \mathbb{R}^2 \times \ldots \times \mathbb{R}^2$, we defined $A^r(P) = A(\bigcup_{i=1}^N B(p_i, r))$, see Chapter 4 for the computation of the gradient $\nabla_{p_i} A^r(P)$. Then, the Euler step starting from a point cloud $P^t = (p_i^t)$ is defined as follows:

$$p_i^{t+1} = p_i^t - \tau \nabla_{p_i} A^r(P^t) \tag{2.4}$$

where $\tau > 0$ is a constant, and we set: $P^{t+1} = (p_i^{t+1})$

We will also be interested in weighted versions of functionals: we can weight the gradient of the area by the area of the restricted Voronoi cell or we can weight the gradient of the perimeter by the perimeter of the visible boundary of the restricted Voronoi cell (the circular arcs composing the intersection of the Voronoi cell and the disk) of the restricted region. For the latter, we can divide by 0 if the point is surrounded by other ones and so does not have a visible boundary. We need to choose a small time step in order to avoid this.

We will test the different properties we expect the flows to have:

1. Smooth the point clouds: remove outliers, noise

2. Minimize the area of a union of balls

We did some experiments to validate our results: we first compared the two gradient flows (area and perimeter of the boundary) to a set of points uniformly sampled on an ellipse, see Figure 28.

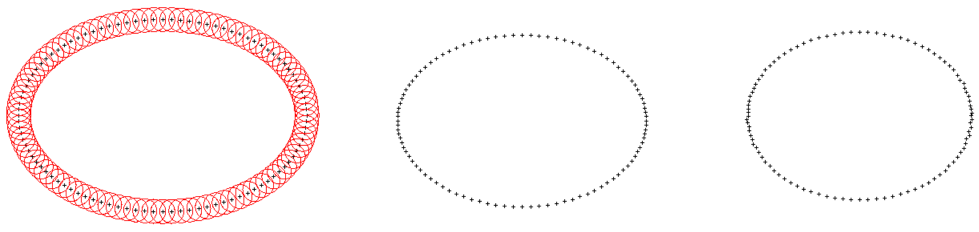
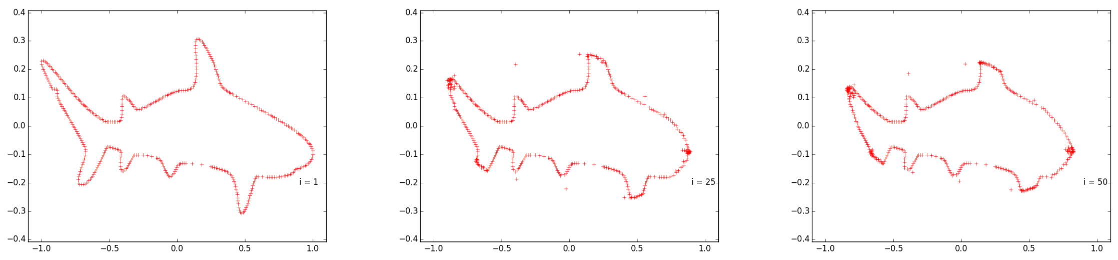


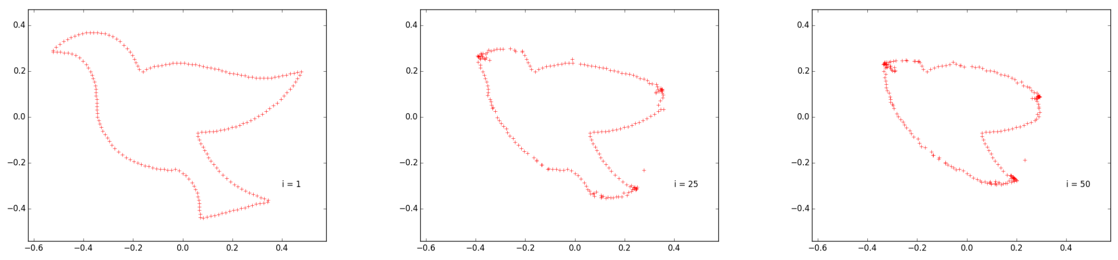
Figure 28 – Initial point set (on the left) and after 100 iterations of the area (middle) / perimeter (right) flow

The two flows seem to have the same behaviour: convergence towards a circle.

Smoothing We also run the perimeter flow on point clouds such as a shark (Figure 29a), a bird (Figure 29b) and a square (Figure 210) to observe the smoothing effect of the flows. We clearly see that the flows tend to smooth the sharp corners.



(a) Shark: 0 / 25 / 50 iterations



(b) Bird: 0 / 25 / 50 iterations

Figure 29 – Examples of perimeter flows

Next, we add some outliers around the ellipse and observe the effects of the two flows: see Figure 211. We see that the outliers are "swallowed" by the initial point set in both cases. But, for the area flow, a hole is created at the same place where the outliers were added. The problem is removed when we use the weighted versions of the area flow.

Furthermore, we added Gaussian noise on these points, see Figure 212. We observe that the two flows remove the noise.

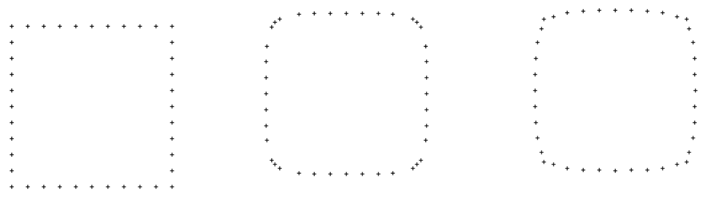


Figure 210 – Initial point set (left) and area (middle), perimeter (right) flows

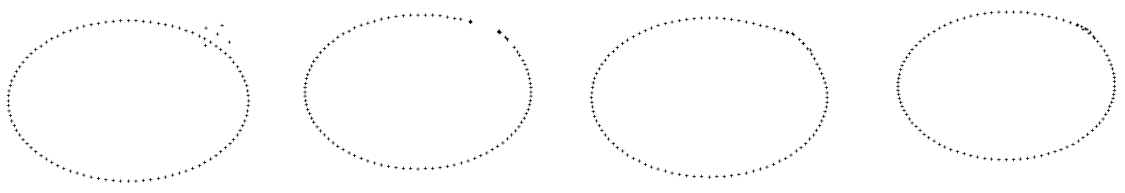


Figure 211 – Initial point set with outliers (left) and area, weighted area and perimeter flows

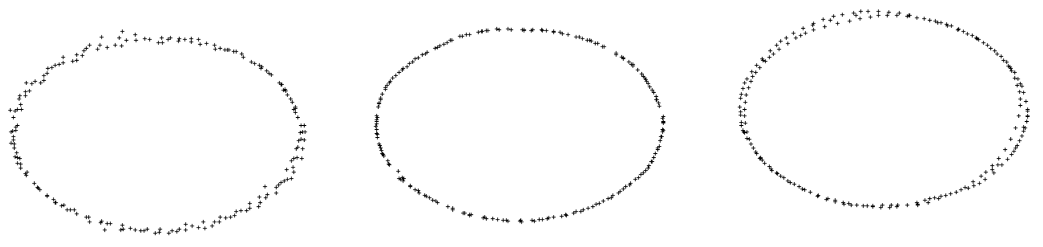


Figure 212 – Initial noisy point set (left) and area (middle), perimeter (right) flows

We noticed that the gradient flow of the area may create holes in the point set which is not the case for the gradient flow of the perimeter. On the contrary, the gradient flow of the perimeter of the boundary will smooth the point set while redistributing the points in an uniform way. This can be explained by looking to a simple case with two intersecting balls:

- *for the perimeter*: the gradients are directed towards the outside and so the balls will be merged because it is easy to see, using the triangle inequality, that in order to minimize the perimeter the balls must come closer.
- *for the area*: it is possible that, in order to minimize the area, we gain more by moving the balls apart rather than closer.

The chosen radius will also influence the smoothing: points which are too far away from other points (at distance greater than the radius) will not move. The bigger the radius is, the more points will move in big groups. Indeed, the radius indicates how the neighbours of a point are taken into account. The more neighbours we take into account, the more "global" the movement will be.

We also add varying oscillations to our ellipse in order to see the adaptive part of the algorithm. We generate oscillations with one or two amplitudes. For one constant amplitude, see Figure 213. For two different amplitudes see Figure 214. For the second figure, the radius is chosen to be between the two amplitudes. We clearly see that the area flow create holes whereas the perimeter flow smooth the point cloud.

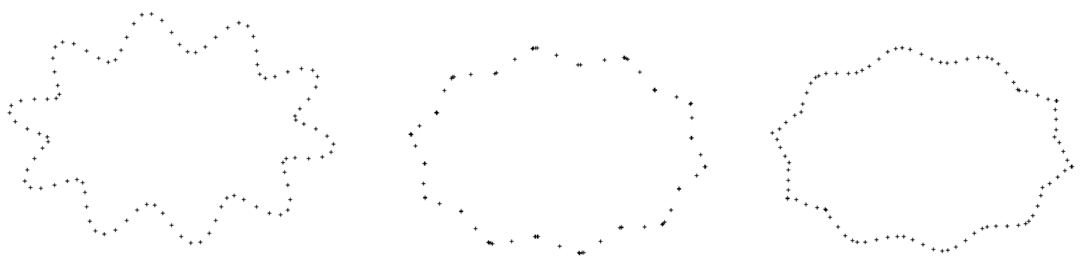


Figure 213 – Area / perimeter flow on an ellipse with oscillations (one amplitude)

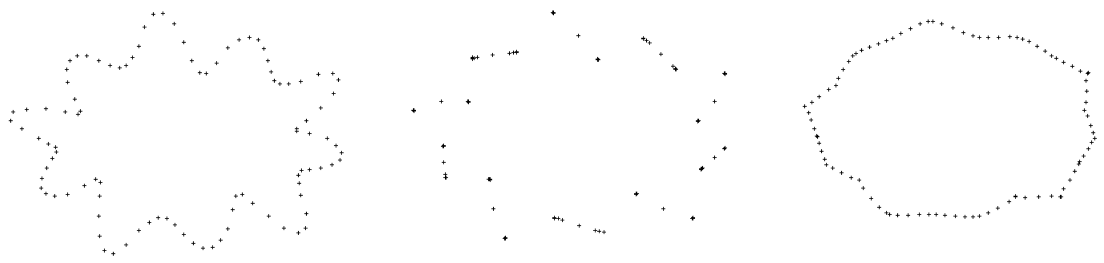


Figure 214 – Area / perimeter flow on an ellipse with oscillations (two amplitudes)

Lastly, we generate points on a line segment and fixed the two points of the extremities. Then, we apply our flow on this point set. We expect the flow to smooth the point set: points should get closer and closer to an uniformly sampled set of points. See Figure 215. The radius is chosen in a way so that the union of the disks is composed of 5 connected components. The results are coherent with our expectations: final points are more uniformly distributed than the initial ones.

