HAL open science

Numerical Mesh Processing

Gabriel Peyré

To cite this version:

Gabriel Peyré. Numerical Mesh Processing. 2008. hal-00365931

HAL Id: hal-00365931

https://hal.science/hal-00365931v1

Preprint submitted on 5 Mar 2009

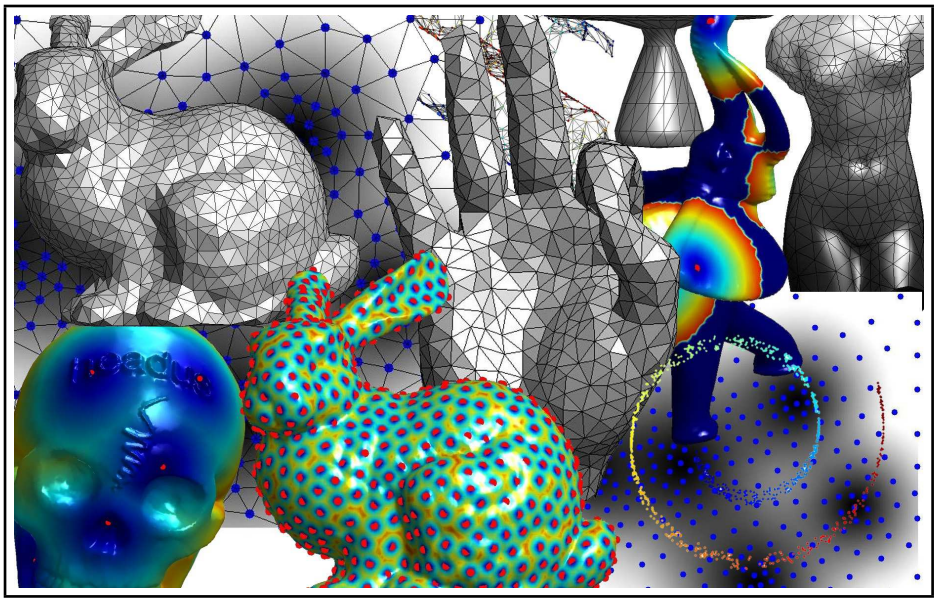
HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

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.

Numerical Mesh Processing

Gabriel Peyré
Ceremade, Université Paris Dauphine,
Place du Marchal De Lattre De Tassigny,
75775 Paris Cedex 16, France
gabriel.peyre@ceremade.dauphine.fr
http://www.ceremade.dauphine.fr/∼peyre/

April 6, 2008



These course notes are an introduction to the geometry of surfaces both from a local (differential), global (geodesic) and multiscale point of view. The emphasis is put on the discrete computation of differential and geodesic quantities on 3D meshes. These computational tools can be used to perform various mesh processing tasks such as surface denoising, compression or recognition. All the numerical experiments shown in these notes can be reproduced with a set of Matlab scripts available in the courses section of my webpage.

Contents

1 Linear Mesh Processing **1**
1.1 Surface Discretization with Triangulated Mesh . . . 1
1.1.1 Continuous Geometry of Surfaces . . . 1
1.1.2 Discretization of Surfaces with Triangulations . . . 2
1.2 Linear Mesh Processing . . . 3
1.2.1 Functions on a Mesh . . . 3
1.2.2 Local Operators . . . 4
1.2.3 Approximating Integrals on a Mesh . . . 5
1.2.4 Example on a Regular Grid . . . 7
1.2.5 Gradients and Laplacians on Meshes . . . 8
1.2.6 Examples in 1D and 2D . . . 9
1.2.7 Example of a Parametric Surface . . . 10
1.3 Diffusion and Regularization on Surfaces . . . 10
1.3.1 Heat Diffusion . . . 10
1.3.2 Spectral Decomposition . . . 12
1.3.3 Spectral Theory on a Regular Grid . . . 14
1.3.4 Spectral Resolution of the Heat Diffusion . . . 15
1.3.5 Quadratic Regularization . . . 16
1.3.6 Application to Mesh Compression . . . 16
1.3.7 Application to Mesh Parameterization . . . 18
1.3.8 Application to Mesh Flattening . . . 19

2 Geodesic Mesh Processing **21**
2.1 Manifold Geometry of Surfaces . . . 21
2.1.1 Riemannian Manifold . . . 21
2.1.2 Geodesic Distances . . . 22
2.1.3 Mapping Between Surfaces . . . 23
2.2 Numerical Computations of Geodesic Distances . . . 24
2.2.1 Front Propagation Algorithms . . . 24
2.2.2 Geodesic Computation on a Graph . . . 25
2.2.3 Geodesic Computation on a Square Grid . . . 26
2.2.4 Geodesic computation on a triangulation . . . 28
2.3 Applications and Extensions of Fast Marching . . . 31
2.3.1 Shape Analysis . . . 31
2.3.2 Heuristically Driven Propagation . . . 33
2.4 Surface Sampling . . . 36
2.4.1 Farthest Point Sampling . . . 36
2.4.2 Triangulations . . . 37
2.4.3 Centroidal Tesselation . . . 38

3 Multiresolution Mesh Processing **43**
3.1 Semi-regular Meshes . . . 43
3.1.1 Nested Multiscale Grids. . . . 43
3.1.2 Semi-regular Triangulation. . . . 43
3.1.3 Spherical Geometry Images . . . 45
3.2 Subdivision Curves . . . 46
3.3 Subdivision Surfaces . . . 47
3.3.1 Interpolation Operators . . . 48

3.3.2 Some Classical Subdivision Stencils 48
3.3.3 Invariant Neighborhoods 51
3.3.4 Convergence of Subdivisions 52
3.4 Wavelets on Meshes 53
3.4.1 Multiscale Biorthogonal Bases on Meshes 53
3.4.2 The Lifting Scheme 54
3.4.3 Imposing vanishing moments. 56
3.4.4 Lifted Wavelets on Meshes 56
3.4.5 Non-linear Mesh Compression 57

Bibliography **61**

1 Linear Mesh Processing

This chapter exposes the basics of surface approximation with 3D meshes and the way to process such meshes with linear operators. In particular, it studies filtering on 3D meshes and explains how a Fourier theory can be built to analyze these filters.

1.1 Surface Discretization with Triangulated Mesh

1.1.1 Continuous Geometry of Surfaces

In this course, in order to simplify the mathematical description of surfaces, we consider only globally parameterized surfaces. We begin by considering surfaces embedded in euclidean space $\mathcal{M} \subset \mathbb{R}^k$.

Definition 1 (Parameterized surface)**.** *A parameterized surface is a mapping*

$$u \in \mathcal{D} \subset \mathbb{R}^2 \mapsto \varphi(u) \in \mathcal{M}.$$

Of course, most surfaces do not benefit from such a simple parameterization. For instance, a sphere should be split into two parts in order to be mapped on two disks $\mathcal{D}_1, \mathcal{D}_2$. These topological difficulties require the machinery of manifolds in order to incorporate a set of charts $\mathcal{D} = \{\mathcal{D}_i\}_i$ that overlap in a smooth manner. All the explanations of this course extend seamlessly to this multi-charts setting.

A curve is defined in parameter domain as a 1D mapping $t \in [0,1] \mapsto \gamma(t) \in \mathcal{D}$. This curve can be traced over the surface and its geometric realization is $\bar{\gamma}(t) \overset{\text{def.}}{=} \varphi(\gamma(t)) \in \mathcal{M}$. The computation of the length of γ in ambient k-dimensional space $\mathbb{R}^k$ follows the usual definition, but to do the computation over the parametric domain, one needs to use a local metric defined as follow.

Definition 2 (First fundamental form)**.** *For an embedded manifold $\mathcal{M} \subset \mathbb{R}^k$, the first fundamental form is*

$$I_\varphi = \left(\langle \frac{\partial \varphi}{\partial u_i}, \frac{\partial \varphi}{\partial u_j} \rangle \right)_{i,j=1,2}.$$

This local metric I_φ defines at each point the infinitesimal length of a curve as

$$L(\gamma) \overset{\text{def.}}{=} \int_0^1 \|\bar{\gamma}'(t)\| \mathrm{d}t = \int_0^1 \sqrt{\gamma'(t)^{\mathrm{T}} I_\varphi(\gamma(t)) \gamma'(t)} \mathrm{d}t.$$

This fundamental form is an intrinsic invariant that does not depends on how the surfaces is isometrically embedded in space (since the length depends only on this tensor field I_φ). In contrast, higher order differential quantities such as curvature might depend on the bending of the surface and are thus usually not intrinsic (with the notable exception of invariants such as the gaussian curvature). In this course, we restrict ourselves to first order quantities since we are mostly interested in lengths and the intrinsic study of surfaces.

Example 1 (Isometry and conformality). A surface $\mathcal{M}$ is locally isometric to the plane if $I_\varphi = \mathrm{Id}_2$. This is for instance the case for a cylinder. The mapping φ is said to be conformal if $I_\varphi(u) = \lambda(u)\mathrm{Id}_2$. It means that the length of a curve over the plane is only locally scaled when mapped to the surface. In particular, the angle of two interescting curves is the same over the parametric domain and over the surface. This is for instance the case for the stereographic mapping between the plane and a sphere.

1.1.2 Discretization of Surfaces with Triangulations

Mesh Data Structure A triangulated mesh is a discrete structure that can be used to approximate a surface embedded in Euclidean space $\mathbb{R}^k$. It is composed of a topological part $M = (V, E, F)$ and a geometrical realization $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$. It is important to make the distinction between these two parts since many algorithms rely only on geometry (point clouds processings such as dimension reduction) or on topology (such as compression).

The topology M of the mesh is composed of

- *Vertices* (0D): this is an abstract set of indices $V \simeq \{1, \ldots, n\}$.
- *Edges* (1D): this is a set of pair of vertices $E \subset V \times V$. This set is assumed to be symmetric

$$(i, j) \in E \quad \Longleftrightarrow \quad i \sim j \Leftrightarrow (j, i) \in E.$$

- *Faces* (2D): this is a collection of 3-tuples of vertices $F \subset V \times V \times V$, with the additional compatibility condition

$$(i, j, k) \in F \quad \Longrightarrow \quad (i, j), (j, k), (k, i) \in E.$$

We further assumer that there is no isolated edges

$$\forall\, (i, j) \in E, \quad \exists\, k,\ (i, j, k) \in F.$$

The set of edges can be stored in a symmetric matrix $A \in \mathbb{R}^{n \times n}$ such that $A_{ij} = 1$ if $(i, j) \in E$ and $A_{ij} = 0$ otherwise. This matrix is often stored as a sparse matrix since the number of edges is usually much smaller than n^2. The set of vertices and edges form a non-oriented graph $\mathcal{G} = (V, E)$. Faces are often stored as a matrix $A_F \in \{1, \ldots, n\}^{3 \times m}$ where m is the number of faces and a column $((A_F)_{i,1}, (A_F)_{i,2}, (A_F)_{i,3})$ stores the indices of a face. In a triangulation, the face matrix A_F allows to recover the edge incidence matrix A. The face data structure allows to really capture the 2D geometry of surfaces, which is not possible with graphs alone.

The geometric realization $\mathcal{M}$ is defined through a spacial localization of the vertices (for instance in 3D space)

$$\mathcal{V} \overset{\text{def.}}{=} \{x_i \setminus i \in V\} \subset \mathbb{R}^3.$$

This allows to define a piecewise linear mesh

$$\mathcal{F} \overset{\text{def.}}{=} \bigcup_{(i,j,k) \in F} \mathrm{Conv}(x_i, x_j, x_k) \subset \mathbb{R}^3,$$

where the convex envelop $\mathrm{Conv}(x, y, z)$ of three points is the Euclidean triangle generated by (x, y, z).

This piecewise linear realization $\mathcal{M}$ can be displayed as a 3D surface on a computer screen. This is performed through a perspective projection of the points and a linear interpolation of color and light inside the triangle. Figure 1.1 shows an example of 3D display, with a zoom on the faces of the mesh.

Adjacency Relationships From the basis topological information given by $M = (V, E, F)$, one can deduce several adjacency data-structures that are important to navigate over the triangulation.

Definition 3 (Vertex 1-ring)**.** *The vertex 1-ring of a vertex $i \in V$ is*

$$V_i \overset{\text{def.}}{=} \{j \in V \setminus (i, j) \in E\} \subset V.\text{‘} \tag{1.1}$$

The s-ring is defined by induction as

$$\forall\, s > 1, \quad V_i^{(s)} = \left\{ j \in V \setminus (k, j) \in E \quad \textit{and} \quad k \in V_i^{(s-1)} \right\}. \tag{1.2}$$

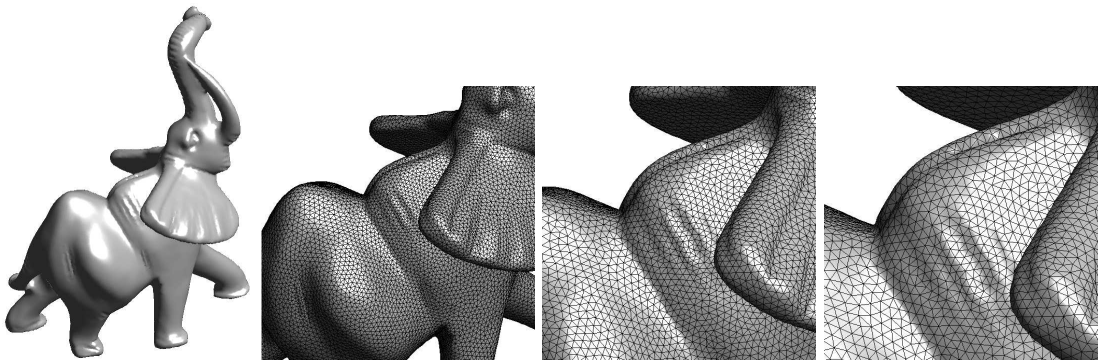


Figure 1.1: *Example of display of a 3D mesh.*

Definition 4 (Face 1-ring)**.** *The face 1-ring of a vertex $i \in V$ is*

$$F_i \stackrel{\text{def.}}{=} \{(i,j,k) \in F \setminus i,j \in V\} \subset F.$$

The geometrical realization of a vertex 1-ring is

$$\mathcal{V}_i = \bigcup_{(i,j,k)\in V_i} \mathrm{Conv}(x_i, x_j, x_k).$$

A triangulated mesh is a manifold mesh if all the rings $\mathcal{V}_i$ for $i \in V$ are homeomorphic to either a disk (for interior vertices) or to a half disk (for boundary vertices). This ensures that the geometrical mesh really has the topology of a 2D surface embedded in $\mathbb{R}^3$ (possibly with boundaries). In particular, it implies that there is at most two faces connected to each edge

$$\forall\, (i,j) \in E, \quad \#\{k \setminus (i,j,k) \in F\} \leqslant 2.$$

As an application of these local rings, one can compute a normal at each point using a simple rule

$$\forall\, f = (i,j,k) \in F, \quad \overrightarrow{n_f} \stackrel{\text{def.}}{=} \frac{(x_j - x_i) \wedge (x_k - x_i)}{\|(x_j - x_i) \wedge (x_k - x_i)\|}.$$

and where

$$\forall\, i \in V, \quad \overrightarrow{n_i} \stackrel{\text{def.}}{=} \frac{\sum_{f\in F_i} \overrightarrow{n_f}}{\|\sum_{f\in F_i} \overrightarrow{n_f}\|}.$$

These normals are used to define for instance a light intensity $I(i) = \max(\langle n_i, \ell(i), , \rangle 0)$, where $\ell(i)$ is the incident light. In practice one uses a infinite light source $\ell(i) = \ell =$constant or a local spot located at position $s \in \mathbb{R}^3$ through $\ell(i) = (v_i - s)/\|v_i - s\|$. This light intensity is interpolated on the whole mesh during display.

1.2 Linear Mesh Processing

The light intensity I is a particular example of a function defined at each vertex of the mesh. Mesh processing is intended to process such functions and we thus define carefully vector spaces and operators on meshes.

1.2.1 Functions on a Mesh

In this course, a function is a discrete set of values defined at each vertex location.

Definition 5 (Linear space on a mesh)**.** *A function on a mesh is a mapping $f \in \ell^2(\mathcal{V}) \simeq \ell^2(V) \simeq \mathbb{R}^n$ and can be viewed equivalently as*

$$f : \left\{ \begin{array}{ccc} \mathcal{V} & \longrightarrow & \mathbb{R} \\ x_i & \longmapsto & f(x_i) \end{array} \right. \quad \Longleftrightarrow \quad f : \left\{ \begin{array}{ccc} V & \longrightarrow & \mathbb{R} \\ i & \longmapsto & f_i \end{array} \right. \quad \Longleftrightarrow \quad f = (f_i)_{i \in V} \in \mathbb{R}^n.$$

The linear space of the functions on a mesh is equipped with an Hilbert space structure that allows to quantify approximation error and compute projections of functions.

Definition 6 (Inner product and norm)**.** *One defines the following inner product and norm for vector $f, g \in \mathbb{R}^n$*

$$\langle f, g \rangle \overset{\text{def.}}{=} \sum_{i \in V} f_i g_i \quad \textit{and} \quad \|f\|^2 = \langle f, f \rangle.$$

In order to modify (process) functions on a mesh (such as a light intensity I), this course considers only linear operations that are defined through a large matrix.

Definition 7 (Linear operator A)**.** *A linear operator A is defined as*

$$A : \ell^2(V) \to \ell^2(V) \quad \Longleftrightarrow \quad A = (a_{ij})_{i,j \in V} \in \mathbb{R}^{n \times n} \textit{ (matrix).}$$

and operate on a function f as follow

$$(Af)(x_i) = \sum_{j \in V} a_{ij} f(x_j) \Longleftrightarrow \quad (Af)_i = \sum_{j \in V} a_{ij} f_j.$$

Example 2. If the coordinates of the point of a mesh are written $x_i = (x_i^1, x_i^2, x_i^3) \in \mathbb{R}^3$, then the X-coordinate defines a function $f : i \in V \mapsto x_i^1 \in \mathbb{R}$. A geometric mesh $\mathcal{M}$ is thus 3 functions defined on M.

Mesh processing is the task of modifying functions $f \in \ell^2(V)$. For instance, one can denoise a mesh $\mathcal{M}$ as 3 functions on M. The usual strategy applies a linear operator $f \mapsto Af$. Sometimes, A can computed from M only (for instance for compression) but most of the times it requires both M and $\mathcal{M}$.

1.2.2 Local Operators

In most applications, one can not store and manipulate a full matrix $A \in \mathbb{R}^{n \times n}$. Furthermore, one is usually interested in exploiting the local redundancies that exist in most usual functions $f \in \mathbb{R}^n$ defined on a mesh. This is why we restrict our attention to local operators that can be conveniently stored as sparse matrices (the zeros are not kept in memory).

Definition 8 (Local operator)**.** *A local operator $W \in \mathbb{R}^{n \times n}$ satisfies $w_{ij} = 0$ if $(i, j) \notin E$.*

$$(Wf)_i = \sum_{(i,j) \in E} w_{ij} f_j.$$

A particularly important class of local operators are local smoothings (also called filterings) that perform a local weighted sum around each vertex of the mesh. For this averaging to be consistent, we define a normalized operator $\tilde{W}$ whose set of weights sum to one.

Definition 9 (Local averaging operator)**.** *A local normalized averaging is $\tilde{W} = (\tilde{w}_{ij})_{i,j \in V} \geqslant 0$ where*

$$\forall (i, j) \in E, \quad \tilde{w}_{ij} = \frac{w_{ij}}{\sum_{(i,j) \in E} w_{ij}}.$$

It can be equivalently expressed in matrix form as

$$\tilde{W} = D^{-1} W \quad \textit{with} \quad D = \mathrm{diag}_i(d_i) \quad \textit{where} \quad d_i = \sum_{(i,j) \in E} w_{ij}.$$

The smoothing property corresponds to $\tilde{W}1 = 1$ which means that the unit vector is an eigenvector of W with eigenvalue 1.

Example 3. In practice, we use three popular kinds of averaging operators.

- *Combinatorial weights:* they depends only on the topology (V, E) of the vertex graph

$$\forall\,(i,j) \in E, \quad w_{ij} = 1.$$

- *Distance weights:* they depends both on the geometry and the topology of the mesh, but do not require faces information,

$$\forall\,(i,j) \in E, \quad w_{ij} = \frac{1}{\|x_j - x_i\|^2}.$$

- *Conformal weights:* they depends on the full geometrical realization of the 3D mesh since they require the face information

$$\forall\,(i,j) \in E, \quad w_{ij} = \cot(\alpha_{ij}) + \cot(\beta_{ij}). \tag{1.3}$$

Figure 1.2 shows the geometrical meaning of the angles α_{ij} and β_{ij}

$$\alpha_{ij} = \angle(x_i, x_j, x_{k_1}) \qquad \text{and} \qquad \beta_{ij} = \angle(x_i, x_j, x_{k_2}),$$

where $(i, j, k_1) \in F$ and $(i, j, k_2) \in F$ are the two faces adjacent to edge $(i, j) \in E$. We will see in the next section the explanation of these celebrated cotangent weights.

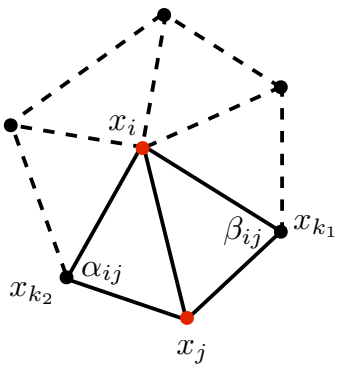


Figure 1.2: *One ring around a vertex i, together with the geometrical angles α_{ij} and β_{ij} used to compute the conformal weights.*

One can use iteratively a smoothing in order to further filter a function on a mesh. The resulting vectors $\tilde{W}f, \tilde{W}^2, \ldots, \tilde{W}^k f$ are increasingly smoothed version of f. Figure 1.3 shows an example of such iterations applied to the three coordinates of mesh. The sharp features of the mesh tend to disappear during iterations. We will make this statement more precise in the following, by studying the convergence of these iterations.

1.2.3 Approximating Integrals on a Mesh

Before investigating algebraically the properties of smoothing operators, one should be careful about what are these discrete operators really approximating. In order for the derivation to be simple, we make computation for a planar triangulation M of a mesh $\mathcal{M} \subset \mathbb{R}^2$.

In the continuous domain, filtering is defined through integration of functions over the mesh. In order to descretize integrals, one needs to define a partition of the plane into small cells centered around a vertex or an edge.

Definition 10 (Vertices Voronoi)**.** *The Voronoi diagram associated to the vertices is*

$$\forall\, i \in V, \quad E_i = \{x \in \mathcal{M} \setminus \forall\, j \neq i, \|x - x_i\| \leqslant \|x - x_j\|\}$$

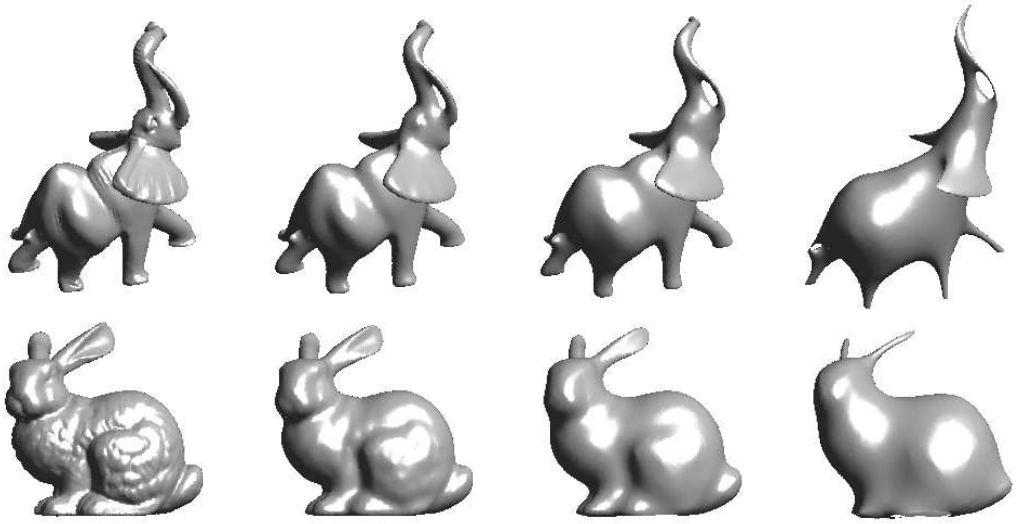


Figure 1.3: *Examples of iterative smoothing of a 3D mesh.*

Definition 11 (Edges Voronoi)**.** *The Voronoi diagram associated to the edges is*

$$\forall\, e = (i,j) \in E, \quad E_e = \{x \in \mathcal{M} \setminus \forall\, e' \neq e, d(x,e) \leqslant d(x,e')\}$$

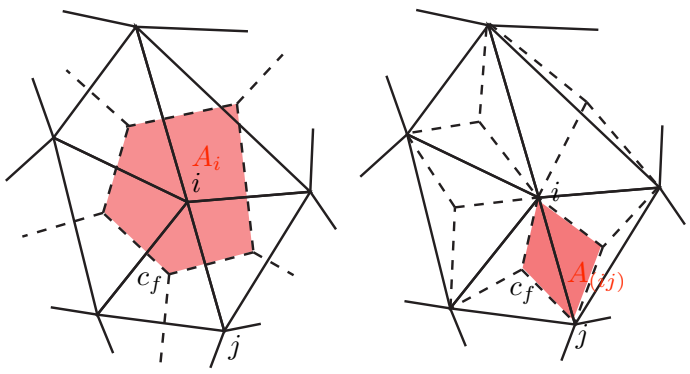


Figure 1.4: *Left: vertex Voronoi cell, right: delaunay Voronoi cell. The point c_f is the orthocenter of a face $f = (i,j,k)$.*

These Voronoi cells indeed form a partition of the mesh

$$\mathcal{M} = \bigcup_{i \in V} E_i = \bigcup_{e \in E} E_e.$$

The following theorem gives the formula for the area of these cells.

Theorem 1 (Voronoi area formulas)**.** *For all $e = (i,j) \in E$, $\forall\, i \in V$, one has*

$$A_e = Area(E_e) = \frac{1}{2}\|x_i - x_j\|^2 \left(\cot(\alpha_{ij}) + \cot(\beta_{ij})\right)$$

$$A_i = Area(E_i) = \frac{1}{2} \sum_{j \in N_i} A_{(ij)}.$$

With these areas, one can approximate integrals on vertices and edges using

$$\int_{\mathcal{M}} f(x)\mathrm{d}x \approx \sum_{i \in V} A_i\, f(x_i) \approx \sum_{e=(i,j) \in E} A_e\, f([x_i, x_j]).$$

Of particular interest is the approximation of the so-called Dirichelet energy $\int_{\mathcal{M}} \|\nabla_x f\|^2 \mathrm{d}x$. In order to compute it on a triangular mesh, one can use a finite difference approximation of the gradient of a function at the point $x_{ij} = (x_i + x_j)/2$ along an edge (i, j)

$$\langle \nabla_{x_{ij}} f, \frac{x_i - x_j}{\|x_i - x_j\|} \rangle \approx \frac{f(x_i) - f(x_j)}{\|x_i - x_j\|}.$$

This leads to the following approximation of the Dirichlet energy

$$\int_{\mathcal{M}} \|\nabla_x f\|^2 \mathrm{d}x \approx \sum_{(i,j)\in E} A_{(i,j)} \langle \nabla_{x_{ij}} f, \frac{x_i - x_j}{\|x_i - x_j\|} \rangle^2 \approx \sum_{(i,j)\in E} A_{(i,j)} \frac{|f(x_j) - f(x_i)|^2}{\|x_j - x_i\|^2} \tag{1.4}$$

$$= \sum_{(i,j)\in E} w_{ij} |f(x_j) - f(x_i)|^2 \qquad \text{where} \qquad w_{ij} = \cot(\alpha_{ij}) + \cot(\beta_{ij}). \tag{1.5}$$

This discrete formulation shows that the correct weights to approximate the Dirichlet energy are the cotangent one, already introduced in equation (1.3).

1.2.4 Example on a Regular Grid

A regular grid is an uniform discretization with n points of $[0, 1)$ (in 1D) or $[0, 1)^2$ (in 2D). One usually assumes periodic boundary conditions, which means that each side of the square is associated with its opposite.

Since the geometry of a regular grid is invariant under translation, local averaging operators can be computed as convolution on $D = (\mathbb{Z}/p\mathbb{Z})^d$ where $n = p^d$ for d the dimension of the domain ($d = 1$ or $d = 2$)

$$\forall\, i \in D, \quad \tilde{W} f(i) = \sum_{k\in D} f(k) \tilde{w}(i - k),$$

where the operation $+$ and $-$ should be computed modulo p and $\tilde{w}(k) = \tilde{W}(0, k)$ is the convolution kernel.

Example 4 (Averaging). The uniform averaging filter is defined as

$$\tilde{W} f(i) = \frac{1}{|N|} \sum_{k\in N} f(i + k),$$

where N is the set of neighbors of the point 0 and $|N| = 2^d$. In this case, in dimension 1, $\tilde{w} = (1, 0, 1)/2$, where this notation assumes that $\tilde{w}$ is centered at the point 0.

In order to study translation invariant operators like local filtering, one needs to use the discrete Fourier transform that diagonalizes these operators.

Definition 12 (Discrete Fourier transform)**.** *The 1D discrete Fourier transform $\Phi(f) \in \mathbb{C}^n$ of the vector $f \in \mathbb{C}^n$*

$$\Phi(f)(\omega) = \hat{f}(\omega) \overset{\text{def.}}{=} \frac{1}{n} \sum_k f_k e^{\frac{2\mathrm{i}\pi}{n} k\omega}.$$

A similar definition can be given for the 2D discrete Fourier transform. The main property of the Fourier transform is the following diagonalization result.

Theorem 2 (Convolution and Fourier)**.** *For any vector f, one has*

$$\Phi(\tilde{W}^k f) = \Phi(\tilde{w} * \ldots * \tilde{w} * f) \quad \Longrightarrow \quad \Phi(\tilde{W}^k f)(\omega) = \widehat{\tilde{w}}(\omega)^k \, \hat{f}(\omega).$$

The main interest of this tools is that $\Phi(f)$ can be computed in $O(n \log(n))$ operations with the FFT algorithm. Using the following theorem, it gives an alternative expression of a local filtering. This expression in the Fourier domain can be used to speed up the computation of $\tilde{w} * f$ if $\tilde{w}$ has a lot of non zero entries (which is not the case in our setting of local operators). It is also useful to analyze theoretically the behavior of iterated filterings.

Theorem 3 (Convergence)**.** *For any function f defined on a regular grid in 1D or 2D, one has*

$$\tilde{W}^k f \overset{k\to+\infty}{\longrightarrow} \frac{1}{|V|}\sum_{i\in V} f_i$$

This Fourier theory can only be developed for domains that have a group structure that enables translation invariant filtering. In particular, it does not carry over easily to an arbitrary surface. In the remaining, we define a corresponding theory for graphs and triangulated surfaces using the eigenvector of Laplacian operators. This Fourier transform on meshes enables the analysis of the convergence of many filtering schemes.

