Discrete Differential Operators for Computer Graphics

Thesis by
Mark Meyer

Advisor
Alan H. Barr

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Science

California Institute of Technology
Pasadena, California

(Defended May 2004)

May 27, 2004

Acknowledgements

There are many people without whose guidance and support this work would not have been possible. To them I now offer my thanks. First and foremost, I thank my advisor, Alan Barr, for his sage advice and unwavering enthusiasm and encouragement. He has provided a wonderful insight into the world of serious research and helped to nurture my interest in solving challenging intellectual problems. The combination of his knowledge, creativity, and approachability still amazes me.

I am also deeply indebted to the other faculty members who offered unofficial advisement in both research and life. To Mathieu Desbrun, who showed through both his teaching and personal example that with a strong work ethic and creativity, great problems could be tackled. His intelligence and broad range of knowledge didn't hurt either – and were always of great help in my research. To Peter Schroeder, who pointed out new directions, challenged existing ideas, and always offered his time and expertise to provide constructive suggestions. I am also very grateful for the help and support of the staff who never complained when my ambition (and disorganization) asked them to go far beyond the call of duty: Louise Foucher, Jeri Chittum, and David Felt.

Special thanks to my officemate Min Chen for always bringing interesting problems and stimulating discussions, as well as an understanding shoulder whenever I needed it. Thanks to all my colleagues who helped me through the years to keep my academic focus as well as my personal sanity. Thank you, Pierre Alliez, Eitan Grinspun, Zoë Wood, Matt Hanna, Jessie Stumpfel, Jon Alferness, Steven Schkolne, Tran Gieng, Nathan Litke and Catherine Wong.

Finally, and most importantly, I thank my family and my friends who have become my family. Throughout the years you have offered unconditional love and understanding, many times when my actions did not warrant it, all while asking nothing in return. I am a product of your love and support. Saying thank you doesn't seem like enough

The Research on differential operators was partially supported by the STC for Computer Graphics and Scientific Visualization (ASC-89-20219), IMSC - an NSF Engineering Research Center (EEC-9529152), an NSF CAREER award (CCR-0133983), NSF (DMS-9874082, ACI-9721349, DMS-9872890, and ACI-9982273), the DOE (W-7405-ENG-48/B341492), Intel, Alias|Wavefront, Pixar, Microsoft, and the Packard Foundation.

The work on smoothing was supported by the Academic Strategic Alliances Program of the Accelerated Strategic Computing Initiative (ASCI/ASAP) under subcontract B341492 of DOE contract W-7405-ENG-48. Additional support was provided by NSF (ACI-9624957, ACI-9721349, DMS-9874082, DMS-9872890, and ASC-89-20219 (STC for Computer Graphics and Scientific Visualization)), Alias|Wavefront and through a Packard Fellowship.

The research on remeshing was supported in part by IMSC NSF Engineering Research Center (EEC-9529152), by the ECG project of the EU No IST-2000-26473, and by a NSF CAREER award (CCR-0133983).

The work on parameterization was supported by IMSC, an NSF Engineering Research Center (EEC-9529152), and a NSF CAREER award (CCR-0133983).

Discrete Differential Operators for Computer Graphics

by

Mark Meyer

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Science

Abstract

This thesis presents a family of discrete differential operators. Since these operators are derived taking into account the continuous notions of differential geometry, they possess many similar properties. This family consists of first- and second-order properties, both geometric and parametric. These operators are then analyzed and their practical use is tested in several example applications.

First, the operators are used in a smoothing application. Due to the properties of the operators, the resulting smoothing algorithm is general, efficient and robust to sampling problems. The smoothing can be applied to many different inputs ranging from images to surfaces to volume data.

Second, a surface remeshing technique using the operators is presented. Given the operators, we present an algorithm that resamples a surface mesh according to several geometric criteria (integrated curvature, directional curvature, geometric distortion). The resulting algorithm is efficient, general and user-tunable.

Next, a surface mesh parameterization technique is presented. Using geometric invariants associated with the discrete operators, we present an efficient, tunable parameterization algorithm that is robust to sampling irregularities in the input model. Using the properties of the differential operators allows us to make a parameterization algorithm that relies only on geometric information and not the original parameterization of the input model.

Finally, we conclude and present future work including physical simulation and sampling theory.

Table of Contents

List of Figures

Chapter 1

Introduction

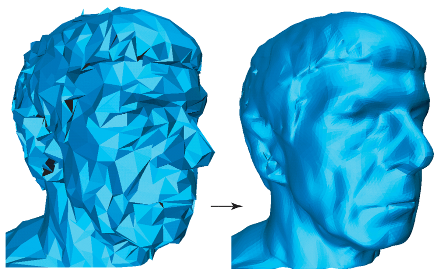
1.1 Motivation

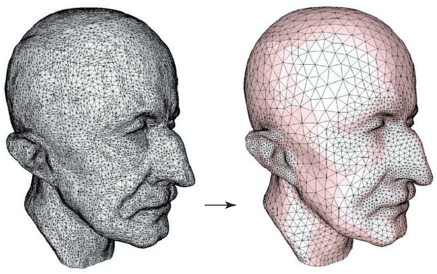
The study and estimation of differential quantities on surfaces (such as curvatures) are fundamental to applications in many fields. Computer Graphics is no exception with many potential applications including (see Figure 1.1):

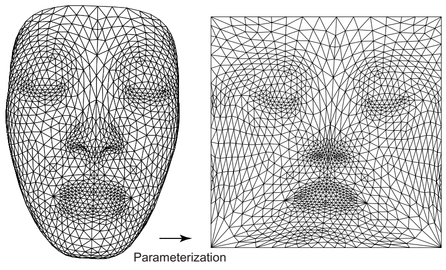
Rendering / Shading: Shading computations are used in almost every Computer Graphics application. Programmable shaders allow artists and designers to achieve a wide variety of effects ranging from physically based subsurface scattering to cartoon/cel-shading. These shaders often use a surface's differential properties for filtering, aiding texture lookups, and general color computations. Non-photorealistic shading techniques [DFRS03], such as diagramatic and toon shading, often use curvatures and their derivatives to determine where to place suggestive contours and where to place shading discontinuities.

Smoothing: Surface smoothing and enhancement algorithms are becoming more and more commonplace in the fields of Computer Graphics, Computer Vision and CAGD (Computer Aided Geometrical Design). The use of scanning devices to create surface models has increased rapidly in recent years. While these devices can produce highly detailed models of real-world objects, they also have the inherent problem of noise and scanning errors. Curvature flow as well as other PDEs (partial differential equations) can be used to remove this noise by smoothing the surface.

(a)


(b)


(c)



(d)

Figure 1.1: *Some applications of differential operators: (a) Non-photorealistic renderings [DFRS03] can use curvatures (and their derivatives) to determine where to draw suggestive contours, (b) a surface smoothing technique using curvature flow to reduce the noise caused by laser scanning, (c) a remeshing algorithm uses curvature to place more samples in regions of high curvature, where more detail is present, (d) a parameterization algorithm uses differential quantities to minimize distortion.*

Simplification / Remeshing: Mesh simplification and remeshing procedures are often used to resample a mesh to create another with a reduced (or increased) number of samples while maintaining some notion of error (usually geometric accuracy). Mesh simplification algorithms [HG99] often use curvature estimates to determine which areas can be simplified and to guarantee optimal triangulations. Similarly, remeshing algorithms [AMD02] are typically driven by curvature to sample with increased density in regions of high curvature.

Parameterization: Surface parameterization algorithms involve computing a mapping between a surface and some other domain (often a flattened (2D) version of the surface). This mapping can then be used to transfer quantities from the chosen domain to the surface for operations such as texture mapping. Additionally, algorithms can be computed in (or cached on) the parameter domain when it may be difficult to apply the same algorithm to the surface directly – care must be taken, however, to account for the distortion and differences between the surface and the parameter domain. To reduce this problem, these parameterizations are often computed while trying to minimize some notion of distortion (angle differences, length differences, area differences, etc.). These distortion measures often involve differential properties of the surface such as curvatures (and their integrals) and thus can benefit from robust and accurate estimations of these differential properties.

Simulation: With the increased computing power enjoyed today, simulation is becoming a more and more important tool in many fields including Computer Graphics. Simulation allows artists and animators to achieve realistic motions and fine details without requiring the manual specification of every vertex or shaded pixel. This allows the creation of secondary motions and small-scale details that would otherwise have been prohibitively expensive. Simulations also provide predictive power allowing for more efficient tests and experiments that may have been impractical using other means (due to time constraints, expense, safety, etc.). PDEs computed using differential quantities are often used to drive intricate simulations of clothing, skin, muscles, clouds, and fluids just to name a few.

Although differential surface properties have been well studied in other fields, Computer Graphics has one important difference: while most fields use a continuous notion for a surface, in Computer Graphics, we often use a discrete (or at most $\mathcal{C}^0$) description of the surface – namely triangle meshes. Because of this difference, it is difficult to directly transfer the existing formulae and results to Computer Graphics applications. In fact, despite extensive use of triangle meshes in Computer Graphics and the obviously many uses of differential operators, there is no consensus on the most appropriate way to estimate simple geometric attributes such as normal vectors and curvatures on discrete surfaces.

1.2 Contributions

This thesis defines one method for extending the continuous defintion and estimating differential properties on discrete surfaces. We define and derive operators that compute the first- and second-order differential attributes (normal vector **n**, mean curvature κ_H, Gaussian curvature κ_G, principal curvatures κ_1 and κ_2, and principal directions $\mathbf{e}_1$ and $\mathbf{e}_2$) for piecewise linear surfaces such as arbitrary triangle meshes. We present a unified framework for deriving such quantities resulting in a set of operators that is consistent, accurate, robust (in both regular and irregular sampling) and simple to compute. Additionally, as we show, many of these operators can be generalized to any 2-manifold (or even 3-manifold) in an arbitrary dimension embedding space.

We then demonstrate the accuracy and usefulness of these operators in several different geometry processing applications:

- An efficient and robust **smoothing algorithm** that uses the differential operators to integrate a surface flow PDE.
- A fast and tunable **remeshing algorithm** that uses the differential operators in conjunction with user requests to determine where to place the surface samples.
- A **parameterization technique** that uses the differential properties of the surface to define the distortion metric to be minimized.

1.3 Thesis Overview

The remainder of this dissertation is organized as follows:

Chapter 2 reviews notions and formulae from continuous differential geometry and details why a *local spatial average* of differential attributes over the immediate 1-ring neighborhood is a good choice to extend the continuous definition to the discrete setting. We then present a formal derivation of the Mean Curvature Normal, Gaussian Curvature, Principal Curvatures, and Principal Directions for triangle meshes using the mixed Finite-Element/Finite-Volume paradigm. The relevance of our approach is demonstrated by showing the optimality of our operators under mild smoothness conditions. Finally, the accuracy of our operators is compared to that of previous techniques.

Chapter 3 describes a technique for smoothing using the discrete differential operators. This technique is general, efficient and robust against sampling rate changes due to the properties of the operators themselves. Additionally, the smoothing algorithm can be applied to many different forms of data including images, surfaces, volume data, and vector fields.

Chapter 4 describes a technique for remeshing a triangulated surface. This technique uses the discrete differential operators to drive the placement of samples and the direction of edges. The resulting algorithm is extremely efficient, general and robust.

Chapter 5 describes a technique for parameterizing a triangulated 2-manifold. This technique uses geometric invariants (associated with the properties of our discrete differential operators) to compute sampling invariant, intrinsic parameterizations. These invariants produce efficiently solvable linear systems resulting in an invaluable tool for mesh parameterization.

Conclusions and Future Work are then discussed to complete the thesis.

Chapter 2

Discrete Differential Operators

In this chapter, we present a unified and consistent set of flexible tools to approximate important geometric attributes, including normal vectors and curvatures on arbitrary triangle meshes. We present a consistent derivation of these first- and second-order differential properties using *averaging Voronoi cells* and the mixed Finite-Element/Finite-Volume method, and compare them to existing formulations. Building upon previous work in discrete geometry, these operators are closely related to the continuous case, guaranteeing an appropriate extension from the continuous to the discrete setting: they respect most intrinsic properties of the continuous differential operators.

Since differential geometry provides a well researched, formal basis for describing the differential properties of a surface, we begin with a review of several important quantities from differential geometry (a more complete discussion of differential geometry can be found in one of the many great texts such as [dC76, Gra98, DHKW92]). This is followed by a discussion of previous techniques for computing differential quantities on discrete surfaces. We then present our technique for extending continuous differential operators to the discrete domain using spatial averaging.