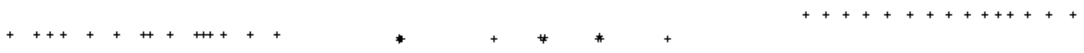
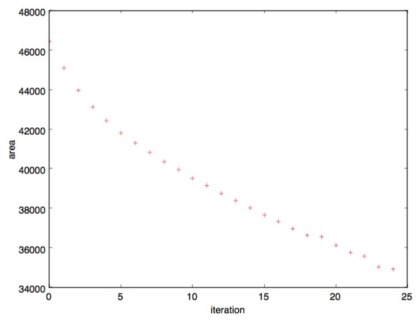


Figure 215 – Left: Initial point set. Middle and right: area / perimeter flow of points on a segment after 50 iterations

Minimization We also check that the area of a union of balls decreases with the number of iterations, see Figure 216.



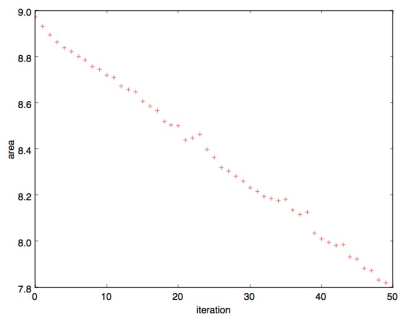


Figure 216 – Evolution of the area on different point clouds with the area flow : square / ellipse

2.4 Conclusion

In summary, the two flows (area and perimeter of the boundary) have the following common properties:

- Smooth the point set: remove the outliers / noise
- Can be used to estimate the mean curvature

But, there are differences between the two flows. The main one is that the area flow has the tendency to create holes in the point cloud. This difference is removed when we use the weighted version of the flow.

We also saw that we can use these flows to estimate the mean curvature on point clouds.

— 3 —
3D case

3.1 Introduction

In this part, we focus on point clouds in 3D that sample a surface. Our goal is to do the same work as the one done in 2D that is to say smooth the point cloud using a mean curvature flow approach. This means computing the volume of a union of balls centered at the points and move the points in the opposite direction to the one given by the gradient of the volume. An algorithm for computing the volume can be found in [4]. Since we expect the same results as in the 2D case (see Chapter 4 for a justification), we preferred to focus on another kind of flow: an anisotropic one.

The idea of this flow is to replace the union of balls with a union of convex polyhedra. The choice of the polyhedron will directly influence the directions in which the points will be moved.

To do that, we will replace the Euclidean ball $B(0,1)$ with a convex polyhedron $B_N(0,1)$ which can be considered as the unit ball for a certain norm N. This norm will be called polyhedral (see Section 3.2). Equation (2.1) remains valid if we replace $B(p,r)$ with $B_N(p,r)$ and V by V_N (the Voronoi diagram computed for the norm N) only if the points are in general position. Indeed, see Figure 31 for a look at a particular case where the chosen polyhedron is a cube (norm L^∞). We see that if we choose two aligned points, the bisector is not a line and so the Voronoi diagram does not partition the plane anymore. Furthermore, the computation of the Voronoi cells for a polyhedral norm is a tough work (see [13] for a thorough study). In Appendix *Voronoi diagrams for polyhedral norms*, we study some properties of the Voronoi diagram for a polyhedral norm and we show that we can symbolically perturb V_N in order to get a partition.

To compute the volume of $\bigcup_{p\in P} B_N(p,r)$, we implemented two different methods: a naive one and one based on inclusion-exclusion formula. In the naive one, we make the following approximation: instead of summing up over all intersections $V_N(p,P) \cap B_N(p,r)$, we sum up over all $V(p,P) \cap B_N(p,r)$. The influence of this choice under the functional we minimized is described in Section 4.5. We will see that the second approached based on inclusion-exclusion formulae is also an approximated one. There is also one method which is exact but we did not have the time to implement it in our internship, it is based on 3D arrangements and overlays.

Firstly, we will define formally what we call a polyhedral norm. Then, we will explain the naive method. Secondly, we will explain the inclusion-exclusion formula we used in our second method. Then, we will talk about the choices we made for the implementation and finally, we

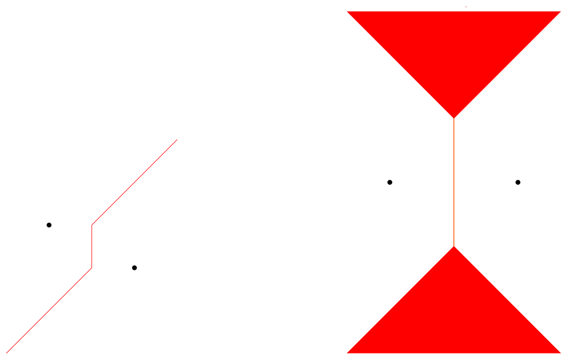


Figure 31 – Bisector for the L^∞ norm: two non aligned points (on the left) and two aligned points (on the right)

will run experiments to validate our expectations.

3.2 Polyhedral norm

A polyhedral norm is a function $N : \mathbb{R}^d \to \mathbb{R}$ defined as follows:

$$\forall x \in \mathbb{R}^d,\ N(x) = \max_i (x|v_i) \tag{3.1}$$

where the v_i are given vectors from $\mathbb{R}^d$. This definition implies that the unit ball for the norm N is a polyhedron. Indeed, if $x \in B_N(0,1)$, then :

$$N(x) \le 1 \iff \forall i,\ (x|v_i) \le 1$$

So, the unit ball is defined by linear constraints. More precisely, it is an intersection of half-spaces and so is a convex polyhedron K. The vectors v_i are the normal vectors to the facets of K. We will denote by $B_K(0,1)$ the unit ball defined by the convex polyhedron K.

3.3 Volume of a union of polyhedra

In this section, we will show how to compute an approximation of the volume of a union of polyhedra using two different methods: a naive one and one based on inclusion-exclusion formula.

3.3.1 Naive method

We want to compute :

$$Vol(\bigcup_{p \in P} B_N(p,r)) = \sum_p Vol(\bigcup_p B_N(p,r) \cap V(p,P)) \tag{3.2}$$

The idea of this method is to approximate $Vol(\bigcup_p B_N(p,r) \cap V(p,P))$ by $Vol(B_N(p,r) \cap V(p,P))$. So, we just have to compute the intersection of half-spaces (see Section 3.4 for details). It is an approximation because, for example, if we want to compute the perimeter of the boundary of squares in 2D, there is a difference between what we want to compute and what we actually compute (see Figure 32).

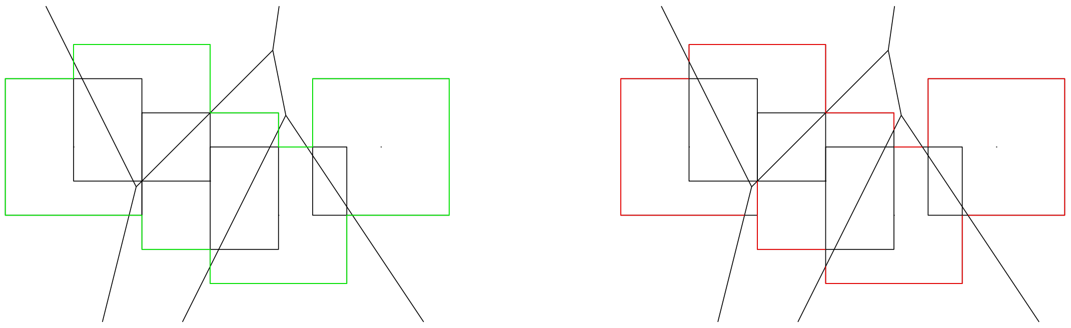


Figure 32 – In green, what we want to compute and in red what we actually compute

3.3.2 Inclusion-exclusion formula

The inclusion-exclusion formula is a well-known formula which can be used to compute the indicator function of a union of sets: given a finite number of sets $A = \{A_1, \dots, A_N\}$, we have:

$$\mathbf{1}_{\bigcup A_i} = \sum_{\emptyset \neq X \subseteq A} (-1)^{cardX-1} \mathbf{1}_{\bigcap X} \tag{3.3}$$

This formula can also be expressed using the notion of nerve as shown in [3]. We define the nerve of $A = \{A_x, x \in X\}$ to be the simplicial complex [1] where a simplex σ exists between the points $x_1 \dots, x_k$ if $\bigcap_{i=1}^{k} A_{x_i} \neq \emptyset$. Then, we can write the inclusion-exclusion formula as:

$$\mathbf{1}_{\bigcup A_x} = \sum_{\sigma \in Nerve(A)} (-1)^{\dim \sigma} \mathbf{1}_{\bigcap \sigma}$$

σ represents any simplex in the nerve. Its dimension is defined as: $\dim \sigma = card \sigma - 1$. If we consider a union of polyhedra, we can write a similar formula:

$$\mathbf{1}_{\bigcup B_N(p,r)} = \sum_{\sigma \in Nerve(\mathscr{B}_N)} (-1)^{\dim \sigma} \mathbf{1}_{\bigcap \sigma} \tag{3.4}$$

where $\mathscr{B}_N$ is the collection of all balls $B_N(p,r),\ p \in P$.

Now, since the nerve can be a really big object, we want to restrict the computation to what we call the α-complex of a set of points:

Definition 2. *The α-complex of a set of points X denoted by $Del(X,\alpha)$ is a subset of the Delaunay triangulation. To each simplex of the Delaunay triangulation, we can associate a characteristic radius: the radius of the smallest empty circle containing the simplex.*

Now, the α-complex contains all the simplices of the Delaunay triangulation whose characteristic radius is smaller than α.

[1]A simplicial complex is a generalization of a triangulation: it is a collection of simplices like vertices, edges, triangles, tetrahedra...