1.2.5 Gradients and Laplacians on Meshes

Gradient operator A gradient operator defines directional derivatives on a triangulation. It maps functions defined on vertices to functions defined on the set of oriented edges

$$\bar{E} \overset{\text{def.}}{=} \{(i,j) \in E \setminus i > j\}.$$

Definition 13 (Gradient)**.** *Given a local averaging W, the gradient operator G is defined as*

$$\forall\,(i,j) \in E,\ i<j, \quad (Gf)_{(i,j)} \overset{\text{def.}}{=} \sqrt{w_{ij}}(f_j - f_i) \in \mathbb{R}.$$

This mapping can be viewed equivalently as

$$\begin{array}{lll} G : \ell^2(V) \longrightarrow \ell^2(E), & \text{or} & G : \mathbb{R}^n \longrightarrow \mathbb{R}^p \quad \text{where} \quad p = |E|, \\ & \text{or} & G \in \mathbb{R}^{n\times p} \quad \text{(a matrix)}. \end{array}$$

The value of $(Gf)_e$ for an edge $e = (i,j)$ can be thought as a derivative along direction $\overrightarrow{x_i x_j}$.
Example 5. For the local averaging based on square distances, one has

$$w_{ij} = \|x_i - x_j\|^{-2}, \quad (Gf)_{(i,j)} = \frac{f(x_j) - f(x_i)}{\|x_i - x_j\|}.$$

which is exactly the finite difference discretization of a directional derivative.

One a regular grid, one can note that

- Gf discretizes $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)^{\mathrm{T}}$.
- $G^{\mathrm{T}} v$ discretizes $\mathrm{div}(v) = \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y}$.

Laplacian Operator A Laplacian operator is a discrete version of a second order derivative operator.

Definition 14 (Laplacian)**.** *Given a local averaging W, the Laplacian operator D is defined as*

$$L \overset{\text{def.}}{=} D - W, \quad \textit{where} \quad D = \mathrm{diag}_i(d_i), \quad \textit{with} \quad d_i = \sum_j w_{ij}.$$

In the remaining, we also make use of normalized operators, which have an unit diagonal.

Definition 15 (Normalized Laplacian)**.** *The normalized Laplacian is defined as*

$$\tilde{L} \overset{\text{def.}}{=} D^{-1/2} L D^{-1/2} = \mathrm{Id}_n - D^{-1/2} W D^{1/2} = \mathrm{Id}_n - D^{1/2} \tilde{W} D^{-1/2}.$$

This normalized Laplacian correspond to the weighted graph Laplacian used in graph theory, see for instance [4].

Remark 1. One can note that

– Laplacians are symmetric operators $L, \tilde{L} \in \mathbb{R}^{n\times n}$.
– L acts like a (second order) derivative since $L1 = 0$.
– in contrast, the normalized Laplacian is not a real derivative since $\tilde{L}1 \neq 0$ in general.

The main interest of the gradient operator is that it factorizes the Laplacian as follow.

Theorem 4 (Laplacian factorization)**.** *One has*

$$L = G^{\mathrm{T}}G \quad \textit{and} \quad \tilde{L} = (GD^{-1/2})^{\mathrm{T}}(GD^{-1/2}).$$

This theorem proves in particular that L and $\tilde{L}$ are symmetric positive definite operators. The inner product defined by the Laplacian can be expressed as an energy summed over all the edges of the mesh

$$\langle Lf,\, f\rangle = \|Gf\|^2 = \sum_{(i,j)\in E} w_{ij}\|f_i - f_j\|^2.$$

In the particular case of the cotangent weights introduced in equation (1.3), one can see that the Laplacian norm $\langle Lf,\, f\rangle$ is exactly the finite differences approximation of the continuous Dirichlet energy $\int_{\mathcal{M}} |\nabla_x f|\mathrm{d}x$ derived in equation (1.5). This is why these cotangent weights are the best choice to compute a Laplacian that truly approximates the continuous Laplace Beltrami operator (see definition 16).

A similar expression is derived for the normalized laplacian

$$\langle \tilde{L}f,\, f\rangle = \|GD^{-1/2}f\|^2 = \sum_{(i,j)\in E} w_{ij}\left\|\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right\|^2.$$

Of particular interest for the study of filtering on meshes is the behavior of the spectrum of the Laplacian. We can first study its kernel.

Theorem 5 (Kernel of the Laplacian)**.** *If M is connected, then*

$$\textit{ker}(L) = \textit{span}(1) \quad \textit{and} \quad \textit{ker}(L) = \textit{span}(D^{1/2}).$$

1.2.6 Examples in 1D and 2D

In 1D, all local weights are equivalent since the points are equi-spaced. The corresponding Laplacian is a convolution that can be written as

$$(Lf)_i = \frac{1}{h^2}\left(2f_i - f_{i+1} - f_{i-1}\right) = \frac{1}{h^2} f * (-1, 2, 1)\,,$$

where it is important to remember that the notation $(-1, 2, 1)$ means that the vector is centered around 0.

This discrete 1D Laplacian is the finite difference approximation of the continuous Laplacian on the torus $\mathcal{T}$ of the segment $[0, 1)$ modulo 1. Up to a minus sign, this Laplacian is just the second order derivative

$$L \overset{h\to 0}{\longrightarrow} -\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(x_i)$$

One should be careful with our notation that consider positive semi-definite Laplacian, that have the opposite sign with respect to second order derivative operators (which are definite negative).

The gradient operator corresponds to a discretization of the first order derivative $f \mapsto f'$ (which is anti symmetric). The continuous counterpart of the factorization $L = G^{\mathrm{T}}G$ is the integration by part formula on the torus

$$\int_{\mathcal{T}} f''(x)g(x)\mathrm{d}x = -\int_{\mathcal{T}} -f(x)g'(x)\mathrm{d}x \quad \Longrightarrow \quad \int_{\mathcal{T}} f''(x)f(x)\mathrm{d}x = -\int_{\mathcal{T}} |f'(x)|^2 \leqslant 0.$$

The discrete Laplacian on a 2D grid can also be written as a 2D convolution

$$(Lf)_i = \frac{1}{h^2}\left(4f_i - f_{j_1} - f_{j_2} - f_{j_3} - f_{j_4}\right) = \frac{1}{h^2} f * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

where $\{j_k\}_k$ are the four neighbors of the point i. This operator is the finite difference approximation to the continuous 2D Laplacian

$$L \overset{h\to 0}{\longrightarrow} -\frac{\partial^2 f}{\partial x^2}(x_i) - \frac{\partial^2 f}{\partial y^2}(x_i) = -\Delta f(x_i).$$

The factorization $Lf = G^{\mathrm{T}}Gf$ corresponds to the decomposition $\Delta f = \mathrm{div}(\nabla f)$.

1.2.7 Example of a Parametric Surface

We recall that a parameterized surface is a mapping $u \in \mathcal{D} \subset \mathbb{R}^2 \mapsto \varphi(u) \in \mathcal{M}$. Whereas the continuous Laplacian is simple to define on the plane using partial derivatives, its definition on a surface requires the intervention of an arbitrary parameterization φ which makes its expression cumbersome.

Definition 16 (Laplace-Beltrami)**.** *The Laplace-Beltrami operator on a parametric surface $\mathcal{M}$ is defined as*

$$\sqrt{g}\Delta_{\mathcal{M}} \overset{\text{def.}}{=} \frac{\partial}{\partial u_1}\left(\frac{g_{22}}{\sqrt{g}}\frac{\partial}{\partial u_1} - \frac{g_{12}}{\sqrt{g}}\frac{\partial}{\partial u_2}\right) + \frac{\partial}{\partial u_2}\left(\frac{g_{11}}{\sqrt{g}}\frac{\partial}{\partial u_2} - \frac{g_{12}}{\sqrt{g}}\frac{\partial}{\partial u_1}\right)$$

where $g = \det(I_\varphi)$ and $I_\varphi = (g_{ij})_{i,j=1,2}$.

The Laplacian is however an intrinsic operator that does not depends on the chosen parameterization, as shown by the following approximation theorem.

Remark 2 (Laplacian using averaging).

$$\Delta_{\mathcal{M}} f(x) = \lim_{h\to 0} \frac{1}{|B_h(x)|}\int_{y\in\mathcal{M}} f(y)\mathrm{d}y \quad \text{where} \quad B_h(x) = \{y \setminus d_{\mathcal{M}}(x,y) \leqslant h\}$$

where $d_{\mathcal{M}}$ is the geodesic distance on $\mathcal{M}$ and $h = \max_{(i,j)\in E} \|x_i - x_j\|$ is the discretization precision.

1.3 Diffusion and Regularization on Surfaces

1.3.1 Heat Diffusion

The main linear PDE for regularization of functions is the heat equation that governs the isotropic diffusion of the values of a function in time.

Definition 17 (Heat diffusion)**.** *$\forall\, t > 0$, one defines $F_t : M \to \mathbb{R}$ solving*

$$\frac{\partial F_t}{\partial t} = -D^{-1}LF_t = -(\mathrm{Id}_n - \tilde{W})F_t \qquad \textit{and} \qquad \forall\, i \in V,\ F_0(i) = f(i)$$

In order to compute numerically the solution of this PDE, one can fix a time step $\delta > 0$ and use an explicit discretization in time $\bar{F}_k$ as $F_0 = f$ and

$$\frac{1}{\delta}\left(\bar{F}_{k+1} - \bar{F}_k\right) = -D^{-1}L\bar{F}_k \quad \Longrightarrow \quad \bar{F}_{k+1} = \bar{F}_k - \delta D^{-1}L\bar{F}_k = (\mathrm{Id} - \delta)\bar{F}_k + \delta\tilde{W}\bar{F}_k. \tag{1.6}$$

If δ is small enough, one hopes that the discrete solution $\bar{F}_k$ is close to the continuous time solution F_t for $t = \delta k$. This is indeed the case as proven later in these notes.

Remark 3. In order for this scheme to be stable, one needs $\delta < 1$. This is be proven later using the extension of Fourier theory to meshes.

Remark 4. If $\delta = 1$, then the discretization of the Heat equation corresponds to iterative smoothing since $\bar{F}_k = \tilde{W}^k f$. In this case stability is not guaranteed but only pathological meshes give unstable filtering (see theorem 13).

Instead of using the explicit discretization in time (1.6), one can use an implicit scheme which compute an approximate solution $\tilde{F}_k$ at step k by solving

$$\frac{1}{\delta}\left(\tilde{F}_{k+1} - \tilde{F}_k\right) = -D^{-1}L\tilde{F}_{k+1} \quad \Longrightarrow \quad ((\delta+1)\mathrm{Id}_n - \delta\tilde{W})\tilde{F}_{k+1} = \tilde{F}_k. \tag{1.7}$$

Computing $\tilde{F}_k$ requires the solution of a sparse linear system at each step k. The implicit scheme (1.7) is thus computationally more involved than the explicit scheme (1.6). We will however see later that the implicit scheme is always stable for any value of $\delta \leqslant 1$.

Example 6 (Mesh smoothing). In order to smooth a mesh whose points are $x_i = (x_i^1, x_i^2, x_i^3)$, one can perform a heat diffusion for each component $f_i = (x_i^k), k = 1, 2, 3$. Figure 1.5 shows an example of such a smoothing.

In practice, mesh smoothing is used to denoise a function $f = f_0 + \sigma g$ where $g \in \mathbb{R}^n$ is a realization of a gaussian white noise (each entry $g(i)$ are independent and follow a gaussian law with unit variance). The difficult task it to find an optimal stopping time t to minimize $\|F_t - f_0\|$, which is not available since one does not know f_0. For uniformly smooth surfaces, the theory predicts that a linear filtering such as the heat equation requires a stopping time proportional to the noise level σ. This is however false for more complex surfaces such as the one used in computer graphics. In these case, alternate non linear diffusions such as non-linear PDE or wavelet thresholding usually perform better, see [24] for an overview of these methods in image processing.

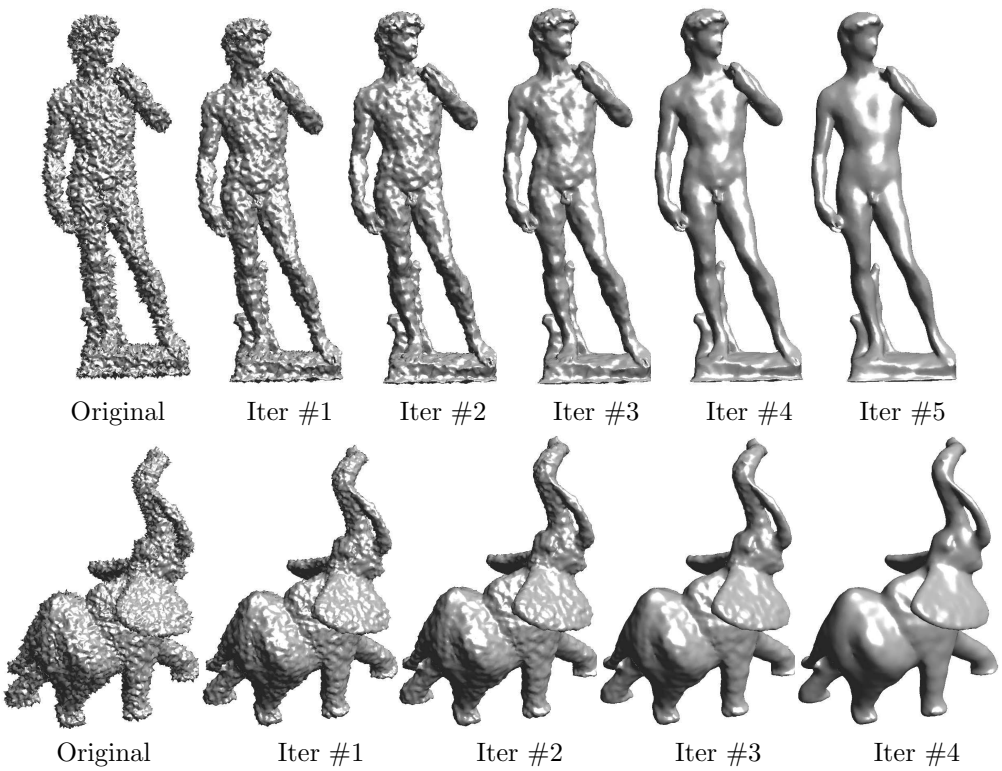


Figure 1.5: *Examples of mesh denoising with the heat equation.*

Other differential equations. One can solve other partial differential equations involving the Laplacian over a 3D mesh $M = (V, E, F)$. For instance, one can consider the wave equation, which defines, for all $t > 0$, a vector $F_t \in \ell^2(V)$ as the solution of

$$\frac{\partial^2 F_t}{\partial t^2} = -D^{-1} L F_t \qquad \text{and} \qquad \begin{cases} F_0 = f \in \mathbb{R}^n, \\ \frac{\mathrm{d}}{\mathrm{d}t} F_0 = g \in \mathbb{R}^n, \end{cases} \tag{1.8}$$

In order to compute numerically the solution of this PDE, one can fix a time step $\delta > 0$ and use an explicit discretization in time $\bar{F}_k$ as $F_0 = f$, $F_1 = F_0 + \delta g$ and for $k > 1$

$$\frac{1}{\delta^2}\left(\bar{F}_{k+1} + \bar{F}_{k-1} - 2\bar{F}_k\right) = -D^{-1} L \bar{F}_k \quad \Longrightarrow \quad \bar{F}_{k+1} = 2\bar{F}_k - \bar{F}_{k-1} - \delta^2 D^{-1} L \bar{F}_k.$$

Figure 1.6 shows examples of the resolution of the wave equation on 3D meshes.

Figure 1.6: *Example of evolution of the wave equation on 3D mesh. The initial condition f is a superposition of small positive and negative gaussians.*

1.3.2 Spectral Decomposition

In order to better understand the behavior of linear smoothing on meshes, one needs to study the spectral content of Laplacian operators. This leads to the definition of a Fourier theory for meshes. The decomposition $\tilde{L} = (GD^{-1/2})^{\mathrm{T}}(GD^{-1/2})$ of the Laplacian implies that it is a positive semi-definite operator. One can thus introduce the following orthogonal factorization.

Theorem 6 (Eigen-decomposition of the Laplacian)**.** *It exists a matrix U, $\quad U^{\mathrm{T}} U = \mathrm{Id}_n$ such that*

$$\tilde{L} = U \Lambda U^{\mathrm{T}} \qquad \textit{where} \qquad \Lambda = \mathrm{diag}_\omega(\lambda_\omega), \qquad \lambda_1 \leqslant \ldots \leqslant \lambda_n.$$

The eigenvalues λ_ω correspond to a frequency index that ranks the eigenvectors u_ω of $U = (u_\omega)_\omega$. One can first state some bounds on these eigenvalues.

Theorem 7 (Spectral bounds)**.** $\forall\, i,\ \lambda_i \in [0, 2]$ *and*

- *If M is connected then $0 = \lambda_1 < \lambda_2$.*
- *$\lambda_n = 2$ if and only if M is 2-colorable.*

We recall the definition of a colorable graph next.

Definition 18 (Colorable graph)**.** *A graph (V, E) is k-colorable if it exist a mapping $f : V \to \{1, \ldots, k\}$ such that*

$$\forall\, (i, j) \in E, \quad f(i) \neq f(j).$$

A 2-colorable graph is also called bi-partite. A 2-colorable mesh is pathological for filtering since one can split the set of vertices into two parts without inner connexions. The filtering process can oscillate by exchanging values between these sets, thus never converging.

The orthogonal eigen-basis $U = (u_\omega)_\omega$ is an orthogonal basis of the space $\mathbb{R}^n \simeq \ell^2(V)$, which can be written as

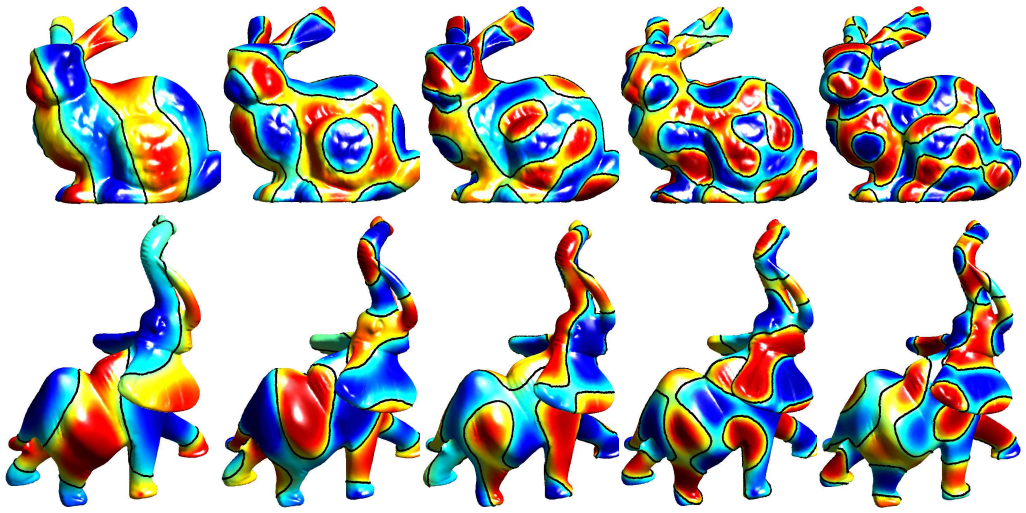
$$u_\omega : \left\{ \begin{array}{ccc} V & \longrightarrow & \mathbb{R} \\ i & \longmapsto & u_\omega(x_i) \end{array} \right.$$

The orthogonality means that $\langle u_\omega, u_{\omega'} \rangle = \delta_\omega^{\omega'}$. This basis allows to compute an orthogonal decomposition of any functions f

$$\forall\, f \in \ell^2(V), \quad f = \sum_\omega \langle f, u_\omega \rangle u_\omega.$$

Having such a tool allows to split a function f in elementary contributions $\langle f, u_\omega \rangle$ with a control in the energy because of orthogonality

$$\|f\|^2 = \sum_\omega |\langle f, u_\omega \rangle|^2.$$



$\omega = 2$ $\omega = 4$ $\omega = 8$ $\omega = 12$ $\omega = 16$

Figure 1.7: *Examples of eigenvectors u_ω of the Laplacian $\tilde{L}$. The blue colors indicated negative values, red colors positive ones. The black curve is the 0 level set of the eigenvector.*

Figure 1.7 shows some examples of eigenfunctions depicted using color ranging from blue (negative values of the eigenfunction) to red (positive values). One can see that these functions are oscillating, in a way similar to the traditional Fourier basis. In some sense (made more precise latter), this basis is the extension of the Fourier basis to meshes. A function u_ω corresponding to a large spectral value λ_ω is highly oscillating and corresponds thus intuitively to a high frequency atom.

Extracting numerically eigenvectors from a large matrix is a difficult problem. If the matrix is sparse, a method of choice consists in using iterative powers of a shifted version of the laplacian. One starts from a random initial vector v_0 and iterates

$$v_{k+1} = \frac{w_{k+1}}{\|w_{k+1}\|} \qquad \text{where} \qquad w_{k+1} = (\tilde{L} - \lambda \mathrm{Id}_n)^{-1} v_k. \tag{1.9}$$

These iterates converges to the eigenvectors corresponding to the eigenvalue the closest to λ, as staten in the following theorem.

Theorem 8 (Inverse iterations)**.** *For a given shift λ, lets denote*

$$\omega^\star = \operatorname*{argmin}_\omega |\lambda - \lambda_\omega| \qquad \textit{and} \qquad \omega^+ = \operatorname*{argmin}_{\omega \neq \omega^\star} |\lambda - \lambda_\omega|$$

If $|\lambda - \lambda_{\omega^\star}| < |\lambda - \lambda_{\omega^+}|$, then

$$v_k \overset{k \to +\infty}{\longrightarrow} u_{\omega^\star} \quad \textit{and} \quad \langle L v_k,\, v_k \rangle \overset{k \to +\infty}{\longrightarrow} \lambda_{\omega^\star}.$$

The speed of convergence of these inverse iterations is governed by the conditioning of $(\tilde{L} - \lambda \mathrm{Id}_n)^{-1}$ since

$$\|v_k - u_{\omega^\star}\| \leqslant C \rho(\lambda)^k \qquad \text{where} \qquad \rho(\lambda) \overset{\text{def.}}{=} \frac{|\lambda - \lambda_{\omega^\star}|}{|\lambda - \lambda_{\omega^+}|} < 1.$$

The smallest $\rho(\lambda)$ is, the faster the method converges.

In order to compute an iteration (1.9) of the method, one needs to solve a sparse linear system $A w_{k+1} = v_k$ whith $A = \tilde{L} - \lambda \mathrm{Id}_n$. In order to do so, one can use a direct method such as LU factorization. The advantage of such an approach is that the factorization is computed once for all and can be re-used to solve very quickly at each step k. These factorization are however quite slow to compute especially for large matrices. For large problems, one can solve this linear system using an iterative algorithm such as conjugate gradient. These iterative method are attractive for sparse matrices, but a fast convergence requires $1/\rho(\lambda)$, the conditioning of $\tilde{L} - \lambda \mathrm{Id}_n$ to be not large, with is contradictory with the constraint for iterations 1.9 to converge fast.

1.3.3 Spectral Theory on a Regular Grid

In the particular case of a 1D or 2D lattice, the eigenfunctions defined earlier correspond exactly to the Fourier basis used in the discrete Fourier transform.

Theorem 9 (Spectrum in 1D)**.** *For a 1D regular lattice,*

$$u_\omega(k) = \frac{1}{\sqrt{n}} \exp\left(\frac{2\mathrm{i}\pi}{n} k\omega\right) \qquad \textit{and} \qquad \lambda_\omega = 4 \sin^2\left(\frac{2\pi}{n}\omega\right).$$

Theorem 10 (Spectrum in 2D)**.** *For a 2D regular lattice, $n = n_1 n_2$, $\omega = (\omega_1, \omega_2)$*

$$u_\omega(k) = \frac{1}{\sqrt{n}} \exp\left(\frac{2\mathrm{i}\pi}{n} \langle k,\, \omega \rangle\right) \qquad \textit{and} \qquad \lambda_\omega = 4\left(\sin^2\left(\frac{2\pi}{n_1}\omega_1\right) + \sin^2\left(\frac{2\pi}{n_2}\omega_2\right)\right).$$

As already mentioned, on a mesh, the eigenvectors of $\tilde{L}$ correspond to a extension of the Fourier basis to meshes. The definition of the Fourier transform on meshes requires a little care since a diagonal normalization by D is used as defined next.

Definition 19 (Manifold-Fourier transform)**.** *For $f \in \ell^2(V)$,*

$$\Phi(f)(\omega) = \hat{f}(\omega) \overset{\text{def.}}{=} \langle D^{1/2} f,\, u_\omega \rangle \quad \Longleftrightarrow \quad \Phi(f) = \hat{f} = U^{\mathrm{T}} D^{1/2}.$$

where $(u_\omega)_\omega$ are the eigenvectors of $\tilde{L}$.

One can note that there is still a degree of freedom in designing this Fourier transform since one can use any local weighting (for instance combinatorial, distance or conformal). Depending on the application, one might need to use weights depending only on the topology of the mesh (combinatorial for mesh compression).

A major theoretical interest of this Fourier transform is that it diagonalizes local averaging operators.

Theorem 11 (Spectral smoothing)**.** *One has* $\Phi\tilde{W}\Phi^{-1} = \mathrm{Id}_n - \Lambda$ *and thus for any function* f

$$\widehat{\tilde{W}f}(\omega) = (1-\lambda_\omega)\hat{f}(\omega)$$

This diagonalization allows to prove the convergence of iterative smoothing.

Theorem 12 (Convergence of iterated smoothing)**.** *If* $\lambda_n < 2$ *(i.e.* M *is not 2-colorable), then for any function* f

$$\tilde{W}^k f \overset{k\to+\infty}{\longrightarrow} \frac{1}{n}\sum_{i\in V} f_i.$$

1.3.4 Spectral Resolution of the Heat Diffusion

Recall that the heat diffusion is defined as

$$\forall\, t>0, \quad \frac{\partial F_t}{\partial t} = -D^{-1}LF_t = -(\mathrm{Id}_n - \tilde{W})F_t$$

Using the manifold Fourier expansion $\hat{F}_t \overset{\text{def.}}{=} U^{\mathrm{T}} D^{1/2} F_t$, this differential equation can be re-written as

$$\frac{\partial \hat{F}_t(\omega)}{\partial t} = -\lambda_\omega \hat{F}_t(\omega) \quad \Longrightarrow \quad \hat{F}_t(\omega) = \exp(-\lambda_\omega t)\hat{f}(\omega). \tag{1.10}$$

This allows to study the convergence of the continuous heat equation.

Theorem 13 (Convergence of heat equation)**.** *If* $\mathcal{M}$ *is connected,*

$$F_t \overset{t\to+\infty}{\longrightarrow} \frac{1}{n}\sum_{i\in V} f_i.$$

Recall that the heat equation is discretized using the following explicit and implicit schemes, equations (1.6) and (1.7)

$$\begin{cases} \bar{F}_k = (1-\delta)\bar{F}_k + \delta\tilde{W}\bar{F}_k, \\ ((1+\delta)\mathrm{Id}_n - \delta\tilde{W})\tilde{F}_{k+1} = \tilde{F}_k. \end{cases}$$

These filtering iterations can be re-written over the Fourier domain as

$$\begin{cases} \widehat{\bar{F}_{k+1}}(\omega) = (1-\delta\lambda_\omega)\widehat{\bar{F}_k}(\omega), \\ \widehat{\tilde{F}_{k+1}}(\omega) = \frac{1}{(1+\delta\lambda_\omega)}\widehat{\bar{F}_k}(\omega). \end{cases}$$

This allows to state the stability and convergence of the finite difference discretization.

Theorem 14 (Convergence of discretization)**.** *The explicit scheme is stable if* $\delta < 1$*. The implicit scheme is always stable. One has*

$$\begin{cases} \bar{F}_{t/\delta} \overset{\delta\to 0}{\longrightarrow} F_t, \\ \tilde{F}_{t/\delta} \overset{\delta\to 0}{\longrightarrow} F_t. \end{cases}$$

with the restriction that for the explicit scheme, the mesh must not be 2-colorable.

Other Differential Equations. The manifold Fourier transform can also be used to solve the wave equation (1.8) since

$$\frac{\partial^2 \hat{F}_t(\omega)}{\partial t^2} = -\lambda_\omega \hat{F}_t(\omega) \quad \Longrightarrow \quad \hat{F}_t(\omega) = \cos(\sqrt{\lambda_\omega}t)\hat{f}(\omega) + \frac{1}{\sqrt{\lambda_\omega}}\sin(\sqrt{\lambda_\omega}t)\hat{g}(\omega).$$

1.3.5 Quadratic Regularization

Instead of using a PDE for regularization, one can try to find a new function that is both close to the original one f and that is smooth in a certain sense. This leads to the notion of quadratic regularization, where one uses a Laplacian as a smoothness prior on the recovered function.

Definition 20 (Quadratic regularizer)**.** *For $t > 0$, one defines*

$$F_t^q = \operatorname*{argmin}_{g \in \mathbb{R}^n} \|f - g\|^2 + t\|\tilde{G}g\|^2 \qquad \textit{where} \qquad \tilde{G} = GD^{-1/2}.$$

This optimization replaces $f \in \ell^2(V)$ by $F_t^{\mathrm{q}} \in \ell^2(V)$ with small gradients. This optimization can be found in closed form by inverting a sparse linear system.

Theorem 15 (Solution of quadratic regularization)**.** *F_t^q is unique and*

$$F_t^q = (\mathrm{Id}_n + t\tilde{L})^{-1} f.$$

Over the Fourier domain, this inversion reads

$$\hat{F}_t^{\mathrm{q}}(\omega) = \frac{1}{1 + t\lambda_\omega} \hat{f}(\omega).$$

This corresponds to an attenuation of the high frequency content of f, in a way very similar to equation (1.10).

Once again, similarly to the heat equation, the spectral expression of the quadratic regularizer allows to study its convergence for large t.

Theorem 16 (Convergence of quadratic regularization)**.** *If $\mathcal{M}$ is connected,*

$$F_t^q \overset{t \to +\infty}{\longrightarrow} \frac{1}{n} \sum_{i \in V} f_i.$$

1.3.6 Application to Mesh Compression

We have shown how the Fourier basis on meshes can be used to compute in a diagonal fashion filtering, heat diffusion and quadratic regularization. This Fourier transform is however of little interest in practice, since the original filterings (or finite difference approximation of the heat equation) are usually faster to compute directly than over the Fourier domain. The Fourier transform is thus mainly of theoretical interest in these cases since it allows to prove convergence results.

Another class of applications makes use of an orthogonal expansion such as the Fourier one to perform mesh compression. This section shows how to compute a linear M-term approximation in this Fourier basis and to do mesh compression. We refer to the survey [1] for more advanced non-linear mesh compression methods.

The orthogonal basis $U = (u_\omega)_\omega$ of $\ell^2(V) \simeq \mathbb{R}^n$, where $\tilde{L} = U\Lambda U^{\mathrm{T}}$ allows to define a linear approximation as followed.

Definition 21 (Linear M-term approximation)**.** *For any $M > 0$, the linear M-term approximation of f is*

$$f = \sum_{\omega=1}^{n} \langle f, u_\omega \rangle u_\omega \quad \overset{M\text{-term approx.}}{\Longrightarrow} \quad f_M \overset{\text{def.}}{=} \sum_{\omega=1}^{M} \langle f, u_\omega \rangle u_\omega.$$

The quality of the approximation is measured using the error decay, which can in turn be estimated using the removed coefficients

$$E(M) \overset{\text{def.}}{=} \|f - f_M\|^2 = \sum_{\omega > M} |\langle f, u_\omega \rangle|^2.$$

A good orthogonal basis U is a basis for which $E(M)$ decays fast on the signals of interest. Equivalently, a fast decay of E with M corresponds to a fast decay of $|\langle f,\, u_\omega\rangle|$ for large ω. Figure 1.8 shows the decay of the Fourier spectrum for two different functions defined on a 3D mesh. The smooth function (left in the figure) exhibits a fast decay of its spectrum, meaning that it can be well approximated with only a few Fourier coefficients.