2.1 Notions from Differential Geometry

Let S be a surface (2-manifold) embedded in $\mathbb{R}^3$, described by an arbitrary (local) parameterization of 2 variables, $\mathbf{X}(u, v)$, around a point $\mathbf{p}$. For each point on the surface S, we can locally approximate the surface by its tangent plane, orthogonal to the *normal vector* **n**. Ignoring the surface orientation, the normal vector of S at a point **p** is given by:

$$\mathbf{n} = \frac{\mathbf{X}_u \times \mathbf{X}_v}{\|\mathbf{X}_u \times \mathbf{X}_v\|} ,$$

where the subscripts indicate partial derivatives.

Using these same derivatives, we can describe the local shape of the surface S near **p**. The first fundamental form describes the change on the surface for a given small change in the parameters (u, v), and is given by

$$\mathbf{I}(u, v, du, dv) = d\mathbf{X} \cdot d\mathbf{X} = d\mathbf{u}^T \mathcal{G} d\mathbf{u},$$

with $d\mathbf{u} = (du, dv)^T$, and

$$\mathcal{G} = \begin{pmatrix} \mathbf{X}_u \cdot \mathbf{X}_u & \mathbf{X}_u \cdot \mathbf{X}_v \\ \mathbf{X}_u \cdot \mathbf{X}_v & \mathbf{X}_v \cdot \mathbf{X}_v \end{pmatrix} .$$

The second fundamental form describes the change in the unit normal for a given small change in the parameters (u, v), and is defined by

$$\mathbf{II}(u, v, du, dv) = -d\mathbf{X} \cdot d\mathbf{n} = d\mathbf{u}^T \mathcal{D} d\mathbf{u},$$

where

$$\mathcal{D} = \begin{pmatrix} \mathbf{n} \cdot \mathbf{X}_{uu} & \mathbf{n} \cdot \mathbf{X}_{uv} \\ \mathbf{n} \cdot \mathbf{X}_{uv} & \mathbf{n} \cdot \mathbf{X}_{vv} \end{pmatrix} .$$

Note that the first fundamental form is invariant to the choice of surface parameterization as well as rigid motions of the surface (rotations and translations). Since it does not depend on the embedding of the surface, the first fundamental form is an *intrinsic* property. The second fundamental form, on the other hand, does depend on the embedding of S and is therefore known as an *extrinsic* property of the surface.

Using these two fundamental forms, we can locally describe the surface shape using the

shape operator or **Weingarten map** β:

$$\beta(\mathbf{t}) = \mathcal{G}^{-1}\mathcal{D}\mathbf{t} = -\nabla_{\mathbf{t}}\mathbf{n},$$

where **t** is a vector in the tangent plane at **p** and $-\nabla_{\mathbf{t}}\mathbf{n}$ is the directional derivative of the normal **n** in the direction **t**:

$$\nabla_{\mathbf{t}}\mathbf{n} = \lim_{\alpha\to 0} \frac{\mathbf{n}(\mathbf{p}+\alpha\mathbf{t}) - \mathbf{n}(\mathbf{p})}{\alpha}.$$

Therefore, the shape operator, β, is a linear operator mapping $T_{\mathbf{p}}S \to T_{\mathbf{p}}S$, where $T_{\mathbf{p}}S$ is the tangent space of S at **p**. It measures the change in normal in the direction **t** and, as we shall see, is useful for measuring the bending and local shape of the surface.

2.1.1 Curvatures and Principal Directions

Local bending of the surface is measured by *curvatures*. For every unit direction **t** in the tangent plane, the normal curvature $\kappa^N(\mathbf{t})$ is defined as the curvature of the curve that belongs to both the surface itself and the plane containing both **n** and **t**:

$$\kappa^N(\mathbf{t}) = \frac{\beta(\mathbf{t})\cdot\mathbf{t}}{|\mathbf{t}|^2}.$$

The two *principal curvatures* κ_1 and κ_2 of the surface S, with their associated orthogonal directions $\mathbf{e}_1$ and $\mathbf{e}_2$, are the extremum values of the normal curvatures over all directions **t** (see Figure 2.1(a)). If we parameterize the directions **t** by θ, the angle between $\mathbf{e}_1$ and **t**, the normal curvature can be expressed in terms of the principal curvatures:

$$\kappa^N(\theta) = \kappa_1 cos^2(\theta) + \kappa_2 sin^2(\theta).$$

The *mean curvature* κ_H is defined as the average of the normal curvatures:

$$\kappa_H = \frac{1}{2\pi}\int_0^{2\pi} \kappa^N(\theta)d\theta. \tag{2.1}$$

Using the above relation for normal curvatures, leads to the well-known definition:

$$\kappa_H = \frac{\kappa_1 + \kappa_2}{2} \, .$$

The *Gaussian curvature* κ_G is defined as the product of the two principle curvatures:

$$\kappa_G = \kappa_1 \kappa_2 \, . \tag{2.2}$$

These latter two curvatures represent important local properties of a surface. Points on the surface are often classified based on their mean and Gaussian curvatues – if $\kappa_G > 0$ the point is *elliptic*, if $\kappa_G < 0$ the point is *hyperbolic*, if $\kappa_G = 0$ and $\kappa_H \neq 0$ the point is *parabolic*, and if $\kappa_G = \kappa_H = 0$ the point is *planar*.

Lagrange noticed that $\kappa_H = 0$ is the Euler-Lagrange equation for surface area minimization. This gave rise to a considerable body of literature on minimal surfaces and provides a direct relation between surface area minimization and mean curvature flow:

$$2\kappa_H \, \mathbf{n} = \lim_{diam(\mathcal{A}) \to 0} \frac{\nabla \mathcal{A}}{\mathcal{A}} \, ,$$

where $\mathcal{A}$ is a infinitesimal area around a point **p** on the surface, $diam(\mathcal{A})$ its diameter, and ∇ is the gradient with respect to the (x, y, z) coordinates of **p**. We will make extensive use of the mean curvature normal κ_H **n**. Therefore, we will denote by **K** the operator that maps a point **p** on the surface to the vector $\mathbf{K}(\mathbf{p}) = 2\kappa_H(\mathbf{p}) \, \mathbf{n}(\mathbf{p})$. **K** is also known as the Laplace-Beltrami operator for the surface S. Note that in the remainder of this chapter we will make no distinction between an operator and the value of this operator at a point as it will be clear from context. Gaussian curvature can also be expressed as a limit:

$$\kappa_G = \lim_{diam(\mathcal{A}) \to 0} \frac{\mathcal{A}^{\mathcal{G}}}{\mathcal{A}} \, , \tag{2.3}$$

where $\mathcal{A}^{\mathcal{G}}$ is the area of the image of the Gauss map (also called the spherical image) associated with the infinitesimal surface $\mathcal{A}$.

2.1.2 Principal Quadric

The portion of a surface S near a point **p** can be locally represented by the height field (or Monge patch) $h(x, y) = z$, with **p** at the origin and the normal **n** at **p** in the direction of the z-axis. Using a Taylor Expansion of h and dropping the higher-order terms results in a quadratic surface which approximates S to second-order:

$$z = \frac{1}{2}(h_{xx}x^2 + 2h_{xy}xy + h_{yy}y^2),$$

where the derivatives of h are evaluated at **p**. This surface is known as the *principal quadric* of S at **p**. At an elliptic point, the principal quadric is an elliptic paraboloid; at a hyperbolic point, it is a hyperbolic paraboloid; at a parabolic point, it is a parabolic cylinder; and at a planar point, it is a plane. Note that, since the principal quadric encodes the same differential information as the surface S at **p**, computing the principal quadric is often used as a way to compute the the differential properties of S.

2.2 Previous Work

Due to the importance of these differential properties in many computer graphics applications, it is no surprise that they have been heavily researched. This section describes several methods for computing differential properties on triangle meshes. In some methods, the vertex normal is computed at the same time as the curvatures. However, some methods require an estimate of the vertex normal to compute the curvature properties. Therefore, we begin by desribing methods for computing the normal at a vertex before discussing techniques for computing the curvatures such as quadric fitting and direct extensions of continuous equations.

In the following sections, assume T is an oriented triangle mesh with or without boundary. Also assume that the orientation is consistent (neighboring triangles have their normals pointing towards the same side of the surface). For a vertex **p**, we denote the set of 1-ring neighbors as $N_1(\mathbf{p})$ and the number of such neighbors m. Similarly, the 1-ring neighborhood of **p** is the set of all triangles in T incident to **p**.

2.2.1 Vertex Normal Estimation

Since the surface normal is such a fundamental quantity in computer graphics – useful in algorithms such as shading, culling, even in computing other differential properties – the computation of vertex normals from a triangle mesh has been studied for many years. It is fairly common to approximate the normal at a vertex **p** on a triangle mesh by a weighted average of the normals of the triangles incident to **p**:

$$\mathbf{n} = \frac{\sum_i w_i \mathbf{n}_i}{\| \sum_i w_i \mathbf{n}_i \|} ,$$

where $\mathbf{n}_i$ are the normals of the triangles incident to **p**.

While the averaging of normals is fairly standard, the choice of the weights w_i is not. Gouraud [Gou71] used an uniformly weighted average, i.e., $w_i = 1$. Depending on the arrangement of triangles around **p**, this can produce greatly varying normals. To reduce this problem, Thürmer and Wüthrich [TW98] use the angle incident to **p** on the i-th face as the weight, $w_i = \theta_i$. This fits their claim that the normal vector should only be defined very locally, however, this normal remains consistent only if the faces are subdivided linearly, introducing vertices which are not on a smooth surface. Max [Max99] derived weights by assuming that the surface locally approximates a sphere:

$$w_i = \frac{sin\theta_i}{\|\mathbf{pp}_i\| \; \|\mathbf{pp}_{i+1}\|} ,$$

where $\mathbf{p}_i$ are the (ordered) neighbors of **p** in face i. These weights are therefore exact if the object is a (even irregular) tessellation of a sphere. However, it is unclear how this approximation behaves on more complex meshes, since no error bounds are defined. Additionally, many meshes have local sampling adapted to local flatness, contradicting the main property of this approach. Even for a property as fundamental as the surface normal, we can see that several (often contradictory) formulæ exist.

2.2.2 Principal Quadric Fitting

One of the most common ways of computing the differential properties at the vertices of a triangle mesh is by locally fitting a continuous surface and computing the curvatures on this continuous surface [Ham93]. Since we are intereseted in second-order derivative properties, fitting a quadric intuitively makes sense. In fact, it has been shown that fitting higher-order surfaces has little advantage [KLM98] over fitting quadrics. Therefore, in this section, we will describe techniques for fitting quadrics to triangle meshes and how to recover the associated differential properties [1].

While the parameters of the principal quadric could be directly estimated using an procedure to fit to the 1-ring neighbors of **p**, this results in a non-linear optimization problem. More commonly, the surface normal is first computed, then the quadric is fit in a rotated space. The steps are as follows:

1. Estimate the surface normal **n** at **p**

2. Construct the rotation matrix $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)^T$:

$$\mathbf{r}_1 = \frac{(\mathbf{I}-\mathbf{n}\mathbf{n}^T)\mathbf{i}}{\|(\mathbf{I}-\mathbf{n}\mathbf{n}^T)\mathbf{i}\|}, \qquad \mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1, \qquad \mathbf{r}_3 = \mathbf{n}$$

3. Map the 1-ring data ($\mathbf{q}_i$) into the rotated frame:

$$\tilde{\mathbf{q}}_i = \mathbf{R}\,(\mathbf{q}_i - \mathbf{p})$$

4. Using the rotated 1-ring neighbors $\tilde{\mathbf{q}}_i$, fit the quadric using least squares:

$$\tilde{z} = a\tilde{x}^2 + b\tilde{x}\tilde{y} + c\tilde{y}^2$$

5. Solve for κ_1, κ_2, and θ (the angle between the $\tilde{x}$ axis and the first principal direction):

$$\kappa_{1,2} = a + c \pm \sqrt{(a-c)^2 + b^2}$$
$$\theta = \frac{1}{2} atan2(b, a-c)$$

[1] This section loosely follows the exposition laid out in [MV97].

Note that this method relies heavily on the accuracy of the normal vector computation. This dependence can be reduced by fitting an extended quadric and iteratively adjusting the normal. The steps remain the same as before except for steps 4 and 5 which become:

4. Fit the extended quadric:

$$\tilde{z} = a\tilde{x}^2 + b\tilde{x}\tilde{y} + c\tilde{y}^2 + d\tilde{x} + e\tilde{y}$$

   This gives a new estimate for the surface normal:

$$\mathbf{n} = \frac{(-d, -e, 1)^T}{(d^2 + e^2 + 1)^{\frac{1}{2}}}$$

   From this, a new rotation **R** can be computed (step 2) and steps 2, 3, 4 can be repeated until the change in normal is below a threshold.