Now, we can wonder if the formula 3.4 is still valid if we replace $Nerve(\mathscr{B}_N)$ by $Del(P,r)$. If we want to prove this assertion, we may use the technique used to prove the inclusion-exclusion formula for a union of balls. The proof starts by defining the subcomplex L_x induced by a point x. It is defined as the nerve of the set of all the polyhedrons that contains a x. Formally, $L_x = Nerve(\{B_N(p,r),\ x \in B_N(p,r)\})$.

Then, when we evaluate the indicator function at x of the union, we can decompose it in two parts: the simplices of L_x and the other ones.

$$\begin{aligned}
\mathbf{1}_{\bigcup B_N(p,r)}(x) &= \sum_{\sigma \in L_x} (-1)^{\dim \sigma} \mathbf{1}_{\bigcap \sigma}(x) + \sum_{\sigma \notin L_x} (-1)^{\dim \sigma} \underbrace{\mathbf{1}_{\bigcap \sigma}(x)}_{=0 \text{ since } x \notin \sigma} \\
&= \sum_{\sigma \in L_x} (-1)^{\dim \sigma} \\
&= \chi(L_x)
\end{aligned}$$

Here, χ is the Euler characteristic.

Now, if we show that L_x is contractible (can be continuously deformed to a point), we will have that $\chi(L_x) = 1$. Obviously, we also have $\mathbf{1}_{\bigcup B_N(p,r)}(x) = 0$ if x is not in the union and we will conclude that the formula 3.4 is true. But it appears that there exists cases where the formula is not correct: see Figure 33. In this figure, we can see that in the α-complex for the L^∞ norm, the triangle, which was present in the α-complex for the L^2 norm, does not exist. It means that in the formula the intersection between the three squares will not be taken into account and so the final result will be incorrect. During this internship, we did not have the time to estimate the error made by using this formula.

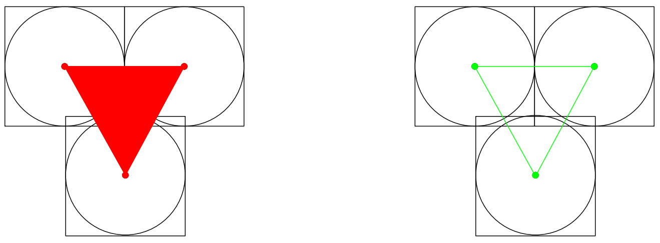


Figure 33 – Counter example where the formula is not correct, in red the L^2 α-complex and in green the L^∞ α-complex.

3.4 Implementation details

In this section, we will describe briefly how we implemented the two methods previously described.

Intersection computation For both methods, we need a way to compute an intersection of half-spaces: for the first one, we need to compute the intersection of a Voronoi cell and a convex polyhedron and for the second one, we need to compute intersections of convex polyhedra.

For the first method, the naive one, a Voronoi cell is represented implicitly as a list of half-spaces (the planes defining the boundary of the cell). Half-spaces are computed using the

Delaunay triangulation. If we want to compute the Voronoi cell of p, we look at the neighbours of p in the Delaunay triangulation. Then, for every neighbour v, the corresponding half-space is the positive part of the bisector plane between p and v. The latter is the plane passing through the midpoint of the segment $[p, v]$ and whose normal vector is $\vec{pv}$. Since the traversal order of the vertices of the Delaunay triangulation is not, in general, the same as the insertion one, we need to associate an index to each point (using a `std::map`).

Recall that a convex polyhedron K represents the unit ball for a polyhedral norm N. K is internally represented as a list of normal vectors to each of its facets. Using this representation, one can compute the translated polyhedron $B_N(p, r)$ for a point p and a radius r by noticing that it is the intersection of the half-spaces H_n where H_n is defined, for any normal vector n, as follows: $H_n = \{x \in \mathbb{R}^3,\ (x|n) \leq (p|n) + r\}$

In order to construct this intersection, we used the duality which allows us to replace the computation of an intersection by the computation of a convex hull (see [15]) of dual points. The duality is defined as follows: for any halfspace defined as the positive part of a plane p which does not contain the origin and whose Cartesian equation is $ax + by + xz + d = 0$, the corresponding dual point is given by $(-\frac{a}{d}, -\frac{b}{d}, -\frac{c}{d})$.

Concretely, we compute the intersection in the following manner (see *Half-space intersection* for a simplified version of the algorithm):

1. Compute the dual points while remembering which point is associated to which plane.
2. Compute the convex hull of these dual points. It gives us a dual polyhedron. To each vertex of this polyhedron, we associate the corresponding primal plane.
3. To compute the primal polyhedron:
   a) We first compute the primal vertices which are the dual of the dual facets: each dual facet has at least 3 vertices. We know the corresponding primal planes for these vertices. Then, the corresponding primal vertex is the intersection of the 3 primal planes.
   b) Secondly, primal facets are constructed by circulating around the dual vertices. For each dual vertex, we circulate on the facets around this vertex. For each dual facet, we add the corresponding primal vertex to the primal facet.

For the second method, we also need a way to compute the α-complex of a set of points. This can be done using the `Alpha_shapes_3` package of `CGAL` and the associated classification methods.

Automatic differentiation integration Let us now explain in more details the integration of the automatic differentiation tool. `CGAL` has the particularity to make things easy for the programmer to change the number type by using the concept of a *Kernel*. A *Kernel* is a class that describes how numbers are stored in memory, how we can construct things (like points, planes, bisectors...) and how to evaluate predicates on objects (collinearity test, coplanarity test...).

There are multiple predefined kernels but we will quickly talk about two particular ones. `Simple_cartesian<NT>` is the most basic one: a number will just be represented by `NT` and so it has inexact predicates evaluation and inexact constructions since it is subject to numerical errors.

`Exact_predicates_inexact_constructions_kernel` (in short `Epick`) is a Kernel which uses a technique called *filtering* to ensure that the predicates are always evaluated in an exact manner. In short, the predicate is first evaluated in an inexact way using interval arithmetic. If the predicate, for example, has to check whether a value is zero or not then, using interval arithmetic, it will need to check whether the resulting interval contains zero or not. If this interval does not contains zero, then a decision can be made. Otherwise, the computation is done again but this time using exact arithmetic which, in all cases, will return an answer.

For the automatic differentiation, we replaced the `NT` type with a custom class `AD`. `AD` is a class with two members: a value and a vector of derivatives. It overloads all the classical arithmetic operators such as addition, subtraction, multiplication, division. It also overloads, using the chain rule, some mathematical functions such as `sqrt`, `atan2`... For example, the implementation of `sqrt` for AD looks like this:

```
AD sqrt(const AD &x) {
  using std::sqrt;
  double sqrtx = sqrt(x.value());
  return AD(sqrtx, x.derivatives() * (double(0.5) / sqrtx));
}
```

Then, for interfacing `AD` with `CGAL`, we choose the `Simple_cartesian<AD>` kernel. The choice of `Simple_cartesian` may seem inappropriate at first glance because the constructions are inexact but it is enough for our applications. In practice, we use a combination of `Epick` and `Simple_cartesian<AD>`. For example, when computing an intersection of half-spaces, we construct the dual using `Epick` to have the exact combinatorics of the dual polyhedron. Then, we use `Simple_cartesian<AD>` when we effectively construct the primal polyhedron.

3.5 Experiments

We compare the two different methods using various polyhedra and point clouds. Let us first observe that naive method is, computationally speaking, less costly. Indeed, for each iteration, it requires to compute n intersections where n is the number of points whereas the inclusion-exclusion one requires to compute $n+e+f+c>n$ intersections where e, f, c are the numbers of edges, triangles and tetrahedra in the α-complex.

First, we test the naive method on 16 points sampled on a circle, see Figure 34 for an example where the polyhedron is a cube. We observe that the gradients are oriented in the interior for the 4 points which are close to the corners of the cube. This will make the flow converge towards a cube.

Then, we test both methods on a point cloud composed of the 8 corners of a cube and with a cube as the polyhedron, see Figure 35 for the results. We observe that both gradients are correctly oriented.

Now we test that if we choose a polyhedron sufficiently close from the unit sphere, we expect the result to be the same as in the 2D case: gradients should be oriented like the outward normal with a norm proportional to the mean curvature. See Figure 36 for an example where the point cloud is a sphere and the polyhedron is a discretized sphere. See also 37 for another example of normal estimation on a more complex point cloud.

We observe that the gradients are oriented like the outward normals and that norms of these gradients seem relatively constant. This confirms the fact that we can simulate in 3D what we

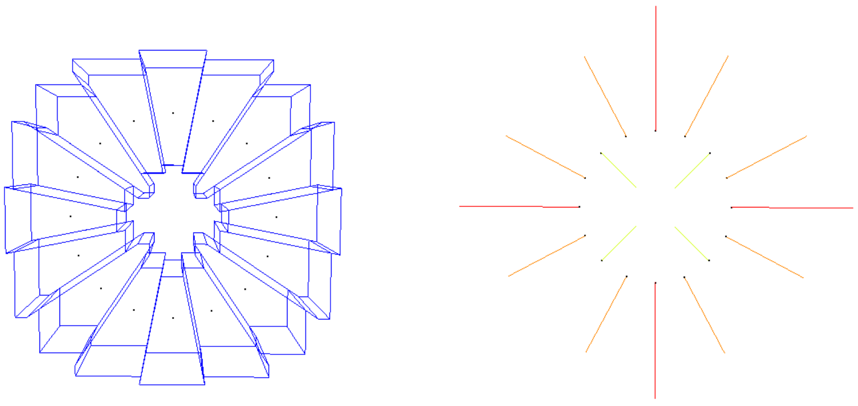


Figure 34 – Intersections of Voronoi cells and a cube (on the left) and gradients of the volume (on the right)

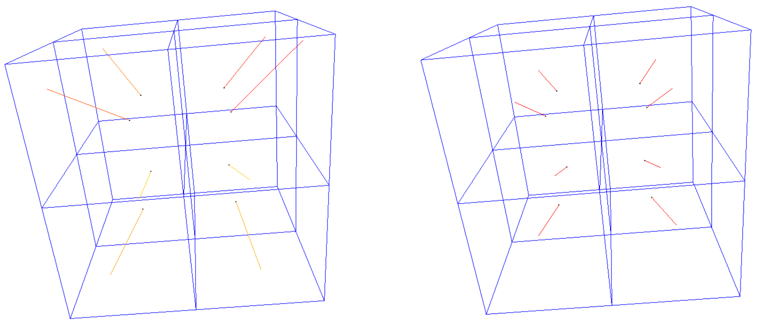


Figure 35 – Naive method (on the left) and inclusion-exclusion method (on the right). Solid blue lines represent the intersections of the Voronoi cells with a cube

did in 2D (mean curvature flow).