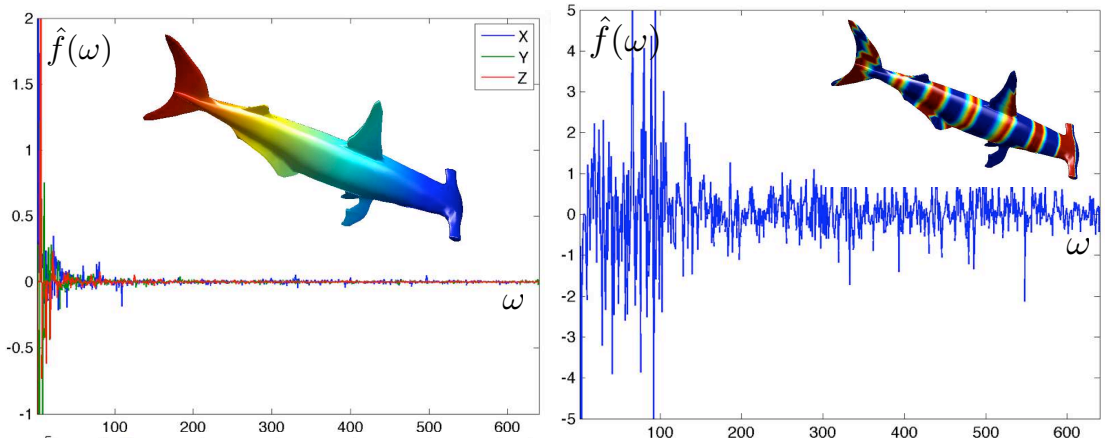


Figure 1.8: *Examples of Fourier spectrum for a smooth and a non-smooth function.*

We recall that the Fourier atoms

$$\forall\, \omega \in \mathbb{Z}, \quad u_\omega(x) = \frac{1}{\sqrt{2\pi}} e^{\mathrm{i}\omega x}$$

are the eigenvectors of the compact, symmetric, semi-definite negative operator $f \mapsto f''$ (that should be defined on the Hilbert space of twice Sobolev derivable functions). This set of function is also an Hilbert basis of the space $\mathrm{L}^2(\mathbb{R}/(2\pi\mathbb{Z}))$ of 2π-periodic square integrable functions and a Fourier coefficient is $\hat{f}(\omega) \overset{\text{def.}}{=} \langle f,\, u_\omega\rangle$.

Approximation theory studies this linear error decay for classical functional spaces. One can for instance study the Fourier expansion over euclidean spaces.

Theorem 17 (Fourier in 1D)**.** *If f is C^α regular on $\mathbb{R}/(2\pi\mathbb{Z})$,*

$$|\hat{f}(\omega)| \leqslant \|f^{(\alpha)}\|_\infty |\omega|^{-\alpha}.$$

This result can be proven with a simple integration by parts. A slightly more difficult result shows that the linear approximation error decays like $M^{-\alpha}$.

Theorem 18 (Fourier approximation)**.** *If f is C^α on $\mathbb{R}/(2\pi\mathbb{Z})$, then it exist $C > 0$ such that*

$$\sum_\omega |\omega|^{2\alpha} |\langle f,\, u_\omega\rangle|^2 < +\infty \quad \Longrightarrow \quad E(M) \leqslant C M^{-\alpha}.$$

This kind of results can be extended to continuous surfaces thanks to the continuous Laplacian. We suppose that $\mathcal{M}$ is a surface parameterized by φ, and a function $f = \varphi \circ \bar{f}$ is defined on it. By definition, this function f is C^α if $\bar{f}$ is C^α in euclidean space. For a compact surface $\mathcal{M}$, the Laplace-Beltrami operator $\Delta_\mathcal{M}$ is symmetric (for the inner product on the surface), is negative semi-definite and has a discrete spectrum $\Delta_\mathcal{M} u_\omega = -\lambda_\omega u_\omega$ for $\omega \in \mathbb{N}$. The functions $\{u_\omega\}_\omega$ are an orthogonal basis for function of finite energy on the surface $\mathrm{L}^2(\mathcal{M})$. The inner product of an arbitrary smooth function $f \in \mathrm{C}^\alpha(\mathcal{M})$ can be bounded using integration by parts

$$\langle f,\, u_\omega\rangle = \frac{1}{\lambda_\omega^k} \langle \Delta_\mathcal{M}^k f,\, u_\omega\rangle \quad \Longrightarrow \quad |\langle f,\, u_\omega\rangle| \leqslant \frac{\|f\|_{\mathrm{C}^\alpha}}{\lambda_\omega^{\alpha/2}}.$$

This proves the efficiency of the Fourier basis on surfaces to approximate smooth functions.

When computing the M-term approximation f_M of f one removes the small amplitude Fourier coefficients of the orthogonal expansion of f. Figure 1.9 shows some examples of mesh approximation where one retains an increasing number of Fourier coefficients. Mesh compression is only a step further, since one also need to code the remaining coefficients. This requires first quantifying the coefficients up to some finite precision and then binary code these coefficients into a file.

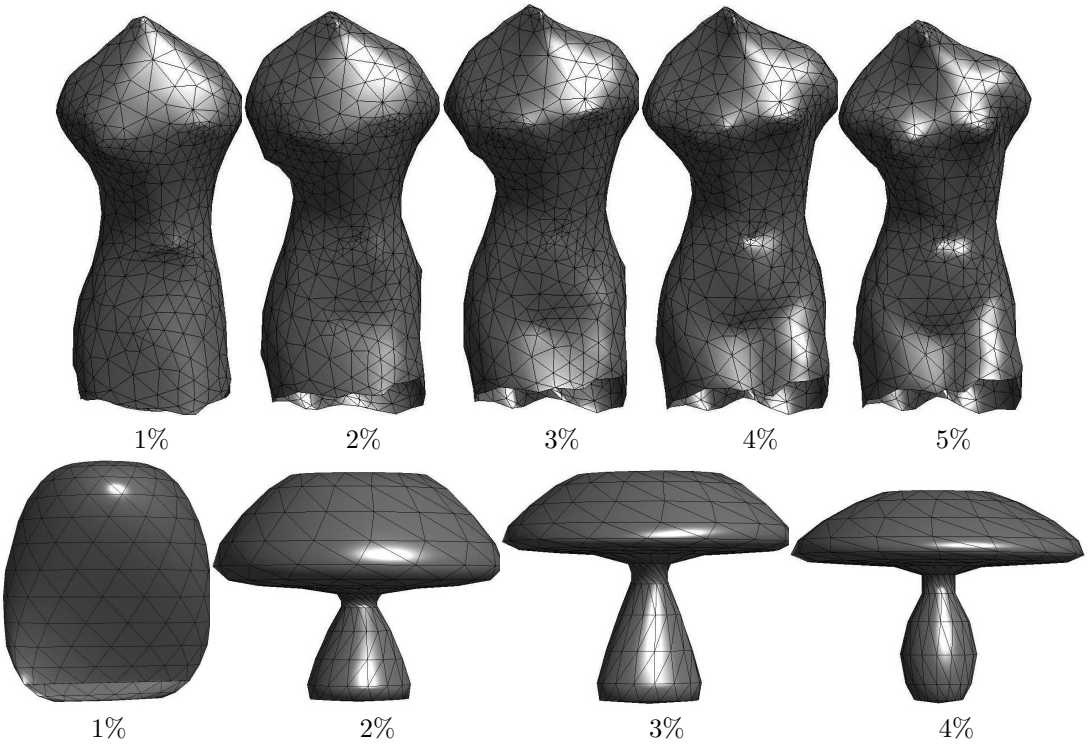


Figure 1.9: *Examples of spectral mesh compression.*

1.3.7 Application to Mesh Parameterization

This section is restricted to the study of meshes that can be globally parameterized on a plane. It means that they are topologically equivalent to a 2D disk. More complex meshes should be first segmented in cells that are equivalent to a disk.

A parameterization of a continuous surface $\mathcal{M}$ is a bijection

$$\psi : \mathcal{M} \longrightarrow \mathcal{D} \subset \mathbb{R}^2.$$

A similar definition applies to a discrete mesh where one computes a 2D position $\psi(i)$ for all the vertices $i \in V$ and then interpolates linearly the mapping to the whole piecewise linear geometric mesh. This section explains the basics of linear methods for mesh parameterization. We refer to various surveys [12, 35] for more details on mesh parameterization.

Usually, a 2D mesh is computed from range scanning or artistic modeling, so it does not come with such a parameterization. In order to perform texture mapping or more general mesh deformations, it is however important to use such a parameterization. Since many bijections are possible to layout the mesh in 2D, the mapping ψ has to satisfy additional smoothness assumptions. Classically, one requires that each coordinate of ψ has a vanishing Laplacian (it is thus harmonic) outside a set of constrained vertices that enforce boundary conditions.

More precisely, $\psi = (\psi_1, \psi_2)$ is the solution of

$$\begin{cases} \forall\, i \notin \partial\mathcal{M}, & (L\psi_1)(i) = (L\psi_2)(i) = 0 \\ \forall\, i \in \partial\mathcal{M}, & \psi(i) = \psi^0(i) \in \partial\mathcal{D}, \end{cases}$$

where $\partial\mathcal{M}$ is the boundary of the mesh, which consists in vertices whose face ring is not homeomorphic to a disk but rather to a half disk. This formulation requires the solution of two sparse linear systems (one for each coordinate of ψ).

The boundary condition $\psi^0(i)$ for $i \in \partial\mathcal{M}$ describes a 1D piecewise linear curve in the plane, that is fixed by the user. In the following, we will see that this curve should be convex for the parameterization to be bijective.

Remark 5. For such an harmonic parameterization, each point is the average of its neighbors since

$$\forall\, i, \quad \psi(i) = \frac{1}{\sum_j w_{ij}} \sum_{(i,j)\in E} w_{i,j}\psi(j).$$

The powerful feature of this linear parameterization method is that it can be proven to produce a valid (bijective) parameterization as long as the constrained position (boundary values of ψ) are along a convex curve.

Theorem 19 (Tutte theorem)**.** *If $\forall\,(i,j) \in E,\ w_{ij} > 0$, and if $\partial\mathcal{D}$ is a convex curve, then ψ is a bijection.*

Figure 1.10 shows several examples of parameterizations. One is free to use any laplacian (combinatorial, distance or conformal) as long as it produces positive weights. There is a issue with the conformal weights, which can be negative if the mesh contains obtuse triangles. In practice however it leads to the best results. The efficiency of a parameterization can be measured by some amount of distortion induced by the planar mapping. Linear methods cannot hope to cope with large isoperimetric distortions (for instance large extrusions in the mesh) since harmonicity leads to clustering of vertices.

1.3.8 Application to Mesh Flattening

One of the difficulty with linear parameterization methods is that they require to set up the positions of the vertices along the boundary of the mesh. In order to let the boundary free to evolve and find some optimal shape, one can replace the fixed point constraint by a global constraints of unit variance as follow

$$\min_{\psi_1,\psi_2\in\mathbb{R}^n} \|\tilde{G}\psi_1\|^2 + \|\tilde{G}\psi_2\|^2 \qquad \text{with} \qquad \begin{cases} \|\psi_i\| = 1, \\ \langle \psi_1,\, \psi_2\rangle = 0, \\ \langle \psi_i,\, 1\rangle = 0. \end{cases}$$

This optimization problem also has a simple global solution using eigenvectors of the Laplacian.

Theorem 20 (Mesh flattening solution)**.** *The mesh flattening solution is given by*

$$\mathit{Span}(\psi_1, \psi 2) = \mathit{Span}(u_1, u_2) \quad \textit{where} \quad \tilde{L} = U\Lambda U^{\mathrm{T}}.$$

In order to compute this flattening, one thus needs to extract 2 eigenvectors from a sparse matrix. Note however that, in contrast to linear parameterization schemes, this flattening is not ensured to be bijective. Figure 1.11 shows that for meshes with large distortion, this flattening indeed leads to wrong parameterizations.

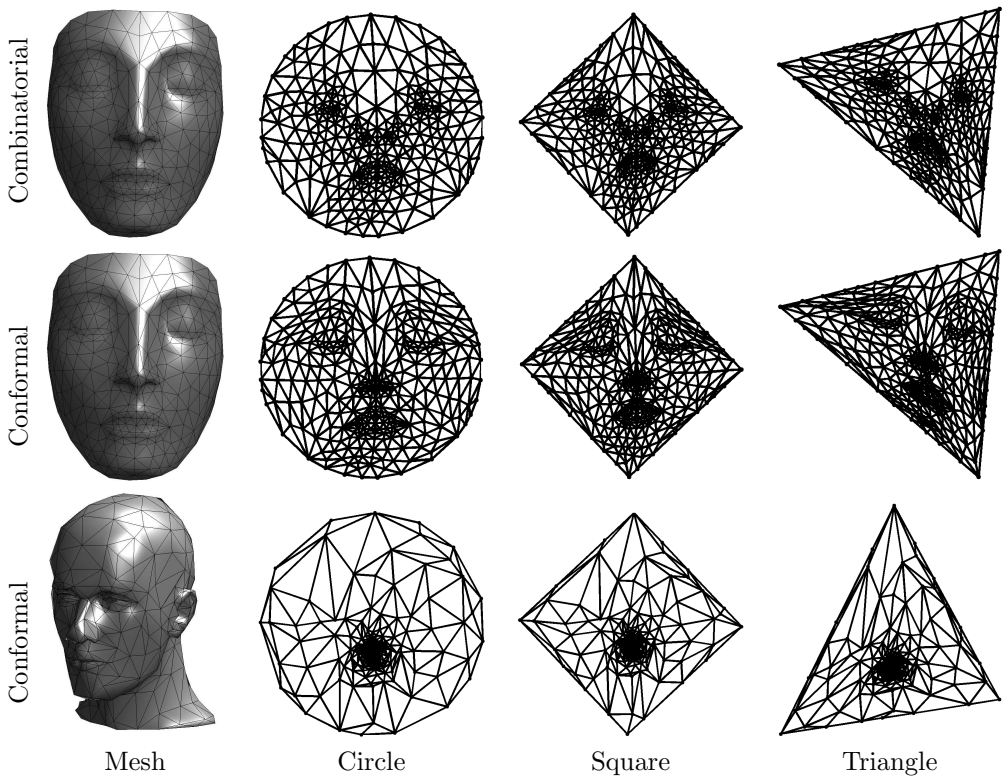


Figure 1.10: *Examples of mesh parameterizations.*

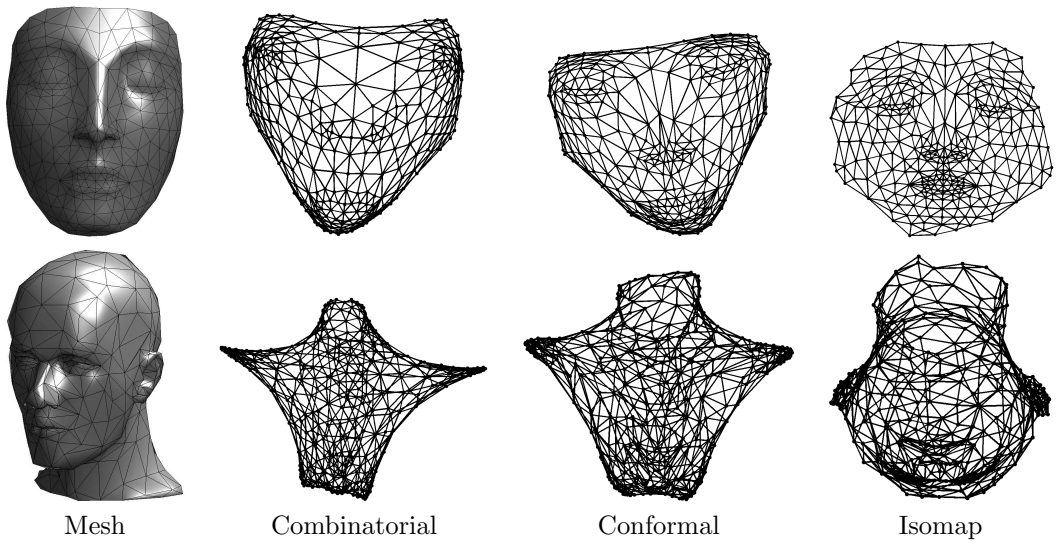


Figure 1.11: *Examples of mesh flattening.*

2 Geodesic Mesh Processing

This chapter studies the numerical computation of geodesic distance on Riemannian manifolds. It shows both the algorithms and applications to surface processing, in particular sampling, meshing and comparison of surfaces.

The original book on Fast Marching methods is [34]. For other applications to computer graphics and image processing one can see [26] and [17]. The recent book [3] treats all the details of the geometry of non-rigid surfaces, including geodesic distance computation and shape comparison. One can also see the two books [27, 28] that contain review articles with some applications of Fast Marching and geodesic methods.

2.1 Manifold Geometry of Surfaces

2.1.1 Riemannian Manifold

We have seen in definition 1 that a parameterized surface is embedded into some Euclidean domain $\mathbb{R}^k$, which allows to define a local metric thanks to the first fundamental form I_φ. It is however possible to consider directly a field of positive definite tensors on a parametric domain $\mathcal{D} = \mathbb{R}^s$ (in practice here $s = 2$ for surfaces or $s = 3$ for volumes). With a slight abuse in notations, we will even assimilate the resulting abstract surface $\mathcal{M}$ with $\mathcal{D}$. Once again, we consider only surfaces globally mapped some Euclidean domain, but handling generic surfaces requires to split the manifold into overlapping charts.

Definition 22 (Riemannian manifold)**.** *A Riemannian manifold is an abstract parametric space $\mathcal{M} \subset \mathbb{R}^s$ equipped with a metric $x \in \mathcal{M} \mapsto H(x) \in \mathbb{R}^{s\times s}$ positive definite.*

Using the Riemannian metric, one can compute the length of a curve $(\gamma(t))_{t=0}^T \in \mathcal{M}$

$$L(\gamma) \overset{\text{def.}}{=} \int_0^T \sqrt{\gamma'(t)^{\mathrm{T}} H(\gamma(t))\gamma'(t)}\mathrm{d}t.$$

This length is invariant under a reparameterization of the curve, which means that for any strictly increasing $\psi : [0, T'] \to [0, T]$, one has

$$L(\gamma) = L(\gamma \circ \psi).$$

This notion of curve length is well defined for smooth curves γ, but it can be extended to piecewise smooth curves by splitting the integral into each part where the curve is smooth.

At each location x, the Riemannian tensor can be diagonalized as follow

$$H(x) = \lambda_1(x)e_1(x)e_2(x)^{\mathrm{T}} + \lambda_2(x)e_2(x)e_2(x)^{\mathrm{T}} \quad \text{with} \quad \lambda_1 \leqslant \lambda_2,$$

and e_1, e_2 are two orthogonal eigenvector fields. In fact, e_i should be understood as direction (unoriented) field since both e_i and $-e_i$ are eigenvectors of the tensor. A curve γ passing at location $\gamma(t) = x$ with speed $\gamma'(t)$ has a shorter local length if $\gamma'(t)$ is collinear to $e_1(x)$ rather to an other direction. Hence shortest paths (to be defined in the next section) tends to be tangent to the direction field e_1.

In practice, the Riemannian metric H is given by the problem one wishes to solve. In image processing, the manifold is the image plane $\mathcal{M} = [0, 1]^2$ equipped with a metric derived from the image (for instance its gradient).

Example 7. Figure 2.1 shows some some frequently used geodesic metric spaces:

- *Euclidean space:* $\mathcal{M} = \mathbb{R}^s$ and $H(x) = \mathrm{Id}_s$.
- *2D shape:* $\mathcal{M} \subset \mathbb{R}^2$ and $H(x) = \mathrm{Id}_2$.
- *Parametric surface:* $H(x) = I_\varphi(x)$ is the first fundamental form.
- *Isotropic metric:* $H(x) = W(x)\mathrm{Id}_s$, $W(x) > 0$ being some weight function.
- *Image processing:* given an image $I : [0,1]^2 \to \mathbb{R}$, one can use an edge-stopping weight $W(x) = (\varepsilon + \|\nabla_x I\|)^{-1}$. This way, geodesic curves can be used to perform segmentation since they will not cross boundaries of the objects.
- *fMRI imaging:* $\mathcal{M} = [0,1]^3$, and $H(x)$ is a field of diffusion tensors acquired during a scanning experiment. For fMRI imaging, the direction field e_1 indicates the direction of elongated fibers of the white matter.

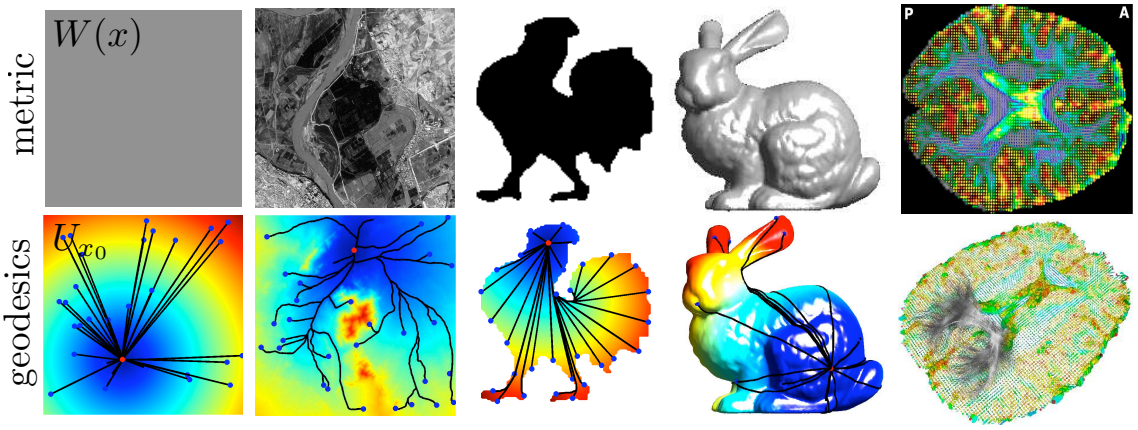


Figure 2.1: *Examples of metric (top row) and geodesics (bottom row). From left to right: euclidean, isotropic, shape, surface and 3D Riemannian manifold metrics (diffusion tensor). The blue/red colormap indicates the geodesic distance to the starting point.*

2.1.2 Geodesic Distances

The local Riemannian metric $H(x)$ allows to define a global metric on the space $\mathcal{M}$ using shortest paths. This corresponds to the notion of geodesic curves.

Definition 23 (Geodesic distance)**.** *Given some Riemannian space $(\mathcal{M}, H)$ with $\mathcal{M} \subset \mathbb{R}^s$, the geodesic metric is defined as*

$$\forall\,(x,y) \in \mathcal{M}^2, \quad d_{\mathcal{M}}(x,y) \overset{\text{def.}}{=} \min_{T>0, \gamma \in \mathcal{P}_T(x,y)} L(\gamma)$$

where $\mathcal{P}_T(x,y)$ denotes the set of piecewise smooth curves joining x and y

$$\mathcal{P}_T(x,y) \overset{\text{def.}}{=} \{\gamma \setminus \gamma(0) = x \quad \textit{and} \quad \gamma(T) = y\}\,.$$

The shortest path between two points according to the Riemannian metric is called a geodesic. If the metric H is well chosen, then geodesic curves can be used to follow salient features on images and surfaces.

Definition 24 (Geodesic curve)**.** *A geodesic curve $\gamma(t) \in \mathcal{P}_T(x,y)$ is such such that $L(\gamma) = d_{\mathcal{M}}(x,y)$.*

A geodesic curve between two points might not be unique, think for instance about two anti-podal points on a sphere. A complete surface is a surface for which for any pair of points $(x,y) \in \mathcal{M}^2$, it

exists a geodesic in $\mathcal{P}_T(x,y)$. In this course we consider only compact connected surfaces that are complete. For instance the space $\mathcal{M} = [0,1]^2 - \{(1/2,1/2)\}$, which is not closed, is not a complete manifold for the euclidean metric since there is no shortest path in $\mathcal{M}$ between $x = (0,0)$ and $y = (1,1)$.

In order to perform the numerical computation of geodesic distances, we fix a starting point x_0 and consider only distance and geodesic curves from that point.

Definition 25 (Distance map)**.** *The distance map to a starting point $x_0 \in \mathcal{M}$ is defined as $U_{x_0}(x) \overset{\text{def.}}{=} d_{\mathcal{M}}(x_0, x)$.*

The main theorem that characterizes the geodesic distance is the following, that replaces the optimization problem of finding the minimum distance by a non-linear partial differential equation.

Theorem 21 (Eikonal equation)**.** *If the metric H is continuous, then for any $x_0 \in \mathcal{M}$, the map U_{x_0} is the unique viscosity solution of the Hamilton-Jacobi equation*

$$\|\nabla_x U_{x_0}\|_{H(x)^{-1}} = 1 \qquad \textit{with} \qquad U_{x_0}(x_0) = 0, \tag{2.1}$$

where $\|v\|_A = \sqrt{v^{\mathrm{T}} A v}$.

It is important to notice that, even if the metric $x \mapsto H(x)$ is a smooth function, the distance function U_{x_0} might not be continuous. This is why the machinery of viscosity solution is needed to give a sense to the solution of the Hamilton-Jacobi equation. See for instance [7] for an introduction to viscosity solutions and a proof of theorem 21.

It is possible to define the distance map $U_S(x)$ to a compact set of points $S \subset \mathcal{M}$ as the shortest path between x and any point in S

$$d_{\mathcal{M}}(S,x) \overset{\text{def.}}{=} U_S(x) \overset{\text{def.}}{=} \min_{y \in S} d_{\mathcal{M}}(x,y).$$

The function U_S also satisfy the Eikonal equation (2.1), but with the boundary condition $U_S(x) = 0$ for $x \in S$. In this course, we restrict ourself to initial conditions that are either a single point $S = \{x_0\}$ or a finite number of such points $S = \{x_i\}_{i=1}^m$.

Once the distance map U_{x_0} has been computed by solving the Eikonal equation (2.1), one can extract any geodesic joining x_0 using a gradient descent on the function U_{x_0}.

Theorem 22 (Gradient descent)**.** *The geodesic curve γ between x_1 and x_0 solves*

$$\gamma'(t) = -\frac{H(\gamma(t))^{-1} \nabla_{\gamma(t)} U_{x_0}}{\|H(\gamma(t))^{-1} \nabla_{\gamma(t)} U_{x_0}\|} \qquad \textit{with} \qquad \gamma(0) = x_1.$$

The geodesic curve γ extracted using this gradient descent is parameterized with unit speed since $\|\gamma'\| = 1$.

Example 8. For an isotropic metric $H(x) = W(x)\mathrm{Id}_x$, the Eikonal equation and the geodesic extraction are

$$\|\nabla_x U_{x_0}\| = W(x) \qquad \text{and} \qquad \gamma'(t) = -\frac{\nabla_x U_{x_0}}{\|\nabla_x U_{x_0}\|}.$$

2.1.3 Mapping Between Surfaces

The notion of a parametric surface $(\mathcal{M}, \varphi)$ allows to compare the surface $\mathcal{M}$ with its parametric domain thanks to the first fundamental form. For instance, a surface is isometric to euclidean space if $I_\varphi = \mathrm{Id}_s$ for some parameterization φ.

However, in most situations, one wants to compare two different surfaces, maybe embedded in different euclidean spaces and of course having different parameterizations with the same parametric domain. Given two parameterized surfaces $(\mathcal{M}_1, \varphi_1)$ and $(\mathcal{M}_2, \varphi_2)$, a matching is a bijective function $f : \mathcal{M}_1 \to \mathcal{M}_2$. Such a matching allows to define an induced parameterization of $\mathcal{M}_2$: $\tilde{\varphi}_2 = f \circ \varphi_1$. This leads to the notion of isometry between surfaces.

Definition 26 (Isometry)**.** *$\mathcal{M}_1$ is isometric to $\mathcal{M}_2$ if it exists a matching f such that*

$$\forall\, u,\ I_{\varphi_2}(u) = I_{\tilde{\varphi}_2}(u).$$

This definition is local and requires only the comparison of the first fundamental forms. The difficulty is that it requires the use of arbitrary parameterizations φ_1 and φ_2, although the notion of isometry is intrinsic to the surface. The following theorem states that isometry can in fact be checked in a global and intrinsic fashion by looking at geodesic lengths.

Theorem 23. *f is isometric if and only if*

$$\forall\, (x, y) \in \mathcal{M}^2,\ d_{\mathcal{M}_1}(x, y) = d_{\mathcal{M}_2}(f(x), f(y)).$$

This theorem thus gives a new definition of isometry that can be used in numerical applications to check if a mapping f is isometric, or even to optimize an unknown isometric mapping f between two surfaces.

One can also relax the notion of isometry and ask only for a conservation of angles. This leads to the notion of conformality between surfaces, which generalizes conformal parameterizations.

Definition 27 (Conformality)**.** *$\mathcal{M}_1$ conformally equivalent to $\mathcal{M}_2$ if it exists f such that*

$$\forall\, u,\ I_{\varphi_2}(u) = \lambda(u) I_{\tilde{\varphi}_2}(u) \quad \textit{with} \quad \lambda(u) \in \mathbb{R}.$$

2.2 Numerical Computations of Geodesic Distances

In order to make all the previous definitions effective in practical situation, one needs a fast algorithm to compute the geodesic distance map U_{x_0}. This section explains algorithms based on front propagation that enables to compute the distance map by propagating the distance information from the starting point x_0.

2.2.1 Front Propagation Algorithms

Depending on the properties of the metric, one needs to consider several algorithms, that all rely on the idea of front propagation. This family of algorithms allows to order to computations in such a way that each point of the discretization grid is visited only once. This ordering is feasible for distance computation because the distance value of a grid point only depends (and can be computed) from a small number of points having only smaller distances. If one can order the grid points with increasing distance, then one gets a coherent ordering of the computations. Of course, this is not that easy since this distance ordering would require the knowledge of the solution of the problem (the distance itself). But depending on the application, it is possible to devise a selection rule that actually select at each step the correct grid point.

A front propagation labels the points of the grid according to a state

$$S(x) \in \{Computed, Front, Far\}.$$

During the iterations of the algorithm, a point can change of label according to

$$Far \mapsto Front \mapsto Computed.$$

Computed points $S(x) = Computed$ are those that the algorithm will not consider any more (the computation of $U_{x_0}(x)$ is done for these points). Front points $S(x) = Front$ are the points that are being processed (the value of $U(x) \approx U_{x_0}(x)$ is well defined but might change in future iterations). Far points $S(x) = Far$ are points that have not been yet processed.

In practice, a front propagation algorithm requires three key ingredients:

- Given a point x in the grid, a local set of neighbors $Neigh(x)$ connected to x.
- A priority $\mathcal{P}(x)$ among points x in the front, that allows to select the point to process at a given iteration. In most application, this priority is computed as the current value of the distance $\mathcal{P}(x) \overset{\text{def.}}{=} U(x)$. Section 2.3.2 shows how to change this priority in order to speed up computations.
- A procedure $x \mapsto Update(x) \in \mathbb{R}$ that computes the distance value approximating $U_{x_0}(x)$ knowing the value $U_{x_0}(x)$ for computed point and an approximate value for points in the front. This procedure usually solves some kind of equation that discretizes the Eikonal equation (2.1) one wishes to solve.

Listing 1 gives the details of the front propagation algorithm that computes a distance map U approximating $U_{x_0}(x)$ on a discrete grid. The following section details for actual implementations of the $Update$ procedure for different metrics.

The numerical complexity of this scheme is $O(n \log(n))$ for a discrete set of n points. This is because all the points are visited (tagged *Computed*) once, and the selection of $\min \mathcal{P}$ from the front points takes at most $\log(n)$ operations with a special heap data structure (although in practice it takes much less and the algorithm is nearly linear in time).