5. Estimate the differential parameters from the extended quadric.

Note that this method could be further extended by adding a translation term f to the extended quadric to account for error in the position of **p**.

2.2.3 Statistical Methods

When trying to estimate the differential properties of a discrete surface, one of the biggest problems is noise. For this reason, several researchers have looked into statistical methods similar to those in signal processing using covariance matrices. These statistical methods have the benefit of being relatively insensitive to noise. The surface covariance matrix for the 1-ring neighborhood of **p** is:

$$\mathcal{C}_1 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{q}_i - \bar{\mathbf{q}})(\mathbf{q}_i - \bar{\mathbf{q}})^T,$$

where $\bar{\mathbf{q}}$ is the centroid of the neighbors of **p**. Note that the eigenvectors of this matrix are two tangent vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ and the normal **n**, making the matrix $\mathcal{C}_1$ similar to the first fundamental form matrix $\mathcal{G}$.

Similarly, one can define a covariance matrix that is analogous to the second fundamental form matrix $\mathcal{D}$. This is done by projecting the edge from $\mathbf{p}$ to $\mathbf{q}_i$ onto the tangent plane and scaling it by the orthogonal distance from $\mathbf{q}_i$ to the tangent plane:

$$\mathbf{d}_i = \left[(\mathbf{p} - \mathbf{q}_i) \cdot \mathbf{n}\right] (\mathbf{p} - \mathbf{q}_i)^T \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix},$$

and then defining the covariance matrix:

$$\mathcal{C}_2 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{d}_i - \bar{\mathbf{d}})(\mathbf{d}_i - \bar{\mathbf{d}})^T.$$

The eigenvectors of this covariance matrix then estimate the principal directions.

2.2.4 Extensions From Differential Geometry

In addition to the methods mentioned above, several researchers have attempted to directly use the formulae from continuous differential geometry on triangulated surfaces in order to compute the differential properties of the triangle mesh itself. Many researchers have used curves formed by the surface to determine the curvature. For instance, [CS92] creates circles using triples of $\mathbf{q}_i\mathbf{p}\mathbf{q}_j$, where $\mathbf{q}_i$, $\mathbf{q}_j$ are neighbors of $\mathbf{p}$, and then uses the curvatures of these circles along with Meusnier's theorem from continuous differential geometry (which relates the curvature of surface curves to the curvature of the surface) to compute the differential properites of the surface. Others have used the difference in normals between adjacent vertices as a measure of the normal curvature and then used these normal curvatures to derive the full differential properties of the surface.

[AvD95] use the fact that the mean curvature is always the average of a normal curvature in one direction and a normal curvature in the perpendicular direction to show that the mean curvature at the points along an edge is one half the dihedral angle (the angle between the two adjacent faces). Using this, they average the mean curvatures from the edges around a vertex to produce the mean curvature at the vertex.

Taubin proposed the most complete derivation of surface properties, leading to a discrete approximation of the curvature tensors for polyhedral surfaces [Tau95a]. Let us consider the matrix:

$$\mathcal{M} = \frac{1}{2\pi} \int_{-\pi}^{\pi} \kappa^N(\mathbf{t})\, \mathbf{t}\, \mathbf{t}^T d\theta\,,$$

where **t** is a tangent vector. It can be shown that this 3×3 symmetric matrix has eigenvalues of $0, \lambda_1, \lambda_2$ with the corresponding eigenvectors $\mathbf{n}, \mathbf{e}_1, \mathbf{e}_2$. The principal curvatures of the surface can then be computed as:

$$\kappa_1 = 3\lambda_1 - \lambda_2, \qquad \kappa_2 = 3\lambda_2 - \lambda_1 \ \ .$$

By first estimating the normal, and then projecting each outgoing edge $\mathbf{pq}_i$ to define a unit tangent $\mathbf{t}_i$, Taubin proposed to approximate the matrix $\mathcal{M}$ as:

$$\mathcal{M} = \sum_{\mathbf{q}_i \in N_1(\mathbf{p})} w_i\, \kappa^N(\mathbf{t}_i)\, \mathbf{t}_i\, \mathbf{t}_i^T\,,$$

where the weights w_i are proportional to the areas of the triangles adjacent to the edge $\mathbf{pq}_i$, constrained so that they sum to unity. The normal curvature $\kappa^N(\mathbf{t}_i)$ is approximated by constructing an osculating circle using **p**, **n**, and $\mathbf{q}_i$ and computing the inverse of its radius:

$$\kappa^N(\mathbf{t}_i) = \frac{2\mathbf{n} \cdot (\mathbf{q}_i - \mathbf{p})}{\|\mathbf{q}_i - \mathbf{p}\|^2}\,.$$

Once the matrix $\mathcal{M}$ is computed, the principal curvatures and directions can easily be recovered using standard eigenvector decomposition techniques.

This thesis is closely related to these works since we also derive the differential properties of a triangulated surface using extensions of properties from continuous differential geometry. In order to preserve fundamental invariants from the continuous domain, we have followed a path initiated by Federer, Fu, Polthier, and Morvan to name a few [Fu93, PP93, PS98, Mor01, TM02]. This series of work proposed simple expressions for the total curvatures, as well as the Dirichlet energy for triangle meshes, and derived discrete methods to compute minimal surfaces or geodesics. We refer the reader to the overview compiled by Morvan [Mor01]. Note also the tight connection with the “Mimetic Discretizations” used in computational physics by Shashkov, Hyman, and Steinberg [HS97, HSS97]. Although it shares a lot of similarities with all these approaches, our work offers a different, unified derivation that ensures accuracy and tight error bounds, leading to simple formulæ that are straightforward to implement.

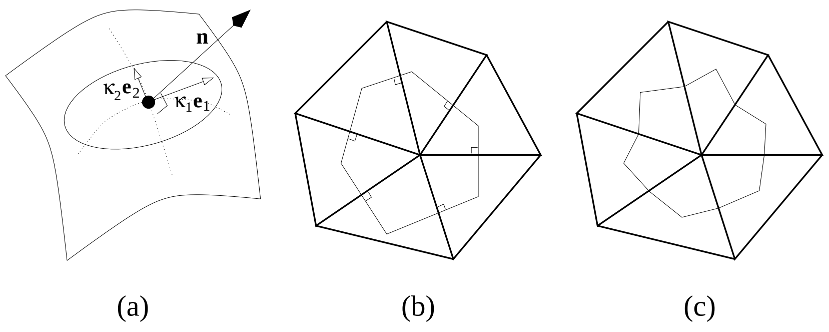


Figure 2.1: *Local regions: (a) an infinitesimal neighborhood on a continuous surface patch; (b) a finite-volume region on a triangulated surface using Voronoi cells, or (c) Barycentric cells.*

2.3 Discrete Properties As Spatial Averages

Most of the smooth definitions for differential properties described above need to be reformulated for $\mathcal{C}^0$ surfaces. We can consider a mesh as either the limit of a family of smooth surfaces, or as a linear (yet assumedly "good") approximation of an arbitrary surface. Since we wish for the total (integrated) value of the property to be independent of the number of samples in the triangle mesh, we define properties (geometric quantities) of the surface at each vertex as *spatial averages* around this vertex. This area-averaging is known as the finite volume method. Although this thesis uses piecewise constant weighting functions (in the finite-element/finite-volume sense), more complex weighting functions can easily be incorporated into the area-averaging scheme.

By using these spatial averages, we extend the definition of curvature or normal vector from the continuous case to discrete meshes. Moreover, this definition is appropriate when, for example, geometric flows must be integrated over time on a mesh as a vertex will be updated according to the average behavior of the surface around it. Therefore, the piecewise linear result of the flow will be a correct approximation of the smoothed surface if the initial triangle mesh was a good approximation of the initial surface. Since we make no assumption on the smoothness of the surface, we will restrict the average to be within the immediately neighboring triangles, the 1-ring neighborhood. For example, we define the discrete Gaussian curvature, $\widehat{\kappa}_G$, at a vertex P as:

$$\widehat{\kappa}_G = \frac{1}{\mathcal{A}} \iint_{\mathcal{A}} \kappa_G \ \ dA,$$

where $\mathcal{A}$ is a properly selected area around P. Note however that we will not distinguish between the (continuous) pointwise and the (discrete) spatially averaged notation, except when there may be ambiguity.

2.3.1 General Procedure Overview

The next sections describe how we derive accurate numerical estimates of the first- and second-order operators at any vertex on an arbitrary mesh. We first restrict the averaging area to a family of special local surface patches denoted $\mathcal{A}_M$. These regions will be contained within the 1-ring neighborhood of each vertex, with piecewise linear boundaries crossing the mesh edges at their midpoints (Figures 2.1(b) and (c)). We show that this choice guarantees correspondences between the continuous and the discrete case. We then find the precise surface patch that optimizes the accuracy of our operators, completing the operator derivation. These steps will be explained in detail for the first operator, the mean curvature normal operator, **K**, and a more direct derivation will be used for the Gaussian curvature operator κ_G, the two principal curvature operators κ_1 and κ_2, and the two principal direction operators $\mathbf{e}_1$ and $\mathbf{e}_2$. All these operators take a vertex $\mathbf{x}_i$ and its 1-ring neighborhood as input, and provide an estimate in the form of a simple formula that we will frame for clarity.

2.4 Discrete Mean Curvature Normal

We now provide a simple and accurate numerical approximation for both the normal vector, and the mean curvature for surface meshes in 3D.

2.4.1 Derivation of Local Integral Using FE/FV

To derive a spatial average of geometric properties, we use a systematic approach which mixes finite elements and finite volumes. Since the triangle mesh is meant to visually represent the surface, we select a linear finite element on each triangle, that is, a linear interpolation between the three vertices corresponding to each triangle. Then, for each vertex, an associated surface patch (so-called finite volume in the Mechanics literature), over which the average will be computed, is chosen. Two main types of finite volumes are common in practice, see Figures 2.1(b-c). In each case, their piecewise linear boundaries connect the midpoints of the edges emanating from the center vertex and a point within each adjacent triangle. For the point inside each adjacent triangle, we can use either the barycenter or the circumcenter. The surface area formed from using the barycenters is denoted $\mathcal{A}_{\text{Barycenter}}$ while the surface area using the circumcenters is recognized as the local Voronoi cell and denoted $\mathcal{A}_{\text{Voronoi}}$. In the general case when this point could be anywhere, we will denote the surface area as $\mathcal{A}_{\text{M}}$.

We now wish to compute the integral of the mean curvature normal over the area $\mathcal{A}_{\text{M}}$. Since the mean curvature normal operator, also known as Laplace-Beltrami operator, is a generalization of the Laplacian from flat spaces to manifolds [DHKW92], we first compute the Laplacian of the surface with respect to the *conformal space* parameters u and v. As in [Dzi91] and [PP93], we use the current surface discretization as the conformal parameter space, that is, for each triangle of the mesh, the triangle itself defines the local surface metric. With such an induced metric, the Laplace-Beltrami operator simply turns into a Laplacian $\Delta_{u,v}\mathbf{x} = \mathbf{x}_{uu} + \mathbf{x}_{vv}$ [DHKW92]:

$$\iint_{\mathcal{A}_{\text{M}}} \mathbf{K}(\mathbf{x}) \;\; dA = -\iint_{\mathcal{A}_{\text{M}}} \Delta_{u,v}\mathbf{x} \, du \, dv. \tag{2.4}$$

Using Gauss's theorem as described in Appendix A.1, the integral of the Laplacian over a surface going through the midpoint of each 1-ring edge of a triangulated domain can be expressed as a function of the node values and the angles of the triangulation. The integral of the Laplace-Beltrami operator thus reduces to the following simple form:

$$\iint_{\mathcal{A}_{\text{M}}} \mathbf{K}(\mathbf{x}) dA = \frac{1}{2} \sum_{j \in N_1(\mathbf{x}_i)} (cot\ \alpha_{ij} + cot\ \beta_{ij}) \ (\mathbf{x}_i - \mathbf{x}_j), \tag{2.5}$$

where α_{ij} and β_{ij} are the two angles opposite to the edge in the two triangles sharing the edge $(\mathbf{x}_i, \mathbf{x}_j)$ as depicted in Figure 2.2(a), and $N_1(\mathbf{x}_i)$ is the set of 1-ring neighbor vertices of vertex i.