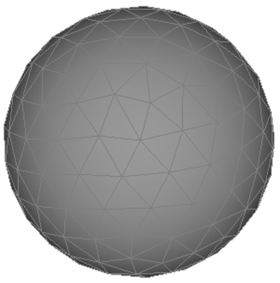
We also do other experiments to check the convergence of the flow. Figure 38 gives examples where the point cloud is a sphere and the polyhedron is a cube, a bipyramid and an icosahedron. On these examples, we can see that the flow seems to converge towards the polyhedron we choose.

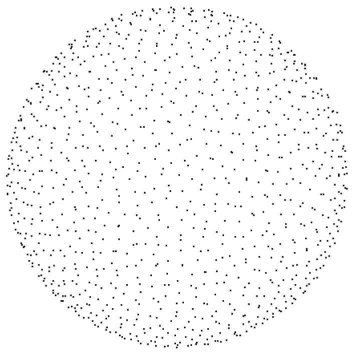
3.6 Conclusion

In summary, we saw, in this chapter, another type of flow: an anisotropic one. We used the same techniques as the ones developed in Chapter 2 by replacing a union of balls with a union of convex polyhedra.

We also saw that, using a discretization of a sphere for the polyhedron, we can obtain similar results as the ones obtained in 2D. Using this technique, we can estimate mean curvature and simulate discrete mean curvature flow on 3D point clouds.

We observed that the obtained flow has different properties than the mean curvature one: the convergence shape will depend on the polyhedron we chose.


(a) Discretized sphere with 200 planes


(b) Initial point cloud

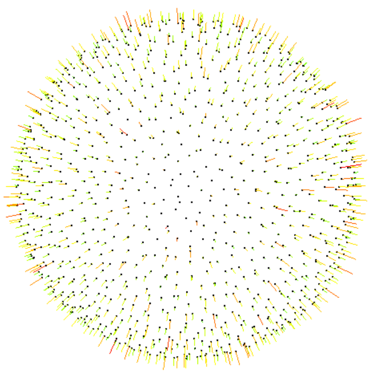

(c) Gradients of the volume

Figure 36 – Polyhedron: discretized sphere / Point cloud: 1000 points on a sphere

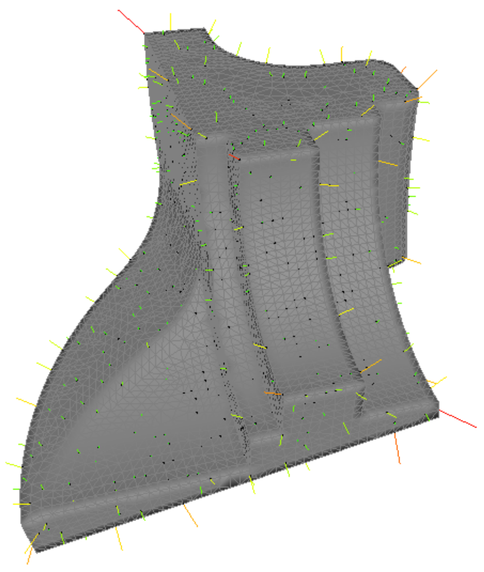
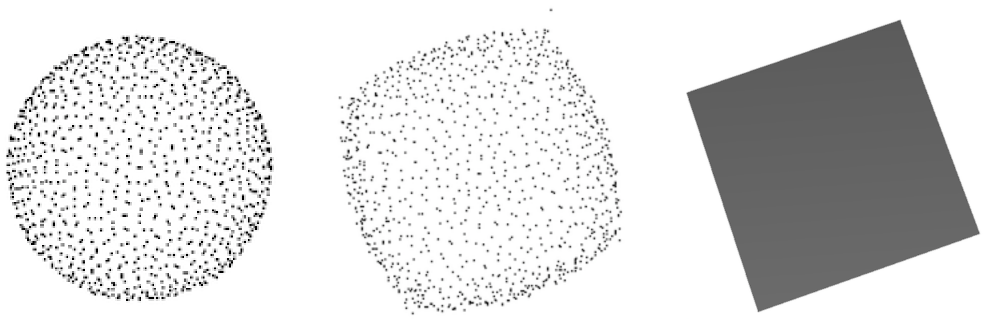
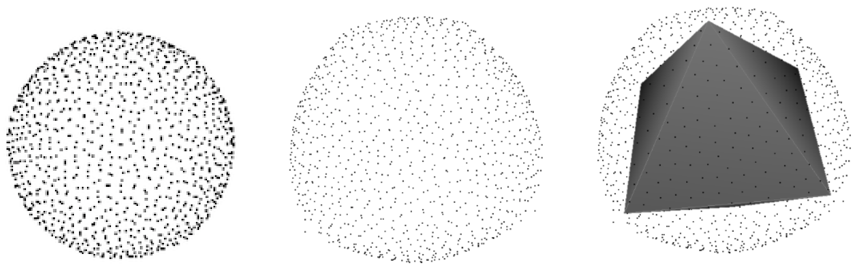


Figure 37 – Normal estimation on points sampled on the `fandisk` model


(a) Cube


(b) Bipyramid

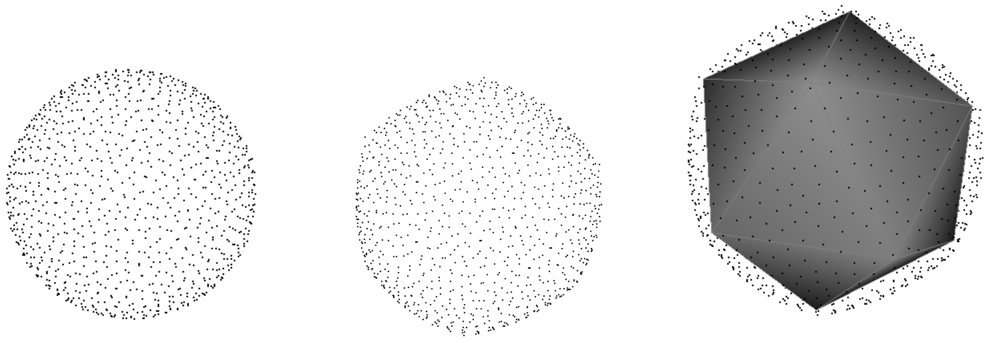

(c) Icosahedron

Figure 38 – Flow of a sphere under different polyhedra after 10 iterations

— 4 —
Theory

In this chapter, all the objects will be considered, for a matter of simplification, to be smooth i.e. C^∞. Also, we will use the term "surface" to refer to any smooth compact hypersurface of $\mathbb{R}^d$. We will also call the area of a surface S denoted by $A(S)$ its $(d-1)$-volume. The d-dimensional volume will be denoted by Vol^d. In Chapter 2, $Vol^d = Area$ and in Chapter 3, $Vol^d = Vol$.

4.1 Mean Curvature Flow

In order to define what we call the Mean Curvature Flow, we will first need the notions of an embedding and the deformation of a surface:

Definition 3. *Let S_0 be a hypersurface of $\mathbb{R}^d$. A map $X : S_0 \to \mathbb{R}^d$ is a smooth embedding if it is injective and that $\forall x \in S_0$, the differential $DX(x)$ is an injective linear map.*

Definition 4. *For a surface $S_0 \subseteq \mathbb{R}^d$, we call a deformation of S_0 a smooth function $X : S_0 \times I \to \mathbb{R}^d$ where $I \subseteq \mathbb{R}$ (which represents the time) with the following properties:*

- $X(.,0) = id_{S_0}$.
- *for any $t \in I$, we denote $X_t = X(.,t)$ and $S_t = X_t(S_0)$. X_t is an embedding of S_0 in $\mathbb{R}^d$.*

Then, the mean curvature flow is a deformation $X : S_0 \times [0,T] \to \mathbb{R}^d$ of a surface S_0 such that:

$$\frac{\partial X_t(x)}{\partial t} = \vec{H}_{S_t}(x) \tag{4.1}$$

where $\vec{H}_{S_t}(x)$ is the mean curvature vector (see Definition 7) of S_t at x and $\frac{\partial X_t(x)}{\partial t}$ is the normal velocity at x.

In simple words, Equation 4.1 says that at each step we move each point of the hypersurface S_t in the direction of the normal by an amount given by the mean curvature at this point.

Now, we will recall that the mean curvature flow belongs to a more general category of geometric flows: the gradient flows (see [11]). In our case, it follows from the fact that the mean curvature vector can be identified with the gradient of a certain functional, namely the area of the surface. The following proposition makes this identification more rigorous:

Proposition 1. *For a surface $S_0 \subseteq \mathbb{R}^d$ and a deformation $X : S_0 \times [0,T] \to \mathbb{R}^d$ defined by $X(p,t) = p + tV(p)$ where $V : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field, we have:*

$$\lim_{t \to 0} \frac{A(S_t) - A(S_0)}{t} = \int_{S_0} V(p)\vec{H}_{S_0}(p)dp \tag{4.2}$$

where $A(S)$ denotes the area of the surface S.

This proposition allows us to identity the gradient of A to the vector field associated to the mean curvature vector on S_0.

To prove this proposition, we will need the next two definitions: the projection on a hypersurface and the reach of an hypersurface.

Definition 5. *For a compact set $K \subseteq \mathbb{R}^d$, we define the projection of $x \in \mathbb{R}^d$ on K to be any point $p \in K$ such that $||x-p|| = d(x,K)$. The set of all the projections of x on K is a subset of K that we will denote by $proj_K(x)$.*

Definition 6. *For a compact set $K \subseteq \mathbb{R}^d$, we define its reach by:*

$$reach(K) = \max\{r \geq 0,\ \forall x \in R^d \text{ s.t. } d(x,K) \leq r,\ card(proj_K(x)) = 1\} \tag{4.3}$$

In other words, the reach is the maximum r we can take such that the projection of any x at distance smaller than r of K is uniquely defined on K.