1. *Initialization:* $U(x_0) \leftarrow 0$, $S(x_0) \leftarrow Front$, $\forall\, y \neq x_0,\ S(y) \leftarrow Far$.
2. *Select point:* $x \longleftarrow \underset{S(z)=Trial}{\operatorname{argmin}}\ \mathcal{P}(z)$.
3. *Tag:* $S(x) \leftarrow Computed$.
4. *Update neighbors:* for all $y \in Neigh(x)$,
   - If $S(y) = Far$, then $S(y) \leftarrow Front$ and $U(y) \leftarrow Update(y)$.
   - If $S(y) = Front$, then $U(y) \leftarrow \min(U(y), Update(y))$.
   - Recompute the priority $\mathcal{P}(y)$.
5. *Stop:* If $x \neq x_1$, go back to 2.

Table 1: Front propagation algorithm.

2.2.2 Geodesic Computation on a Graph

A graph is a discrete data structure that links together points with edges. This is a crude discretization of many problems (such as those involving surfaces). In particular it is poorer than the mesh data structure introduced in section 1.1.2 since it lacks two dimensional connectivity information given by the faces of the mesh. However, in many applications including high dimensional data processing, it is the only way to capture the local connectivity information.

Definition 28 (Graph)**.** *A graph is a data structure $\mathcal{G} = (V, E)$ where $V = \{1, \ldots, n\}$ and $E \subset V \times V$ is symmetric.*

Similarly to the notion of local averaging weights introduced in section 1.2.2, a metric on a graph is a weight defined on each edge.

Definition 29 (Graph metric)**.** *A metric H on a graph is given by $\forall\, (i, j) \in E,\ H(i, j) > 0$.*

It is important to note that the metric $H(i, j)$ is usually very different from the local averaging weights $W(i, j)$ introduced in section 1.2.2. In general, $H(i, j)$ is small to represent the fact that i and j share similarities, and in this case $W(i, j)$ should be large (to produce strong averaging between values at i and j).

Example 9 (Geometric realization). In practice, a graph often comes with some geometric realization, which is a set of positions $\mathcal{V} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^k$. In this case, the metric can be defined as

$$H(i, j) \overset{\text{def.}}{=} \|x_i - x_j\|.$$

If the set of points is sampled from some continuous surface, this edge metric is a crude discretization of the metric defined by the embedding of the surface in euclidean space. One can note that in this case, if one uses the local averaging weights W based on distance, one has $W(i,j) = H(i,j)^{-2}$.

Definition 30 (Path)**.** *A path π is $\pi = (i_1, \ldots, i_m) \subset V^m$ with $(i_k, i_{k+1}) \in E$. The starting point is $\mathcal{S}(\pi) = i_1$ and the ending point is $\mathcal{E}(\pi) = i_m$.*

It is important to note that a path π is a topological structure. In order to compute its length, one just needs to use the metric.

Definition 31 (Length)**.** *The length of a path $\pi \subset V^m$ is*

$$L(\pi) \stackrel{\text{def.}}{=} \sum_{i=1}^{n} W(v_i, v_{i+1}).$$

This path length allows to define a geodesic metric on vertices of the graph.

Definition 32 (Geodesic distance on a graph)**.** *The geodesic distance between $(x,y) \in V^2$ is defined as*

$$d_{\mathcal{G}}(x,y) = \min_{\pi} \ \{L(\pi) \setminus \mathcal{S}(\pi) = x, \mathcal{E}(\pi) = y\}$$

In order to compute the geodesic map $U_{x_0}(x) = d_{\mathcal{G}}(x_0, x)$, one can use the front propagation algorithm, listing 1 with the following update procedure

$$Update(x) = \min_{(x,y)\in E} \ \big(U(y) + H(x,y)\big).$$

This corresponds to the Dijkstra propagation that is classical in graph theory, see for instance [6].

2.2.3 Geodesic Computation on a Square Grid

The classical Fast Marching algorithm, introduced by Sethian [34], is a fast procedure to solve the Eikonal equation (2.1) for an isotropic metric $H(x) = W(x)\mathrm{Id}_k$ for a uniform regular grid that discretizes $[0,1]^k$. We recall this procedure for a planar domain $k = 2$ although it can be extended to any dimension.

In order to capture the viscosity solution of an Hamilton Jabobi equation, one cannot use standard finite differences because of the apparition of chocks and singularities in the solution of the equation. One needs to choose, at each grid point, the optimal finite difference scheme (differentiation on the left or on the right to approximate d/dx for instance). This optimal differentiation should be chosen in the direction where the solution of the equation decreases. This is called an upwind finite difference scheme, and on a 2D grid it leads to find $u = Update(x)$ at a grid point $x = x_{i,j}$ that is the smallest solution of

$$\begin{aligned} &\max(u - U(x_{i-1,j}),\, u - U(x_{i+1,j}), 0)^2 + \\ &\max(u - U(x_{i,j-1}),\, u - U(x_{i,j+1}), 0)^2 = h^2 W(x_{i,j})^2. \end{aligned}$$

Sethian shown that the smallest solution of this equation leads to a stable and convergent scheme that can be safely used in the front propagation algorithm listing 1.

Figure 2.2 shows some examples of front propagation with the Fast Marching. The colored area shows, at some given step of the algorithm, the set of computed points (its boundary being the set of front points). During the iterations, the front propagates outwards until all the grid points are visited. The numerical complexity of this scheme is $O(n \log(n))$ for a grid of n points.

Figure 2.3 shows examples of distance functions to a starting point x_0 with the corresponding geodesics $\gamma(t)$ extracted from some ending point x_1. The front propagation is stopped when

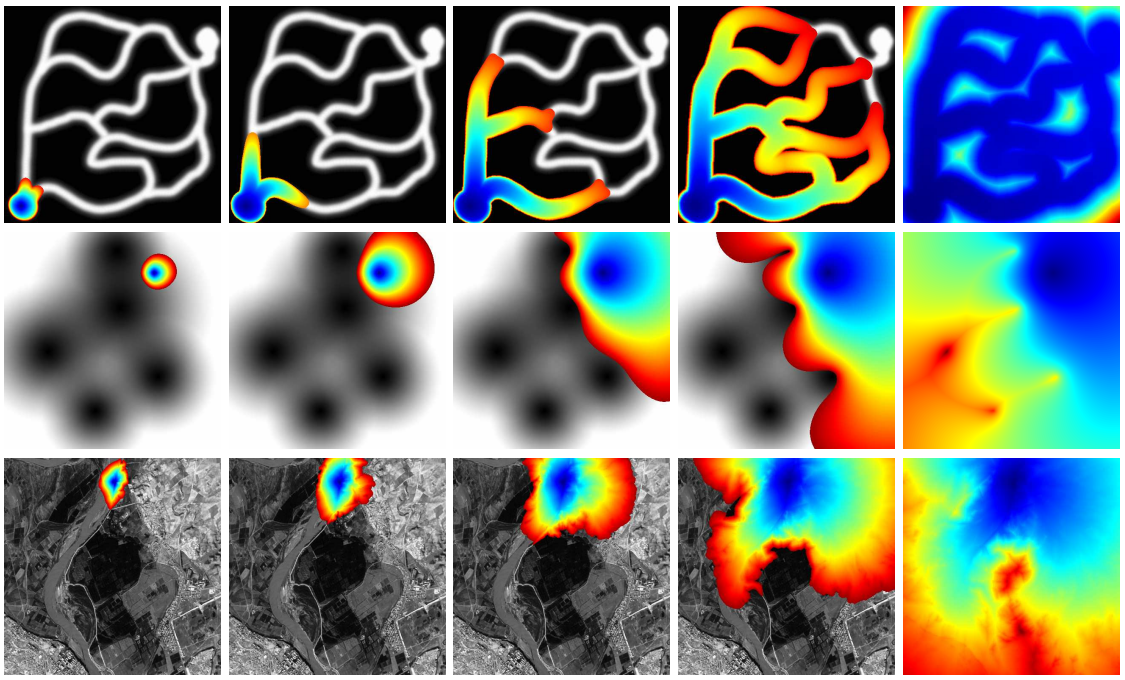


Figure 2.2: *Examples of front propagation. The colormap indicates the values of the distance functions at a given iteration of the algorithm.*

$S(x_1) = Computed$ to avoid performing useless computations. The idea of using geodesics in order to extract salient curves in images as been introduced in [18].

In practice, the difficult task is to design a metric W in order have meaningful geodesics. Here are some examples of possible choices, for image processing with an input image f:

- *Pixel value based potential:* in many application, one simply wishes to extract curves with a constant value c. In this case, one can use a potential like

$$W(x) = \frac{1}{\varepsilon + |f(x) - c|}.$$

  Figure 2.3, left and middle, shows examples of such curves extractions.
- *Gradient avoiding potential:* for edge detection, one wants geodesics that avoid traversing image edges. In this case, the potential should be small in area of large gradient in the image

$$W(x) = G_{\sigma_2} * \frac{1}{\varepsilon + \|\nabla_x f * G_{\sigma_1}\|},$$

  where G_σ are gaussian filter. One usually uses a very small σ_1 but a larger σ_2 to cope with noise.
- *Gradient attracting potential:* for road tracking in satellite images, the structures to extract are usually very thin, and one thus would only like to follow regions with elongated gradient. To avoid cancellation in the gradient, only a smoothing on the magnitude of the gradient should be performed

$$W(x) = \varepsilon + G_{\sigma_2} * \|\nabla_x f\|.$$

  Figure 2.3, right, shows an example of such a road extraction.

The Fast Marching works the same way in any spacial dimension k and in particular can be used to extract shortest paths in 3D volumetric medical data. Such a volume is a discretization of a mapping $f : [0,1]^3 \mapsto \mathbb{R}$. Figure 2.4 shows two alternative ways to explore a 3D cube of a scanned medical data. The first way is simply to visualize 2D slices that are traditional images, for instance $(f(x,y,z_0))_{x,y}$ for some fixed $z_0 \in [0,1]$. An alternative way is to perform a 3D display with a semi-transparent mapping that removes more or less parts of the data. The transparency at point (x,y,z) is defined as $\rho(f(x,y,z))$ where $\rho : [f_{\min}, f_{\max}] \to [0,1]$ is the α-mapping. A third

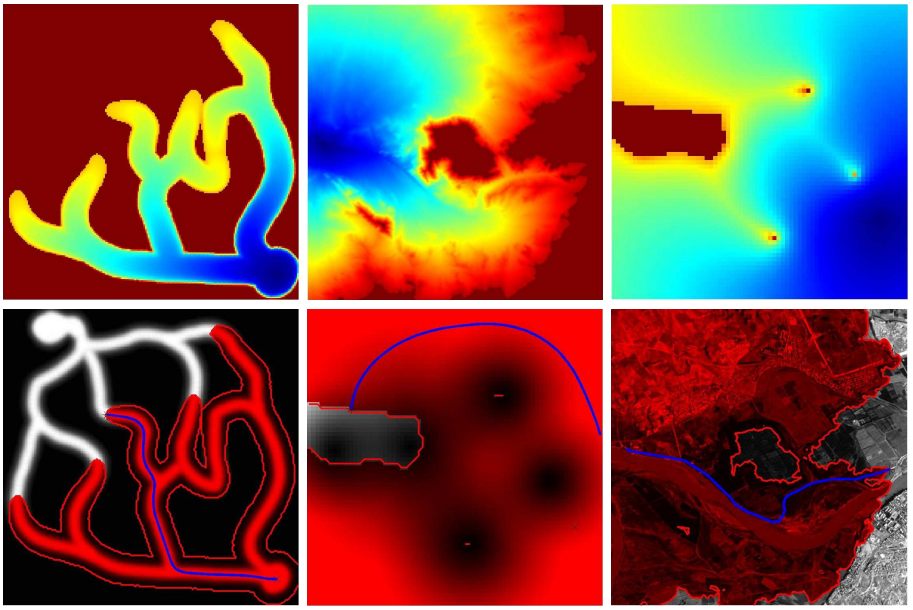


Figure 2.3: *Example of distance functions (top row) and geodesics (bottom row).*

way (applied in figure 2.5 where one can see in red the front of the Fast Marching propagation) is to display various isosurfaces which are level sets $\{(x, y, z) \setminus f(x, y, z) = c\}$ for some c.

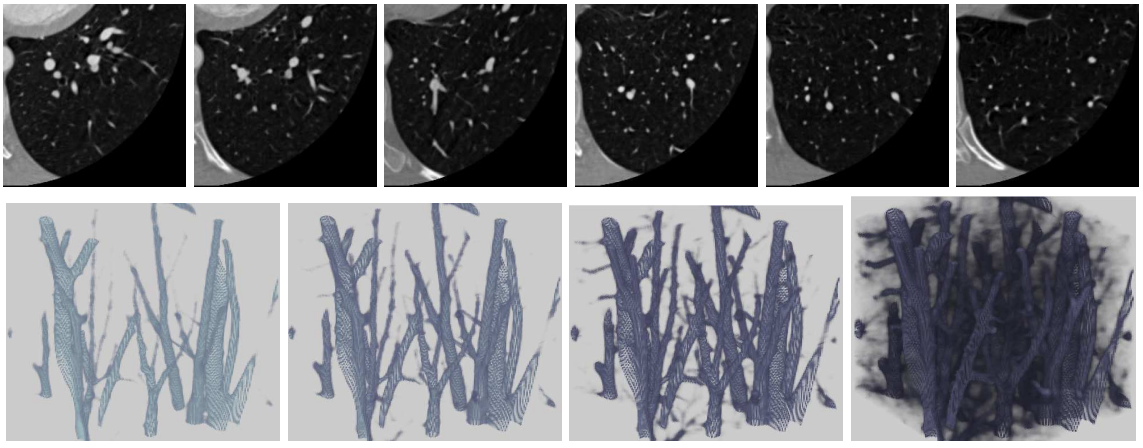


Figure 2.4: *Example of volume data visualization with slices (top row) and semi-transparent mapping (bottom row).*

Figure 2.5 shows some examples of geodesic extraction on a medical image that represents tubular structures (blood vessels) around the heart. The potential $W(x)$ is chosen as $W(x) = (|f(x) - f(x_0)| + \varepsilon)^{-1}$ where x_0 is a point given by the user and supposed to lie inside some vessel. A geodesic follows nicely a vessel since its density is constant and thus the value of f is approximately equal to $f(x_0)$ inside the vessel.

2.2.4 Geodesic computation on a triangulation

The classical Fast Marching algorithm is restricted to isotropic metrics on a regular grid. It can thus only be used to compute geodesics on surface parameterized conformaly on a 2D square. This setting is useful for image and volumetric data processing, but in order to deal with arbitrary surfaces embedded in $\mathbb{R}^k$, one needs to use algorithms designed for triangulated meshes.

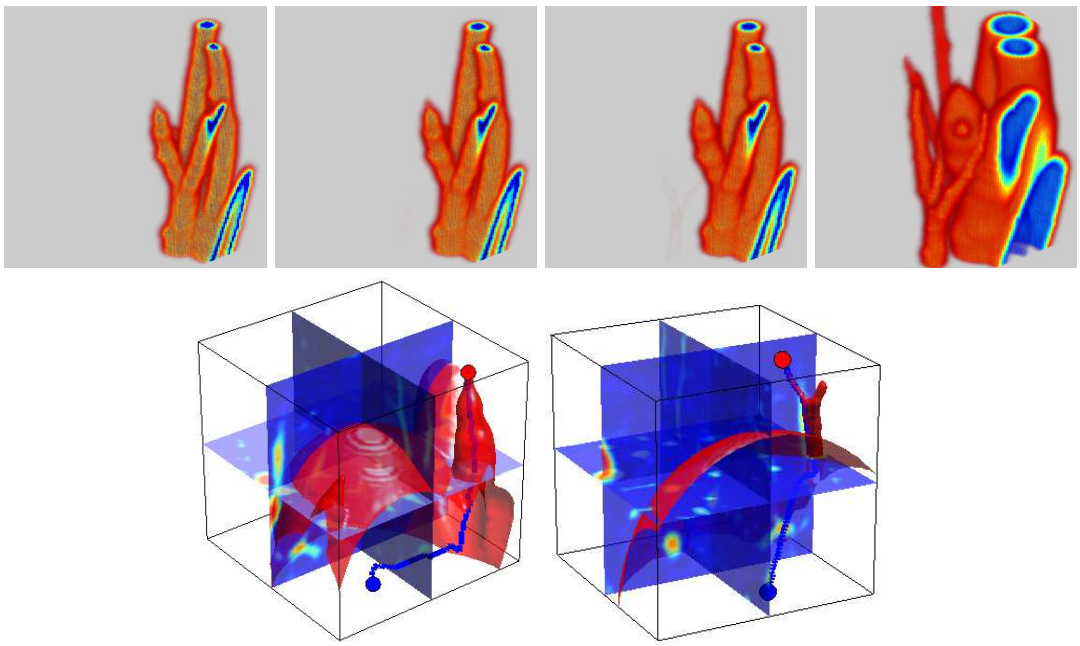


Figure 2.5: *Example of volumetric Fast Marching evolution (top row) and geodesic extractions (bottom row).*

Kimmel and Sethian [19]have developed a version of the Fast Marching algorithm for surface $\mathcal{M} \subset \mathbb{R}^n$ with metric $W(x)$ for x in embedded space $\mathbb{R}^n$. In the continuous setting, a parameteric surface $(\mathcal{M}, \varphi)$ embedded with a metric $W(x)$ in ambient space correspond to a Riemannian manifold with a metric $I_\varphi(\bar{x})W(\varphi(\bar{x}))$ in parameter space $\bar{x} \in \mathbb{R}^2$.

The algorithm of Kimmel and Sethian works on a triangulated mesh and treats the triangles of this mesh as locally flat and equipped with an isotropic metric W. In order to compute the update value at a given vertex x_0, it computes an update value $Update_f(x_0)$ for each triangle $f \in F_{x_0}$ in the face 1-ring around x_0, $F_{x_0} = \{f_1, \ldots, f_k\}$. The resulting Fast Marching update step is defined as

$$Update(x_0) = \min_{f \in F_{x_0}} Update_f(x_0).$$

In order to derive the expression for $Update_f(x_0)$, one considers a planar triangle $f = (x_0 = 0, x_1, x_2)$ and denote as $X = (x_1, x_2) \in \mathbb{R}^{2\times 2}$. In the following, we assume, without loss of generality that $x = 0$.

If only $U(x_1)$ or $U(x_2)$ is available for computation (meaning that one of the points has not been reached by the front), then the update procedure is simply

$$Update_f(x_0) = \min(U(x_0), U(x_1) + W(x_0)\|x_1\|, U(x_2) + W(x_0)\|x_2\|),$$

which corresponds to a simple Dijkstra-like update (propagation along the 1D edges of the triangle). Otherwise, the known distances are $u = (U(x_1), U(x_2))^{\mathrm{T}}$ and one wishes to solve for $Update_f(x) = p = U(x)$.

The discretization of the Eikonal equation leads to consider a linear approximation of the geodesic distance map U inside the current triangle f.

$$U(x) \approx \langle g,\, x\rangle + p \qquad \text{where} \qquad \nabla_x U \approx g.$$

One can evaluate this linear equation at x_1 and x_2, which leads to the following quadratic equation in the unknown p

$$\begin{cases} u = X^{\mathrm{T}} g + p1 \\ \|g\|^2 = W(x)^2 \end{cases} \quad \Longrightarrow \quad 1^{\mathrm{T}} Q 1 p^2 + (21^{\mathrm{T}} Q u)p + (u^{\mathrm{T}} Q u - W(x)^2) = 0,$$

where we have denoted $Q = (X^{\mathrm{T}}X)^{-1}$. This matrix Q account for the local deformation of the surface (which corresponds to the anisotropy of the metric). The only admissible solution to this problem is

$$p = \frac{1Qu + \sqrt{(1^{\mathrm{T}}Qd)^2 + 1^{\mathrm{T}}Q1(u^{\mathrm{T}}Qu - W(x)^2)}}{1^{\mathrm{T}}Q1}$$

Some care should be taken to compute the value of the update on triangle f so that the computed value is larger than the available distances at x_1 and x_2

$$Update_f(x) = \begin{cases} p \quad \text{if} \quad p \geqslant \max(U(x_1), U(x_2)), \text{ otherwise:} \\ \min(U(x_0), U(x_1) + W(x_0)\|x_1\|, U(x_2) + W(x_0)\|x_2\|). \end{cases}$$

There is some technical difficulties with this scheme on triangulation that contains obtuse angles that we shall ignore here.

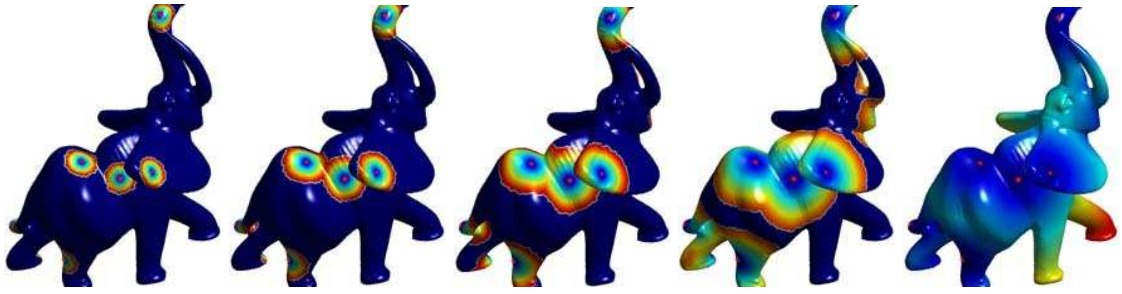


Figure 2.6: *Example of Fast Marching propagation on a triangulated mesh.*

Figure 2.6 shows an example of propagation on a triangulated surface. The colored region corresponds to the point that are computed (its boundary being the front).

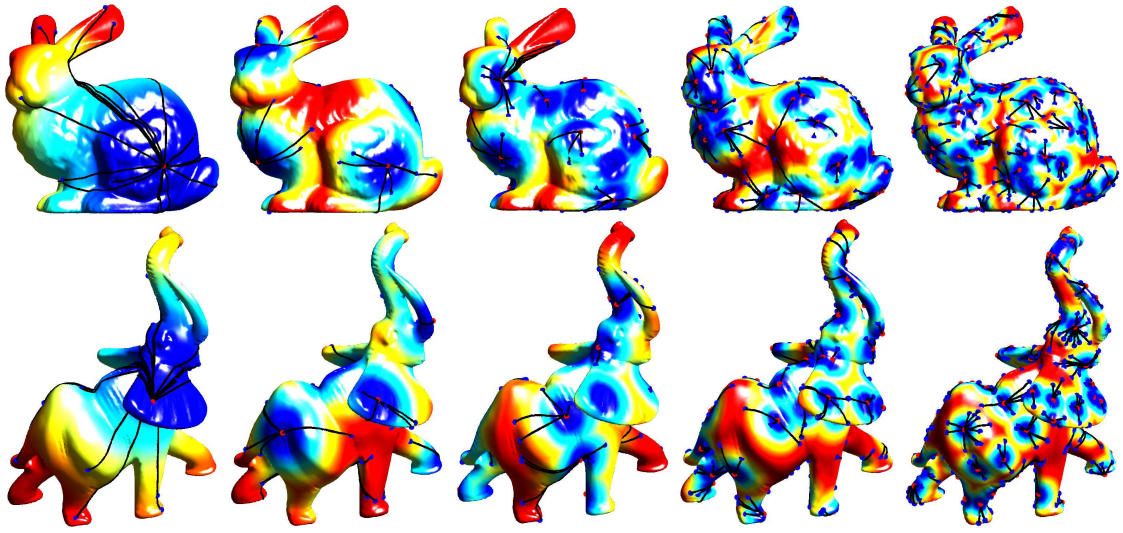


Figure 2.7: *Examples of geodesic extraction on a mesh with an increasing number of starting points.*

The propagation can be started from several starting points $S = \{x_1, \ldots, x_m\}$ in order to compute the geodesic distance $d_{\mathcal{M}}(S, x)$. Figure 2.7 shows examples of such distances to several points together with geodesics. A geodesic γ links a point x to its closest point in S.

2.3 Applications and Extensions of Fast Marching

2.3.1 Shape Analysis

In order to analyze the shape of planar objects, one can consider the metric space obtained by restricting the plane to the inside of a planar domain.

Definition 33 (2D shape)**.** *A 2D shape S is a connected, closed compact set $S \subset \mathbb{R}^2$, with a piecewise-smooth boundary ∂S.*

The geodesic distance inside such a shape is obtained by constraining the curve to lie inside S.

Definition 34 (Geodesic distance in S)**.** *The geodesic distance in S for the uniform metric is*

$$d_S(x,y) \overset{\text{def.}}{=} \min_{\gamma \in \mathcal{P}(x,y)} L(\gamma) \qquad \text{where} \qquad L(\gamma) \overset{\text{def.}}{=} \int_0^1 |\gamma'(t)| \mathrm{d}t.$$

where $\mathcal{P}(x,y) \subset S$ are the paths with starting point x and ending point y.

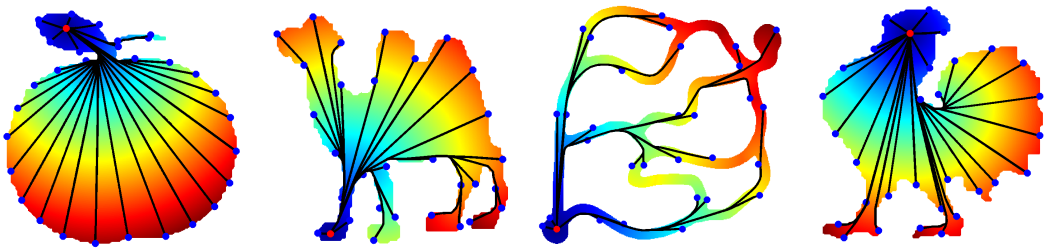


Figure 2.8: *Geodesics inside a 2D shape.*

Figure 2.8 shows examples of shapes together with the geodesic distance to a starting point. The geodesic curve is the union of segments inside S and pieces of the boundary ∂S.

The geodesic distance can be used to define several functions on the 2D shape. This section studies the eccentricity of a shape, as introduced by [15] to perform shape recognition.

Definition 35 (Eccentricity)**.** *The eccentricity $\mathrm{E}_S : \mathcal{M} \mapsto \mathbb{R}$ is*

$$\mathrm{E}_S(x) \overset{\text{def.}}{=} \max_{y \in S} d_S(x,y) = \max_{y \in \partial S} d_S(x,y).$$

Figure 2.9 (top row) shows several examples of eccentricity. The colormap indicates in blue points with small eccentricity. The points for which the minimum in the definition of E_S is obtained are called eccentric.

Definition 36 (Eccentric points)**.** *An eccentric point $x \in \mathcal{E}(S)$ satisfies $\exists y, \mathrm{E}_S(y) = d(x,y)$.*

These eccentric points define regions of influence which perform a segmentation of the shape as follow

$$S = \bigcup_{x \in \mathcal{E}(S)} \{y \in S \setminus \mathrm{E}_S(y) = d(x,y)\}$$

. These eccentric points are in fact located along the boundary.

Theorem 24 (Location of eccentric points)**.** *One has $\mathcal{E}(S) \subset \partial S$.*

A more general definition of eccentricity allows to replace the maximum by a weighted average of geodesic distances.

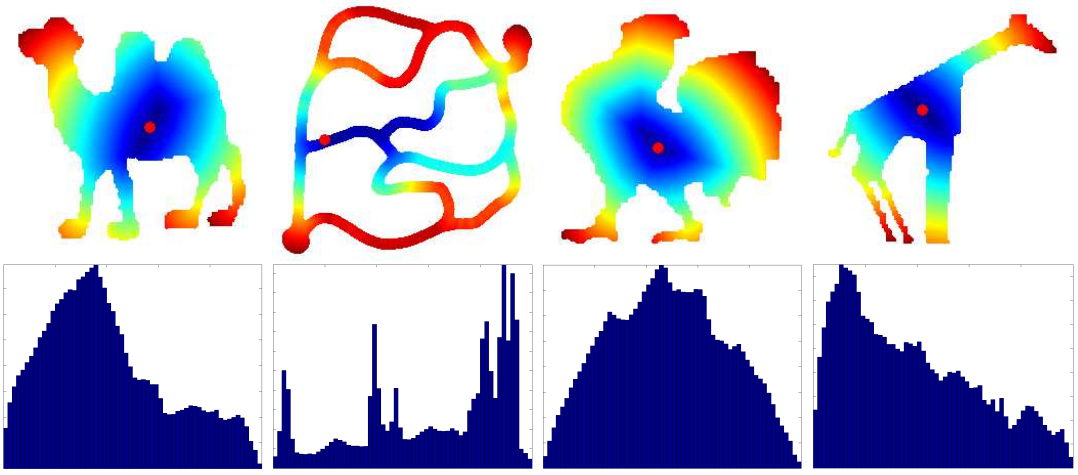


Figure 2.9: *Example of eccentricity E_S and corresponding histograms h_S.*

Definition 37 (α-eccentricity)**.** *The α eccentricity of some shape S is defined as*

$$\mathrm{E}_S^\alpha(x) \overset{\text{def.}}{=} \left(\int_S d_S(x,y)^\alpha \mathrm{d}y \right)^{1/\alpha}.$$

This eccentricity allows to generalize the notion of gravity center to the geodesic setting.

Definition 38 (Euclidean gravity center)**.** *The Euclidean gravity center is*

$$\operatorname*{argmin}_x \int_S \|x-y\|^2 \mathrm{d}y.$$

The α-eccentric center is

$$\operatorname*{argmin}_x \mathrm{E}_S^\alpha(x).$$

Remark 6. For $\alpha = 2$, the eccentric center is called geodesic gravity center (and equivalent to the Euclidean center for an uniform metric).

Having defined a function such as E_S inside a shape S, one can collect information about the shape using the histogram of that function.

Definition 39 (Descriptors)**.** *The eccentricity histogram descriptor $h_S \in \mathbb{R}^m$ of a shape is*

$$\forall\, i = 1, \ldots, m, \quad h_S(i) = \frac{1}{|S|} \# \left\{ x \in S \setminus \frac{i-1}{m} \leqslant \frac{\mathrm{E}_S(x) - \min(\mathrm{E}_S)}{\max(\mathrm{E}_S) - \min(\mathrm{E}_S)} < \frac{i}{m} \right\}.$$

In particular, one can compare shapes by measuring the distance between the histograms

$$\delta(h, \tilde{h})^2 \overset{\text{def.}}{=} \sum_{i=1}^m (h(i) - \tilde{h}(i))^2.$$

These histograms are invariant if one modifies a shape isometrically. In the plane, geodesic isometry of shapes are not interesting since they are rotations and translations. One can however consider approximate isometries such as articulations, that are useful to model deformations of planar shapes, as defined in [22].

Definition 40 (ε-articulated object)**.** *An articulated object S can be split as*

$$S = \bigcup_{i=1}^m S_i \bigcup_{i \neq j} J_{ij},$$

with $\mathrm{diam}(J_{ij}) \leqslant \varepsilon$.

Definition 41 (Articulation)**.** *An articulation is a mapping between two articulated shapes S, S' such that*

$$f : S \to S' = \bigcup_{i=1}^{m} S'_i \bigcup_{i \neq j} J'_{ij}$$

is rigid on $S_i \mapsto S'_i$.

The eccentricity is approximately invariant for shapes that are modified by articulation.