Note that this equation was already obtained by minimizing the Dirichlet energy over a triangulation in [PP93]. Additionally, it can be shown to be the formula for the gradient of the surface area of the mesh (see Appendix A.2). This confirms, in the discrete setting, the area minimization nature of the mean curvature normal as derived by Lagrange. We can therefore express our previous result using the following general formula, valid for *any triangulation*:

$$\iint_{\mathcal{A}_\mathrm{M}} \mathbf{K}(\mathbf{x}) dA = 2\nabla \mathcal{A}_{\text{1-ring}}. \tag{2.6}$$

where $\mathcal{A}_{\text{1-ring}}$ is the 1-ring area around the vertex $\mathbf{x}_i$, and ∇ is the gradient with respect to the (x, y, z) coordinates of $\mathbf{x}_i$.

Notice that the formula results in a zero value for any flat triangulation, regardless of the shape or size of the triangles of the locally-flat (zero curvature) mesh since the gradient of the area is zero for any locally flat region.

Although we have found an expression for the integral of the mean curvature normal independent of which of the two finite volume discretizations is used, one finite volume region must be chosen in order to provide an accurate estimate of the spatial average. We show in the next section that Voronoi cells provide provably tight error bounds under reasonable assumptions of smoothness.

2.4.2 Voronoi Regions for Tight Error Bounds

We now show that using Voronoi regions provides provably tight error bounds for the discrete operators by comparing the local spatial average of mean curvature with the actual pointwise value. Given a $\mathcal{C}^2$ surface tiled by small patches $\mathcal{A}_i$ around n sample points $\mathbf{x}_i$, we can define the error E created by local averaging of the mean curvature normal compared to its pointwise value at $\mathbf{x}_i$ as:

$$\begin{aligned} E &= \sum_i \iint_{\mathcal{A}_i} \left\| \mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{x}_i) \right\|^2 dA \\ &\leq \sum_i \iint_{\mathcal{A}_i} C_i^2 \|\mathbf{x} - \mathbf{x}_i\|^2 \, dA \\ &\leq C_{max}^2 \sum_i \iint_{\mathcal{A}_i} \|\mathbf{x} - \mathbf{x}_i\|^2 \, dA, \end{aligned}$$

where C_i is the Lipschitz constant of the Beltrami operator over the smooth surface patch $\mathcal{A}_i$, and C_{max} the maximum of the Lipschitz constants. The Voronoi region of each sample point *by* *definition* minimizes $\|\mathbf{x} - \mathbf{x}_i\|$ since they contain the closest points to each sample, thus minimizing the bound on the error E due to spatial averaging [DFG99]. Furthermore, if we add an extra assumption on the sampling rate with respect to the curvature such that the Lipschitz constants from patch to patch vary slowly with a ratio ϵ, we can actually guarantee that the Voronoi cell borders are less than $O(\epsilon)$ away from the optimal borders. As this still holds in the limit for a triangle mesh, we use the vertices of the mesh as sample points, and pick the Voronoi cells of the vertices as associated finite-volume regions. This will guarantee optimized numerical estimates and, as we will see, determining these Voronoi cells requires few extra computations.

2.4.3 Voronoi Region Area

Given a non-obtuse triangle P, Q, R with circumcenter O, as depicted in Figure 2.2(b), we must now compute the Voronoi region for P. Using the properties of perpendicular bisectors, we find : $a+b+c = \pi/2$, and therefore, $a = \pi/2 - \angle Q$ and $c = \pi/2 - \angle R$. The Voronoi area for point P lies within this triangle if the triangle is non-obtuse, and is thus: $\frac{1}{8}(|PR|^2 cot\angle Q + |PQ|^2 cot\angle R)$. Summing these areas for the whole 1-ring neighborhood, we can write the non-obtuse Voronoi area for a vertex $\mathbf{x}_i$ as a function of the neighbors $\mathbf{x}_j$:

$$\mathcal{A}_{\text{Voronoi}} = \frac{1}{8} \sum_{j \in N_1(\mathbf{x}_i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ \|\mathbf{x}_i - \mathbf{x}_j\|^2. \tag{2.7}$$

Since the cotangent terms were already computed for Eq. (2.5), the Voronoi area can be computed very efficiently. However, if there is an obtuse triangle among the 1-ring neighbors or among the triangles edge-adjacent to the 1-ring triangles, the Voronoi region either extends beyond the 1-ring, or is truncated compared to our area computation. In either case our derived formula no longer stands.

2.4.4 Extension To Arbitrary Meshes

The previous expression for the Voronoi finite-volume area does not hold in the presence of obtuse angles. However, the integral of the Laplace-Beltrami operator given in equation (2.6) holds even for obtuse 1-ring neighborhoods – the only assumption used is that the finite-volume region goes through the midpoint of the edges. It is thus *still valid even in obtuse triangulations*.

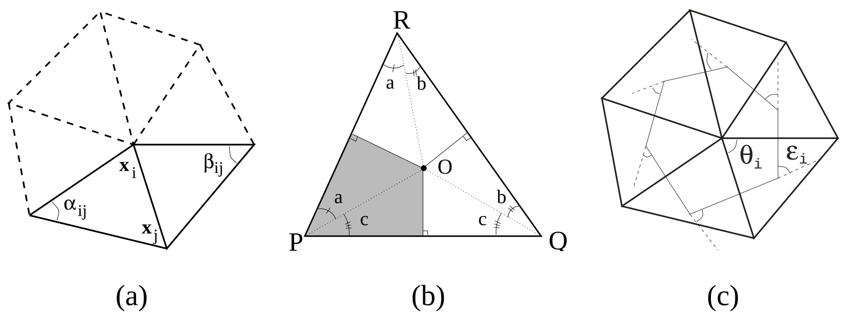


Figure 2.2: *(a) 1-ring neighbors and angles opposite to an edge; (b) Voronoi region on a non-obtuse triangle; (c) External angles of a Voronoi region.*

Therefore, we could simply divide the integral evaluation by the barycenter finite-volume area in lieu of the Voronoi area for vertices near obtuse angles to determine the spatial average value. We use a slightly more subtle area, to guarantee a perfect tiling of our surface, and therefore, optimized accuracy as each point on the surface is counted once and only once. We define a new surface area for each vertex **x**, denoted $\mathcal{A}_{\text{Mixed}}$: for each non-obtuse triangle, we use the circumcenter point, and for each obtuse triangle, we use the midpoint of the edge opposite to the obtuse angle. Algorithmically, this area around a point **x** can be easily computed as detailed in Figure 2.3. Note that the derivation for the integral of the mean curvature normal is still valid for this mixed area since the boundaries of the area remain inside the 1-ring neighborhood and go through the midpoint of each edge. Moreover, these mixed areas tile the surface without overlapping. This new cell definition is equivalent to a local adjustment of the diagonal mass matrix in a finite element framework in order to ensure a correct evaluation. The error bounds are not as tight when local angles are more than $\pi/2$, and therefore, numerical experiments are expected to be worse in areas with obtuse triangles.

2.4.5 Discrete Mean Curvature Normal Operator

Now that the mixed area is defined, we can express the mean curvature normal operator **K** defined in Section 2.1 using the following expression:

Mean Curvature Normal Operator

$$\boxed{\mathbf{K}(\mathbf{x}_i) = \frac{1}{2\mathcal{A}_{\text{Mixed}}} \sum_{j \in N_1(\mathbf{x}_i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ (\mathbf{x}_i - \mathbf{x}_j)} \tag{2.8}$$

```
$\mathcal{A}_{\text{Mixed}} = 0$
For each triangle $T$ from the 1-ring neighborhood of **x**
    If $T$ is non-obtuse,      // Voronoi safe
        // Add Voronoi formula (see Section 2.4.3)
        $\mathcal{A}_{\text{Mixed}} += $ Voronoi region of **x** in $T$
    Else                       // Voronoi inappropriate
        // Add either area($T$)/$4$ or area($T$)/$2$
        If the angle of $T$ at **x** is obtuse
            $\mathcal{A}_{\text{Mixed}} += $ area($T$)/2
        Else
            $\mathcal{A}_{\text{Mixed}} += $ area($T$)/4
```

Figure 2.3: *Pseudo-code for region $\mathcal{A}_{Mixed}$ on an arbitrary mesh*

From this expression, we can easily compute the mean curvature value κ_H by taking half of the magnitude of this last expression. As for the normal vector, we can just normalize the resulting vector $\mathbf{K}(\mathbf{x}_i)$. In the special (rare) case of zero mean curvature (flat plane or local saddle point), we simply average the 1-ring face normal vectors to evaluate **n** appropriately.

It is interesting to notice that using the barycentric area as an averaging region results in an operator very similar to the definition of the mean curvature normal by Desbrun *et al.* [DMSB99], since $\mathcal{A}_{\text{Barycenter}}$ is a third of the whole 1-ring area $\mathcal{A}_{\text{1-ring}}$ used in their derivation – however, our new derivation uses *non-overlapping* regions and is therefore more accurate. At this time, we are not aware of a proof of convergence for this operator. However, our tests have shown no divergence as we refine a mesh, as long as we do not degrade the mesh quality (the triangles must not degenerate). We will give more precise numerical results in Section 2.7.1 showing the improved quality of our new estimate.

2.5 Discrete Gaussian Curvature

In this section, the Gaussian curvature κ_G for bivariate (2D) meshes embedded in 3D is studied. We will demonstrate that a derivation similar to the above is easily obtained.

2.5.1 Expression of the Local Integral of κ_G

Similar to what was done for the mean curvature normal operator, we first need to find an exact value of the integral of the Gaussian curvature κ_G over a finite-volume region on a piecewise linear surface. From Eq. (2.3), we could compute the integral over an area $\mathcal{A}_{\text{M}}$ as the associated spherical image area (also called the image of the Gauss map). Instead, we use the *Gauss-Bonnet theorem* [DHKW92, Gra98, AZ67] which proposes a very simple equality, valid over any surface patch. Applied to our local finite-volume regions, the Gauss-Bonnet theorem simply states:

$$\iint_{\mathcal{A}_{\text{M}}} \kappa_G \, dA = 2\pi - \sum_j \epsilon_j \, ,$$

where the ϵ_j are the external angles of the boundary, as indicated in Figure 2.2(c). Note that this simplified form results from the fact that the integral of geodesic curvature on the piece-wise linear boundaries is zero. If we apply this expression to a Voronoi region, the external angles are zero across each edge (since the boundary stays perpendicular to the edge), and the external angle at a circumcenter is simply equal to θ_j, the angle of the triangle at the vertex $\mathbf{x}_i$. Therefore, the integral of the Gaussian curvature (also called total curvature) for non-obtuse triangulations is: $2\pi - \sum_j \theta_j$. This result is still valid for the mixed region and is proven using a similar geometric argument. This result was already proven by Polthier and Schmies [PS98], who considered the area of the image of the Gauss map for a vertex on a polyhedral surface. Therefore, analogous to Eq. (2.6), we can now write for the 1-ring neighborhood of a vertex $\mathbf{x}_i$:

$$\iint_{\mathcal{A}_{\text{M}}} \kappa_G dA = 2\pi - \sum_{j=1}^{\#f} \theta_j \, ,$$

where θ_j is the angle of the j-th face at the vertex $\mathbf{x}_i$, and $\#f$ denotes the number of faces around this vertex. Note again that this formula holds for any surface patch $\mathcal{A}_{\text{M}}$ within the 1-ring neighborhood whose boundary crosses the edges at their midpoint.

2.5.2 Discrete Gaussian Curvature Operator

To estimate the local spatial average of the Gaussian curvature, we use the same arguments as in 2.4.2 to claim that the Voronoi cell of each vertex is an appropriate local region to use for good error bounds. In practice, we use the mixed area $\mathcal{A}_{\text{Mixed}}$ to account for obtuse triangulations. Since the mixed area cells tile the whole surface without any overlap, we will satisfy the (continuous) Gauss-Bonnet theorem: the integral of the discrete Gaussian curvature over an entire sphere for example will be equal to 4π *whatever the discretization used* since the sphere is a closed object of genus zero. This result ensures a robust numerical behavior of our discrete operator. Our Gaussian curvature discrete operator can thus be expressed as:

Gaussian Curvature Operator

$$\boxed{\kappa_G(\mathbf{x}_i) = (2\pi - \sum_{j=1}^{\#f} \theta_j)/\mathcal{A}_{\text{Mixed}}} \tag{2.9}$$

Notice that this operator will return zero for any flat surface, as well as any roof-shaped 1-ring neighborhood, guaranteeing a satisfactory behavior for trivial cases. Note also that *convergence conditions* (using fatness or straightness) exist for this operator [Fu93, TM02], proving that if the triangle mesh does not degenerate, the approximation quality gets better as the mesh is refined. We postpone numerical tests until Section 2.7.1.