The reach has some interesting properties (see [14] for a more detailed study):

- For example, if we consider a circle in the plane of center c and of radius R, then every point in the plane has a unique projection on the circle except c which is at distance R from every point on the circle: $proj_C(c) = C$. Consequently, its reach is R.
- Another more general example is that any convex set has infinite reach. This is because the projection on a closed convex set is always uniquely defined.
- The reach is also related to the Local Feature Size (LFS) introduced in [2] for surface reconstruction problems: $LFS(p) = d(p, \mathscr{M})$ $reach(M) = \min_{p \in M} LFS(p)$ where $\mathscr{M}$ is the medial axis of M.
- The reach is also always upper bounded by the minimum radius of curvature. There are no converse result giving a lower bound on the reach in term of the curvature. Indeed, consider the following example: two spheres of radius R at distance ε, then the radius of curvature is always R but the reach of the union is $\frac{\varepsilon}{2}$.

Then, some facts about the curvature of embedded hypersurfaces:

Definition 7. *For an hypersurface $S \subseteq \mathbb{R}^d$, we have:*

- *the map $p \in S \to \vec{n}_S(p)$ where $\vec{n}_S(p)$ is the normal vector of S at p is called the Gauss map and its differential is called the shape operator. The shape operator at p is a linear map from $T_p(S)$ to $T_p(S)$. $T_p(S)$ is the tangent space of S at p: it is the space which is orthogonal to $\vec{n}_S(p)$. For example, in 3D, $T_p(S)$ is the tangent plane of S at p.*

- *the shape operator is diagonalizable: its eigenvectors $P_S^i(p)$ are called the principal curvature directions and its eigenvalues are the $d-1$ principal curvatures at p: $k_S^1(p),\dots,k_S^{d-1}(p)$. So, in the basis formed by $P_S^i(p)$, the matrix of the shape operator is $diag(k_S^1(p),\dots,k_S^{d-1}(p))$.*

- *we will also denote $\vec{H}_S(p)$ the mean curvature vector of S at p: $\vec{H}_S(p) = \sum_{i=1}^{d-1} k_S^i(p)\vec{n}_S(p)$.*

We will need the notion the r-offset of a compact set K:

Definition 8. *For $r > 0$ and a compact set K, we define the r-offset of K denoted by K^r by:*

$$K^r = \bigcup_{p\in K} B(p,r)$$

This definition is interesting since the only assumption about K is that it is compact, it can either represent a point cloud or a surface.

And finally, we will need the following two lemmas which allow us to compute the differential of deformations of surfaces.

Lemma 1. *Given a surface S, and a smooth function $\phi : S \to \mathbb{R}$, we define a deformation of S: $X(p,t) = p + t\phi(p)\vec{n}_S(p)$. Then, we have:*

- *if $t \leq \frac{reach(S)}{\max\phi}$ then X_t is an embedding of S in $\mathbb{R}^d$.*

- *the differential of X_t at p is a linear map from $T_p(S)$ to $\mathbb{R}^d$ and:*

$$DX_t(p) = p + tD\phi(p)\vec{n}_S(p) + t\phi(p)D\vec{n}_S(p) \tag{4.4}$$

  In a basis composed of the principal curvature directions of S at p and $\vec{n}_S(p)$, then we have:

$$DX_t(p) = \begin{pmatrix} 1+t\phi(p)k_S^1(p) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & & 1+t\phi(p)k_S^{d-1}(p) \\ & 1+tD\phi(p) & \end{pmatrix} \tag{4.5}$$

- *For any function $\chi : X_t(S) \to \mathbb{R}$, we have the following change of variable result:*

$$\int_{X_t(S)} \chi(x)dx = \int_S \chi(X_t(p))J_{X_t}(p)dp \tag{4.6}$$

  where $J_{X_t}(p) = (1+t\phi(p)k_S^1(p))\dots(1+t\phi(p)k_S^{d-1}(p))$ is the Jacobian of X_t at $p \in S$.

Lemma 2. *Given a surface S, $r = reach(S)$, and a deformation $f : S\times[-r,r] \to \mathbb{R}^d$ defined by $F(p,t) = p + t\vec{n}_S(x)$, then:*

- *F is a diffeomorphism from $S\times[-r,r]$ to S^r*

- *the differential of F at p is a linear map from $T_p(S) \times \mathbb{R}$ to $\mathbb{R}^d$. In the basis formed by the principal curvature directions at p and the normal $\vec{n}_S(p)$, we have:*

$$DF(p,t) = diag(1 + tk_S^1(p), \ldots, 1 + tk_S^{d-1}(p), 1) \tag{4.7}$$

So its Jacobian is:

$$J_F(p,t) = \prod_{i=1}^{d-1}(1 + tk_S^i(p)) \tag{4.8}$$

- *For any function $\chi : S^r \to \mathbb{R}$, we have the following change of variable result:*

$$\int_{S^r} \chi(x)dx = \int_S \int_{-r}^{r} \chi(F(p,t))J_F(p,t)dtdp \tag{4.9}$$

Then, we can tackle the proof of the proposition 1.

Sketch of proof of Proposition 1. We use the deformation $X : S_0 \times [0,T]$ where $X(p,t) = p + t\phi(p)\vec{n}_{S_0}(p)$ for $\phi : S_0 \to \mathbb{R}$ is a smooth function. Then, we have, using the lemma 1 with $\chi = 1$:

$$\begin{aligned}
A(S_t) &= \int_{S_t} 1dx = \int_{S_0} J_X(p)dp \\
&= \int_{S_0} \prod_{i=1}^{d-1}(1 + t\phi(p)k_{S_0}^i(p)) \\
&= \int_{S_0} 1dp + t\int_{S_0} \phi(p)\vec{H}_{S_0}(p)dx + O(t^2) \\
&= A(S_0) + t\int_{S_0} \phi(p)\vec{H}_{S_0}(p)dx + O(t^2)
\end{aligned}$$

So:

$$\lim_{t\to 0} \frac{A(S_t) - A(S_0)}{t} = \int_{S_0} \phi(p)\vec{H}_{S_0}(p)dp$$

This equation tells us exactly that if we perturb a surface by ϕ then the variation of the area functional is linked to the mean curvature. In other words, $\vec{H}_{S_0}$ is the L^2 gradient of the area functional. □

This proposition will be at the basis of our work. We will discretize the hypersurface by taking points sampled on it and try to approximate the area of the hypersurface by something which can be computed using only the point cloud.

4.2 Discretization of the area

In this section, we will try to compute an approximation of the area of an hypersurface with knowing only points sampled on it. We can relate the area of an hypersurface to the volume of its tubular neighbourhood using a variant of the tube formula [18].

Proposition 2. *If S is a smooth hypersurface whose reach is positive and $reach(S) > r > 0$, then:*

$$A(S) = \frac{Vol^d(S^r)}{2r} + O(r^2) \tag{4.10}$$

The constants in the error term of the development only depend on the curvature of S.

Proof. We have: $Vol^d(S^r) = \int_{S^r} 1dx$. Now, we will use the following deformation: $X(p,t) = p + t\vec{n}_S(p)$ Using this substitution and Lemma 2 with $\chi = 1$:

$$\begin{aligned}
Vol^d(S^r) &= \int_S \int_{-r}^{r} J_f(p)dtdp \\
&= \int_S \int_{-r}^{r} \prod_{i=1}^{d-1} (1 + t\kappa_i(p))dtdp \\
&= \int_S \int_{-r}^{r} \left(1 + \sum_{k=1}^{d-1} t^k \sum_{i_1 < \ldots < i_k} \kappa_{i_1}(p) \ldots \kappa_{i_k}(p) \right) dtdp \\
&= 2r \int_S 1dp + \int_K \int_{-r}^{r} \sum_{k=1}^{d-1} t^k \sum_{i_1 < \ldots < i_k} \kappa_{i_1}(p) \ldots \kappa_{i_k}(p)dtdp \\
&= 2rA(S) + O(r^2)
\end{aligned}$$

So: $\frac{Vol^d(S^r)}{2r} = A(S) + O(r^2)$. □

This proposition is related to the Minkowski content. It is used to measure the area of the boundary of a set: the $d-1$ dimensional volume in dimension d. It is defined as follows:

$$\lim_{\varepsilon \to 0} \frac{Vol^d(S^\varepsilon) - Vol^d(S)}{\varepsilon}$$

This is exactly what we computed: a Taylor expansion of $Vol^d(S^r)$.

Next, we want to approximate $Vol^d(S^r)$ by $Vol^d(P^r)$ where $P \subset S$ is a point cloud. We will need a formalization of the notion of sampling:

Definition 9. *A set of points p_i is said to be an ε-sampling of a manifold M if the set of the balls $B(p_i,\varepsilon)$ verifies: $M \subseteq \bigcup_i B(p_i,\varepsilon)$ (see [2]).*

The following proposition bounds the error of this approximation:

Proposition 3. *For an hypersurface S with positive reach and an ε-sampling $P \subseteq S$, we have:*

$$0 \leq Vol^d(S^r) - Vol^d(P^r) \leq \frac{\varepsilon^2}{r} A(S) + O(\frac{\varepsilon^4}{r^2}) \tag{4.11}$$

Proof. Since $P \subseteq S$, then $Vol^d(S^r) \leq Vol^d(P^r)$. Then, we parametrize the boundary of P^r by $f : S \to \partial P^r$ where $f(p) = p + r(p)\vec{n}_S(p)$ for $r(p) \in [0,r]$ (see Figure 41).