Theorem 25 (Articulation and isometry)**.** *If f is an articulation, then* $\begin{cases} |d_S(x,y) - d_{S'}(x,y)| \leqslant m\varepsilon \\ |\,\mathrm{E}_S(x) - \mathrm{E}_{S'}(x)| \leqslant m\varepsilon \end{cases}$.

Starting from a shape library $\{S_1, \ldots, S_p\}$, one can use the shape signature h_S to do shape retrieval using for instance a nearest neighbor classifier, as shown in pseudo-code 2. Figure 2.10 shows examples of typical shape retrievals.More complex signatures can be constructed out of geodesic distances and un-supervised recognition can also be considered.

1. *Dataset:*shapes $\{S_1, \ldots, S_p\}$ (binary images).
2. *Preprocessing:* compute eccentricity descriptors h_{S_i}.
3. *Input:* shape S.
4. *Retrival:* return $i^\star = \underset{i}{\operatorname{argmin}}\, \delta(h_S, h_{S_i})$.

Table 2: Shape retrieval process.

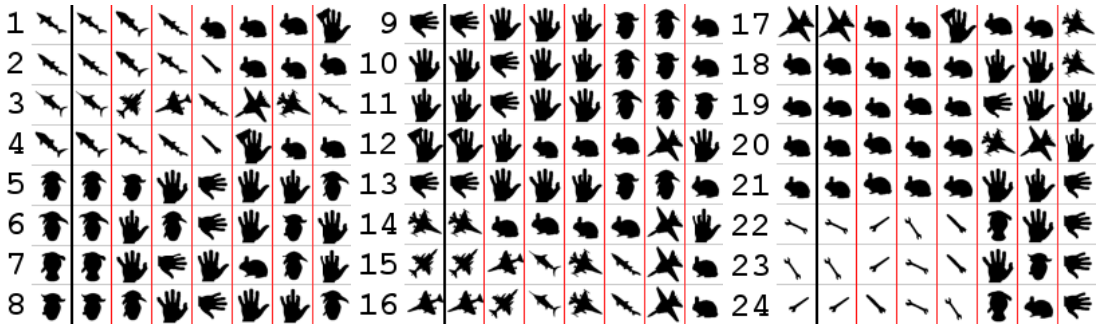


Figure 2.10: *Examples of shape recognitions. The shape on the left is the input S and the second shape in each raw is $S_{i^\star}$.*

2.3.2 Heuristically Driven Propagation

The various implementations of the front propagation algorithm, pseudo-code 1, use a simple priority $\mathcal{P}(x) = U_{x_0}(x)$, where $U(x) \approx d_{\mathcal{M}}(x_0, x)$ is the current value of the distance to the starting point. This strategy leads to an isotropic grow of the front which enforces the exploration of a large area of the computational grid. The advantage of using this priority is that it does not favor any points and thus produces provable valid approximations of geodesic distance (both on a graph with Dijkstra and on a square/triangular grid with Fast Marching).

In order to reduce the computational burden, one could think about using more aggressive ordering of the front that favors some specific direction in the front. The hope is that the front would advance faster in the direction of the goal x_1 one wishes to reach. Ultimately, one would like the front to explore only points along the geodesic $\gamma \in \mathcal{P}_T(x_0, x_1)$ joining the starting point to the ending point.

If one has an oracle: $V(x) \approx d(x_1, x)$ that estimates the remaining geodesic distance from the current point x to the end x_1, one can use as priority map

$$\mathcal{P}(x) = U(x) + V(x).$$

The map V is called an heuristic since the exact distance $d(x_1, x)$ is not available in practice. The value of a good heuristic close to the real distance is revealed by the following theorem.

Theorem 26 (Geodesic segment)**.** *The function $\psi(x) = d(x_0, x) + d(x_1, x)$ is minimal and constant $\psi(x) = d(x_0, x_1)$ along the geodesic path joining x_0 and x_1.*

In the setting of graph theory, the Dijkstra algorithm can be replaced by the A* (A-star), [25], which uses an heuristic to speed up computations. The following theorem proves the validity of this approach.

Theorem 27 (A* validity)**.** *If the heuristic satisfies $V(x) \leqslant d(x_1, x)$, then the curve $\gamma \in \mathcal{P}(x_0, x_1)$ extracted from the front propagation, algorithm 1, is a geodesic between x_0 and x_1.*

Over a continuous domain, one can invoke a similar (but weaker) theorem.

Theorem 28 (Explored area)**.** *If the heuristic satisfies $V(x) \leqslant d(x_1, x)$, then the geodesic $\gamma \in \mathcal{P}_1(x_0, x_1)$ between x_0 and x_1 satisfies*

$$\{\gamma(t) \setminus t \in [0, 1]\} \subset \{x \setminus \mathcal{P}(x) = U(x) + V(x) \leqslant \mathcal{P}(x_1)\}.$$

This theorem shows why it is important to approximated the geodesic distance by below, since otherwise the region explored by the algorithm might not contain the true geodesic.

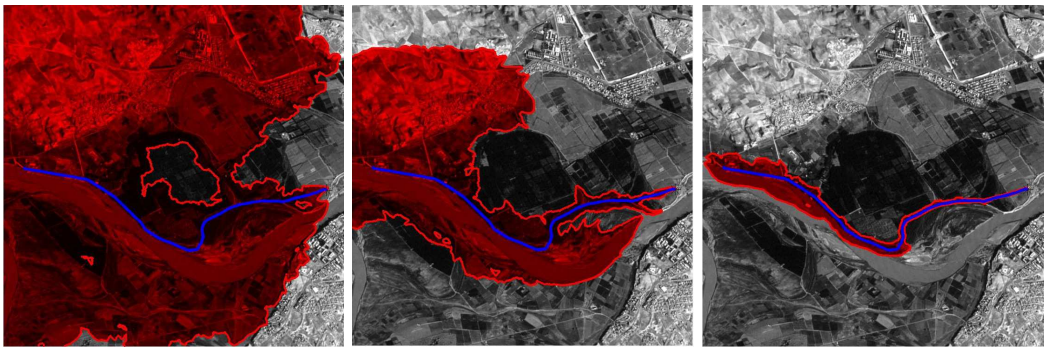


Figure 2.11: *Example of propagations with a priority $\mathcal{P}(x) = U(x) + \lambda V(x)$ for $\lambda = 0, 0.5, 0.9$.*

Figure 2.11shows examples of heuristics that approximate the true remaining distance by bellow. One can see how the explored area of the propagation progressively shrinks while containing the true geodesic. Such an heuristic is however impossible to use in practice since one does not have direct access to the remaining distance during the propagation.

Many strategies can be used to estimate an heuristic. For instance, on a Riemannian metric $(\mathcal{M}, H(x))$, one could use

$$V(x) = \rho \|x - x_1\| \qquad \text{where} \qquad \rho = \min_{x \neq 0, v \neq 0} \|v\|_{H(x)}.$$

In this case, ρ is the minimum eigenvalue of all the tensors $H(x)$. This heuristic estimates the geodesic distance with an Euclidean distance and satisfies $V(x) \leqslant d(x_1, x)$.

For a propagation on a graph (A* algorithm) that is embedded in Euclidean space according to $i \in V \mapsto x_i \in \mathbb{R}^k$, one could also define

$$\forall\, i \in V, \quad V(i) = \|x_{i_1} - x_i\|,$$

where i_1 is the index of the ending point. This heuristic also satisfy $V(i) \leqslant d(i_1, i)$.

These Euclidean heuristics performs poorly on spaces that are not relatively flat. In order to compute more accurate heuristic, we use an expression of the geodesic distance as a minimization.

Theorem 29 (Reversed triangular inequality)**.** *For all $(x, y) \in \mathcal{M}$, one has*

$$d(x, y) = \sup_z \Big(|d(x, z) - d(z, y)| \Big).$$

If one restricts the minimum to a small subset of landmark points $\{z_1, \ldots, z_n\} \subset \mathcal{M}$, one can define the following approximate distance

$$\tilde{d}_{z_1 \ldots z_n}(x, y) = \sup_{k=1 \ldots n} \Big(|d_k(x) - d_k(y)| \Big),$$

This kind of approximation has been used first in graph theory [13] and it is defined in a continuous setting in [29]. This leads to an heuristic $V(x) = \tilde{d}(x, x_1)$ that has the following properties.

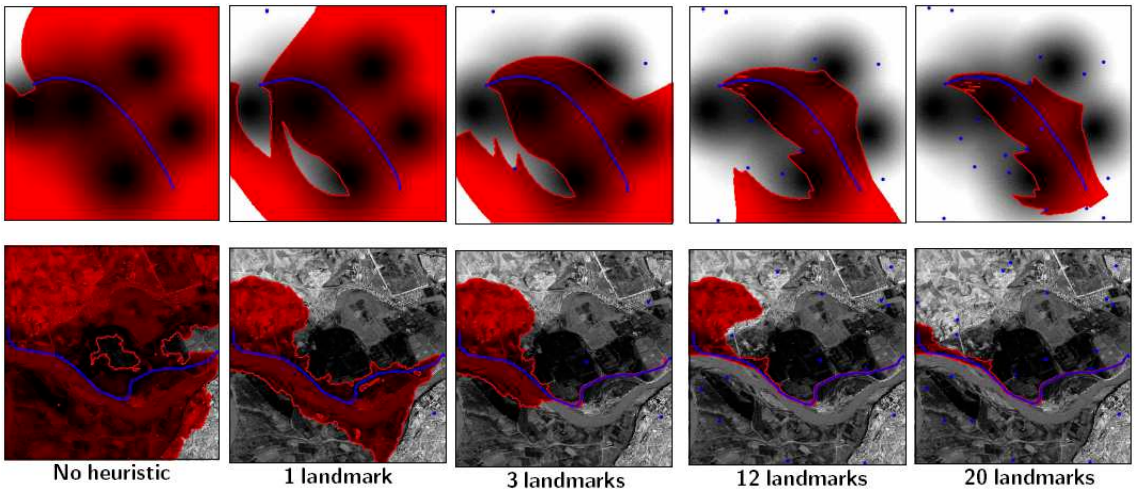


Figure 2.12: *Heuristically driven propagation in 2D with an increasing number of landmark points.*

Theorem 30 (Convergence of heuristic)**.** *One has $\tilde{d} \leqslant d$ and $\tilde{d} \overset{n \to +\infty}{\longrightarrow} d$.*

In a numerical application that requires the extraction of many geodesics in real time over a large domain, one can pre-compute (off-line) the set of distance maps to the landmarks $\{d(x, z_i)\}_{i=1}^m$. At run time, this set of distances is used to compute the heuristic and speed up the propagation. Figure 2.12 shows how the quality of the heuristic increases with the number of landmarks. Figure 2.13 shows an application to geodesic extraction on 3D meshes.

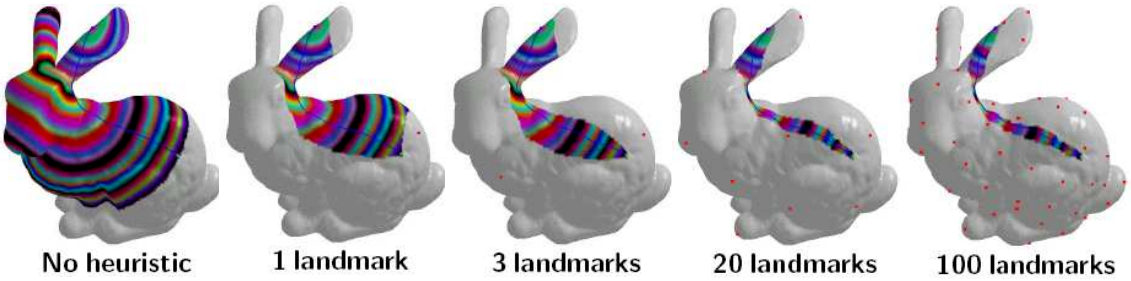


Figure 2.13: *Heuristically driven propagation on a 3D mesh with landmark points.*

2.4 Surface Sampling

In order to acquire discrete samples from a continuous surface, or to reduce the number of samples of an already acquired mesh, it is important to be able to seed evenly a set of points on a surface. This is relevant in numerical analysis to have a good accuracy in computational simulations, or in computer graphics to display 3D models with a low number of polygons. In practice, one typically wants to enforce that the samples are approximately at the same distance ones from each others. The numerical computation of geodesic distances is thus a central tool, that we are going to use both to produce the sampling and to estimate the connectivity of a triangular mesh.

2.4.1 Farthest Point Sampling

A sampling of a Riemannian surface $\mathcal{M}$ is a set of points $\{x_1, \ldots, x_n\} \subset \mathcal{M}$. If the surface is parameterized by $\varphi : [0,1]^2 \mapsto \mathcal{M}$, the easiest way to compute a sampling is to seed points regularly over the parametric domain

$$\forall\, (i,j) \in \{1, \ldots, \sqrt{n}\}^2, \quad x_{i,j} = \varphi(i/\sqrt{n}, j/\sqrt{n}).$$

This strategy performs poorly if the mapping φ introduces heavy geodesic distortion and the sampling might not be regular any more for the geodesic metric on the surface. In order to ensure the quality of a sampling, one can use the notion of a well separated covering.

Definition 42 (ε-covering)**.** *A sampling $\{x_1, \ldots, x_n\} \subset \mathcal{M}$ is an ε-covering if*

$$\bigcup_i B_\varepsilon(x_i) = \mathcal{M} \qquad \textit{where} \qquad B_\varepsilon(x) \stackrel{\text{def.}}{=} \{y \setminus d_{\mathcal{M}}(x,y) \leqslant \varepsilon\}\,.$$

Definition 43 (ε-separated)**.** *A sampling $\{x_1, \ldots, x_n\} \subset \mathcal{M}$ is ε-separated if*

$$\max(d_{\mathcal{M}}(x_i, x_j)) \leqslant \varepsilon.$$

The farthest point sampling algorithm is a simple greedy strategy able to produce quickly a good sampling. This algorithm has been introduced in image processing to perform image approximation [11]. It is used in [30] together with geodesic Delaunay triangulation (to be defined in the next section) to do surface remeshing.

Pseudo code 3 gives the details of this iterative algorithm. In particular, note that the update of the distance $d(x)$ to the set of already seeded points goes faster at each iteration since the domain of update is smaller when the number of points increases.

1. *Initialization:* $x_1 \leftarrow$random, $d(x) \leftarrow d_{\mathcal{M}}(x_1, x)$, set $i = 1$.
2. *Select point:* $x_{i+1} = \operatorname*{argmax}_x d(x)$, $\varepsilon = d(x_{i+1})$.
3. *Local update of the distance:* $d(x) \leftarrow \min(d(x), d_{\mathcal{M}}(x_{i+1}, x))$.
   This update is restricted to the set of points $\{x \setminus d_{\mathcal{M}}(x_{i+1}, x) < d(x)\}$.
4. *Stop:* If $i < n$ or $\varepsilon > \varepsilon_0$, set $i \leftarrow i + 1$ and go back to 2.

Table 3: Farthest point sampling algorithm.

The output sampling of the algorithm enjoys the property of being a well separated covering of the manifold.

Theorem 31 (Farthest seeding properties)**.** *The farthest point sampling $\{x_1, \ldots, x_n\}$ is an ε-covering that is ε-separated for*

$$\varepsilon = \max_{i=1,\ldots,n} \min_{j=1,\ldots,n} d_{\mathcal{M}}(x_i, x_j).$$

Note however that there is no simple control on the actual number of samples n required to achive a given accuracy ε. We refer to [5] for an in-depth study of the approximation power of this greedy sampling scheme.

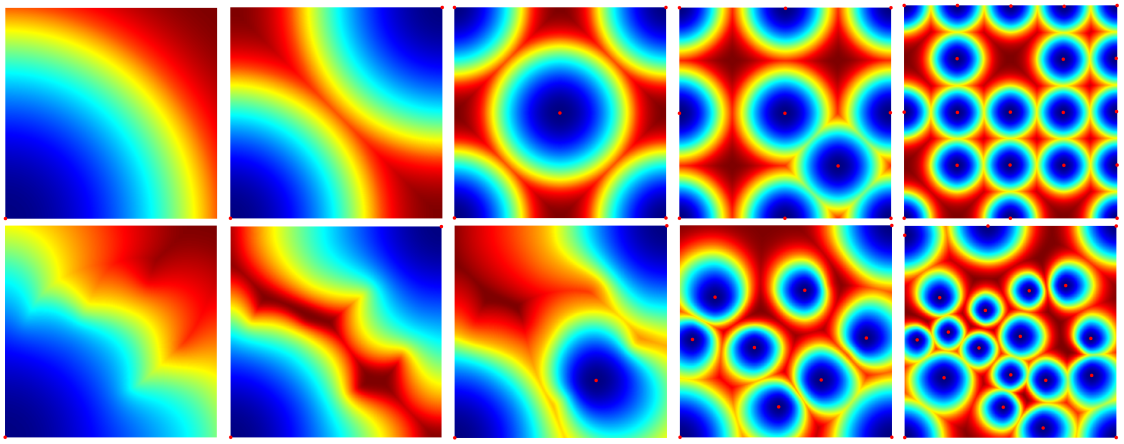


Figure 2.14: *Examples of farthest point sampling (the colormap indicates the distance function to the seeds).*

Figure 2.14 shows examples of farthest point sampling with an uniform (top row) and a spatially varying isotropic metric $W(x)$ (bottom row). One can see that this scheme seeds more points in areas where the metric W is large. One can thus control the sampling density by modifying the metric W.

2.4.2 Triangulations

Having computed, for instance with farthest points, a sampling $\{x_i\}_{i\in V} \subset \mathcal{M}$, the next step is to compute some connectivity between the samples in order to build a graph, or even better, a triangulation. The problem of surface remeshing has been studied extensively in computer graphics, see the survey [2]. This section explains a solution based on the geodesic Delaunay triangulation.

The following definition generalizes the notion of an Euclidean Voronoi diagram, definition 10, to an arbitrary surface.

Definition 44 (Voronoi segmentation)**.** *The Voronoi segmentation of a sampling $\{x_i\}_{i\in V} \subset \mathcal{M}$ is*

$$\mathcal{M} = \bigcup_i V_i \qquad \textit{with} \qquad (V_i)_{i=1}^m \overset{\text{def.}}{=} \textit{Voronoi}_{\mathcal{M}}(\{x_i\}_i)$$

where

$$V_i \overset{\text{def.}}{=} \{x \setminus \forall j \neq i,\ d_{\mathcal{M}}(x, x_i) \leqslant d_{\mathcal{M}}(x, x_j).\}$$

Each Voronoi cell V_i is thus composed of points that are closer to x_i than to any other sampling point. The boundary between two adjacent cells V_i and V_j is thus a piece of curve at equal distance between x_i and x_j. One can then compute the graph dual to a given partition, which joins together pair of adjacent cells. This leads to the notion of Delaunay graph.

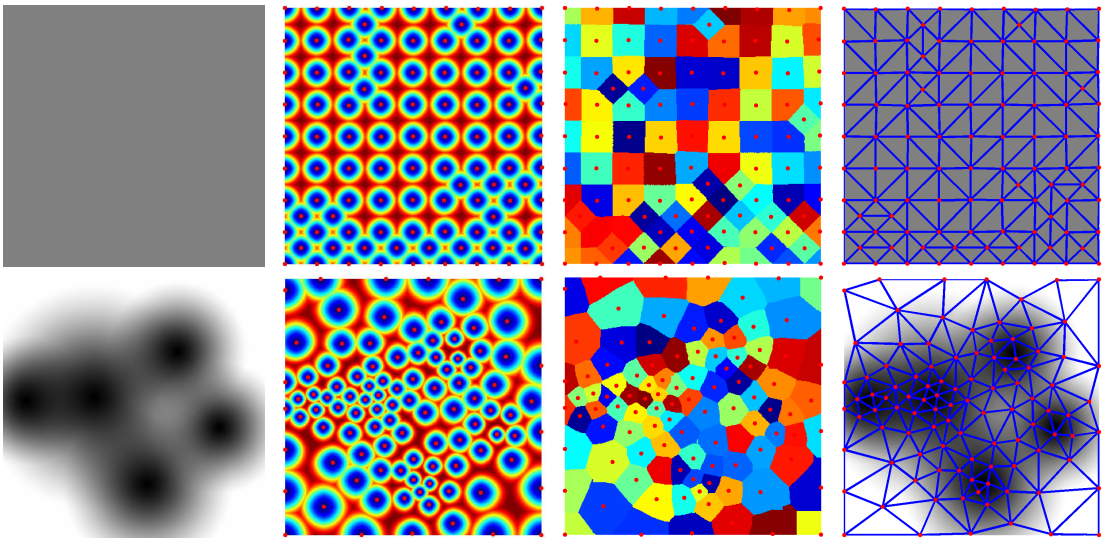
Definition 45 (Geodesic Delaunay graph)**.** *The Delaunay graph (V, E) of a samplind $\{x_i\}_{i\in V} \subset \mathcal{M}$ is defined for $V = \{1, \ldots, n\}$ as*

$$E = \{(i, j) \in V \setminus \partial V_i \cap \partial V_j \neq 0\}.$$

The main interest of this Delaunay graph is that, if the number of points is large enough to capture the topology of the surface (for instance at least 4 points are needed on a sphere), then one gets a valid triangulation.

Theorem 32. *For a large enough number of points, the Delaunay graph is a valid triangulation.*

This theorem means that one can find a set of faces F such that (V, E, F) is a triangulated mesh. One can see [21] for a theoretical study of geodesic Delaunay triangulations.



Metric Sampling Voronoi Delaunay

Figure 2.15: *Examples of sampling and triangulations.*

This Delaunay triangulation can thus be used to perform a geodesic meshing or re-meshing of any Riemannian surface, as explained in [30]. Figure 2.15 shows examples of Voronoi segmentations on the plane for various isotropic Riemannian metrics $W(x)$. The Delaunay graph allows to define a planar mesh of points evenly sampled according to the metric.

Figure 2.16 shows examples of Voronoi cells on a surface embedded in $\mathbb{R}^3$.

In order to mesh the interior of a planar shape $S \subset \mathbb{R}^2$, one can use the Euclidean metric inside the shape and compute a geodesic Delaunay triangulation. Figure 2.17 shows some examples of shape meshing with this uniform metric. This triangulation is however very close to the usual definition of a planar Euclidean Delaunay triangulation. In contrast, one can use a non-uniform metric $W(x)$ and compute a sampling inside the shape that conforms itself to this density. Figure 2.17 shows a sampling and meshing that uses a metric $W(x) = (\varepsilon + d(x, \partial S))^{-1}$ that tends to seed more points on the boundary of the shape S.

Figure 2.18 shows an example of uniform remeshing of a 3D surface with an increasing number of points. Figure 2.19 shows how one can adapt the density by defining a non-constant isotropic metric on the surface.

An option to compute this metric is to use a texture mapped on the surface. Starting from some parametric surface: $\varphi : \mathcal{D} \subset [0,1]^2 \to \mathcal{M}$, a texture T is a mapping $T : [0,1]^2 \to \mathbb{R}$. It allows to define an isotropic metric using for instance an edge adaptive function

$$\forall\, x \in \mathcal{D},\ H(x) = \psi_T(x)\mathrm{Id}_2.$$

where the edge-based stopping function is $\psi_T(x) = (\|\nabla_x T\| + \varepsilon)^{-1}$. Figure 2.20 shows examples of remeshing with a texture-adapted metric with a decreasing value of ε (increasing adaptivity).

2.4.3 Centroidal Tesselation

For some applications, the sampling quality provided by the greedy farthest point sampling, pseudo code 3, might not be good enough. One can thus try to enhance its quality by a relaxation

Figure 2.16: *Example of Voronoi segmentations for an increasing number of seeding points.*

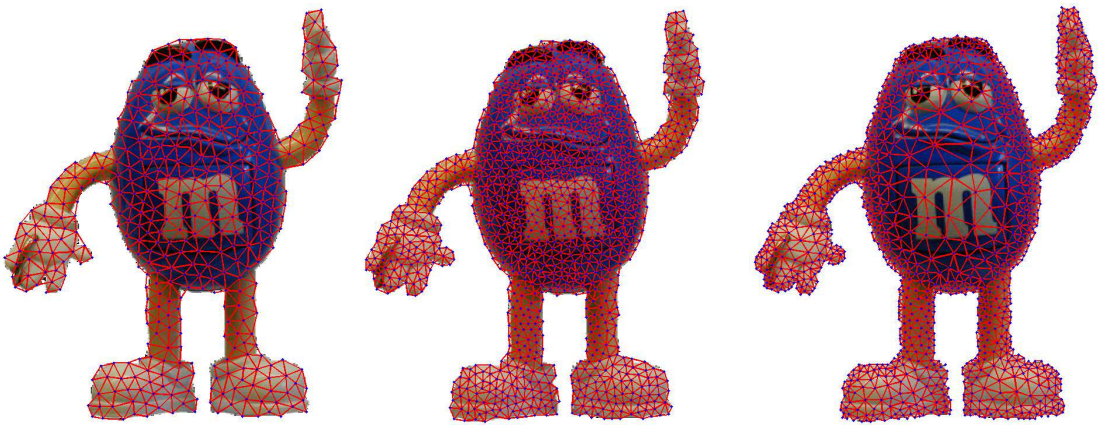


Figure 2.17: *Shape meshing with an increasing number of points. Left and center: uniform meshing, right: adaptive meshing.*

scheme that updates the position of each sample in order for them to be globally more evenly distributed. To give a precise definition of this sampling quality, one can use the geodesic gravity center, already introduced in definition 38 in the special case of a planar shape.

Definition 46 (Geodesic gravity center)**.** *The geodesic center of gravity of $A \subset \mathcal{M}$ is*

$$g_{\mathcal{M}}(A) = \operatorname*{argmin}_x \int_A d_{\mathcal{M}}(x,y)^2 \mathrm{d}y.$$

For an Euclidean metric where $H(x) = \mathrm{Id}$, one gets the traditional definition of gravity center

$$g_{\mathrm{eucl}}(A) = \operatorname*{argmin}_x \int_A \|x-y\|^2 \mathrm{d}y. \quad \Longrightarrow \quad g_{\mathrm{eucl}}(A) = \frac{1}{|A|}\int_A y \mathrm{d}y,$$

where $|A|$ is the Euclidean area of A.

A good sampling is then defined as a sampling whose Voronoi cells are compact and rounded, which is equivalent to each cell having a gravity center equal to its seed.

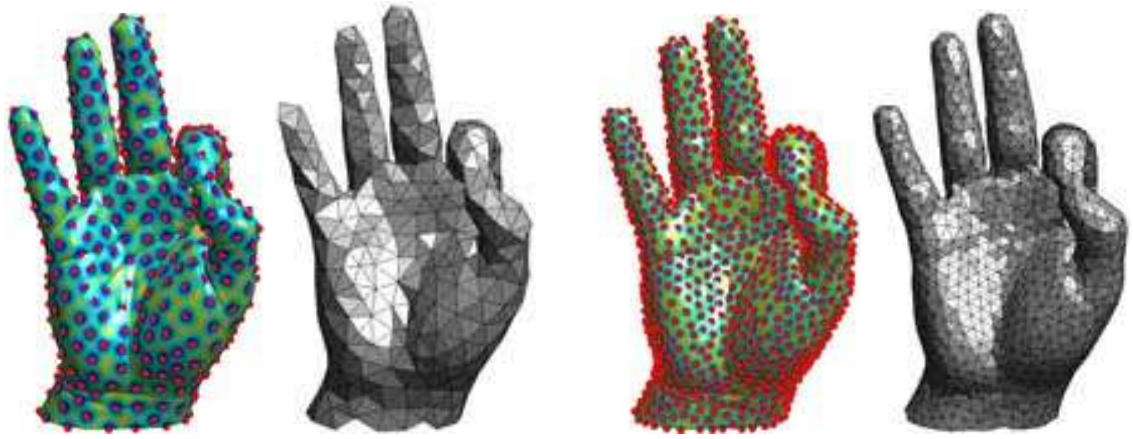


Figure 2.18: *Geodesic remeshing with an increasing number of points. Top: sampling with corresponding distance function, bottom: corresponding geodesic triangulation.*

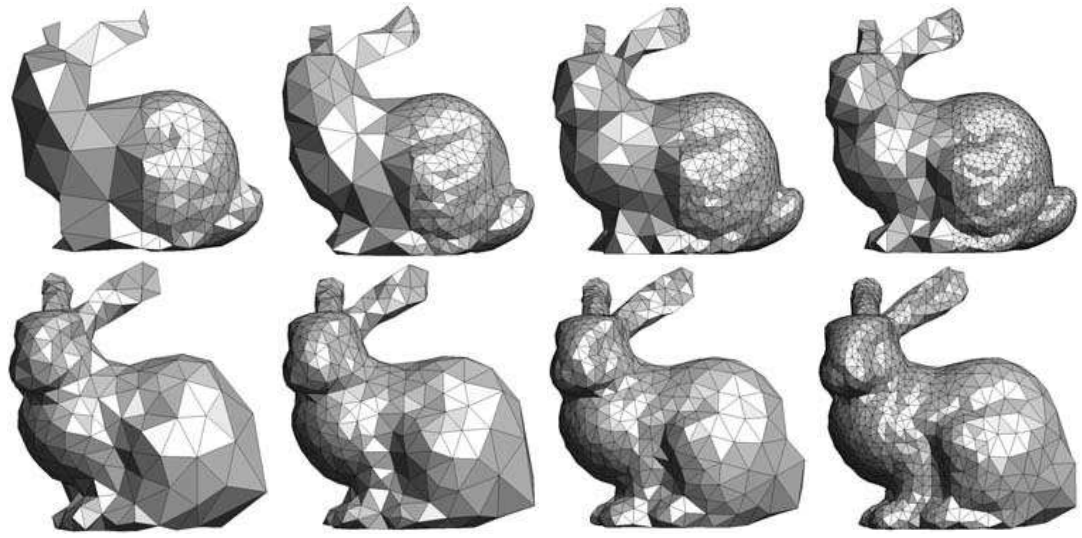


Figure 2.19: *Adaptive remeshing with a binary density function (top row) and a linearly increasing density (bottom row).*

Definition 47 (Centroidal tesselation)**.** *A sampling $\{x_i\}_{i=1}^m$ is centroidal if its Voronoi tessellation $Voronoi_{\mathcal{M}}(\{x_i\}_i) = \{V_i\}_{i=1}^m$ satisfies $x_i = g_{\mathcal{M}}(V_i)$.*

One can see [10] for an overview of the theory and applications of centroidal tesselations with weighted Euclidean metrics. The geodesic centroidal tesselation has been introduced for surface remeshing and segmentation in [30]. See also [2] for other ways to use centroidal tesselations for remeshing.

This definition can be turned into an energy minimization as follow.

Theorem 33 (Variational characterization)**.** *A centroidal tesselation is a local minimizer of*

$$E(\{x_i\}, \{V_i\}_i) = \sum_{i=1}^{m} \int_{V_i} d_{\mathcal{M}}(x_i, y)^2 \mathrm{d}y.$$

Such a local minimizer can be computed by a gradient descent of E, using

$$E_i(x) = \int_{V_i} d_{\mathcal{M}}(x_i, y)^2 \mathrm{d}y \quad \Longrightarrow \quad \nabla_x E_i = \int_{V_i} d_{\mathcal{M}}(x_i, y) \overrightarrow{n_x(y)} \mathrm{d}y,$$

where $n_x(y) = \gamma'(0)/\|\gamma'(0)\|$ is the unit tangent at x to the geodesic $\gamma \in \mathcal{P}_T(x, y)$ joining x to y.