2.6 Discrete Principal Curvatures

We now wish to robustly determine the two principal curvatures, along with their associated directions. Since the previous derivations give estimates of both Gaussian and mean curvature, the only additional information that must be sought are the principal directions since the principal curvatures are, as we are about to see, easy to determine.

2.6.1 Principal Curvatures

We have seen in Section 2.1 that the mean and Gaussian curvatures are easy to express in terms of the two principal curvatures κ_1 and κ_2. Therefore, since both κ_H and κ_G have been derived for triangulated surfaces, we can define the discrete principal curvatures as:

Principal Curvature Operators

$$\boxed{\kappa_1(\mathbf{x}_i) = \kappa_H(\mathbf{x}_i) + \sqrt{\Delta(\mathbf{x}_i)}} \tag{2.10}$$

$$\boxed{\kappa_2(\mathbf{x}_i) = \kappa_H(\mathbf{x}_i) - \sqrt{\Delta(\mathbf{x}_i)}} \tag{2.11}$$

$$\text{with:}\quad \Delta(\mathbf{x}_i) = \kappa_H^2(\mathbf{x}_i) - \kappa_G(\mathbf{x}_i) \text{ and } \kappa_H(\mathbf{x}_i) = \frac{1}{2}\|\mathbf{K}(\mathbf{x}_i)\|.$$

Unlike the continuous case where Δ is always positive, we must make sure that κ_H^2 is always larger than κ_G to avoid any numerical problems, and threshold Δ to zero if it is not the case (an extremely rare occurrence).

Mean Curvature as a Quadrature

In order to determine the principal axes at a vertex, we will first show that the mean curvature from our previous expression can be interpreted as a quadrature of normal curvature samples:

$$\begin{aligned}
\kappa_H(\mathbf{x}_i) &= \frac{1}{2}\,(2\kappa_H(\mathbf{x}_i)\mathbf{n})\cdot\mathbf{n} = \frac{1}{2}\mathbf{K}(\mathbf{x}_i)\cdot\mathbf{n} \\
&= \frac{1}{4\mathcal{A}_{\text{Mixed}}}\sum_{j\in N_1(\mathbf{x}_i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ (\mathbf{x}_i - \mathbf{x}_j)\cdot\mathbf{n} \\
&= \frac{1}{4\mathcal{A}_{\text{Mixed}}}\sum_{j\in N_1(\mathbf{x}_i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}(\mathbf{x}_i - \mathbf{x}_j)\cdot\mathbf{n} \\
&= \frac{1}{\mathcal{A}_{\text{Mixed}}}\sum_{j\in N_1(\mathbf{x}_i)} \left[\frac{1}{8}(cot\ \alpha_{ij} + cot\ \beta_{ij})\ \|\mathbf{x}_i - \mathbf{x}_j\|^2\right]\kappa_{ij}^N,
\end{aligned} \tag{2.12}$$

where we define:

$$\kappa_{ij}^N = 2\frac{(\mathbf{x}_i - \mathbf{x}_j)\cdot\mathbf{n}}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}\,.$$

This κ_{ij}^N can be shown to be an estimate of the normal curvature in the direction of the edge $\mathbf{x}_i\mathbf{x}_j$. The radius R of the osculating circle going through the vertices $\mathbf{x}_i$ and $\mathbf{x}_j$ is easily found using the mean curvature normal estimate as illustrated in Figure A.1(a). Since we must have a right angle at the neighbor vertex $\mathbf{x}_j$, we have $(\mathbf{x}_i - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j - 2R\,\mathbf{n}) = 0$. This implies:

$$R = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{(2\ (\mathbf{x}_i - \mathbf{x}_j) \cdot \mathbf{n})}\ .$$

This proves that κ_{ij}^N is a normal curvature estimate in the direction of the edge $\mathbf{x}_i\mathbf{x}_j$ (as it is the inverse of the radius of the osculating circle). This expression was also used in the context of curvature approximation in [MS92] and [Tau95a].

Therefore, Eq. (2.12) can be interpreted as a quadrature of the integral from Eq. (2.1), with weights w_{ij}:

$$\kappa_H(\mathbf{x}_i) = \sum_{j \in N_1(\mathbf{x}_i)} w_{ij}\ \ \kappa_{ij}^N,$$

where the $w_{ij} = \frac{1}{\mathcal{A}_{\text{Mixed}}}\left[\frac{1}{8}(cot\ \alpha_j + cot\ \beta_j)\ \|\mathbf{x}_i - \mathbf{x}_j\|^2\right]$ sum to one for each i on a non-obtuse triangulation.

2.6.2 Least-Square Fitting for Principal Directions

In order to find the two orthogonal principal curvature directions we can simply compute the eigenvectors of the curvature tensor. Since the mean curvature obtained from our derivation can be seen as a quadrature using each edge as a sample direction, we use these samples to find the best fitting ellipse, in order to fully determine the curvature tensor. In practice, we select the symmetric curvature tensor β as being defined by three unknowns a, b, c:

$$\beta = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

This tensor will provide the normal curvature in any direction in the tangent plane. Therefore, when we use the direction of the edges of the 1-ring neighborhood, we should find:

$$\mathbf{t}_{ij}^T\ \beta\ \mathbf{t}_{ij}\ =\ \kappa_{ij}^N,$$

where $\mathbf{t}_{ij}$ is the unit direction *in the tangent plane* of the edge $\mathbf{x}_i\mathbf{x}_j$. Since we know the normal vector $\mathbf{n}$ to the tangent plane, this direction is calculated using a simple projection onto the tangent plane:

$$\mathbf{t}_{ij} = \frac{(\mathbf{x}_j - \mathbf{x}_i) - [(\mathbf{x}_j - \mathbf{x}_i) \cdot \mathbf{n}]\ \mathbf{n}}{\|(\mathbf{x}_j - \mathbf{x}_i) - [(\mathbf{x}_j - \mathbf{x}_i) \cdot \mathbf{n}]\ \mathbf{n}\|} \ .$$

A conventional least-square approximation can be obtained by minimizing the error E:

$$E(a,b,c) = \sum_j w_j \ \left(\mathbf{t}_{ij}^T \ \beta \ \mathbf{t}_{ij} - \kappa_{ij}^N \right)^2 \ .$$

Adding the two constraints $a + b = 2\kappa_H$ and $ac - b^2 = \kappa_G$, to ensure coherent results, turns the minimization problem into a root-finding problem. Once the three coefficients of the matrix B are found, we find the two principal axes $\mathbf{e}_1$ and $\mathbf{e}_2$ as the two (orthogonal) eigenvectors of β. In practice, all our experiments have demonstrated that the non-linear constraint on the determinant is not necessary (reducing the problem to a linear system). An example of these principal directions is shown in Figure 2.5(b).

Although we could actually determine the principal curvatures (and thus the mean and gaussian curvatures) using an unconstrained least squares procedure (similar to Taubin's work [Tau95a]), we use our operators to compute the curvatures and only use the least squares for the principal directions as the curvature values computed from the least squares are often *less* accurate in practice while the directions are fairly robust. A plausible interpretation for the bad numerical properties of a pure least squares approach is the hypothesis of elliptic curvature variation: although this is perfectly valid for smooth surfaces, this is somewhat arbitrary for coarse, triangulated surfaces. It seems therefore more natural to use our previous operators that rely on differential properties still valid on discrete meshes.

2.7 Operator Quality

Now that we have defined our discrete differential operators, this section examines how the operators perform numerically and visually on several representative meshes.

2.7.1 Numerical Quality of Our Operators

We performed a number of tests to demonstrate the accuracy of our approach in practice. First, we compared our operators to the well-known second-order accurate Finite Difference operators on several discrete meshes approximating simple surfaces such as spheres, or hyperboloids, where the curvatures are known analytically. In order to do so, we used special surfaces defined as height fields over a flat, regular grid so that the FD operators can be computed and tested against our results. The table below lists some representative results:

$\overline{\% error}$	FD κ_H	[DMSB99] κ_H	our κ_H	FD κ_G	our κ_G
Sphere patch	0.20	0.17	0.16	0.4	1.2
Paraboloid	0.0055	0.0038	0.0038	0.01	0.02
Torus (irregular)	-	0.047	0.036	-	0.05

Table 2.1: *Comparison of our operators with Finite Differences. The error is measured in mean percent error compared to the exact, known curvature values. Dashes "-" indicate that the FD tests cannot be performed since the triangulation is irregular. The angles θ_j needed for the Gaussian curvature were computed using the C function* `atan2`, *instead of* `acos` *or* `asin` *since* `acos` *and* `asin` *would significantly deteriorate the precision of the results.*

Overall, the numerical quality of our operators is equivalent to FD operators for regular sampling. A major advantage of our new operators over FD operators is that these differential-geometry based operators can *still* be used on irregular sampling, with the same order of accuracy.

We also tested our operators against one of the most widely used curvature estimation techniques [Tau95a]. We tested several simple surfaces (spheres, parametric surfaces, etc.) to determine the effect of sampling on the operators. The surfaces were created with 258 points, quadrisected and reprojected to create surfaces of 1026, 4098 and 16386 points. In all cases, the average percent error of our operators did not exceed 0.07% for mean curvature and 1.3% for gaussian curvature. The previous method had average errors of up to 1.8% for mean curvature and exceeding 10% in some instances for gaussian curvature.

Finally, we tested the effects of irregularity on the operators. In irregular areas of the surfaces (such as the area joining two regions of different sampling rates), our operators performed with

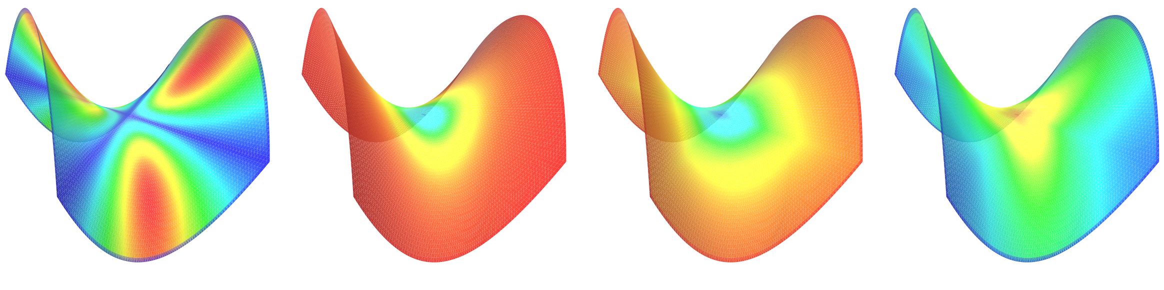


Figure 2.4: *Curvature plots of a triangulated saddle using pseudo-colors: (a) Mean, (b) Gaussian, (c) Minimum, (d) Maximum.*

the same order of accuracy as in the fairly regular regions (less than 0.2% average error for mean curvature and below 1.8% average error for gaussian curvature in regions of mild irregularity). The accuracy of our operators decreases as the irregularity (angle and edge length dispersion) increases, but, in practice, the rate at which the error increases is low.

2.7.2 Visual Inspection of Meshes

Producing high quality meshes is not an easy task. Checking if a given mesh is appropriately smooth requires a long inspection with directional or point light sources to detect any visually unpleasant discontinuities on the surface. Curvature plots (see Figure 2.4), using false color to texture the mesh according to the different curvatures, can immediately show problems or potential problems since they will reveal the variation of curvatures in an obvious way. Figure 2.5 demonstrates that even if a surface (obtained by a subdivision scheme) looks very smooth, a look at the mean curvature map reveals flaws such as discontinuities in the variation of curvature across the surface. Conversely, curvature plots can reveal unsuspected details on existing scanned meshes, like the veins on the horse. We tested our operators on a wide variety of meshes from simple geometric shapes to artist sculpted models to highly detailed scanned models. We found that our operators produced results visually consistent with the expected curvatures.