We have:

$$\begin{aligned}
r(p)^2 + \varepsilon^2 = r^2 &\iff (r - r(p))(r + r(p)) = \varepsilon^2 \\
&\iff r - r(p) = \frac{\varepsilon^2}{r + r(p)} \leq \frac{\varepsilon^2}{r}
\end{aligned}$$

Now, we will use the same formula as in the previous proofs:

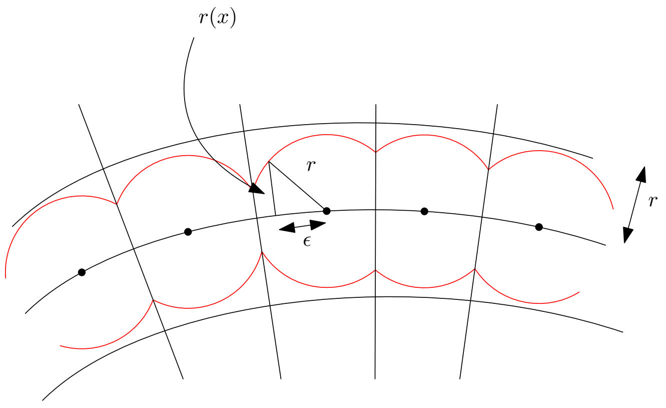


Figure 41 – Parametrization of the offset (in red)

$$\begin{aligned}
Vol^d(S^r) - Vol^d(P^r) &= \int_{S^r \setminus P^r} 1dp \\
&= \int_S \int_{r(p)}^{r} J_f(p,t)dtdp \\
&= \int_S \int_{r(p)}^{r} \left(1 + \sum_{k=1}^{d-1} t^k \sum_{i_1<\ldots<i_k} k_S^{i_1}(p)\ldots S_S^{i_k}(p)\right) dtdp \\
&= \int_S (r - r(p))dp + O((r-r(p))^2) \\
&\leq \frac{\varepsilon^2}{r} A(S) + O(\frac{\varepsilon^4}{r^2})
\end{aligned}$$ □

Now, combining the previous propositions, we will show that we can approximate the area of S by the volume of P^r:

Proposition 4. *Given an hypersurface S, an ε-sampling of S: P, we have:*

$$|\frac{Vol^d(P^r)}{2r} - A(S)| \leq \frac{\varepsilon^2}{2r^2} + O(\frac{\varepsilon^4}{r^3}) + O(r^2) \tag{4.12}$$

So, when $\frac{\varepsilon}{r}$ and r vanish then $\frac{Vol^d(P^r)}{2r}$ converges towards the area of S.

Proof. We will use the propositions 3 and 2. We have, using the triangle inequality:

$$\begin{aligned}
\left|\frac{Vol^d(P^r)}{2r} - A(S)\right| &\leq \left|\frac{Vol^d(P^r)}{2r} - \frac{Vol^d(S^r)}{2r}\right| + \left|\frac{Vol^d(S^r)}{2r} - A(S)\right| \\
&\leq \frac{\varepsilon^2}{2r^2} A(S) + O(\frac{\varepsilon^4}{r^3}) + O(r^2)
\end{aligned}$$ □

At this point, we have shown a way to approximate the area of an hypersurface with a quantity proportional to the volume of the r-offset of a point cloud sampled on the surface. Now, we want to study the gradient of this newly computed quantity.

4.3 Convergence of the gradient

In this section, we will consider the functional $A^r : \mathbb{R}^d, \ldots, \mathbb{R}^d \to \mathbb{R}$ defined as follows:

$$A^r(p_1, \ldots, p_N) = Vol^d(\bigcup_{i=1}^{N} B(p_i, r)) \tag{4.13}$$

We will derive formulae for the gradients of A^r. First, we will define what we mean by "gradients":

Definition 10. *If $F : (\mathbb{R}^d)^N \to \mathbb{R}$ is a function over d-dimensional point clouds, and $p \in (\mathbb{R}^d)^N$, then the j-th ($j \in \{1, \ldots, d\}$) coordinate of $\nabla_{p_i} F(p) \in \mathbb{R}^d$ for $i \in \{1, \ldots, N\}$ is given by:*

$$(\nabla_{p_i} F(p))_j = \lim_{\varepsilon \to 0} \frac{F(p_1, \ldots, p_i + \varepsilon e_j, \ldots, p_N) - F(p)}{\varepsilon}$$

where $(e_1, \ldots, e_d)$ is the standard basis of $\mathbb{R}^d$.

The next proposition gives the gradient of the volume of union of balls :

Proposition 5. *The gradient of the functional A^r at a point $p = (p_1, \ldots, p_N)$ is given by:*

$$\nabla_{p_i} A^r(p) = \int_B \frac{x - p_i}{||x - p_i||} dx \tag{4.14}$$

where $B = \partial B(p_i, r) \cap V(p_i, P)$ is the visible boundary of the ball.

Note that [12] tackles the problem of minimizing a functional over the set of convex bodies. For doing this, the authors compute the derivatives of this functional: we use a similar technique to prove our proposition.

Proof. For this proof, we will use Formula 2.1. Assume that we move the point p_i by a small quantity δp_i, let us study the variation of the area of $\bigcup_i V(p_i, P) \cap B(p_i, r)$.

More formally, let us define $A_\varepsilon = A(V(p_i + \varepsilon\delta p_i, P) \cap B(p_i + \varepsilon\delta p_i, r))$, we want to compute the following limit:

$$\lim_{\varepsilon \to 0} \frac{A_\varepsilon - A_0}{\varepsilon}$$

To compute this quantity, we will define the following sets:

- $B_1 = \{x \in B, (x - p_i | \delta p_i) \geq 0\}$
- $\mathscr{A}_\varepsilon^1 = \bigcup_{x \in B_1} [x, x + \varepsilon\delta p_i]$: "upper" gained area (see Figure 42 for a drawing of the situation in 2D)
- $B_2 = \{x \in B, (x - p_i | \delta p_i) \leq 0\}$
- $\mathscr{A}_\varepsilon^2 = \bigcup_{x \in B_2} [x, x + \varepsilon\delta p_i]$: "lower" lost area.

Then, we have: $A_\varepsilon = A_0 + Vol(\mathscr{A}_\varepsilon^1) - Vol(\mathscr{A}_\varepsilon^2) + O(\varepsilon^2)$

Now, we will evaluate $Vol(\mathscr{A}_\varepsilon^1)$, we will do similarly for $Vol(\mathscr{A}_\varepsilon^2)$.

For any $x \in B$, we define $z(x)$ as the intersection of the half-line D (see Figure 42) with the circle of center $p_i + \varepsilon\delta p_i$ of radius r. We also parametrize the half-line D by: $x + t\frac{x - p_i}{||x - p_i||}$ for $t \geq 0$.

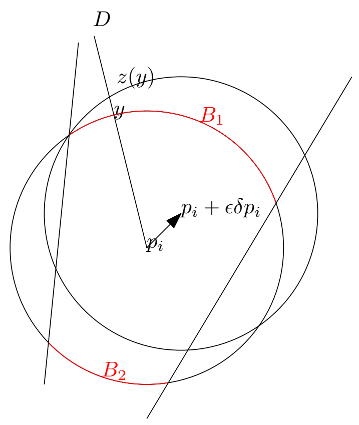


Figure 42 – Situation for a ball $B(p_i,r)$ in 2D

Let us find this intersection point $z(x)$, we assume that t is small by neglecting second order terms $O(t^2)$. We need to find $D \cap C(p_i+\delta p_i, r)$. Let $t \geq 0$ then, we have :

$$||x+t\frac{x-p_i}{||x-p_i||}-(p_i+\varepsilon\delta p_i)||^2 = r^2 \qquad (\star)$$

If we expand this expression, we get:

$$\begin{aligned}
(\star) \iff & ||x-(p_i+\varepsilon\delta p_i)||^2+t^2+2t\left(\frac{x-p_i}{||x-p_i||}|x-(p_i+\varepsilon\delta p_i)\right) = r^2\\
\iff & ||x-p_i||^2-2\varepsilon(x-p_i|\delta p_i)+||\varepsilon\delta p_i||^2+t^2+2t\left(\frac{x-p_i}{||x-p_i||}|x-(p_i+\varepsilon\delta p_i)\right) = r^2\\
\iff & -2\varepsilon(x-p_i|\delta p_i)+2t||x-p_i||+o(\varepsilon^2) = 0 \text{ because } ||x-p_i|| = r\\
\iff & t = t^\star = \varepsilon\left(\frac{x-p_i}{||x-p_i||}|\delta p_i\right)+o(\varepsilon^2)
\end{aligned}$$

Then, $z(y) = x+t^\star\frac{x-p_i}{||x-p_i||}$ and $||x-z(x)|| = t^\star$.
We deduce that :

$$Vol(\mathscr{A}_\varepsilon^1) = \int_{B_1}||x-z(x)||dx = \int_{B_1}\left[\varepsilon\left(\frac{x-p_i}{||x-p_i||}|\delta p_i\right)+o(\varepsilon^2)\right]dx$$

And:

$$Vol(\mathscr{A}_\varepsilon^1)-Vol(\mathscr{A}_\varepsilon^2) = \int_B\left[\varepsilon\left(\frac{x-p_i}{||x-p_i||}|\delta p_i\right)+o(\varepsilon^2)\right]dx$$

Finally, we have, by linearity:

$$\nabla_{p_i}A^r(p) = \int_B\frac{x-p_i}{||x-p_i||}dx \qquad \square$$

Now, we will relate the previously computed gradients to the mean curvature vector of an hypersurface. If we suppose that we have a point cloud that is an ε-sampling of a smooth hypersurface S, then the following proposition gives the link between the gradient of volume of an offset of S and the mean curvature vector.

Proposition 6. *Given an ε-sampling P of a smooth (C^∞) hypersurface S and $r \geq 0$ such that: $\varepsilon \leq r \leq reach(M)$, then, for $p = (p_1, \dots, p_N) \in P^N$:*

$$\frac{\nabla_{p_i} A^r(p)}{r \times Vol^{d-1}(\partial B(p_i, r) \cap V(p_i, P))} = \vec{H}_S(p_i) + O\left(\frac{\varepsilon}{r}\right) + O(r) \tag{4.15}$$

Consequently, when r and $\frac{\varepsilon}{r}$ vanish then $\nabla_{p_i} A^r(p)$ converges towards a quantity proportional to the mean curvature vector of S at p_i.

To prove that, we will need two lemmas extracted from [2] which will tell us how the Voronoi cells look:

Lemma 3. *If S is an hypersurface, P an ε-sampling of S and $p \in P$, then:*

$$V(p,P) \cap S \subseteq B\left(p, \frac{\varepsilon}{1-\varepsilon} LFS(p)\right)$$

Lemma 4. *If S is an hypersurface, P an ε-sampling of S, $p \in P$ and $v \in V(p,P)$ such that $d(p,v) \geq \alpha LFS(p)$ for $\alpha > 0$, then the angle at s between the vector v and the normal at the surface (oriented in the same direction) is at most* $\arcsin(\frac{\varepsilon}{\alpha(1-\varepsilon)}) + \arcsin(\frac{\varepsilon}{(1-\varepsilon)})$.