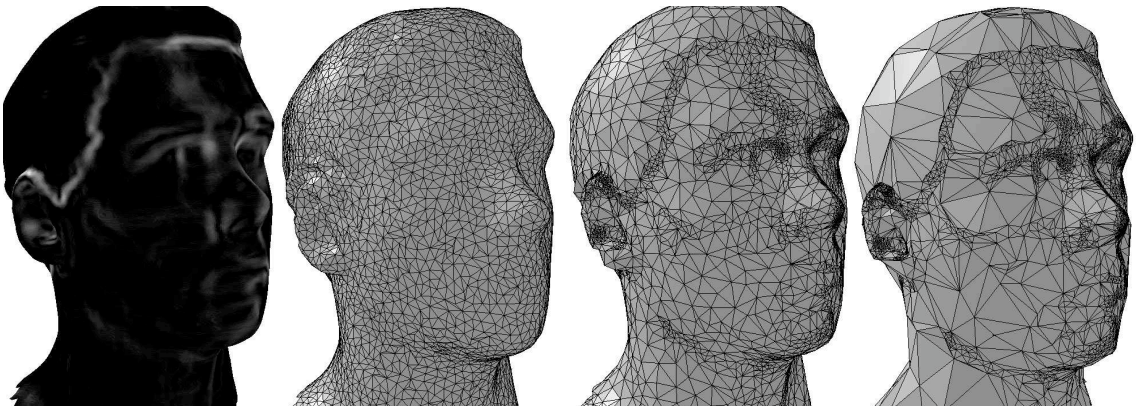


Figure 2.20: *Adaptive remeshing with a density given by a texture.*

A gradient descent of E_i allows to compute the geodesic gravity center of a given region of the surface. It is the central ingredient of the Lloyd algorithm detailed in pseudo code 4, that converges in practice to a local optimum of the energy E and thus a centroidal tesselation. Of course, centroidal tesselation of a given domain are not unique and computing the best one (with the smallest value of E) is a difficult task. In a discrete setting (point clouds in the Euclidean space) the Lloyd algorithm is equivalent to the k-means algorithm used to perform vector quantization and classification.

1. *Initialize:* $\{x_i\}_i \leftarrow$ random.
2. *Update regions:* $\forall\, i, \{V_i\}_i \leftarrow \mathrm{Voronoi}_{\mathcal{M}}(\{x_i\}_i)$.
3. *Update centers:* $\forall\, i, x_i \leftarrow g_{\mathcal{M}}(V_i)$.
4. *Stop:* while not converged, go back to 2.

Table 4: Max-Lloyd relaxation algorithm.

Figure 2.21, top row, shows examples of iterations of the Lloyd algorithm on a square with euclidean metric. Middle and bottom row shows the Lloyd algorithm with a weighted euclidean metric, for which the gravity center are defined as

$$g_\rho(A) = \operatorname*{argmin}_x \int_A \rho(y)\|x-y\|^2 \mathrm{d}y. \quad \Longrightarrow \quad g_\rho(A) = \frac{\int_A \rho(y) y \mathrm{d}y}{\int_A \rho(y)\mathrm{d}y},$$

where $\rho : [0,1]^2 \to \mathbb{R}^+$ is a user-defined weight function. Not that this definition does not corresponds to a geodesic gravity center with an isotropic geodesic metric $H(x) = \rho(x)\mathrm{Id}_2$, but this usually gives similar results. In practical situation, one wants to choose a weigth function $W(x)$ or $\rho(x)$ in order for the size of the triangles near a position x to conform to some sizing field $s(x)$. As explained in [10], the weights should obey approximately the scaling relation $\rho(x) \approx W(x) \sim s(x)^{-(d+2)}$ where d is the dimension of the domain ($d = 2$ for a 2D surface mesh).

Figure 2.22 shows iterations of the algorithm on a 3D surface, equipped with an uniform metric.

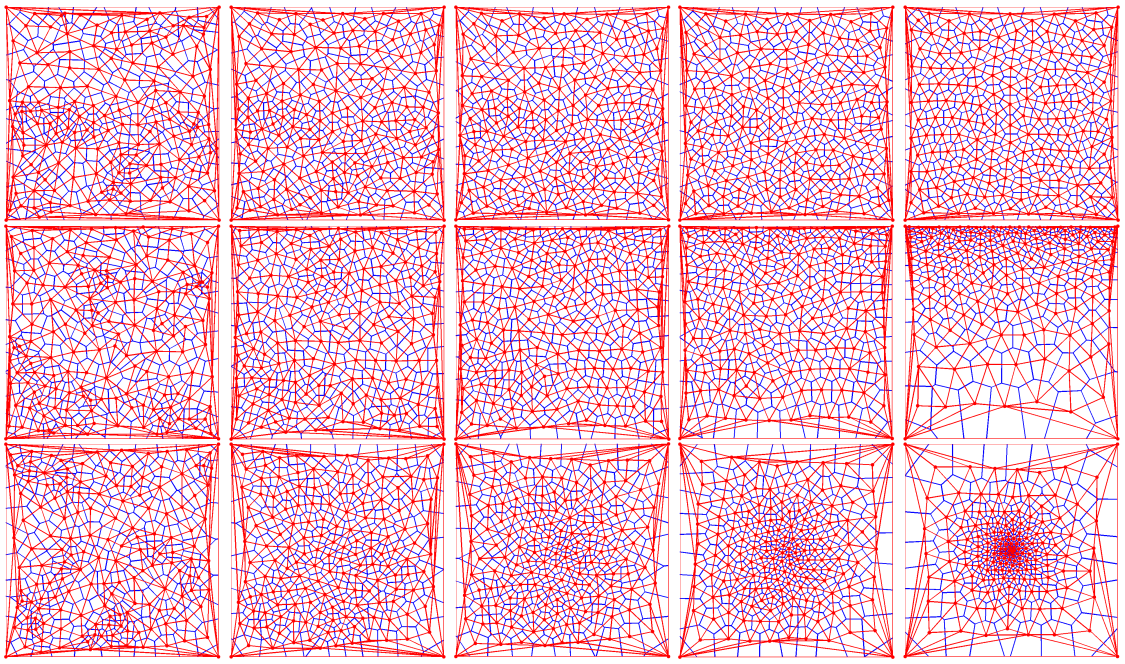


Figure 2.21: *Iterations of the Lloyd algorithm on a square with euclidean metric (top row) and weighted euclidean metric (middle and bottom rows). The Delaunay triangulation is depicted in red and the corresponding Voronoi diagram is in blue.*

Figure 2.22: *Iterations of the Lloyd algorithm on two different surfaces.*

3 High Dimensional Data Analysis

This chapter presents various methods to analyze point clouds in high dimension. The emphasis is put on manifold models that allows to parameterize these point clouds with a small number of parameters. Examples in image processing shows that some image ensemble indeed satisfy these manifold models.

3.1 Manifold Models of High Dimensional Data

3.1.1 Point Clouds and Manifold

3.1.2 Graph Approximation and Geodesic Distances

3.1.3 Image Ensembles

3.2 Flattening and Dimension Reduction

In order to perform data analysis, one often has to reduce the dimension of the input set of points. This can be useful to extract salient features from a data set and to speed up processing. This section shows how several related problems all lead to dimensionality reduction using spectral methods. In particular, surface flattening can be re-casted as a matrix approximation problem with dimensionality reduction.

3.2.1 Point Clouds

A data set is a large point clouds $X = (x_j)_{j=1}^p \in \mathbb{R}^{n\times p}$ with $x_j \in \mathbb{R}^n, n \ll p$. From this set of input data, one can define two matrices that represent in transposed forms the correlation present in the input data (second order statistics).

Definition 48 (Correlation matrix)**.** $C = XX^{\mathrm{T}} \in \mathbb{R}^{n\times n}, C_{ij} = \sum_k x_k(i)x_k(j)$.

Definition 49 (Gram matrix)**.** $K = X^{\mathrm{T}}X \in \mathbb{R}^{p\times p}, K_{ij} = (\langle x_i,\, x_j\rangle)_{i,j}$.

It is important to remember that these matrices do not carry the same amount of information and depending on the application (flattening, dimension reduction, best approximation, etc) one can use either the correlation or the gram matrix.

An important tools from numerical analysis is the singular value decomposition (SVD) that generalizes the orthogonal diagonalization of symmetric matrices to non-symmetric, non-square matrices. Such an orthogonal decomposition is crucial to perform matrix approximation by deleting small spectral values.

Theorem 34 (Singular value decomposition)**.** *Any matrix X can be decomposed as*

$$X = U\Sigma V^{\mathrm{T}} = U\Sigma_n {V_n}^{\mathrm{T}} \qquad \textit{with} \qquad U \in \mathcal{O}(n), V \in \mathcal{O}(p),$$

where $\mathcal{O}(n) \subset \mathbb{R}^{n\times n}$ is the group of orthogonal matrices and

$$\Sigma = \mathrm{diag}(\sigma_i)_i,\ \Sigma_n = \mathrm{diag}(\sigma_1, \ldots, \sigma_n),\ V_n = (v_1, \ldots, v_n) \in \mathbb{R}^{p\times n}.$$

This decomposition satisfies

$$\begin{cases} XX^{\mathrm{T}} = U\Lambda U^{\mathrm{T}} \\ X^{\mathrm{T}}X = V\Lambda V^{\mathrm{T}} \end{cases}.$$

The singular values $\Lambda = \Sigma^2$ are ordered $\lambda_1 \geqslant \ldots \geqslant \lambda_n \geqslant \lambda_{n+1} = 0$.

In order to compare point clouds, one can use various matrix norms, among which the most popular are the Frobenius norm and the norm derived from the vectorial norm.

Definition 50 (Matricial norms)**.**

$$\|X\|_2 \overset{\text{def.}}{=} \operatorname*{argmin}_{\|a\|=1} \|Xa\| \qquad \textit{and} \qquad \|X\|_F^2 = \sum_{ij} X_{ij}^2 = \mathrm{tr}(X^{\mathrm{T}}X) = \mathrm{tr}(XX^{\mathrm{T}}).$$

The properties of these norms are recalled in this theorem.

Theorem 35. *These norms are invariant under isometries, since for any orthogonal matrix U,*

$$\|UX\|_F = \|X\|_F \qquad \textit{and} \qquad \|UX\|_2 = \|X\|_2.$$

These matricial norms satisfy

$$\|X\|_2 = \sigma_1 \qquad \textit{and} \qquad \|X\|_F^2 = \sum_i \sigma_i^2.$$

3.2.2 Linear Dimension Reduction

In order to perform linear approximation of some data set, one needs to use orthogonal projectors.

Definition 51 (Orthogonal projectors)**.** *The set of orthogonal projectors of rank k is defined as*

$$\mathcal{P}_k \overset{\text{def.}}{=} \left\{ P = AA^{\mathrm{T}} \setminus A \in \mathbb{R}^{n\times k} \quad \textit{and} \quad A^{\mathrm{T}}A = \mathrm{Id}_k \right\}$$

The optimal projection of a given data set can be computed easily using a spectral decomposition.

Theorem 36 (Dimension reduction solution)**.** *Up to a rotation, $U_k = (u_1, \ldots, u_k)$ satisfies*

$$U_k {U_k}^{\mathrm{T}} = \operatorname*{argmin}_{P\in\mathcal{P}_k} \|X - PX\|_F = \operatorname*{argmin}_{P\in\mathcal{P}_k} \|X - PX\|_2.$$

The Principal component analysis (PCA) of a data set X corresponds to reducing the dimensionality using the optimal linear projector

$$\begin{cases} X \in \mathbb{R}^{n\times p} & \longmapsto & X_k = {U_k}^{\mathrm{T}} X \in \mathbb{R}^{k\times p} & \text{(dim. reduction)} \\ X_k \in \mathbb{R}^{k\times p} & \longmapsto & \bar{X} \overset{\text{def.}}{=} UX_k & \text{(reconstruction)} \end{cases}$$

The error that occurs from this linear reduction / reconstruction cycle can be monitored using the singular values that have been removed during reduction

Theorem 37 (PCA error)**.** *One has $\|X - \bar{X}\|^2 = \sum_{i>k} \sigma_i^2$.*

It is important to remember that the PCA approximation is optimal only in a linear setting. for input exemplar x_j that corresponds to smooth function (or similarly for data set that are close to gaussian), this linear approximation is very good and is close to the low-frequency Fourier expansion. However for complex data x_j such as points on non-smooth meshes or natural images, this linear setting performs poorly. Non-linear methods such as wavelets can overcome this drawback.

3.2.3 Distance and Gram Matrices

An input data set is treated as a points cloud $X = (x_j)_{j=1}^p \in \mathbb{R}^{n\times p}$ with $x_j \in \mathbb{R}^n$. In order to manipulate this points cloud, this section considers two kinds of euclidean invariants: inner products and distances. Inner products matrices are easy to deal with since they form a convex cone that can be mapped back to the point cloud X using the diagonalization of symmetric operators. In contrast, distance matrices form a complex set difficult to handle. To ease the computation with distance matrices, we consider a simple mapping between inner products and distances.

Definition 52 (Gram matrices)**.** *The set of Gram matrices is defined as*

$$\mathcal{K}_n \overset{\text{def.}}{=} \left\{X^{\mathrm{T}}X \setminus X \in \mathbb{R}^{n\times p}\right\} \subset \mathbb{R}^{p\times p} \qquad \textit{and} \qquad \mathcal{K} = \bigcup_{1\leqslant n\leqslant p} \mathcal{K}_n.$$

The nice property of inner product matrices is that they can be characterized by their spectral content.

Theorem 38 (Kernel factorization)**.** $K \in \mathcal{K}_n \iff K$ *is positive semi-definite and* $\mathrm{rank}(K) = n$.

Remark 7 (Description of the factorization)*.* In particular, an inner product matrix can be mapped back to a point cloud X using diagonalization.

$$\left\{\begin{array}{l} K = V\Lambda V^{\mathrm{T}} \\ \mathrm{rank}(K) = n \end{array}\right. \implies K = X^{\mathrm{T}}X \quad \text{with} \quad \left\{\begin{array}{l} X = \sqrt{\Lambda_n}V_n \\ V_n = (v_1, \ldots, v_n) \end{array}\right. .$$

An other important set of matrices is those that contain pairwise distances between points in Euclidean space.

Definition 53 (Distance matrices)**.** *The set of euclidean distance matrices is defined as*

$$\mathcal{D}_n \overset{\text{def.}}{=} \left\{D = (\|x_i - x_j\|^2)_{i,j} \setminus X \in \mathbb{R}^{n\times p}\right\} \subset \mathbb{R}^{p\times p} \qquad \textit{and} \qquad \mathcal{D} = \bigcup_{1\leqslant n\leqslant p} \mathcal{D}_n.$$

It is not obvious to check wether a given matrix is a distance matrix or not. At least it has to satisfy the following properties of a pre-distance matrix.

Definition 54 (Pre-distance matrices)**.** *The set of pre-distance matrices is defined as*

$$\mathcal{D}^+ \overset{\text{def.}}{=} \left\{D \setminus D \geqslant 0, D^{\mathrm{T}} = D, \mathrm{diag}(D) = 0\right\}.$$

Remark 8*.* One has the inclusions: $\mathcal{D}_1 \subset \mathcal{D}_2 \subset \ldots \subset \mathcal{D}_p \subset \mathcal{D}^+$.

Using the expansion

$$\|x_i - x_j\|^2 = \|x_i\|^2 + \|x_j\|^2 - 2\langle x_i, x_j\rangle = K_{ii} + K_{jj} - 2K_{ij},$$

one can define a mapping between kernels and distance matrices as follow

$$\varphi : \left\{\begin{array}{ccc} \mathcal{K}_n & \longrightarrow & \mathcal{D}_n \\ K & \longmapsto & K\mathbb{I} + \mathbb{I}K - 2K \end{array}\right. \qquad \text{where} \quad \mathbb{I} = 11^{\mathrm{T}}.$$

Remark 9*.* One can readily note that

$$\left\{\begin{array}{rcl} \varphi((X+\lambda)^{\mathrm{T}}(X+\lambda)) = \varphi(X) & \implies & \varphi \text{ is not injective.} \\ \forall D \in \mathcal{D},\ \exists K, \quad D = \varphi(K) & \implies & \varphi \text{ is surjective.} \end{array}\right.$$

In order to make this mapping bijective, one has to consider only centered data sets.

Definition 55 (Centered points)**.** *The set of centered points clouds is*

$$\bar{\mathcal{P}}_n \overset{\text{def.}}{=} \left\{X \in \mathbb{R}^{n\times p} \setminus X1 = 0\right\}.$$

Definition 56 (Centered inner products)**.** *The set of centered inner product matrices is*

$$\bar{\mathcal{K}}_n \overset{\text{def.}}{=} \left\{ X^{\mathrm{T}} X \setminus X \in \bar{\mathcal{P}}_n \right\}.$$

One can center a points cloud and a gram matrix using the following projectors

$$\left\{ \begin{array}{ccc} \mathbb{R}^{n\times p} & \longrightarrow & \bar{\mathcal{P}}_n \\ X & \longmapsto & XJ \end{array} \right. \quad \text{and} \quad \left\{ \begin{array}{ccc} \mathcal{K}_n & \longrightarrow & \bar{\mathcal{K}}_n \\ K & \longmapsto & JKJ \end{array} \right. .$$

where $J \overset{\text{def.}}{=} \mathrm{Id}_p - \mathbb{I}/p$.

The following theorem shows that centered inner product and distance matrices are in bijection.

Theorem 39 (Mapping between gram and distance)**.** *One has the following bijections*

$$\left\{ \begin{array}{l} \psi \circ \varphi = \mathrm{Id}_{\bar{\mathcal{K}}_n}, \\ \varphi \circ \psi = \mathrm{Id}_{\mathcal{D}_n}, \end{array} \right. \quad \textit{where} \quad \psi : \left\{ \begin{array}{ccc} \mathcal{D}_n & \longrightarrow & \bar{\mathcal{K}}_n \\ D & \longmapsto & -\frac{1}{2} JKJ. \end{array} \right.$$

3.2.4 Manifold Flattening

In order to flatten a manifold $\mathcal{M}$, we forget about the location of the points on the manifold but only consider pairwise geodesic distances using the metric $d_{\mathcal{M}}$. This gives an input geodesic distance matrix $\tilde{D} = (d_{\mathcal{M}}(\tilde{x}_i, \tilde{x}_j)^2)_{i,j} \in \mathcal{D}^+$, where $\tilde{x}_i \in \mathcal{M}$.

The flattening corresponds to finding $X = (x_i)_{i=1}^p \in \mathbb{R}^{n\times p}$ such that

$$\varphi(X^{\mathrm{T}} X) \overset{\text{def.}}{=} D \approx \tilde{D}.$$

The underlying idea is to find an euclidean layout of points that respects the geometry of the manifold (the pairwise distances). The input matrix $\tilde{D} \in \mathcal{D}^+$ is not euclidean. To find the layout, one would ideally like to solve

$$\min_{D \in \mathcal{D}_n} \|D - \tilde{D}\|.$$

This is a difficult optimization task, and in order to simplify it, this optimization can be approximated by replacing the distance matrices by gram matrices as follow

$$\min_{K \in \mathcal{K}_n} \|K - \psi(\tilde{D})\|.$$

This optimization has a simple spectral solution, that has been exploited in the Isomap algorithm, introduced in [**?**] (listing **??**). This corresponds to the classical multidimensional scaling theory.

Theorem 40 (Isomap solution)**.** *One has* $K = U \Lambda_n U^{\mathrm{T}}$ *where* $\psi(\tilde{D}) = U^{\mathrm{T}} \Lambda U$ *where*

$$\Lambda = (\lambda_i)_i, \ |\lambda_1| \leqslant \ldots \leqslant |\lambda_p| \qquad \textit{and} \qquad \Lambda_n = \mathrm{diag}(\lambda_1, \ldots, \lambda_n, 0, \ldots, 0).$$

1. Compute geodesic distances $\tilde{D} = (d_{\mathcal{M}}(\tilde{x}_i, \tilde{x}_j)^2)_{i,j}$.
2. Compute centered kernel $K = -J\tilde{D}J/2$ with $J = \mathrm{Id} - \mathbb{I}/p$.
3. Compute spectral decomposition $K = U\Lambda U^{\mathrm{T}}$, $U = (u_j)_j \in \mathbb{R}^{p\times p}$.
4. Embedding locations are

$$X = \sqrt{\Lambda_n} {U_n}^{\mathrm{T}} = \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})(u_1, \ldots, u_n)^{\mathrm{T}} \in \mathbb{R}^{n\times p}.$$

Table 5: Isomap algorithm.

Remark 10 (Equivalence PCA/Isomap)*.* If D is euclidean, i.e. $D = \varphi(X^{\mathrm{T}} X) \in \mathcal{D}$, then isomap is equivalent to PCA.

A flattening is a mapping $f = (f_1, f_2) \to \mathbb{R}^2$. Figure 1.11, right, shows an example of mesh flattening using Isomap. Such a flattening has been used to perform texture mapping [**?**]. Figure 1.11 also compares this flattening method with the Laplacian-based methods introduced in section 1.3.8.

– *Laplacian eigenmaps* enforces local smoothness through

$$f_i = \underset{\|f\|=1}{\operatorname{argmin}} \|Gf\|.$$

In this case, (f_1, f_2) are eigenvectors (#2,#3) of $L = G^{\mathrm{T}}G$.

– *Isomap* enforces global constraints through

$$\|f(x) - f(y)\| \approx d_{\mathcal{M}}(x, y).$$

In this case, (f_1, f_2) are eigenvectors (#1,#2) of $-J(d_{\mathcal{M}(x_i,x_j)})_{ij}J$.

In laplacian methods, the matrix to process is sparse, which eases computations but may leads to poor global control over the flattening.

3.2.5 Bending Invariants

Instead of performing a reduction in dimension (flattening) one can use the Isomap embedding in order to obtain a representation of the surface that is invariant under geodesic isometries, as introduced by [**?**].

Definition 57 (Bending invariants)**.** *Given a surface $\mathcal{M}$, the bending invariant $I_{\mathcal{M}}(x) \in \mathbb{R}^3$ is the output of the Isomap algorithm.*

A geodesic isometry is formally a mapping $\psi : \mathcal{M} \to \mathcal{M}'$ such that

$$\forall\,(x, y) \in \mathcal{M}^2, \quad d_{\mathcal{M}}(x, y) = d_{\mathcal{M}'}(\psi(x), \psi(y)).$$

The main properties of bending invariants is the following theorem.

Theorem 41 (Bending invariance)**.** *Up to rigid motion, $I_{\mathcal{M}}$ is invariant to geodesic isometries:*

$$I_{\mathcal{M}}(x) = v + UI_{\mathcal{M}'}(\psi(x)) \quad \textit{where} \quad U \in \mathcal{O}(3) \quad \textit{and} \quad v \in \mathbb{R}^3.$$

Figure **??** shows some examples of bending invariants. These bending invariants can be used to perform shape recognition with isometric invariance, which is a useful property in face recognition for instance, see [3].

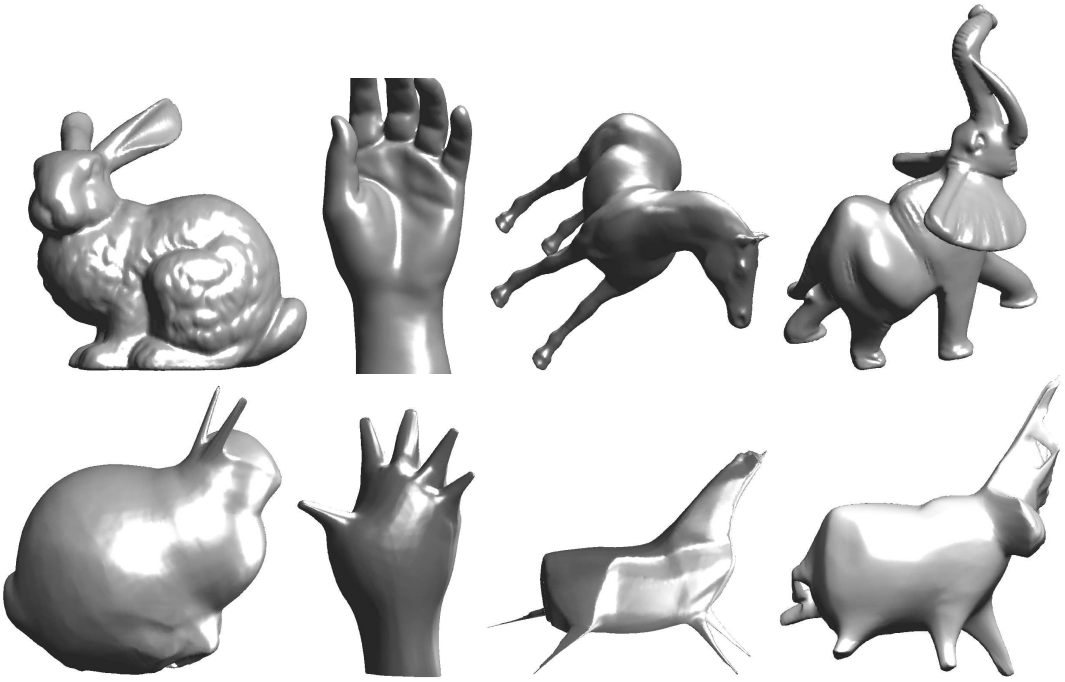


Figure 3.1: *Top: original surfaces, bottom: examples of bending invariants.*

4 Multiresolution Mesh Processing

This chapter shows how computations on a mesh can be performed in a multiscale manner, by considering meshes of increasing resolutions. This leads to the notion of subdivision surfaces and wavelet transform, which are two different tools to interpolates and decompose functions on meshes. Both methods rely on a special kind of meshes whose triangulations can be obtained by applying a regular refinement rule.

4.1 Semi-regular Meshes

4.1.1 Nested Multiscale Grids.

In order to perform multiscale mesh processing, one needs to pack the vertices V of a topological mesh $M = (V, E, F)$ in sets of increasing resolution. As explained in section 1.1.2, it is important to remember that this construction is purely combinatorial, in that no geometrical information (such as actual positions of the vertices in $\mathbb{R}^3$) is required to build the set of multi-resolution meshes. In fact these multiscale grids can be used to actually process the geometrical realization $\mathcal{M}$ of the mesh M as three real valued functions (the three coordinates of the points).

We thus consider a set of nested indexes

$$V_0 \subset V_{-1} \subset \ldots \subset V_L = V$$

which are split according to

$$V_j = V_{j+1} \cup H_{j+1}.$$

Next section describes how to actually compute this set of nested grids using a triangular split, but most of the mathematical tools are in fact valid for arbitrary set of indices, as long as they are embedded in one each other through scales.

For mesh processing, an index $\ell \in V_j$ corresponds to a vertex $x_\ell \in \mathcal{V} \subset \mathbb{R}^3$. The signals to be processed are vectors $f \in \mathbb{R}^n$ of size $n = |V_L|$ defined on the grid V_L. We sometimes write $f \in \ell^2(V_L)$ instead of $f \in \mathbb{R}^n$ to emphasis the domain on which f is indexed. This chapter describes transforms for signals $f \in \ell^2(V_L)$ sampled on the finest grid V_L.

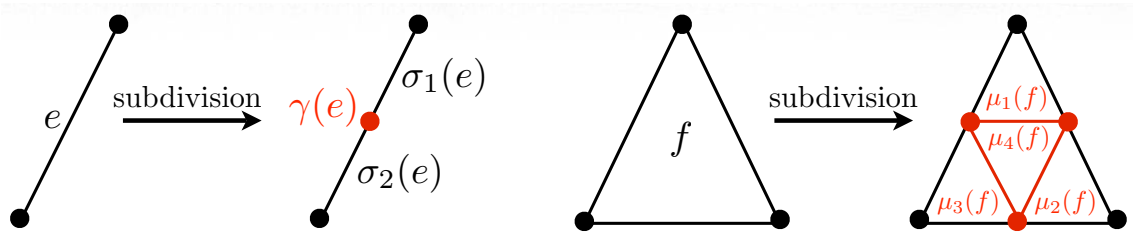


Figure 4.1: *Edge-splitting subdivision.*

4.1.2 Semi-regular Triangulation.

The combinatorial structure of a triangular mesh is defined in section 1.1.2. This chapter considers only a certain class of meshes $M = (V, E, F)$ that can be obtained by a regular split of faces,

starting from an initial coarse triangulation. This splitting leads to a set of multiresolution meshes $M_j = (V_j, E_j, F_j)$ for $J \leqslant j \leqslant 0$, where the full mesh is $M_J = M$.

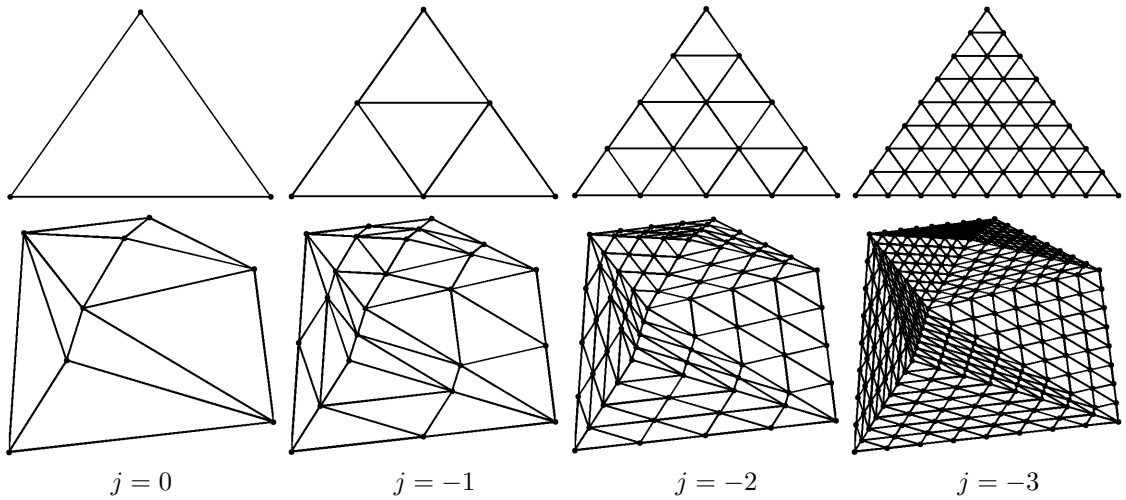


Figure 4.2: *Regular subdivision 1:4 of a single triangle. Regular subdivision of a planar triangulation M_0.*

Starting from this coarse triangulation, one defines by subdivision a multiscale triangulation $(V_j, E_j, F_j)_{L \leqslant j \leqslant 0}$ where

– For each edge $e \in E_j$, a central index $\gamma(e) \in V_{j-1}$ is added to the vertices

$$V_{j-1} = V_j \cup \{\gamma(e) \setminus e \in E_j\}.$$

– Each edge is subdivided into two finer edges

$$\forall\, e = (a, b) \in E_j, \quad \sigma_1(e) = (a, \gamma(e)) \quad \text{and} \quad \sigma_2(e) = (b, \gamma(e)).$$

The subdivided set of edges is then

$$E_{j-1} = \{\sigma_i(e) \setminus i = 1, 2 \quad \text{and} \quad e \in E_j\}.$$

– Each face $f = (a, b, c) \in F_j$ is subdivided into four faces

$$\begin{cases} \mu_1(f) = (a, \gamma(a, b), \gamma(a, c)), \ \mu_2(f) = (b, \gamma(b, a), \gamma(b, c)), \\ \mu_3(f) = (c, \gamma(c, a), \gamma(c, b)), \ \mu_4(f) = (\gamma(a, b), \gamma(b, c), \gamma(c, a)). \end{cases}$$

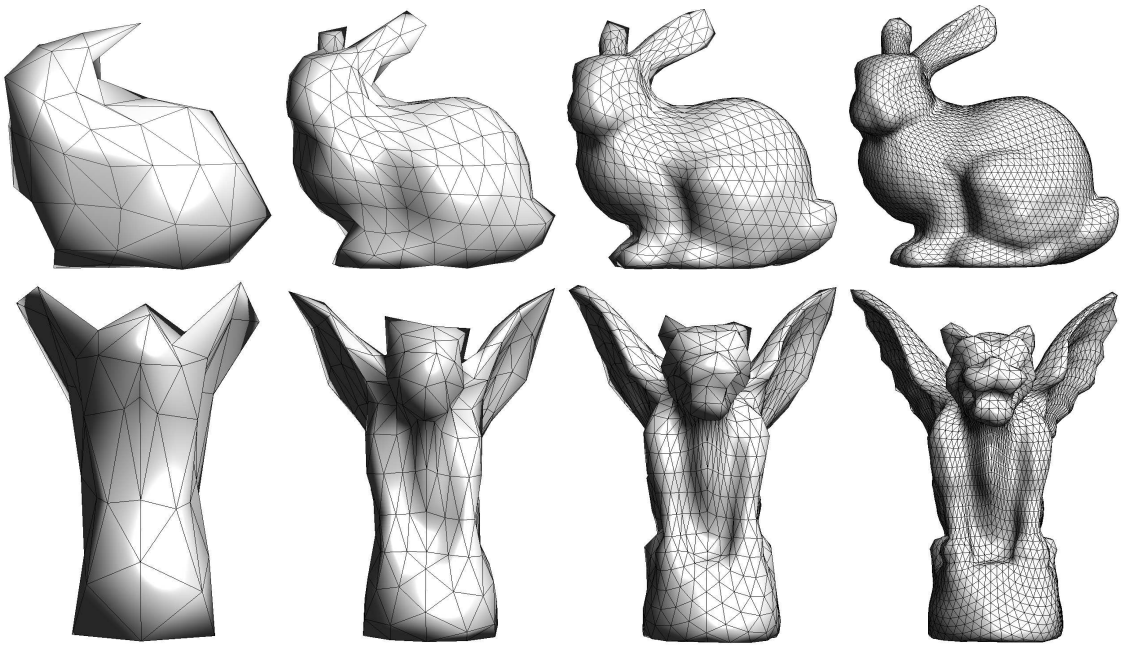
The subdivided set of faces is then

$$F_{j-1} = \{\mu_i(f) \setminus i = 1, 2, 3, 4 \quad \text{and} \quad f \in F_j\}.$$

Figure 3.1 shows the notations related to the subdivision process. Figure 3.2 shows an example of recursive splitting of a triangle and a coarse triangulation. Figure 3.3 shows examples of semi-regular triangulation using a geometric realization (position of the vertices) to create a 3D surface.