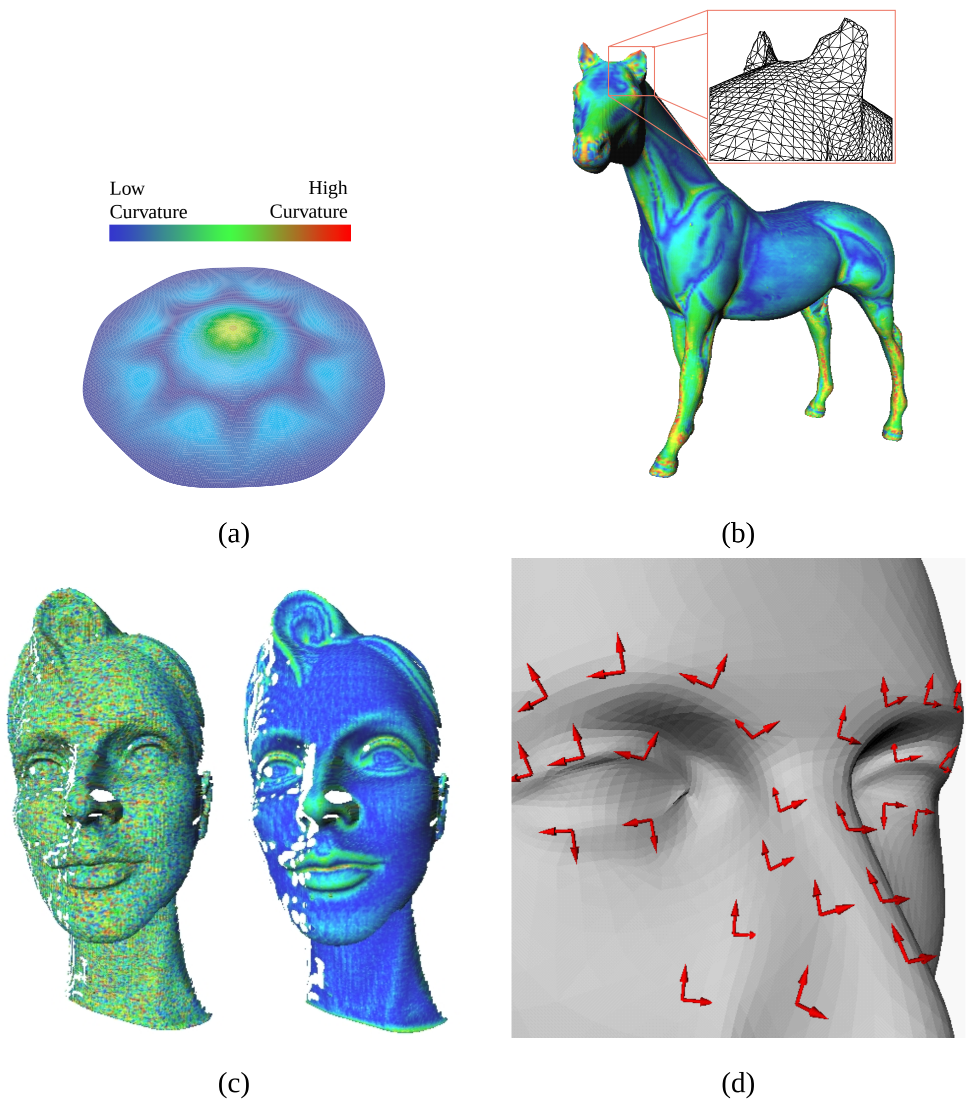


Figure 2.5: *Mean curvature plots revealing surface details for: (a) a Loop surface from an 8-neighbor ring, (b) a horse mesh, (c) a noisy mesh obtained from a 3D scanner and the same mesh after smoothing. Our operator performs well on irregular sampling such as on the ear of the horse. Notice also how the operator correctly computes quickly varying curvatures on the noisy head while returning slowly varying curvatures on the smoothed version. (d) An example of our principal directions computed on a triangle mesh.*

2.8 Discrete Operators in nD

Up to this point, we defined and used our geometric operators for bivariate (2D) surfaces embedded in 3D. We propose in this section to generalize our tools for 2D surfaces to any embedding space dimensionality, as well as extending the formulæ to 3-manifolds (volumes) in n dimensions. This will allow us to apply the same types of algorithms (smoothing techniques, etc.) on datasets such as vector fields, tensor images, or volume data.

2.8.1 Operators for 2-Manifolds in nD

We now extend our operators for 2-parameter surfaces embedded in an arbitrary dimensional space, such as color images (2D surface in 5D), or bivariate vector field (2D surface in 4D).

Beltrami Operator

As we have seen in Sections 2.1 and 2.4.1, the Beltrami operator is in the direction of surface area minimization. In order to extend this operator to higher dimensional space, we must first derive the expression for a surface area in nD. The area of a triangle formed by two vectors **u** and **v** in 3D is $2\mathcal{A} = ||\mathbf{u} \times \mathbf{v}||$. Being proportional to the sine of the angle between vectors, we can also express it as:

$$\begin{aligned} \mathcal{A} &= \frac{1}{2}||\mathbf{u}||||\mathbf{v}||sin(\mathbf{u},\mathbf{v}) = \frac{1}{2}||\mathbf{u}||||\mathbf{v}||\sqrt{1 - cos^2(\mathbf{u},\mathbf{v})} \\ &= \frac{1}{2}\sqrt{||\mathbf{u}||^2||\mathbf{v}||^2 - (\mathbf{u}\cdot\mathbf{v})^2}. \end{aligned} \tag{2.13}$$

This expression is now valid in nD, and is particularly easy to evaluate in any dimension.

We can now derive the gradient of the 1-ring area with respect to the central vertex to find the analog of Eq. (2.5) in nD. We detail this proof in Appendix A.3, but the result is very simple: the previous cotangent formula is still valid in nD if we define the cotangent between two vectors **u** and **v** as:

$$cot(\mathbf{a},\mathbf{b}) = \frac{cos(\mathbf{a},\mathbf{b})}{sin(\mathbf{a},\mathbf{b})} = \frac{\mathbf{a}\cdot\mathbf{b}}{\sqrt{||\mathbf{a}||^2||\mathbf{b}||^2 - (\mathbf{a}\cdot\mathbf{b})^2}}.$$

With this definition, the implementation in nD space is straightforward and efficient, as dot products require little computation.

Gaussian Curvature Operator

The expression of the Gaussian curvature operator Eq. (2.9) still holds in nD. Indeed, the Gaussian curvature is an intrinsic attribute of a 2-manifold, and does not depend on the embedding.

2.8.2 Beltrami Operator for 3-Manifolds in nD

We also extend the previous mean curvature normal operator, valid on triangulated surfaces, to tetrahedralized volumes which are 3-parameter volumes in an embedding space of arbitrary dimension. This can be used, for example, on any MRI volume data (intensity, vector field or even tensor fields). For these 3-manifolds, we can compute the gradient of the 1-ring volume this time to extend the Beltrami operator. Once again, the cotangent formula turns out to be still valid, but this time for the dihedral angles of the tetrahedrons. Appendix A.4 details the derivation to prove this result. This Beltrami operator can still be used to denoise volume data as it minimizes volume just as we denoised meshes through a surface area minimization.

2.9 Conclusion

A complete set of accurate differential operators for any triangulated surface has been presented. We consistently derived estimates for normal vectors and mean curvatures (Eq. (2.8)), Gaussian curvatures (Eq. (2.9)), principal curvatures (Eq. (2.10) and (2.11)), and principal directions (Section 2.6.2), and numerically showed their quality. Extended versions of our operator for surfaces and volumes in higher dimension embedding spaces have also been provided. Our operators perform as well as established methods such as Finite Differences in the regular setting and degrade gracefully as irregularity is increased. In the following chapters, we will show the practical benefits of our operators in several mesh processing algorithms that we have designed.

Chapter 3

Smoothing

Many times, a triangulated surface does not have the smoothness (or *fairness*) required for a given application. This problem has increased recently due to the use of highly detailed computer graphics objects obtained from imperfectly-measured data from the real-world. When this occurs, the mesh must be smoothed to remove undesirable noise and uneven edges while retaining desirable geometric features (see Figure 3.1). In this chapter, we use our discrete differential operators to develop methods to rapidly remove rough features from irregularly triangulated data intended to portray a smooth surface.

Our approach contains several novel features, including an *implicit integration* method to achieve efficiency, stability, and large time-steps; a scale-dependent Laplacian operator to improve the diffusion process; and finally, use of a robust curvature flow operator that achieves a smoothing of the shape itself, distinct from any parameterization. Additional features of the algorithm include automatic exact volume preservation, and hard and soft constraints on the positions of the points in the mesh. Extensions to the smoothing algorithm are also described that allow for feature preservation using anistropic smoothing and simulataneous sampling and shape smoothing using a mixture of Laplacian and curvature flow. The use of higher dimensional smoothing for images, vector fields and volumes is also explored.

We compare our method to previous operators and related algorithms, and prove that our discrete differential operators have several mathematically desirable qualities that improve the appearance of the resulting surface. Finally, we provide a series of examples to graphically and numerically demonstrate the quality of our results.

3.1 Introduction

While the mainstream approach in mesh fairing has been to enhance the smoothness of triangulated surfaces by minimizing computationally expensive functionals, Taubin [Tau95b] proposed in 1995 a signal processing approach to the problem of fairing arbitrary topology surface triangulations. This method is linear in the number of vertices in both time and memory space; large arbitrary connectivity meshes can be handled quite easily and transformed into visually appealing models. Such meshes appear more and more frequently due to the success of 3D range sensing approaches for creating complex geometry [CL96].

Taubin based his approach on defining a suitable generalization of frequency to the case of arbitrary connectivity meshes. Using a discrete approximation to the Laplacian, its eigenvectors become the “frequencies” of a given mesh. Repeated application of the resulting linear operator to the mesh was then employed to tailor the frequency content of a given mesh.

Closely related is the approach of Kobbelt [Kob97], who considered similar discrete approximations of the Laplacian in the construction of fair interpolatory subdivision schemes. In later work this was extended to the arbitrary connectivity setting for purposes of multiresolution editing [KCVS98].

The success of these techniques is largely based on their simple implementation and the increasing need for algorithms which can process the ever larger meshes produced by range sensing techniques. However, a number of issues in their application remain open problems in need of a more thorough examination.

The simplicity of the underlying algorithms is based on very basic, uniform approximations of the Laplacian. For irregular connectivity meshes this leads to a variety of artifacts such as geometric distortion during smoothing, numerical instability, problems of slow convergence for large meshes, and insufficient control over global behavior. The latter includes shrinkage problems and more precise shaping of the frequency response of the algorithms.

In this chapter we consider more carefully the question of numerical stability by observing that Laplacian smoothing can be thought of as time integration of the heat equation on an irreg-

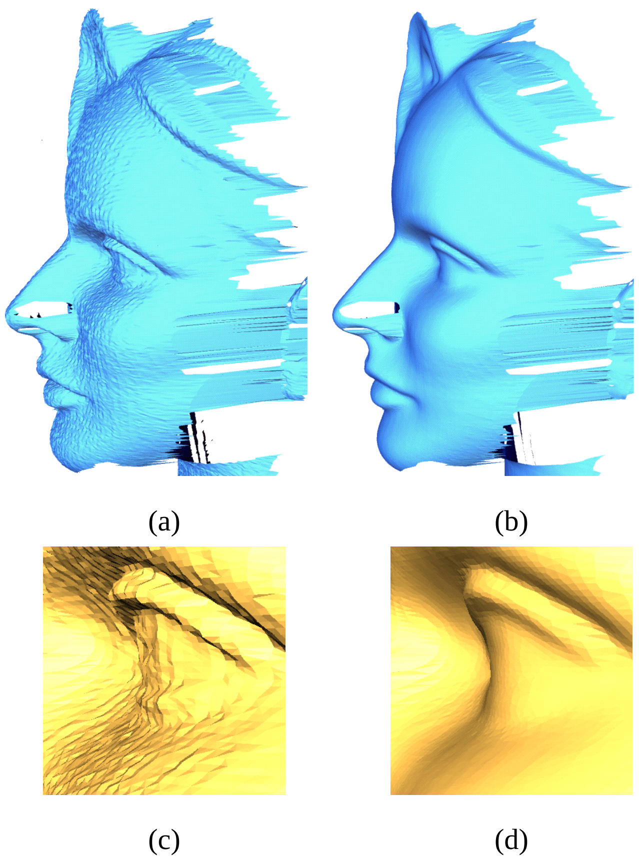


Figure 3.1: *(a): Original 3D photography mesh (41,000 vertices). (b): Smoothed version with the scale-dependent operator in two integration step with* $\lambda dt = 5 \cdot 10^{-5}$*, the iterative linear solver (PBCG) converges in 10 iterations. (c),(d): Close-ups of the eye. All the images in this chapter are flat-shaded to enhance the faceting effect.*

ular mesh. This suggests the use of *implicit integration* schemes which lead to unconditionally stable algorithms allowing for very large time steps. At the same time the necessary linear system solvers run faster than explicit approaches for large meshes. We also consider the question of mesh parameterization more carefully and propose the use of discretizations of the Laplacian which take the underlying parameterization into account. The resulting algorithms avoid many of the distortion artifacts resulting from the application of previous methods. We demonstrate that this can be done at only a modest increase in computing time and results in smoothing algorithms with considerably higher geometric fidelity. Finally a more careful analysis of the underlying discrete differential geometry is used to derive a curvature flow approach, using our previously defined discrete differential operators, which satisfies crucial geometric properties.