These two lemmas tell us that the Voronoi cells are skinny (first lemma) and long (second lemma).

Proof of Proposition 6. We will first do two approximations on the equation (4.14):

1. the first one is to suppose that the offset of S is close enough to the offset of P. Using the computation done in 3, we can say that the error made is in the order of $O(\frac{\varepsilon^2}{r})$.

2. the second one is that if the sampling is dense enough then we can replace p_i by the projection of $x \in V(p_i, P)$ on S: $\forall x \in V(p_i, P),\ p_i = p_S(x) + O(\varepsilon)$, see Figure 43. For this approximation to hold, it is necessary for the pointed out distance to be greater than ε i.e. $r \gg \varepsilon$ which means that $\frac{\varepsilon}{r}$ needs to vanish. This will allow us to say that: $||x - p_i|| = O(r)$.

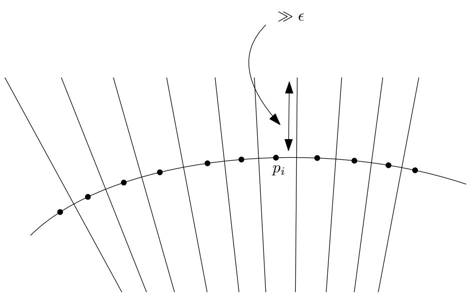


Figure 43 – Shape of the Voronoi cells of densely sampled points

Using these approximations, we get:

$$\begin{aligned}\int_{\partial P^r \cap V(p_i,P)} (x - p_i)dx &\overset{(1)}{=} \int_{\partial S^r \cap V(p_i,P)} (x - p_i)dx + O(\varepsilon^2) \\ &\overset{(2)}{=} \int_{\partial S^r \cap V(p_i,P)} (x - p_S(x))dx + O(\varepsilon)\end{aligned}$$

Then, we will use the substitution $q = p_S(x) \iff x = F^+(q) = q + r\vec{n}_S(q)$ on the upper part of $\partial S^r \cap V(p_i, P)$. Using the lemma 2, we have that F^+ is a diffeomorphism and that $J_{F^+}(q) = (1 + rk_S^1(q)) \dots (1 + rk_S^{d-1}(q))$.

Using an analogous substitution on the lower part of $\partial S^r \cap V(p_i, P)$, we get : $J_{F^-}(q) = (1 - rk_S^1(q)) \dots (1 - rk_S^{d-1}(q))$. By subtracting the two, we obtain: $2rH_S(q) + O(r^2)$.

It follows that:

$$\begin{aligned}\int_{\partial S^r \cap V(p_i,P)} (x - p_S(x))dx &= \int_{S \cap V(p_i,P)} \vec{n}_S(q)(J_{F^+}(q) - J_{F^-}(q))dq \\ &= 2r \int_{S \cap V(p_i,P)} \vec{H}_S(q)dq + O(r^2)\end{aligned}$$

Finally, since we have supposed that S is C^∞ then, we can write: $\vec{n}_S(q) = \vec{n}_S(p_i) + O(\varepsilon)$, $\vec{H}_S(q) = \vec{H}_S(p_i) + O(\varepsilon)$. Error terms are negligible since they depend on the third derivative of the curvature. We deduce that:

$$\nabla_{p_i} A^r(p) = 2r\vec{H}_S(p_i) \int_{S \cap V(p_i,P)} 1dq + O(r^2) + O(\varepsilon)$$

Then, we can write:

$$\int_{S \cap V(p_i,P)} 1dq = Vol^{d-1}(S \cap V(p_i,P)) \simeq \frac{1}{2} Vol^{d-1}(\partial B(p_i,r) \cap V(p_i,P))$$

So:

$$\frac{\nabla_{p_i} A^r(p)}{r \times Vol^{d-1}(\partial B(p_i,r) \cap V(p_i,P))} = \vec{H}_S(p_i) + O\left(\frac{\varepsilon}{r}\right) + O(r) \qquad \square$$

We can summarize the situation with the following diagram:

$$\begin{tikzcd} A^r \arrow[r] \arrow[d, "\nabla"] & A \arrow[d, "\nabla"] \\ \nabla A^r \arrow[r] & \nabla A \end{tikzcd}$$

A^r represents the discretization of the area of an hypersurface: it is $\frac{Vol^d(P^r)}{2r}$ which converges towards the continuous area (see Proposition 4). There is also a convergence property for the gradients: the discretized gradient converges towards the continuous one (see Proposition 6).

In some sense, "discretization" and "gradient" commute: the gradient of the discretization converges towards the continuous gradient.

4.4 Other kinds of discretizations

In this work, we were also interested in other discretizations. Instead of choosing the volume as the functional, we can also choose the area of the boundary or corresponding weighted versions (i.e. the gradient of the volume will be weighted by the corresponding volume of the intersection between the Voronoi cell and a ball).

For the former case (area of the boundary), we have the following lemma:

Lemma 5. *If S is a smooth hypersurface whose reach is positive and reach$(S) > r > 0$, then:*

$$A(S) = \frac{Vol^d(\partial S^r)}{2r} + O(r^2) \tag{4.16}$$

During this internship, we did not prove that the gradients of the area of the boundary of a union of balls are directly connected (like for the volume of the union) to the mean curvature of the sampled surface. But even if we did not made the prove, we made experiments to show this relation.

Finally, we were interested in replacing the Euclidean ball with a convex polyhedron in order to have an anisotropic smoothing. A more detailed study of this problem will be done in the next section.

4.5 3D case

In this section, we will study the influence of the choice we made by replacing, in chapter 3, the Voronoi diagram computed with a polyhedral norm with the standard one. We will show that the function that we minimized is not the volume of the union of convex polyhedra but another functional. For this, we will need the definition of the radial function of a convex set:

Definition 11. *For a convex set K, we define the radial function ρ_K by:*

$$\rho_K(x) = \sup\{r \in \mathbb{R},\ rx \in K\}$$

For a surface S and a deformation $X(p,t) = p + t\vec{n}_S(p)$ for $t \in [-r\rho_K(-\vec{n}_S(p)), r\rho_K(\vec{n}_S(p))]$, let us denote by A' the following functional:

$$A'(S) = \int_S \int_{-r\rho_K(-\vec{n}_S(p))}^{r\rho_K(\vec{n}_S(p))} J_X(p,t)\,dt\,dp \tag{4.17}$$

where K is a convex polyhedron.

We have, in 3D, using the lemma 2:

$$\begin{aligned} A'(S) &= \int_S \int_{-r\rho_K(-\vec{n}_S(x))}^{r\rho_K(\vec{n}_S(x))} \left[1 + (\kappa_1(x) + \kappa_2(x))t + \kappa_1(x)\kappa_2(x)t^2\right] dt\,dx \\ &= \int_S (r\rho_K(\vec{n}_S(x)) + r\rho_K(-\vec{n}_S(x)))dx + O(r^2) \end{aligned}$$

Now, if K is symmetric, i.e. if $x \in K$ then $-x \in K$ which implies that $\rho_K(\vec{n}_S(x)) = -\rho_K(-\vec{n}_S(x))$, then we get:

$$A'(S) = 2r \int_S \rho_K(\vec{n}_S(x))dx + O(r^3) \tag{4.18}$$

So:

$$\frac{1}{2r} A'(S) = \int_S \rho_K(\vec{n}_S(x))dx + O(r^2) \tag{4.19}$$

Thus, the functional we minimized is A' and not the volume of the union.

— 5 —

Conclusion & Perspectives

During this internship, we developed different algorithms for point cloud smoothing. In Chapter 2, we looked at one which allows us to estimate the mean curvature and smooth 2D point clouds using a discretization of the mean curvature flow. Instead of minimizing a continuous area, we minimize a discrete one: the volume of a union of balls. In order to do that, we use a gradient descent based algorithm combined with the Automatic Differentiation technique.

Then, in Chapter 3, we extended this algorithm in 3D by replacing the union of balls with a union of convex polyhedra. The obtained flow has different properties than the normal mean curvature flow: it is anisotropic. We showed that the choice of the convex polyhedron influence the directions of the anisotropic smoothing.

Finally, in Chapter 4, we examined the theoretical properties of our discretization of the area. We proved that the approximation of the area of a surface by the volume of a union of balls is a good under some conditions. We also proved that the flow we constructed in Chapter 2 is indeed a discretization of the mean curvature flow: the gradient of the volume of a union of balls converge towards a quantity proportional to the mean curvature vector.

There is still some place left for improvements:

- We did the proof of the convergence for the gradient of the volume of the union (see Proposition 6). It would be interesting to study the convergence of the gradient of the area of the boundary. Our experiments seem to show that this gradient tends to converge also to a quantity proportional to the mean curvature vector.
- Correct the algorithm in 3D in order to minimize the good functional. Indeed, we saw in Section 4.5 that we do not minimize the volume of the union of convex polyhedra but another, related functional. It would be interesting to manage to find a way to use the true functional.
- Implement an exact computation of the volume of the union in 3D using arrangements and overlays.
- Use the 3D algorithm to simulate crystal growth in physics.

Automatic Differentiation

The automatic differentiation is a technique used for computing the derivatives functions. It relies on the fact that no matter how complex a computer program is, it is composed of a sequence of elementary arithmetic operations (such as addition, subtraction, multiplication, division) and elementary functions (exp, log, cos, sin...). Automatic Differentiation extensively uses the chain rule on these operations to compute derivatives.

It is not the same as symbolic or numerical differentiation (like finite differences). Indeed, automatic differentiation can be used to compute derivatives of programs and not only mathematical functions without using approximations as in numerical differentiation.

Let us suppose that we want to compute the derivative of a function f with a single one dimensional argument x. To do that, we will replace the number type that is to say that we will replace x by $x+\varepsilon x'$ where ε satisfies the property $\varepsilon^2 = 0$ ($x+\varepsilon x'$ is said to be a dual number). Then, we overload all the arithmetical operations: addition, subtraction, multiplication, division. For instance: $(x+\varepsilon x') \times (y+\varepsilon y') = xy+\varepsilon(xy'+x'y)$.