The set of vertices can be classified as

– **Regular vertices** are those who belong neither to the coarse mesh V_0 nor to a boundary of a mesh M_j. These vertices have always 6 neighbors.
– **Extraordinary vertices** are the initial vertices of V_0. They exhibit arbitrary connectivity.
– **Boundary vertices** are those belonging to a mesh boundary. Boundary vertices not in V_0 always have 4 immediate neighbors.



$j = 0$ $j = -1$ $j = -2$ $j = -3$

Figure 4.3: *Examples of semi-regular meshes $(V_j)_j$ for increasing scale j (from left to right).*

Obviously not every meshes can be obtained from such a subdivision process. In practice, an arbitrary mesh, obtained from CAD design or range scanning usually does not have any multiscale structure. It is thus necessary to remesh it in order to modify the connectivity of the mesh. During this process, the position of the vertices in $\mathbb{R}^3$ is modified in order for the geometrical realization to stay close from the original piecewise linear surface. One can see [2] fur a survey of various semi-regular remeshing methods.

4.1.3 Spherical Geometry Images

Starting from some input surfaces $\mathcal{S} \subset \mathbb{R}^3$, one typically wants to compute a semi-regular meshes $(M_j)_{j \geqslant L}$ that approximate $\mathcal{S}$. In most case, the surface $\mathcal{S}$ is actually given as an arbitrary triangulated mesh and this process corresponds to a semi-regular remeshing. Many algorithm have been devised for surface remeshing and we describe here a method [31] that works for surfaces that have the topology of a sphere. It means that the surface has genus 0, without boundary and without handles.

This methods works by computing several intermediate surface-wise parameterization.

- *Spherical parameterization:* each points of the original triangulation of $\mathcal{S}$ is mapped onto the unit sphere. This create a bijective parameterization

$$\varphi_S : S^2 \to \mathcal{S}.$$

  This is a non-linear process that differ from the planar parameterization introduced in section 1.3.7. We do not give the details of such a process, but it requires minimizing the smoothness of the mapping φ_S^{-1} under the constraint that it maps points of $\mathcal{S}$ to unit length vectors (point on the sphere S^2). The algorithm is explained in details in [31].
- *Spherical-tetraedron flattening:* one flatten each quadrant (1/8) of the sphere in order to have a mapping

$$\varphi_T : \text{Octaedron} \to S^2.$$

  One can use for instance a mapping between spherical barycentric coordinate on each quadrant and Euclidean barycentric coordinates on each face of the octahedron.

– *Tetraedron unfolding:* One maps each equilateral face of the octaedron on a rectangular triangle that corresponds to 1/8th of the square $[0,1]^2$

$$\varphi_U : [0,1]^2 \to \text{Octaedron}.$$

– *Regular sampling:* the geometry image is obtained by regularly sampling the square on a uniform grid

$$x_\ell = \varphi_S \circ \varphi_T \circ \varphi_U(\ell/n) \quad \text{for} \quad \ell_i = 0, \ldots, n-1.$$

The mapping $\ell \mapsto x_\ell \in \mathbb{R}^3$ is the geometry image, which can be stored as a 3-channel (color) image.

From such a geometry image x_ℓ, one can easily compute a semi-regular mesh by simply performing a regular 1:4 subdivision of the octaedron. Figure 3.4 shows the steps of the construction of a geometry image, and the resulting semi-regular mesh.

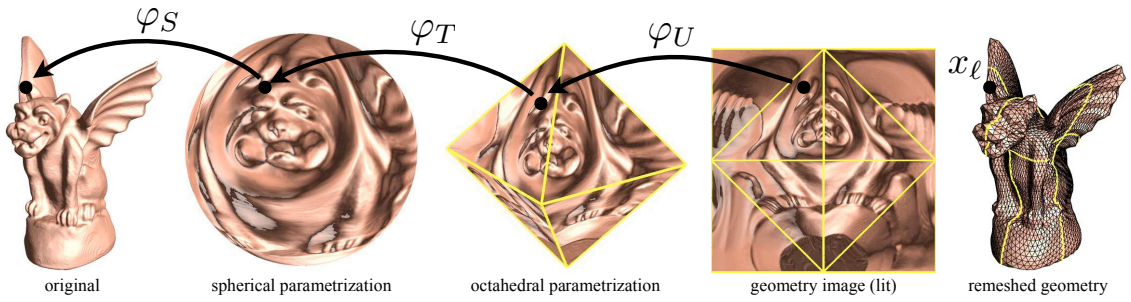


Figure 4.4: *Spherical geometry image construction, taken from [31].*

4.2 Subdivision Curves

Before getting into the detail of subdivision surfaces, we describe the subdivision process in the simpler setting of 1D signals. This leads to the construction of subdivision of 1D functions and subdivision curves.

In this 1D setting, the grid point indexes are dyadic sub-grids of $\mathbb{Z}$

$$\forall j \geqslant L, \quad V_j = \left\{ \ell 2^{j-L} \setminus 0 \leqslant \ell < s_0 2^{-j} \right\},$$

where $s_0 = |V_0|$ is the size of the initial vector f_0 to be subdivided.

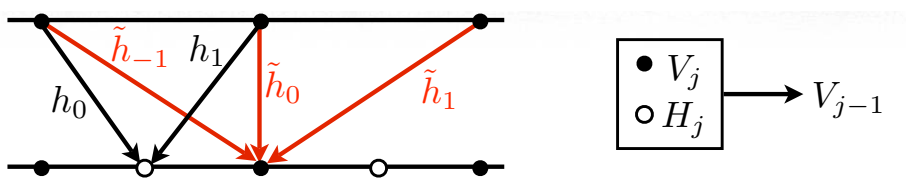


Figure 4.5: *1D subdivision scheme with filters h and $\tilde{h}$. The red curve represent the original signal f^0.*

Each subdivision steps computes, from a set $f_j(\ell) \in \ell^2(V_j)$ of coarse values, a refined vector $f_{j-1} \in \ell^2(V_{j-1})$ defined by

$$\begin{cases} \forall k \in H_j, \; f_{j-1}(k) = \sum_t f_j((k-1)/2 + t)h(t), \\ \forall \ell \in V_j, \; f_{j-1}(\ell) = \sum_t f_j(\ell + t)\tilde{h}(t). \end{cases}$$

where the set of weights h and $\tilde{h}$ acts as local averaging operators. This averaging should be corrected at the boundary, and we use here cyclic boundary conditions which identifies 0 and $s_0 2^{-j}$ in V_j. Figure 3.5 shows a graphical display of these averaging operators.

One can write this subdivision steps as convolution by introducing the global set of weights

$$g = [\ldots, \tilde{h}(-1), h(0), \tilde{h}(0), h(1), \tilde{h}(1), \ldots]$$

since one has

$$f_{j-1} = (f_j \uparrow 2) * g \quad \text{where} \quad a \uparrow 2 = [\ldots, 0, a(-1), 0, a(0), 0, a(1), 0, \ldots].$$

This corresponds to the traditional description of the wavelet low-pass filtering [24].

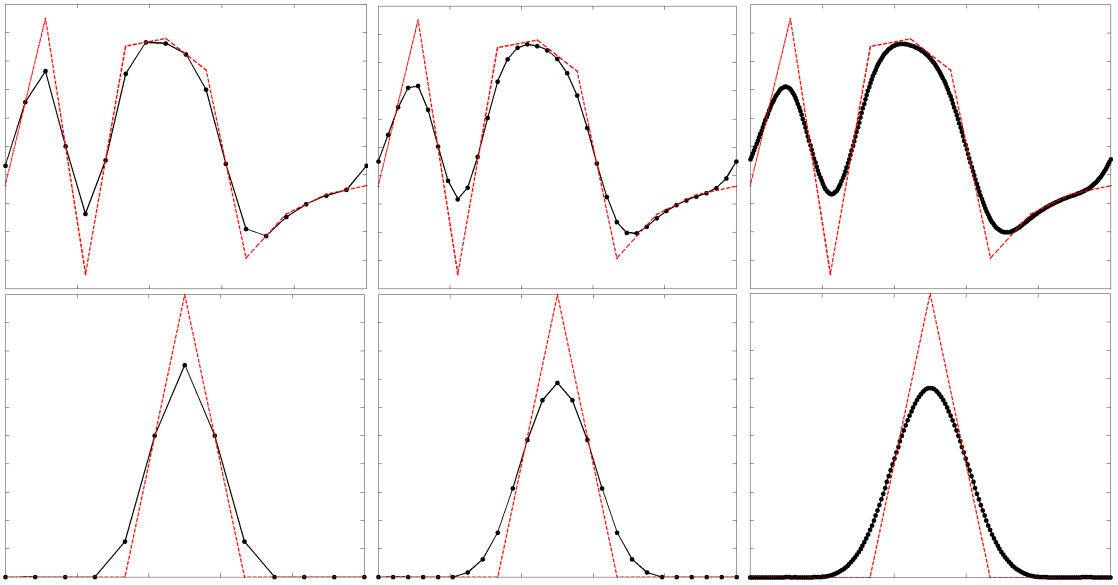


Figure 4.6: *1D subdivision of a signal. Bottom row shows the subdivision from an impulse signal, converging to the scaling function φ.*

Figure 3.6 shows several steps of subdivision, starting from an initial vector of size $|V_0| = 10$.

One can apply this subdivision of functions to a pair of signals

$$(X_0, Y_0) : V_0 \to \mathbb{R}^2$$

which is a control polygon composed of points located in the plane. The subdivision curve converges to the limiting curve

$$(X_j, Y_j) \stackrel{j \to -\infty}{\longrightarrow} (X(t), Y(t))_{t=0}^1 \subset \mathbb{R}^2.$$

An interesting property is that this curve is included in the convex hull of the control polygon

$$(X(t), Y(t))_t \subset \mathrm{Conv}(X_0, Y_0).$$

Figure 3.7 shows examples of subdivision curves.

4.3 Subdivision Surfaces

Subdivision schemes allows to compute a set of progressively refined vectors on a semi-regular mesh. More precisely, from an initial vector $f_0 \in \mathbb{R}^{|V_0|}$ defined on the coarse mesh M_0, local interpolation kernels computes iteratively vectors $f_j \in \mathbb{R}^{|V_j|}$ of finer resolution. When applied to 3 function $(f_0^i)_{i=1,2,3}$ defining the geometrical position of points in $\mathbb{R}^3$, this hierarchical construction defines a subdivision surface. These subdivisions surfaces are used extensively in computer aided geometry and computer graphics. One can see [8] for a survey of subdivision surfaces and their applications.

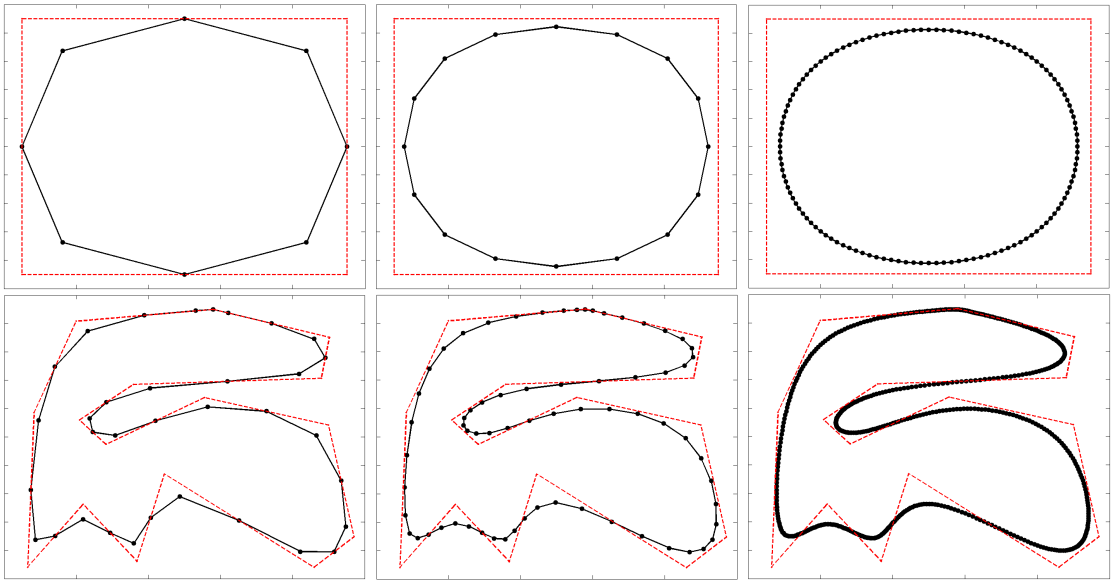


Figure 4.7: *Two examples of subdivision curves. The red curve represent the original curve* (X^0, Y^0).

4.3.1 Interpolation Operators

In order to refine a vector $f_j \in \mathbb{R}^{|V_j|}$ defined on the vertex V_j of the mesh M_j, one uses two interpolators

$$P_j : \ell^2(V_j) \longrightarrow \ell^2(H_j) \qquad \text{and} \qquad \tilde{P}_j : \ell^2(V_j) \longrightarrow \ell^2(V_j). \tag{4.1}$$

A new refined function $f_{j-1} \in \mathbb{R}^{|V_{j-1}|}$ defined on the vertices $V_{j-1} = V_j \cup H_j$ of M_{j-1} is defined by applying these two refinement operators:

$$\forall\, \ell \in V_{j-1}, \quad f_{j-1}(\ell) = \begin{cases} (P_j f_j)(\ell) & \text{if} \quad \ell \in V_j, \\ (\tilde{P}_j f_j)(\ell) & \text{if} \quad \ell \in H_j. \end{cases}$$

Since $V_j \subset V_{j-1}$, the operator $\tilde{P}_j$ only modify slightly the value at vertex in V_j. On the other hand, the operator P_j creates new value at the vertices of H_j that are inserted between V_j and V_{j-1}.

In practical applications, these interpolating operators are local, meaning that the value of $(P_j f_j)(\ell)$ and $(\tilde{P}_j f_j)(\ell)$ depends only on values $f_j(\ell')$ for $\ell' \in V_j$ being close to $\ell \in V_{j-1}$, typically in the 1-ring or 2-ring vertex neighborhood.

A particularly important setting for subdivision scheme is when one apply the subdivision steps in parallel to three vectors (X_j, Y_j, Z_j) starting from three initial vectors describing the position in 3D space of a coarse mesh M_0. This allows to defines finer and finer spacial localization for the vertex of the refined meshes M_j. Figure 3.9 shows an example of such a subdivision surface. In order for the resulting infinitely refined surface to have good properties such as being continuous and even smooth, one needs to design carefully the interpolation operators. Next section gives examples of such operators.

4.3.2 Some Classical Subdivision Stencils

In order to define the interpolation operators P_j and $\tilde{P}_j$ of equation (3.1), one needs to use a naming convention for the neighborhoods of vertices.

For a vertex $\ell \in V_j$, the one ring neighborhood V_ℓ has already been defined in equation (1.1). It is the set of vertices adjacent to ℓ. In a regular point (that does not belongs to V_0 and not on a boundary of the mesh), its size is $|V_\ell| = 6$ since a point has 6 neighbors. This 1-ring is used to define $\tilde{P}_j$.

For a vertex $k \in H_j \subset V_{j-1}$, the butterfly neighborhood is a set of vertices in V_j close to k. This neighborhood is used to define P_j. The two immediate neighbors are

$$(v_k^1, v_k^2) \stackrel{\text{def.}}{=} \{v \in V_j \setminus (v,k) \in E_{j-1}\}.$$

Two other vertices (w_k^1, w_k^2) are defined using the two faces adjacent to edge $(v_k^2, v_k^2) \in E_j$

$$f_k^1 = (v_k^1, v_k^2, w_k^1) \in F_j \quad \text{and} \quad f_k^2 = (v_k^1, v_k^2, w_k^2) \in F_j.$$

For edges E_j on the boundary of M_j, one one face is available, in which case we implicitly assume that $f_1 = f_2$ (reflecting boundary conditions). The four last vertices are defined using faces adjacent to f_1 and f_2:

$$\forall\, i,j = 1,2, \quad f_k^{i,j} \stackrel{\text{def.}}{=} (z_k^{i,j}, v_k^j, w_k^j) \in F_j \quad \text{with} \quad f_k^{i,j} \neq f_j.$$

Once again, reflecting boundary condition are applied for faces on the boundary of the mesh. The butterfly neighborhood is depicted on figure 3.8.

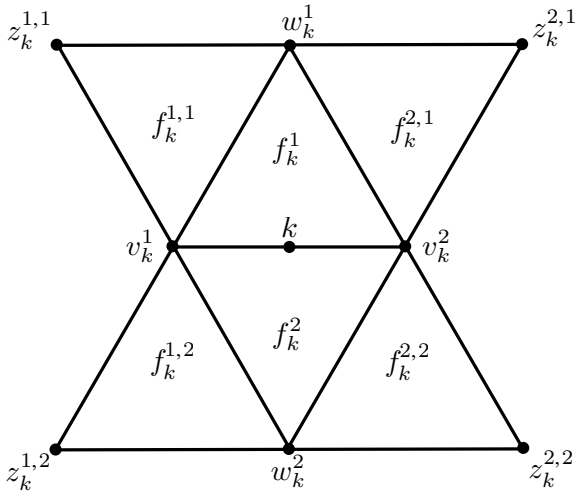


Figure 4.8: *The butterfly neighborhood of a vertex $k \in H_j$.*

Linear Interpolating Scheme The simplest subdivision rule compute values along edge mide point using a simple linear interpolation as follow

$$\begin{cases} \forall\, k \in H_j, & (P_j f_j)(k) = \frac{1}{2}(f(v_k^1) + f(v_k^2)), \\ \forall\, \ell \in V_j, & (\tilde{P}_j f_j)(\ell) = f_j(\ell). \end{cases} \tag{4.2}$$

Since $\tilde{P}_j$ is the identity operator, this scheme is called interpolating. It means that value of f_0 on points of the coarse triangulation are kept during iteration of the subdivision.

Butterfly Interpolating Scheme The linear scheme creates function that are piecewise linear on each face of the coarse triangulation F_0. In order to create smooth surface, one needs to use more points in the butterfly neighborhood as follow

$$\begin{cases} \forall\, k \in H_j, & (P_j f_j)(k) = \frac{1}{2}\sum_{i=1}^2 f(v_k^i) + \frac{1}{8}\sum_{i=1}^2 f(w_k^i) - \frac{1}{16}\sum_{i,j=1}^2 f(z_k^{i,j}), \\ \forall\, \ell \in V_j, & (\tilde{P}_j f_j)(\ell) = f_j(\ell). \end{cases} \tag{4.3}$$

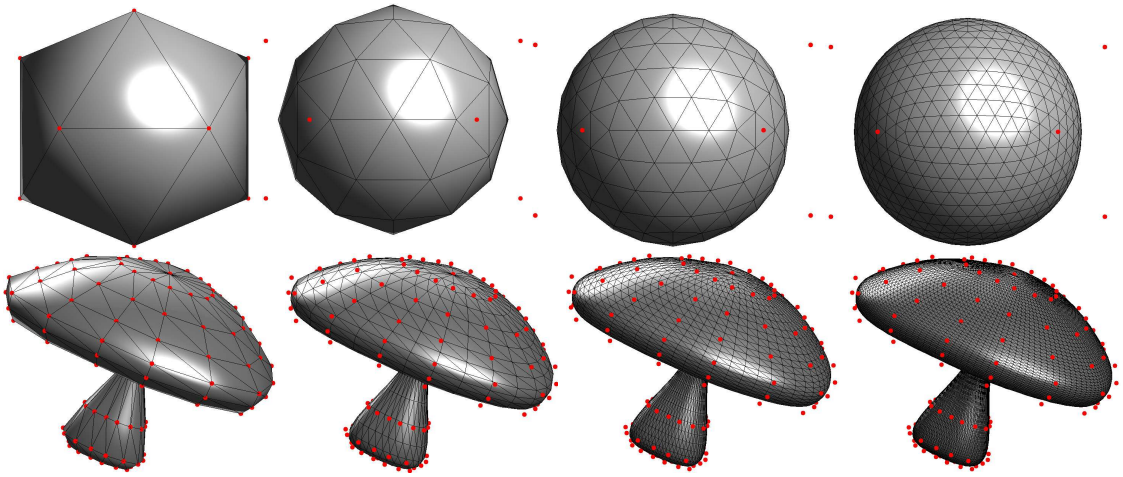
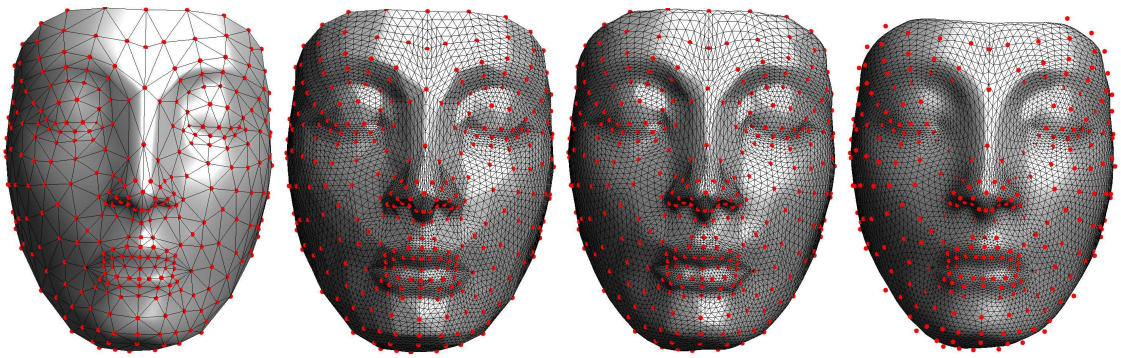


Figure 4.9: *Examples of iterative subdivision using Loop scheme. The points* (X_0, Y_0, Z_0) *of the initial coarse mesh* M_0 *are shown in red.*



Original Linear Butterfly Loop

Figure 4.10: *Examples of subdivision schemes. The points* (X_0, Y_0, Z_0) *of the initial coarse mesh* M_0 *are shown in red. Since the linear and butterfly scheme are interpolating, these points actually belongs to the limiting surface.*

Loop Approximating Scheme In order to gain flexibility in the subdivision design, one can also modify points in V_j during the iterations. This means that $\tilde{P}_j$ is not any more the identity, and that all the values will evolves during the iterations. The question of wether these iterated modification actually converge to a limit value is studied in the next section.

The Loop subdivision rule is defined as

$$\begin{cases} \forall\, k \in H_j, & (P_j f_j)(k) = \frac{3}{8}\sum_{i=1}^{2} f(v_k^i) + \frac{1}{8}\sum_{i=1}^{2} f(w_k^i), \\ \forall\, \ell \in V_j, & (\tilde{P}_j f_j)(\ell) = (1 - |V_\ell|\beta_{|V_\ell|}) f_j(\ell) + \beta_{|V_\ell|}\sum_{\ell' \in V_\ell} f_j(\ell'). \end{cases}$$

where the weights depends on the number of neighbors and are defined as

$$\beta_m \overset{\text{def.}}{=} \frac{1}{m}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos(2\pi/m)\right)^2\right).$$

Other schemes. It is possible to define subdivision schemes using rules that do not involve a regular 1:4 splitting of each coarse face. For instance, in dual schemes such as the one depicted in figure 3.11, the faces of F_j are not included in F_{j-1} but only in F_{j-2}.

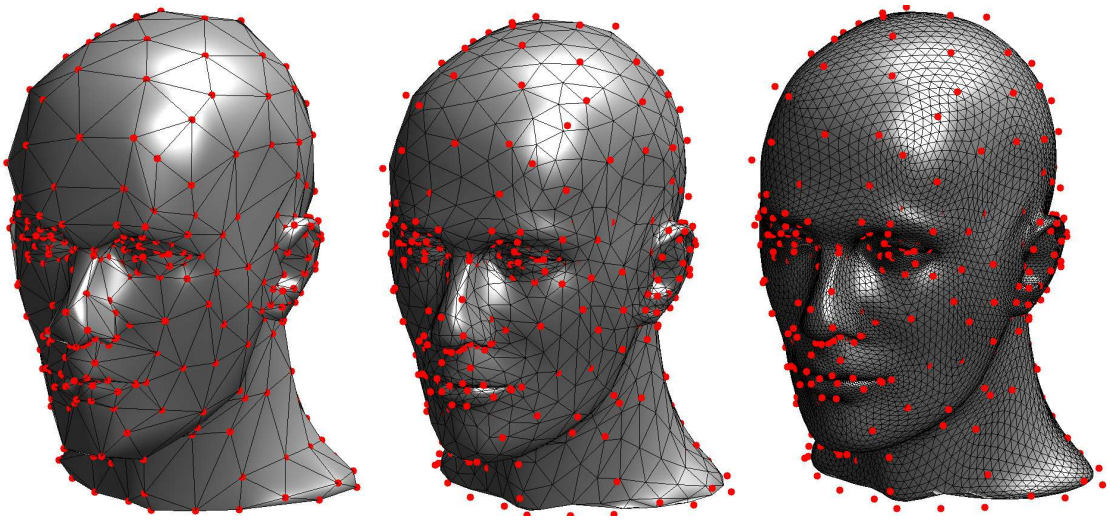


Figure 4.11: *Surface after 0, 1 and 3 step of $\sqrt{3}$ subdivision [20].*

4.3.3 Invariant Neighborhoods

In order to study the convergence of subdivision schemes, one needs to consider independently each vertex $x \in V_{j_0(x)}$, where $j_0(x)$ is the coarser scale at which x appears

$$j_0(x) = \max \{j \setminus x \in V_j\}.$$

Original vertices satisfy $j_0(x) = 0$ and are the only one (except boundary vertices) that have a non-regular connectivity.

The vertex x belongs to the mesh $M_{j_0(x)}$ which is going to be refined through scales $j < j_0(x)$. In order to analyze this refinement, one needs to define an invariant neighborhood $V_j^x \subset V_j$ of x for each scale $j \leqslant j_0(x)$. These neighborhood are the set of points that are required to compute the operators P_j and $\tilde{P}_j$. More precisely, given a vector $f \in \ell^2(V_{j-1})$, the neighborhoods are required to satisfy

$$\begin{cases} \forall\, \ell \in V_{j-1}^x \cap V_j, & (\tilde{P}_j f)(\ell) \text{ depends only on } V_j^x \\ \forall\, k \in V_{j-1}^x \cap H_j, & (P_j f)(k) \text{ depends only on } V_j^x. \end{cases}$$

We further impose that all the invariant neighborhoods have the same size

$$\forall\, j \leqslant j_0(x), \quad \#V_j^x = m_x.$$

Figure 3.12 shows an example of invariant neighborhood which corresponds to the 2-ring $V_\ell^{(2)}$, as defined in (1.2).

Thanks to the invariance of these neighborhood systems, one can restrict the predictors around x and define

$$P_j^x : V_j^x \longrightarrow V_{j-1}^x \cap V_j \qquad \text{and} \qquad \tilde{P}_j^x : V_j^x \longrightarrow V_{j-1}^x \cap H_j.$$

The subdivision matrix $S_j^x \in \mathbb{R}^{m_x \times m_x}$ is then defined as matrix of the following mapping

$$(\tilde{P}_j^x, P_j^x) : V_x^j \longrightarrow V_x^{j-1}.$$

All the subdivision schemes studied in this chapter are invariant, meaning that the subdivision rule does not change through the scales j. This impose that the subdivision matrices are constant $S_j^x = S^x$. In fact, in all the examples given in the previous section, they only depends on the number $|V_x|$ of neighbors in the one ring of x.

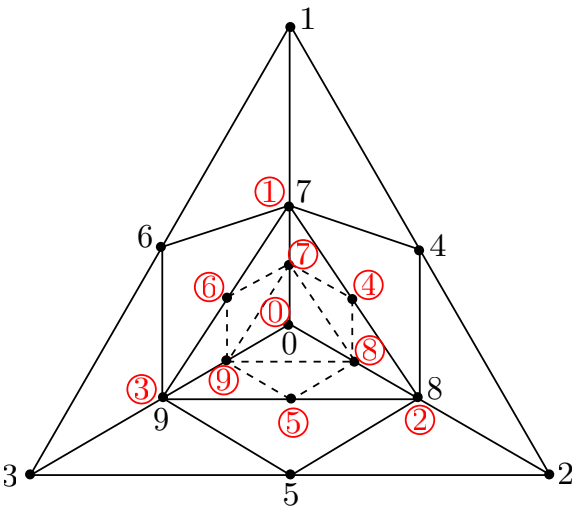


Figure 4.12: *Invariant neighborhood V_j^x and V_{j-1}^x (indexing with red circles) of the Loop subdivision scheme for a vertex of valence $|V_\ell| = 0$. The number in $\{0, \ldots, 9\}$ refers to the numbering of the vertices in V_j^x and V_{j-1}^x*

4.3.4 Convergence of Subdivisions

The value at $x \in V_{j_0(x)}$ of a function $f_j \in \ell^2(V_j)$ obtained by subdividing at scale $j \leqslant j_0(x)$ an initial vector $f_0 \in \ell^2(V_0)$ can be computed as

$$f_j(x) = \left(S^x f_{j+1}^x\right)(x) = \left((S^x)^{j_0(x)-j} f_{j_0(x)}^x\right)(x),$$

where the vector $f_i^x \in \mathbb{R}^{m_x}$ is the restriction of f_i to the set V_j^x.