We detail how these different operators act on meshes, and how users can then decide which one is appropriate in their case. If the user wants to, at the same time, smooth the shape of an object and equalize its triangulation, a scale-dependent diffusion must be used. On the other hand, if only the shape must be filtered without affecting the sampling rate, then curvature flow has all the desired properties. This allows us to propose a novel class of efficient smoothing algorithms for arbitrary connectivity meshes. Using this family of smoothing algorithms as a base, we define several extensions including anisotropic smoothing to retain the features of the mesh while reducing the noise, sampling regularization during the shape smoothing, and the smoothing of other data types such as images, vector fields and volume data.

3.2 Implicit Fairing

In this section, we introduce *implicit fairing*, an implicit integration of the diffusion equation for the smoothing of meshes. We will demonstrate several advantages of this approach over the usual explicit methods. While this section is restricted to the use of a linear approximation of the diffusion term, implicit fairing will be used as a robust and efficient numerical method throughout the chapter, even for non-linear operators. We start by setting up the framework and defining our notation.

3.2.1 Notation and Definitions

In the remainder of this chapter, $\mathbf{X}$ will denote a mesh, $\mathbf{x}_i$ a vertex of this mesh, and e_{ij} the edge (if existing) connecting $\mathbf{x}_i$ to $\mathbf{x}_j$. As in chapter 2, we will call $N_1(\mathbf{x}_i)$ the “neighbors” (or 1-ring neighbors) of $\mathbf{x}_i$, i.e., all the vertices $\mathbf{x}_j$ such that there exists an edge e_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j$ (see Figure 3.2).

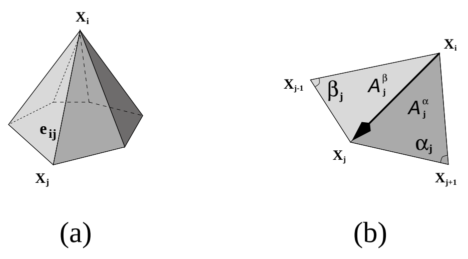


Figure 3.2: *(a) A vertex $\mathbf{x}_i$ and its adjacent faces, (b) one term of its curvature normal formula.*

In the surface fairing literature, most techniques use constrained energy minimization. For this purpose, different fairness functionals have been used. The most frequent functional is the total curvature of a surface $\mathcal{S}$:

$$\mathcal{E}(\mathcal{S}) = \int_{\mathcal{S}} \kappa_1^2 + \kappa_2^2 \, d\mathcal{S}. \tag{3.1}$$

This energy can be estimated on discrete meshes [WW94, Kob97] by fitting local polynomial interpolants at vertices. However, principal curvatures κ_1 and κ_2 depend non-linearly on the surface $\mathcal{S}$. Therefore, many practical fairing methods prefer the membrane functional or the thin-plate functional of a mesh **X**:

$$\mathcal{E}_{membrane}(\mathbf{X}) = \frac{1}{2} \int_{\Omega} \mathbf{X}_u^2 + \mathbf{X}_v^2 \; dudv \tag{3.2}$$

$$\mathcal{E}_{thin\ plate}(\mathbf{X}) = \frac{1}{2} \int_{\Omega} \mathbf{X}_{uu}^2 + 2\,\mathbf{X}_{uv}^2 + \mathbf{X}_{vv}^2 \; dudv. \tag{3.3}$$

Note that the thin-plate energy turns out to be equal to the total curvature only when the parameterization (u, v) is isometric. Their respective variational derivatives corresponds to the Laplacian and the second Laplacian:

$$\mathcal{L}(\mathbf{X}) = \mathbf{X}_{uu} + \mathbf{X}_{vv} \tag{3.4}$$

$$\mathcal{L}^2(\mathbf{X}) = \mathcal{L} \circ \mathcal{L}(\mathbf{X}) = \mathbf{X}_{uuuu} + 2\,\mathbf{X}_{uuvv} + \mathbf{X}_{vvvv}. \tag{3.5}$$

For smooth surface reconstruction in vision, a weighted average of these derivatives has been used to fair surfaces [Ter88]. For meshes, Taubin [Tau95b] used signal processing analysis to show that a combination of these two derivatives of the form: $(\lambda + \mu)\mathcal{L} - \lambda\mu\mathcal{L}^2$ can provide a Gaussian filtering that minimizes shrinkage. The constants λ and μ must be tuned by the user to obtain this non-shrinking property. We will refer to this technique as the $\lambda|\mu$ algorithm.

3.2.2 Diffusion Equation for Mesh Fairing

As stated above, one common way to attenuate noise in a mesh is through a *diffusion process*:

$$\frac{\partial \mathbf{X}}{\partial t} = \lambda \mathcal{L}(\mathbf{X}). \tag{3.6}$$

By integrating Equation 3.6 over time, a small disturbance will disperse rapidly in its neighborhood, smoothing the high frequencies, while the main shape will be only slightly degraded. The Laplacian operator can be linearly approximated at each vertex by the umbrella operator (we will use this approximation in the current section for the sake of simplicity, but will discuss its validity in section 3.4), as used in [Tau95b, KCVS98]:

$$\mathcal{L}(\mathbf{x}_i) = \frac{1}{m} \sum_{j \in N_1(\mathbf{x}_i)} \mathbf{x}_j - \mathbf{x}_i, \tag{3.7}$$

where $\mathbf{x}_j$ are the neighbors of the vertex x_i, and $m = \#N_1(\mathbf{x}_i)$ is the number of these neighbors (valence). A sequence of meshes $(\mathbf{X}^n)$ can be constructed by integrating the diffusion equation with a simple *explicit Euler* scheme, yielding:

$$\mathbf{X}^{n+1} = (I + \lambda dt \mathcal{L})\mathbf{X}^n. \tag{3.8}$$

With the umbrella operator, the stability criterion requires $\lambda dt < 1$. If the time step does not satisfy this criterion, ripples appear on the surface, and often end up creating oscillations of growing magnitude over the whole surface. On the other hand, if this criterion is met, we get smoother and smoother versions of the initial mesh as n grows.

3.2.3 Time-Shifted Evaluation

The implementation of this previous explicit method, called *forward Euler method*, is very straightforward [Tau95b] and has nice properties such as linear time and linear memory size for each filtering pass. Unfortunately, when the mesh is large, the time step restriction results in the need to perform hundreds of integrations to produce a noticeable smoothing, as mentioned in [KCVS98].

Implicit integration offers a way to avoid this time step limitation. The idea is simple: if we approximate the derivative using the new mesh (instead of using the old mesh as done in explicit methods), we will get to the equilibrium state of the PDE faster. As a result of this time-shifted evaluation, stability is obtained unconditionally [PFTV94]. The integration is now: $\mathbf{X}^{n+1} = \mathbf{X}^n + \lambda dt \mathcal{L}(\mathbf{X}^{n+1})$. Performing an implicit integration, this time called *backward Euler method*, thus means solving the following linear system:

$$(I - \lambda dt \mathcal{L})\mathbf{X}^{n+1} = \mathbf{X}^n. \tag{3.9}$$

This apparently minor change allows the user not to worry about practical limitations on the time step. Consequent smoothing will then be obtained safely by increasing the value λdt. However, we now must solve a linear system.

3.2.4 Solving the Sparse Linear System

Fortunately, this linear system can be solved efficiently as the matrix $\mathbf{A} = I - \lambda dt \mathcal{L}$ is sparse: each line contains approximately seven non-zero elements if the Laplacian is expressed using Eq. (3.7) since the average number of neighbors on a typical triangulated mesh is six. We can use a preconditioned bi-conjugate gradient (PBCG) to iteratively solve this system with great efficiency[1]. The PBCG is based on matrix-vector multiplies [PFTV94], which only require linear time computation in our case thanks to the sparsity of the matrix **A**. We review in Appendix A.5 the different options we chose for the PBCG in order to have an efficient implementation for our purposes.

3.2.5 Interpretation of the Implicit Integration

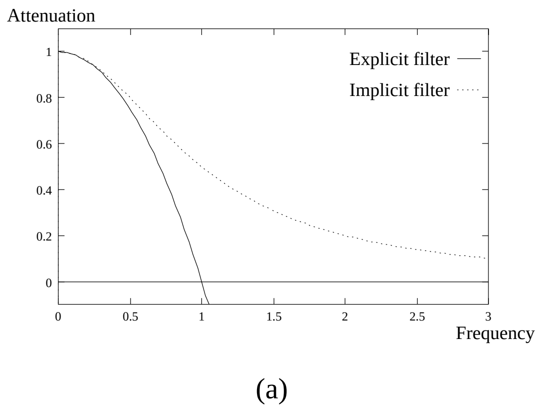
Although this implicit integration for diffusion is sound as is, there are useful connections with other prior work. We review the analogies with signal processing approaches and physical simulation.

[1]We use a bi-conjugate gradient method to be able to handle non-symmetric matrices, to allow the inclusion of constraints (see Section 3.2.7).

Signal Processing

In [Tau95b], Taubin presents the explicit integration of diffusion with a signal processing point of view. Indeed, if $\mathbf{X}$ is a 1D signal of a given frequency ω: $\mathbf{X} = e^{i\omega}$, then $\mathcal{L}(\mathbf{X}) = -\omega^2\mathbf{X}$. Thus, the transfer function for Eq. (3.8) is $1 - \lambda dt\omega^2$, as displayed in Figure 3.3(a) as a solid line. We can see that the higher the frequency ω, the stronger the attenuation will be, as expected.

The previous filter is called FIR (for Finite Impulse Response) in signal processing. When the diffusion process is integrated using implicit integration, the filter in Eq. (3.9) turns out to be an Infinite Impulse Response filter. Its transfer function is now $1/(1 + \lambda dt\omega^2)$, depicted in Figure 3.3(a) as a dashed line. Because this filter is always in $[0, 1]$, we have unconditional stability.



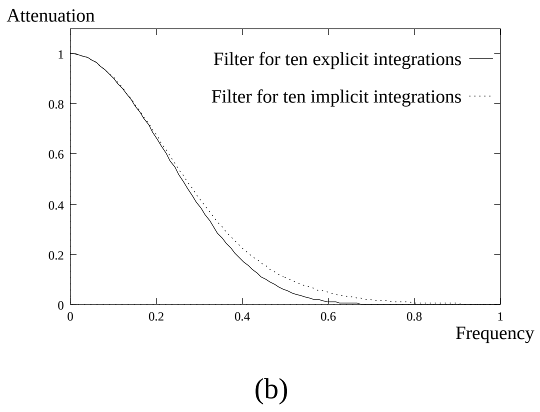


Figure 3.3: *Comparison between (a) the explicit and implicit transfer function for $\lambda dt = 1$, and (b) their resulting transfer function after 10 integrations.*

By rewriting Eq. (3.9) as: $\mathbf{X}^{n+1} = (I - \lambda dt\mathcal{L})^{-1}\mathbf{X}^n$, we also note that our implicit filtering is equivalent to $I + \lambda dt\mathcal{L} + (\lambda dt)^2\mathcal{L}^2 + ...$, i.e., standard explicit filtering plus an infinite sequence of higher-order filtering. Contrary to the explicit approach, one single implicit filtering step performs global filtering.

Mass-Spring Network

Smoothing a mesh by minimizing the membrane functional can be seen as a physical simulation of a mass-spring network with zero-rest length springs that will shrink to a single point in the limit. Recently, Baraff and Witkin [BW98] presented an implicit method to allow large time steps in cloth simulation. They found that the use of an implicit solver instead of the traditional explicit Euler integration considerably improves computational time while still being stable for very stiff systems. Our method is analogous to theirs, but used for a different PDE. We therefore have the same advantages of using an implicit solver over the usual explicit type: *stability* and *efficiency* when significant filtering is called for.

3.2.6 Filter Improvement

Now that the method has been described for the standard diffusion equation, we can consider other equations that may be more appropriate or may give better visual results for smoothing when we use implicit integration.