Then we call f with this number, the result is also a dual number: $f(x+\varepsilon x') = y+\varepsilon y'$. Furthermore, for small values of ε, we have the following first order Taylor expansion: $f(x+\varepsilon x') = f(x)+\varepsilon x' f'(x)+...$ By identification, we get $y = f(x)$ and $y' = x' f'(x)$. x' is called a seed and can be chosen arbitrarily, we can for instance choose $x' = 1$ so that $y' = f'(x)$.

This process can be easily extended to handle functions like $f : \mathbb{R}^n \to \mathbb{R}$ in order to compute gradients. This is what we used during the internship: we considered a function over n points in dimension d as an energy functional $E : \mathbb{R}^{dn} \to \mathbb{R}$.

This technique is interesting because it allows us to compute accurate derivatives of functions very easily and efficiently. Indeed, we only have to change the number type and write the good implementations of the basic arithmetic operations and overloaded mathematical functions. It also allows us to only focus on the computation of the value and not its derivative. Another advantage is that in `CGAL`, this is easy to do since the library can be parametrized by the number type.

Half-space intersection

Here is a sketch of the code used in the 3D case for computing the intersection of half-spaces. The function `construct_dual` constructs the dual polyhedron by attaching to each vertex its corresponding primal plane. The function `primal_from_dual` constructs the primal polyhedron from the dual one.

```
void construct_dual (PlaneIterator pbegin,
                     PlaneIterator pbeyond,
                     Polyhedron& dual) {
    std::vector<Point_3> dual_points;
    std::map<Point_3, Plane_3> point_plane_map;

    // Compute the dual points and associate the dual point to
    // its primal plane.
    for (PlaneIterator pit = pbegin; pit != pbeyond; ++pit) {
        Point_3 dp = CGAL::ORIGIN +
            pit->orthogonal_vector() / (-pit->d());
        dual_points.push_back(dp);
        point_plane_map[dp] = *pit;
    }

    // Compute the convex hull of the dual points
    CGAL::convex_hull_3(dual_points.begin(), dual_points.end(), dual);

    // Tag each vertex with the corresponding primal plane
    for (Vertex_iterator vit = dual.vertices_begin();
         vit != dual.vertices_end();
         ++vit) {
        vit->plane = point_plane_map[vit->point()];
    }
}

void primal_from_dual (Polyhedron const& dual,
                       Polyhedron &primal) {
    Builder B;
    B.begin_surface(m_dual.size_of_facets(),
                    m_dual.size_of_vertices(),
                    m_dual.size_of_halfedges());

    std::map<Facet_handle, size_t> primal_vertices;

    // Compute the primal vertices
    size_t n = 0;
    for (Facet_iterator fit = dual.facets_begin();
         fit != dual.facets_end();
         ++fit, ++n) {
        Halfedge_handle h = fit->halfedge();

        // Primal plane asssociated to facet 'fit'
        Plane_3 p1 = h->vertex()->plane,
                p2 = h->vertex()->next()->plane,
                p3 = h->vertex()->next()->next()->plane;

        Point_3 p = CGAL::intersection(p1, p2, p3);

        B.add_vertex(p);
        // Remember the association between a facet
        // and a primal vertex
        primal_vertices[fit] = n;
    }

    // Compute the primal facets
    for (Vertex_iterator vit = dual.vertices_begin();
         vit != dual.vertices_end();
         ++vit) {
        B.begin_facet();
        Halfedge_around_vertex_circulator h0 = vit->vertex_begin(),
                                          hf = h0;

        B.begin_facet();
        // For each dual edge, add a primal edge
        do {
            B.add_vertex_to_facet(primal_vertices[hf->facet()]);
        } while (++hf != h0);
        B.end_facet();
    }

    B.end_surface();
    primal.delegate(B);
}
```

Voronoi diagrams for polyhedral norms

Given a polyhedral norm N, we define its symbolic perturbation N_ε by: $N_\varepsilon(x) = N + \varepsilon||x||$ where $\varepsilon > 0$ and $||\cdot||$ is an Euclidean norm (typically an L_p norm where $1 < p < +\infty$). We will show that the Voronoi cell of a point x under the norm N_ε converges (when $\varepsilon \to 0$) towards the Voronoi cell defined as the following:

$$V_{N'}(x,X) = \{y \in \mathbb{R}^d,\ \forall x' \in X,\ N(x-y) \leq N(x'-y) \text{ or } (N(x-y) = N(x'-y) \text{ and } ||x-y|| \leq ||x'-y||)\}$$

First, we define the convergence of a set A towards another set B: we say that A converges towards B if the Hausdorff distance of A and B converges towards 0. We recall that the Hausdorff distance is defined as follows:

$$d_H(A,B) = \min\{r \geq 0,\ X \subset Y^r \text{ and } Y \subset X^r\}$$

Now, we prove that $V_{N_\varepsilon}(x,X)$ converges towards $V_{N'}(x,X)$ under the Hausdorff distance. We want to prove that $d_H(V_{N_\varepsilon}(x,X), V_{N'}(x,X)) \to 0$. Let us take a $\delta > 0$, we want to find ε_δ such that if $\varepsilon \leq \varepsilon_\delta$ then $d_H(V_{N_\varepsilon}(x,X), V_{N'}(x,X)) \leq \delta$. The last inequality is equivalent to:

$$V_{N'}(x,X)) \subset V_{N_\varepsilon}(x,X)^\delta \tag{1}$$

$$V_{N_\varepsilon}(x,X) \subset V_{N'}(x,X)^\delta \tag{2}$$

Let us take care of the first inclusion. Let us take $y \in V_{N'}(x,X)$. By definition we have, for a given $x' \in X$:

$$N(x-y) \leq N(x'-y) \text{ or } (N(x-y) = N(x'-y) \text{ and } ||x-y|| \leq ||x'-y||)$$

So, we have two cases:

- $N(x-y) \leq N(x'-y)$: we did not have the time to look at this case during this internship. But we think that we can show that for sufficiently small values of ε, we have: $N(x-y) + \varepsilon||x-y|| \leq N(x'-y) \leq ||x'-y||$, then $y \in V_{N_\varepsilon}(x,X)$.
- $N(x-y) = N(x'-y)$ and $||x-y|| \leq ||x'-y||$: in this case, we have:

$$\begin{aligned}
N_\varepsilon(x-y) &= N(x-y) + \varepsilon||x-y|| \\
&= N(x'-y) + \varepsilon||x-y|| \\
&\leq N(x'-y) + \varepsilon||x'-y|| = N_\varepsilon(x'-y) \\
&\implies y \in V_{N_\varepsilon}(x,X) \\
&\implies y \in V_{N_\varepsilon}(x,X)^\delta \text{ for any } \delta \geq 0
\end{aligned}$$

And now the second inclusion. Let us take $y \in V_{N_\varepsilon}(x,X)$. By definition of N_ε, we have: $\forall x' \in X,\ N(x-y)+\varepsilon||x-y|| \leq N(x'-y)+\varepsilon||x'-y||$. Then, we will show that $y \in V_{N'}(x,X)$. Indeed, using the previous inequality, there are two cases: either $N(x-y) \leq N(x'-y)$ or $N(x-y) = N(x'-y)$. In the former case, we have directly that $y \in V_{N'}(x,X)$ by definition. In the latter case, the inequality reduces to: $||x-y|| \leq ||x'-y||$ and so $y \in V_{N'}(x,X)$ i.e. $y \in V_{N'}(x,X)^\delta$ for any δ.

We can also prove that this new Voronoi diagram has good properties, notably that this new diagram forms a "partition" of $\mathbb{R}^d$. "Partition", here, must be understand in the following sense: for any $p,q \in P$, $p \neq q$, the intersection $V_{N'}(p,P) \cap V_{N'}(q,P)$ has a zero measure. This property allows us to use the formula:

$$Vol(P+rB_N(0,1)) = \sum_{p \in P} Vol(V_{N'}(p,P) \cap B_N(p,r))$$

Indeed, if the Voronoi diagram was not a partition, then some regions would have been counted twice.

Bibliography

[1] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T Silva. Computing and rendering point set surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 9(1):3–15, 2003.

[2] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[3] Dominique Attali and Herbert Edelsbrunner. Inclusion-exclusion formulas from independent complexes. *Discrete & Computational Geometry*, 37(1):59–77, 2007.

[4] Frederic Cazals, Harshad Kanhere, and Sebastien Loriot. Computing the volume of a union of balls: a certified algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):3, 2011.

[5] Frédéric Cazals and Marc Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121–146, 2005.

[6] Antonin Chambolle, Massimiliano Morini, and Marcello Ponsiglione. A nonlocal mean curvature flow and its semi-implicit time-discrete approximation. *SIAM Journal on Mathematical Analysis*, 44(6):4048–4077, 2012.

[7] Adina Ciomaga, Pascal Monasse, and Jean-Michel Morel. Level lines shortening yields an image curvature microscope. In *ICIP*, pages 4129–4132, 2010.

[8] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. Anisotropic geometric diffusion in surface processing. In *Proceedings of the conference on Visualization'00*, pages 397–405. IEEE Computer Society Press, 2000.

[9] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[10] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Richard Zhang. Edge-aware point set resampling. *ACM Transactions on Graphics (TOG)*, 32(1):9, 2013.

[11] Tom Ilmanen, Tom Ilmanen, and Tom Ilmanen. Lectures on mean curvature flow and related equations, 1998.

[12] Thomas Lachand-Robert and Édouard Oudet. Minimizing within convex bodies using a convex hull method. *SIAM Journal on Optimization*, 16(2):368–379, 2005.

[13] Lihong Ma. *Bisectors and Voronoi diagrams for convex distance functions*. PhD thesis, Fernuniv., Fachbereich Informatik, 2000.

[14] Quentin Mérigot. *Détection de structure géométrique dans les nuages de points*. PhD thesis, Université Nice Sophia Antipolis, 2009.

[15] Franco P. Preparata and David E. Muller. Finding the intersection of n half-spaces in time o (nlogn). *Theoretical Computer Science*, 8(1):45–55, 1979.

[16] Jörg Vollmer, Robert Mencl, and Heinrich Mueller. Improved laplacian smoothing of noisy surface meshes. In *Computer Graphics Forum*, volume 18, pages 131–138. Wiley Online Library, 1999.

[17] Joachim Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998.

[18] Hermann Weyl. On the volume of tubes. *American Journal of Mathematics*, pages 461–472, 1939.