In order to analyze the limiting function resulting from an infinite number of subdivision, one can use the eigen vector decomposition of the matrix S^x

$$S^x = \tilde{\Phi} V \Lambda \Phi^{\mathrm{T}} \quad \text{where} \quad \begin{cases} \Phi^{\mathrm{T}} = \tilde{\Phi}^{-1}, \\ \Lambda = \mathrm{diag}(\lambda_i), \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_{m_x}. \end{cases}$$

Since the subdivision matrix S_x is not symmetric, some of the eigenvalues might be complex, and we shall ignore this difficulty here. The fact that P_j and $\tilde{P}_j$ are predictor implies that the subdivision matrix has to satisfy $S^x 1 = 1$, meaning that $\tilde{\varphi}_1 = 1$ is an eigenvector associated to the eigenvalue 1. In the following we further makes the following assumption

$$1 = \lambda_1 < \lambda \stackrel{\text{def.}}{=} \lambda_2 = \lambda_3 < \lambda_4. \tag{4.4}$$

This hypothesis is satisfied by all the subdivision rules introduced in the previous section.

If one write $\Phi = (\varphi_i)_{i=1}^m$ and $\Phi = (\varphi_i)_{i=1}^m$, one has the following decomposition of a vector $f \in \mathbb{R}^{m_x}$

$$f = \sum_{i=1}^{m_x} \langle f, \varphi_i \rangle \tilde{\varphi}_i \quad \text{and} \quad (S^x)^k(x) = \sum_{i=1}^{m_x} \lambda_i^k \langle f, \varphi_i \rangle \tilde{\varphi}_i.$$

One thus has the following asymptotic expansion

$$\frac{1}{\lambda^k}\left(f - \langle f, \varphi_1 \rangle 1\right) = \langle f, \varphi_2 \rangle \tilde{\varphi}_2 + \langle f, \varphi_3 \rangle \tilde{\varphi}_3 + o(1). \tag{4.5}$$

This expression describes the asymptotic behavior of the subdivision scheme at zero order (position) and first order (tangents).

Theorem 42 (Convergence of the subdivision scheme)**.** *If the subdivision matrix S^x of a point x satisfies* (3.4) *then the subdivision process converges at x to the value*

$$f^j(x) \stackrel{j \to -\infty}{\longrightarrow} \langle f_{j_0(x)}^x, \varphi_1 \rangle.$$

The smoothness of the resulting function is more difficult to analyze. A particularly important setting is when one computes the subdivision of 3 function $p_0 = (X_0, Y_0, Z_0) \in \ell^2(V_0)^3$ corresponding to the position in $\mathbb{R}^3$ (geometrical realization) of a coarse mesh M_0. In this case, the subdivided functions $p_j = (X_j, Y_j, Z_j)$ gives refined 3D meshes that converge uniforlmy to a continuous surfaces

$$p(x) = (X(x), Y(x), Z(x)) = (\langle X^x_{j_0}, \varphi_1\rangle, \langle Y^x_{j_0}, \varphi_1\rangle, \langle Z^x_{j_0}, \varphi_1\rangle).$$

Condition (3.4) nearly implies that the resulting surface is smooth. Indeed, the asymptotic expansion (3.5) shows that for a point x' near x in the subdivision domain, the differential vector can be well approximated as a projection on a 2D plane

$$p(x) - p(x') + o(1) \in \mathrm{Span}(\tau^x_2, \tau^x_3) \quad \text{where} \quad \tau^i(x) \overset{\text{def.}}{=} (\langle X^x_{j_0}, \varphi_i\rangle, \langle Y^x_{j_0}, \varphi_i\rangle, \langle Z^x_{j_0}, \varphi_i\rangle).$$

If the vectors τ^x_2 and τ^x_3 are linearly independent, they form a basis of the tangent plane at $p(x)$.

Example of the Loop subdivision. For the Loop interpolation operators defined in equation (3.3.2), the invariant neighborhood V^x_j correspond to the 2-ring of x in the triangulation G_j, as shown in figure 3.12. For a vertex with k neighbors, $|V_x| = k$, the size of these invariant neighborhood is $m_x = 3k + 1$. A particular neighboring for $k = 3$ is depicted in figure 3.12, together with an indexing in $\{0, \ldots, 3k = 9\}$ of the points in V^x_j and V^x_{j-1}. For this indexing, the subdivision matrix reads

$$\begin{pmatrix}
7 & & & & & & & 3 & 3 & 3 \\
1 & 1 & & & 1 & & 1 & 10 & 1 & 1 \\
1 & & 1 & & 1 & 1 & & 1 & 10 & 1 \\
1 & & & 1 & & 1 & 1 & 1 & 1 & 10 \\
1 & & & & 1 & & & 3 & 3 & \\
1 & & & & & 1 & & & 3 & 3 \\
1 & & & & & & 1 & 3 & & 3 \\
1 & & & & & & & 3 & 1 & 1 \\
1 & & & & & & & 1 & 3 & 1 \\
1 & & & & & & & 2 & 1 & 3
\end{pmatrix}$$

where the 0's have been omitted and where the rows should be rescaled to sum to 1. The eigenvalues of this matrix satisfy $\lambda_1 = 1$ and $\lambda_2 = \lambda_3 = 1/3 > \lambda_4$.

4.4 Wavelets on Meshes

4.4.1 Multiscale Biorthogonal Bases on Meshes

The transforms considered in this section are multiscale and indexed by the set of nested grids $(V_j)_{L<j\leqslant J}$. This corresponds to computing a set of coefficients $(d_j)_{L<j\leqslant J} \cup f_J$ from an initial input signal f. These coefficients corresponds to inner products with basis vectors

$$\begin{cases} d_j \in \ell^2(H_j) & \text{where} \quad \forall\, k \in H_j,\ d_j(k) = \langle f, \psi_{j,k}\rangle, \\ f_J \in \ell^2(V_J) & \text{where} \quad \forall\, \ell \in V_J,\ f_J(\ell) = \langle f, \varphi_{J,\ell}\rangle. \end{cases}$$

By analogy with the wavelet setting, the vectors $\psi_{j,k} \in \mathbb{R}^n$ corresponds to primal wavelets and are intended to capture the details present in the signal f at a scale j, whereas the scaling vectors $\varphi_{J,k} \in \mathbb{R}^n$ capture the missing coarse approximation of f at scale J. This decomposition is stopped at any coarse scale $L < J \leqslant 0$.

In order to reconstruct the function f from this set of transformed coefficients, one needs to use a set of bi-orthogonal basis vectors

$$f = \sum_{L<j\leqslant J, k\in H_j} d_j(k)\tilde{\psi}_{j,k} + \sum_{\ell\in V_J} f_J(\ell)\tilde{\varphi}_{J,\ell}.$$

If this reconstruction formula holds for any scale $L < J \leqslant 0$, the set of vectors

$$(\psi_{j,k}, \varphi_{j,\ell})_{k\in H_j, \ell\in V_j}^{L<j\leqslant 0} \qquad \text{and} \qquad (\tilde{\psi}_{j,k}, \tilde{\varphi}_{j,\ell})_{k\in H_j, \ell\in V_j}^{L<j\leqslant 0}, \tag{4.6}$$

is said to be a pair of primal and dual multiscale bases (together with their scaling functions).

The following paragraph shows how one can modify such a pair of multiscale bases while still maintaining the biorthogonality property. This lifting process is useful to design multiscale bases with various properties on complicated domains.

4.4.2 The Lifting Scheme

The lifting scheme is a construction of multiscale biorthogonal bases introduced by Sweldens [36, 37]. It extends the traditional construction of wavelets in two main directions:

- As explained in [9], it allows to implement already existing filter banks more efficiency by splitting the computation into elementary blocks. This computational gain is described at the end of the section together with the factorization of wavelets into lifting steps.
- It allows to define multiscale transforms over domains that are not translation invariant. This section gives two examples of such transforms: a non-separable 2D wavelet transform and wavelets on triangulated meshes.

In order to build wavelets on triangulation, one can specialize the lifting scheme to a particular setting where only two lifting steps are applied.

Forward lifting scheme. The forward algorithm performs the transform

$$(f_{j-1}(\ell))_{\ell\in V_{j-1}} \quad \longrightarrow \quad (d_j(k))_{k\in H_j} \cup (f_j(\ell))_{\ell\in V_j}$$

by applying the following steps

- *Splitting:* this corresponds selecting the coefficient of $f_{j-1}(\ell)$ that are in V_j or in H_j

$$(f_{j-1}(\ell))_{\ell\in V_{j-1}} = (f_j(\ell))_{\ell\in V_j} \cup (f_j(\ell))_{\ell\in H_j}.$$

These two sets of coefficients are treated differently in the two remaining steps of the transform.

- *Predict step:* creates wavelets coefficients d_j by computing local differences between each coefficient in V_j and its neighbors in H_j

$$\forall\, k \in H_j, \quad d_j(k) = f_{j-1}(k) - \sum_{\ell\in V_j} p_j(k,\ell) f_{j-1}(\ell).$$

The coefficients $p_j(k,\ell)$ are weights that determine the predict operator

$$P_j : \left\{ \begin{array}{ccc} \ell^2(V_j) & \longrightarrow & \ell^2(H_j) \\ g & \longmapsto & h = P_j g \end{array} \right. \qquad \text{where} \quad h(k) = \sum_{k\in H_j} p_j(k,\ell) g(\ell).$$

- *Update step:* enhance the properties of each remaining low pass coefficients $f_{j-1}(\ell)$ for $\ell \in V_j$ by pooling locally the wavelets coefficients $d_j(k)$ for k around ℓ

$$\forall\, \ell \in V_j, \quad f_j(\ell) = f_{j-1}(\ell) + \sum_{k\in H_j} u_j(\ell,k) d_j(k).$$

The coefficients $u_j(\ell,k)$ are weights that determine the update operator

$$U_j : \left\{ \begin{array}{ccc} \ell^2(H_j) & \longrightarrow & \ell^2(V_j) \\ h & \longmapsto & g = U_j h \end{array} \right. \qquad \text{where} \quad g(\ell) = \sum_{k\in H_j} u_j(\ell,k) h(k).$$

Figure 3.13, top row, shows the block diagram associated to this forward lifting wavelet transform.

The iterations of the forward lifting transform can also be written in vector and operator format

$$\begin{cases} d_j = f_{j-1}^{H_j} - P_j f_{j-1}^{V_j}, \\ f_j = f_{j-1}^{V_j} + U_j d_j = (\mathrm{Id}_{V_j} - U_j P_j) f_{j-1}^{V_j} + U_j f_{j-1}^{H_j}, \end{cases}$$

where g^A is the restriction of some vector g to the set A.

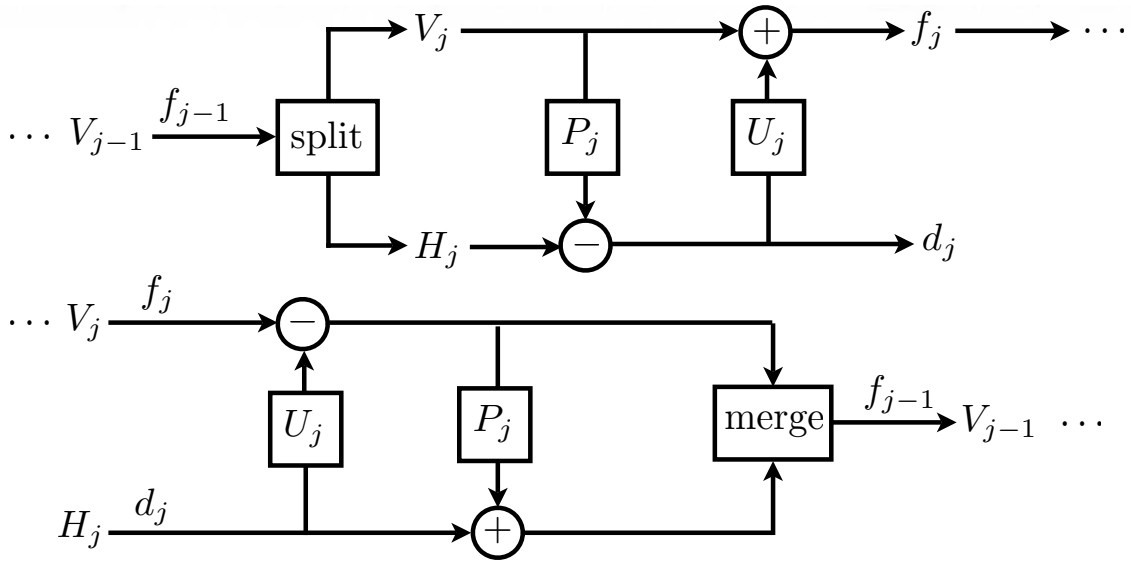


Figure 4.13: *Block diagrams for the forward and backward lifting scheme.*

Backward lifting scheme. The backward transform algorithm does the reverse computation

$$(d_j(k))_{k \in H_j} \cup (f_j(\ell))_{\ell \in V_j} \quad \longrightarrow \quad (f_{j-1}(\ell))_{\ell \in V_{j-1}}$$

One of the main feature of the lifting scheme is that this is achieved by simply reversing the order of the lifting steps and interchanging +/- signs.

– *Inverse update step:*

$$\forall\, \ell \in V_j, \quad f_{j-1}(\ell) = f_j(\ell) - \sum_{k \in H_j} u_j(\ell, k) d_j(k).$$

– *Inverse predict step:*

$$\forall\, k \in H_j, \quad f_{j-1}(k) = d_j(k) + \sum_{\ell \in V_j} p_j(k, \ell) f_{j-1}(\ell).$$

– *Merging:* makes the union of the coefficients computed in the two previous steps

$$(f_{j-1}(\ell))_{\ell \in V_{j-1}} = (f_j(\ell))_{\ell \in V_j} \cup (f_j(\ell))_{\ell \in H_j}.$$

Figure 3.13, top row, shows the block diagram associated to this backward lifting wavelet transform.

The lifting scheme is more general than the algorithm described in this section since several passes of predict/update steps can be applied to further enhance the properties of the resulting transform. However, the steps beyond the two initial ones are difficult to analyze, except in the notable exception of points sampled evenly on a 1D axes, where a factorization algorithm [9] allows to recover traditional wavelet filters.

4.4.3 Imposing vanishing moments.

The operator P_j is called a predictor since the values of $P_j f_{j-1}^{V_j}$ should typically be close to $f_{j-1}^{H_j}$ for the wavelet coefficients d_j to be small. Such predictors have already been constructed in equations (3.2), (3.3) and (3.3.2).

The operator U_j is called an update operator since the additional term $U_j d_j$ should enhance the properties of $f_{j-1}^{V_j}$. This update steps does not appears in the theory of subdivision surface and this section considers a local update operator which guaranty the conservation of the mean value when switching from f_{j-1} to f_j.

Polynomial vectors. In order to select predict and update operator that have good properties, one follow the insight gained from the analysis of the wavelet approximation of signal on the real line. In order to do so, one need analyze the effect of a lifting wavelet transform on polynomials. The most basic constraint enforces one vanishing moment by imposing orthogonality with the constant vector $\Phi_0 = 1$. This constraint does not require to known the spacial location x_ℓ of each index $\ell \in V_L$. In order to impose higher order vanishing moments, one needs to assume some sampling pattern, for instance

$$\forall\, \ell \in V_L, \quad f(\ell) = \bar{f}(x_\ell) \qquad \text{where} \qquad x_\ell \in \mathbb{R}^q$$

and where $\bar{f}$ is a function defined on $\mathbb{R}^q$. For instance, the points x_ℓ might corresponds to a regular sampling of the line (this is the traditional wavelet setting) or to an irregular sampling of a 2D surface embedded in $\mathbb{R}^3$. The next paragraphs describe several situations with different sampling grids. Once the precise locations of the samples are known, one can for instance select Φ_s as some monomials of degree $(s_1, \ldots, s_q)$ over $\mathbb{R}^q$.

Vanishing moment and polynomials reproduction. Having defined these polynomial vectors, one requires that the following constraints are fulfilled.

- *Vanishing moments:* the wavelet coefficients of a low order polynomial should be 0, which implies that
$$\forall\, k \in H_j,\ \langle \Phi_s,\, \psi_{j,\ell} \rangle = 0. \tag{4.7}$$
- *Polynomial reproduction:* coarse coefficients f_j computed from a polynomial f_{f-1} should also be polynomials, which implies that
$$\forall\, \ell \in V_j,\ \langle \Phi_s,\, \varphi_{j,\ell} \rangle = \Phi_s(\ell) \tag{4.8}$$

In order for the wavelets and scaling function to satisfy conditions (3.7) and (3.8), the predict operator P_j and update operator U_j should be designed carefully. One can impose these constraint from the fine scale $j = L$ until the coarse scale $j = 0$. Indeed, if $(\varphi_{j-1,\ell}, \psi_{j-1,\ell})_{k,\ell}$ satisfy conditions (3.7) and (3.8), then, for the scale j

$$\forall\, s \in S, \quad \begin{cases} (3.7) & \Longleftrightarrow \quad P_j \Phi_s^{V_j} = \Phi_s^{H_j}, \\ (3.8) & \Longleftrightarrow \quad {U_j}^{\mathrm{T}} \left(\Phi_s^{V_j} + {P_j}^{\mathrm{T}} \Phi_s^{H_j} \right) = \Phi_s^{H_j}. \end{cases}$$

where $\Phi_s^A \in \ell^2(A)$ is the restriction of Φ_s to A.

In contrast, the constraint (3.8) on the update operator P_j is more involved and the next section shows how to handle it on a triangulation situations for only one vanishing moment $|S| = 1$.

4.4.4 Lifted Wavelets on Meshes

The lifted wavelet bases can be used to process signals $f \in \ell^2(V_L)$ where $\ell \in V_L$ index a sampling x_ℓ of an arbitrary surface. The construction of biorthogonal wavelets on triangulated mesh has been first proposed by Lounsbery et al. [23] and re-casted into the lifting scheme framework by Schroeder and Sweldens [32, 33].

Designing predict operators. The constraints (3.7) on the predictor P_j is easily solved. For instance, for each k, one selects only $|S|$ non vanishing weights $(p_j(k,\ell))_\ell$ and solves a small $|S| \times |S|$ linear system. Furthermore, in the case of a regular triangulation with edges of constant length, predictors with several vanishing moments have been already defined in (3.2), (3.3) and (3.3.2). Figure 3.14 shows the weights for these predictors.

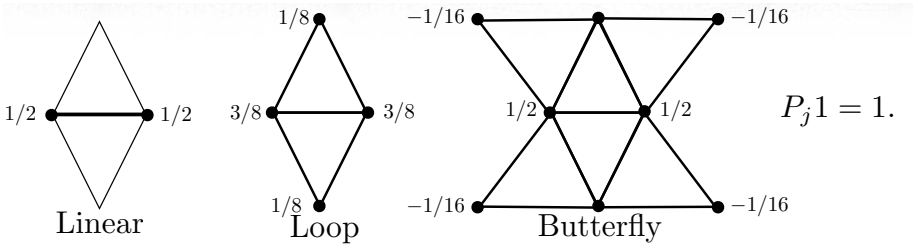


Figure 4.14: *Predict operators on a triangulation.*

One can choose any of these operators, and creates respectively linear, butterfly and Loop wavelets bases. All these predictors have one vanishing moment since they satisfy $P_j 1^{H_j} = 1^{V_j}$. In fact they have more vanishing moments if one consider polynomials Φ_s sampled at points $x_\ell \in \mathbb{R}^2$ of an hexagonal tiling with constant edge length. In practice, if the triangulation under consideration have edges with smoothly varying length, the resulting predictor are efficient to predict the value of smooth functions on the triangulation.

Designing update operators. In order to ensure the reproduction of constant polynomials, we design the update operator so that it depends only on the direct neighbors in H_j of each point in V_j

$$\forall\, \ell \in V_j, \quad V_\ell = \{\gamma(\ell,\ell') \setminus (\ell,\ell') \in E_j\}\,.$$

One then looks for a valid update operator in the following form

$$\forall\, h \in \ell^2(H_j),\ \forall\, \ell \in V_j, \quad (U_j h)(\ell) = \lambda_\ell \sum_{k \in V_\ell} h(k), \tag{4.9}$$

where each λ_ℓ should be fixed in order for condition (3.8) to be satisfied.

In an semi-regular triangulation, $|V_\ell| = 6$ excepted maybe for some points in the coarse grid $\ell \in V_0$. In this setting, the values of λ_ℓ can be computed by a recursion through the scales. In an ideal triangulation where $|V_\ell| = 6$ for all ℓ, one can use a constant weight $\lambda_\ell = \lambda$.

For the predictors defined in (3.2), (3.3) and (3.3.2), one has

$$P_j{}^{\mathrm{T}} 1^{H_j} = 3 \times 1^{V_j} \qquad \text{and} \qquad U_j{}^{\mathrm{T}} 1^{V_j} = 6\lambda 1^{H_j}$$

so setting $\lambda_\ell = 1/24$ solves equation (3.8). Figure 3.15 shows examples of butterfly wavelets on a planar semi-regular triangulation.

4.4.5 Non-linear Mesh Compression

These wavelets can be used to perform an approximation of a function $f \in \ell^2(V_L)$ defined on the fine triangulation. For instance a wavelet approximation can be applied to each coordinate $f_i, i = 1,2,3$ of the actual position $x_\ell = (f_1(\ell), f_2(\ell), f_3(\ell)) \in \mathbb{R}^3$ of the surface points, as done in [14, 16]. This leads to a scheme to approximate and compress a 3D surface using the lifted biorthogonal wavelets associated to the semi-regular triangulation. This is possible because these wavelets depend only on the combinatorial grids V_j and not on the precise position of the samples x_ℓ in 3D.

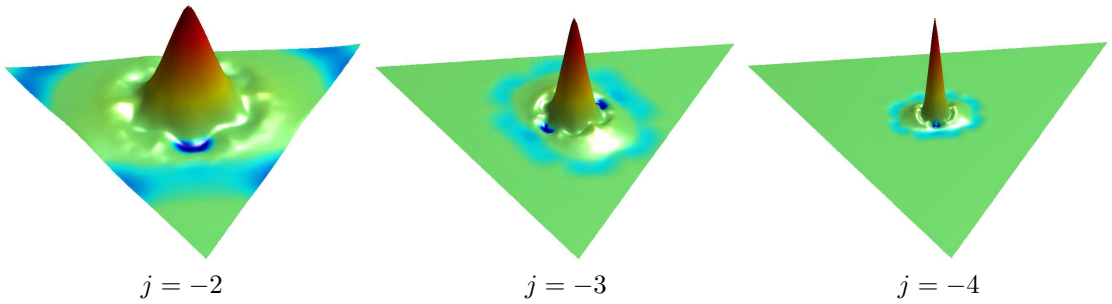


Figure 4.15: *Example of wavelets $\psi_{j,k}$ on a semi-regular triangulation. The height over the triangle (together with the color) indicates the value of the wavelet vector.*

In order to perform a wavelet approximation in this biorthogonal basis, one uses a non-linear thresholding at $T > 0$

$$f = \sum_{(j,k)\in I_T} \langle f,\, \psi_{j,k}\rangle \tilde{\psi}_{j,k}$$

$$\text{where} \quad I_T = \left\{ (j,k) \setminus k \in H_j \quad \text{and} \quad |\langle f,\, \psi_{j,k}\rangle| > T |\operatorname{supp}(\psi_{j,k})|^{-1/2} \right\}.$$

Note that for each coefficient the threshold T is scaled according to the size of the support of the wavelet in order to approximately normalize the wavelets in $\ell^2(V_L)$ norm.

Figure 3.16 shows an example of compression of the position of a vertex in 3D spaces as 3 functions defined on a semi-regular mesh. Figure 3.17 shows an example of compression of a spherical texture map which is a single function defined at each vertex of a semi-regular mesh obtained by subdividing an icosaedron.

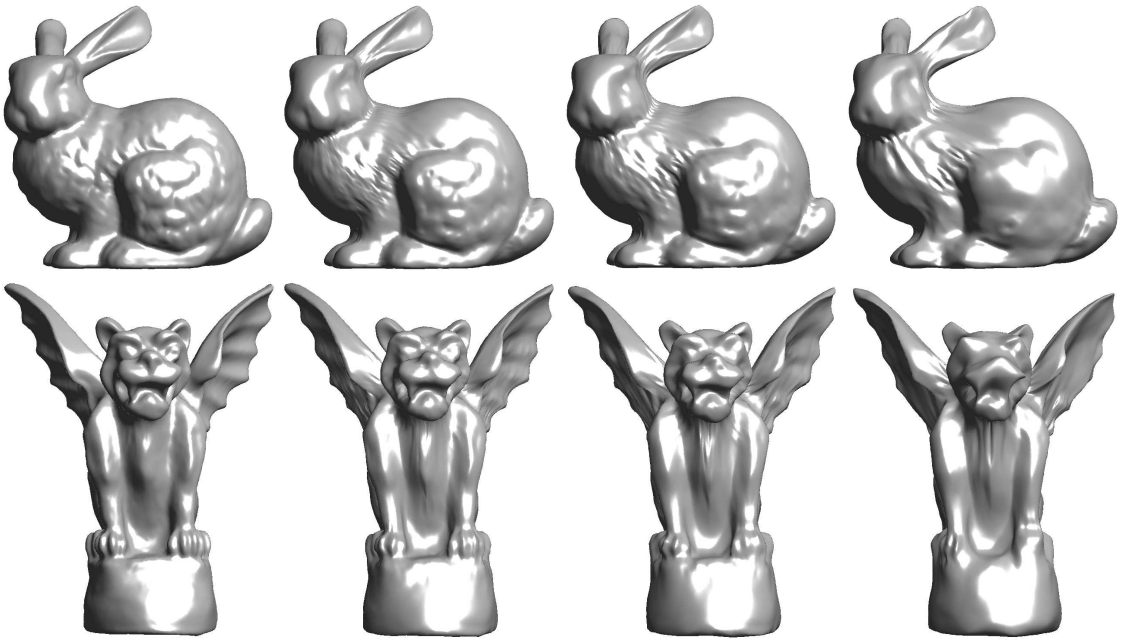
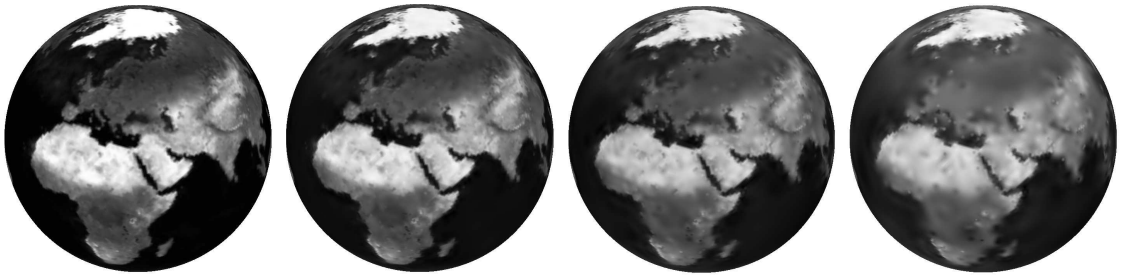


Figure 4.16: *Non-linear wavelet mesh compression with a decreasing number of coefficients.*



100% 10% 5% 2%

Figure 4.17: *Non-linear spherical wavelet compression with a decreasing number of coefficients.*

Bibliography

[1] P. Alliez and C. Gotsman. Recent advances in compression of 3d meshes. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 3–26. Springer Verlag, 2005.

[2] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. In *AIM@SHAPE repport*. 2005.

[3] A. Bronstein, M. Bronstein, and R. Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer, 2007.

[4] F. R. K. Chung. Spectral graph theory. *Regional Conference Series in Mathematics, American Mathematical Society*, 92:1–212, 1997.

[5] K. L. Clarkson. Building triangulations using epsilon-nets. In Jon M. Kleinberg, editor, *STOC*, pages 326–335. ACM, 2006.

[6] T. H. Cormen, C. E. Leiserson, and R. R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[7] M. G. Crandall, H. Ishii, and P-L. Lions. User's guide to viscosity solutions of second order partial differential equations. *BULL.AMER.MATH.SOC*, 27:1, 1992.

[8] P. Schroeder et al. D. Zorin. Subdivision surfaces in character animation. In *Course notes at SIGGRAPH 2000*, July 2000.

[9] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[10] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41:637–676, 1999.

[11] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Trans. Image Processing*, 6(9):1305–1315, September 1997.

[12] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[13] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. *Technical Report MSR-TR-2004-24*, 2004.

[14] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In Alyn Rockwood, editor, *Proceedings of the Conference on Computer Graphics (Siggraph99)*, pages 325–334. ACM Press, August8–13 1999.

[15] A. Ion, G. Peyré, Y. Haxhimusa, S. Peltier, W. G. Kropatsch, and L. Cohen. Shape matching using the geodesic eccentricity transform. In *Proceedings of OAGM'07*, 2007.

[16] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)*, pages 271–278, New York, July 23–28 2000. ACMPress.

[17] R. Kimmel. *Numerical Geometry of Images: Theory, Algorithms, and Applications*. Springer, 2004.

[18] R. Kimmel, A. Amir, and A. M. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *IEEE Trans. on PAMI*, 17(6):635–640, 1995.

[19] R. Kimmel and J.A. Sethian. Computing Geodesic Paths on Manifolds. *Proc. Natl. Acad. Sci.*, 95(15):8431–8435, 1998.

[20] L. Kobbelt. $\sqrt{3}$ subdivision. In Sheila Hoffmeyer, editor, *Proc. of SIGGRAPH'00*, pages 103–112, New York, July 23–28 2000. ACMPress.

[21] G. Leibon and D. Letscher. Delaunay triangulations and voronoi diagrams for riemannian manifolds. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 341–349, New York, NY, USA, 2000. ACM.

[22] H. Ling and D. W. Jacobs. Using the inner-distance for classification of articulated shapes. In *CVPR 2005, 20-26 June 2005, San Diego, CA, USA*, pages 719–726, 2005.

[23] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Trans. Graph.*, 16(1):34–73, 1997.

[24] S. G. Mallat. *A Wavelet Tour of Signal Processing.* Academic Press, 1998.

[25] N.J. Nilsson. *Problem-solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[26] S. J. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer, 2002.

[27] S. J. Osher and N. Paragios. *Geometric Level Set Methods in Imaging, Vision, and Graphics.* Springer-Verlag, July 2003.

[28] N. Paragios, Y. Chen, and O. D. Faugeras. *Handbook of Mathematical Models in Computer Vision.* Springer, 2005.

[29] G. Peyré and L. Cohen. Heuristically driven front propagation for fast geodesic extraction. *International Journal for Computational Vision and Biomechanics*, 1(1), 2007.

[30] G. Peyré and L. D. Cohen. Geodesic remeshing using front propagation. *Int. J. Comput. Vision*, 69(1):145–156, 2006.

[31] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *ACM Transactions on Graphics*, 22(3):340–349, July 2003.

[32] P. Schröder and W. Sweldens. Spherical Wavelets: Efficiently Representing Functions on the Sphere. In *Proc. of SIGGRAPH 95*, pages 161–172, 1995.

[33] P. Schröder and W. Sweldens. Spherical wavelets: Texture processing. In P. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95.* Springer Verlag, Wien, New York, August 1995.

[34] J.A. Sethian. *Level Sets Methods and Fast Marching Methods.* Cambridge University Press, 2nd edition, 1999.

[35] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis.*, 2(2):105–171, 2006.

[36] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computation Harmonic Analysis*, 3(2):186–200, 1996.

[37] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.