We have seen in Section 3.2.1 that both $\mathcal{L}$ and $\mathcal{L}^2$ have been used with success in prior work [Ter88, Tau95b, KCVS98]. When we use implicit integration, as Figure 3.4(a) shows, the higher the power of the Laplacian, the closer to a *low-pass filter* we get. In terms of frequency analysis, it is a better filter. Unfortunately, the matrix becomes less and less sparse as more and more neighbors are involved in the computation. In practice, we find that $\mathcal{L}^2$ is a very good trade-off between efficiency and quality. Using higher-orders affects the computational time significantly, while not always producing significant improvements. We therefore recommend using $(I + \lambda dt\mathcal{L}^2)\mathbf{X}^{n+1} = \mathbf{X}^n$ for implicit smoothing (a precise definition of the umbrella-like operator for $\mathcal{L}^2$ can be found in [KCVS98]).

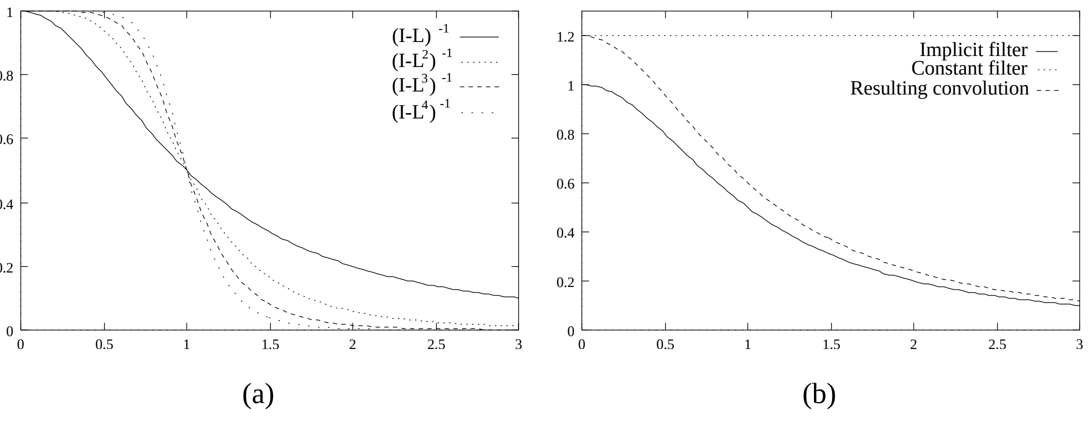


Figure 3.4: *(a): Comparison between filters using $\mathcal{L}$, $\mathcal{L}^2$, $\mathcal{L}^3$, and $\mathcal{L}^4$. (b): The scaling to preserve volume creates an amplification of all frequencies; but the resulting filter (diffusion+scaling) only amplifies low frequencies to compensate for the shrinking of the diffusion.*

We also tried to use a linear combination of both $\mathcal{L}$ and $\mathcal{L}^2$. We obtained interesting results like, for instance, amplification of low or middle frequencies to exaggerate large features (refer to [GSS99] for a complete study of feature enhancement). It is not appropriate in the context of a fixed mesh, though: amplifying frequencies requires refinement of the mesh to offer a good discretization.

3.2.7 Constraints

We can put hard and soft constraints on the mesh vertex positions during the diffusion. For the user, it means that a vertex or a set of vertices can be fixed so that the smoothing happens only on the rest of the mesh. This can be very useful to retain certain details in the mesh.

A vertex $\mathbf{x}_i$ will stay fixed if we impose $\mathcal{L}(\mathbf{x}_i) = 0$ (or more correctly $\lambda = 0$). More complicated constraints are also possible [BW98]. For example, vertices can be constrained along an axis or on a plane by modifying the PBCG to keep these constraints enforced during the linear solver iterations.

We can also easily implement *soft constraints*: each vertex can be weighted according to the desired smoothing that we want. For instance, the user may want to smooth a part of a mesh less than another one, in order to keep desirable features while getting a smoother version. We allow the assignment of a smoothing value between 0 and 1 to attenuate the smoothing spatially: this is equivalent to choosing a variable λ factor on the mesh, and happens to be very useful in practice. Entire regions can be "spray painted" interactively to easily assign this special factor.

3.2.8 Discussion

Even though adding a linear solver step to the integration of the diffusion equation would appear to slow down the problem at first glance, it turns out that we gain significantly by doing so. For instance, the implicit integration can be performed with an arbitrary time step. Since the matrix of the system is very sparse, we actually obtain computational time similar or better than the explicit methods. In the following table, we indicate the number of iterations of the PBCG method for different meshes and it can be seen that the PBCG is more efficient when the smoothing is high. These timings were performed on an SGI High Impact Indigo2 175MHz R10000 processor with 128M RAM.

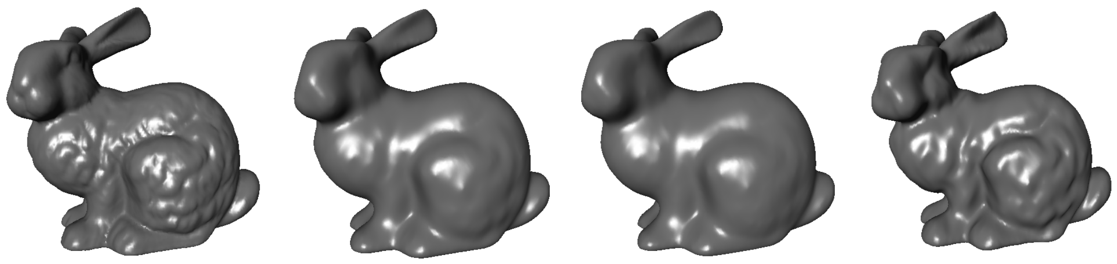


Figure 3.5: *Stanford bunnies: (a) The original mesh, (b) 10 explicit integrations with* $\lambda dt = 1$*, (c) 1 implicit integration with* $\lambda dt = 10$ *that takes only* 7 *PBCG iterations (30% faster), and (d) 20 passes of the* $\lambda|\mu$ *algorithm, with* $\lambda = 0.6307$ *and* $\mu = -0.6732$*. The implicit integration results in better smoothing than the explicit one for the same, or often less, computing time. If volume preservation is called for, our technique then requires many fewer iterations to smooth the mesh than the* $\lambda|\mu$ *algorithm.*

Mesh	Nb of faces	$\lambda dt = 10$	$\lambda dt = 100$
Horse	42,000	8 iterations (2.86s)	37 iterations (12.6s)
Dragon	42,000	8 iterations (2.98s)	39 iterations (13.82s)
Isis	50,000	9 iterations (3.84s)	37 iterations (15.09s)
Bunny	66,000	7 iterations (4.53s)	35 iterations (21.34s)
Buddha	290,000	5 iterations (13.78s)	28 iterations (69.93s)

To be able to compare the results with the explicit method, one has to notice that one iteration of the PBCG is only slightly more time-consuming than one integration step using an explicit method. Therefore, we can see in the following results that our implicit fairing takes about 60% less time than the explicit fairing for a filtering of $\lambda dt = 100$, as we get about 33 iterations compared to the 100 integration steps required in the explicit case. We have found this behavior to be true for all the other meshes as well. The advantage of the implicit method in terms of computational speed becomes more obvious for *large meshes* and/or *high smoothing* value. In terms of quality, Figures 3.5(b) and 3.5 (c) demonstrate that both implicit and explicit methods produce about the same visual results, with a slightly better smoothness for the implicit fairing. Note that we use 10 explicit integrations of the umbrella operator with $\lambda dt = 1$, and 1 integration using the implicit integration with $\lambda dt = 10$ to approximate the same results. Therefore, there is a definite advantage in the use of implicit fairing over the previous explicit methods. Moreover, the remainder of this paper will make heavy use of this method and its stability properties.

3.3 Automatic Anti-Shrinking Fairing

Pure diffusion will, by nature, induce shrinkage. This is inconvenient as this shrinking may be significant for aggressive smoothing. Taubin proposed to use a linear combination of $\mathcal{L}$ and $\mathcal{L} \circ \mathcal{L}$ to amplify low frequencies in order to balance the natural shrinking. Unfortunately, the linear combination depends heavily on the mesh in practice, and this requires fine tuning to ensure both stable and non-shrinking results. In this section, we propose an automatic solution to avoid this shrinking. We preserve the zeroth moment, i.e., the volume, of the object. Without any other information on the mesh, we feel it is the most reasonable invariant to preserve, although surface area or other invariants can be used.

Volume Computation

As we have a mesh given in terms of triangles, it is easy to compute the interior volume. This can be done by summing the volumes of all the oriented pyramids centered at a point in space (the origin, for instance) and with a triangle of the mesh as a base. This computation has a linear complexity in the number of triangles [LK84]. For the reader's convenience, we give the expression of the volume of a mesh in the following equation, where x_k^1, x_k^2 and x_k^3 are the three vertices of the kth triangle:

$$V = \frac{1}{6} \sum_{k=1}^{nbFaces} \mathbf{g}_k \cdot \mathbf{N}_k \, , \tag{3.10}$$

where $\mathbf{g} = (\mathbf{x}_k^1 + \mathbf{x}_k^2 + \mathbf{x}_k^3)/3$ and $\mathbf{N}_k = \overrightarrow{\mathbf{x}_k^1\mathbf{x}_k^2} \wedge \overrightarrow{\mathbf{x}_k^1\mathbf{x}_k^3}$.

3.3.1 Exact Volume Preservation

After an integration step, the mesh will have a new volume V^n. We then want to scale it back to its original volume V^0 to cancel the shrinking effect. We apply a simple scale on the vertices to achieve this. By multiplying all the vertex positions by $s = (V^0/V^n)^{1/3}$, the volume is guaranteed to go back to its original value. As this is a simple scaling, it is harmless in terms of frequencies. To put it differently, this scaling corresponds to a convolution with a scaled Dirac in the frequency domain, hence it amplifies all the frequencies in the same way to change the volume back. The resulting filter, after the implicit smoothing and the constant amplification filter, amplifies the low frequencies of the original mesh to *exactly* compensate for the attenuation of the high frequencies, as sketched on Figure 3.4(b).

The overall complexity for volume preservation is thus linear. With such a process, we do not need to tweak parameters: the anti-shrinking filter is *automatically* adapted to the mesh and to the smoothing, contrary to previous approaches. Note that hard constraints defined in the previous section are applied before the scaling and do not result in fixed points anymore: scaling alters the absolute, but not the relative position.

We can generalize this re-scaling phase to different invariants. For instance, if we have to smooth height fields, it is more appropriate to take the invariant as being the volume enclosed between the height field and a reference plane, which changes the computations only slightly. Likewise, for surfaces of revolution, we may change the way the scaling is computed to exploit this special property. We can also preserve the surface area if the mesh is a non-closed surface. However, in the absence of specific characteristics, preserving the volume gives nice results. According to specific needs, the user can select the appropriate type of invariant to be used.

3.3.2 Discussion

When we combine both methods of implicit integration and anti-shrinking convolution, we obtain an automatic and efficient method for fairing. Indeed, no parameters need be tuned to ensure stability or to have exact volume preservation. This is a major advantage over previous techniques. Yet, we retain all of the advantages of previous methods, such as constraints [Tau95b] and the possibility of accelerating the fairing via multigrid [KCVS98], while additionally offering stability and efficiency. This technique also dramatically reduces the computing time over Taubin's anti-shrinking algorithm: as demonstrated in Figures 3.5(c) and 3.5(d), using the $\lambda|\mu$ algorithm may preserve the volume after fine tuning, but one iteration will only slightly smooth the mesh. The rest of this paper exploits both automatic anti-shrinking and implicit fairing techniques to offer more accurate tools for fairing.

3.4 An Accurate Diffusion Process

Up to this section, we have relied on the umbrella operator (Eq. (3.7)) to approximate the Laplacian on a vertex of the mesh. This particular operator does not truly represent a Laplacian in the physical meaning of this term as we are about to see. Moreover, simple experiments on smooth meshes show that this operator, using explicit or implicit integration, can create bumps or “pimples” on the surface, instead of smoothing it. This section proposes a sounder simulation of the diffusion process, by defining a new approximation for the Laplacian and by taking advantage of the implicit integration.

3.4.1 Inadequacy of the Umbrella Operator

The umbrella operator, used in the previous sections, corresponds to an approximation of the Laplacian in the case of a specific parameterization [KCVS98]. This means that the mesh is supposed to have edges of length 1 and all the angles between two adjacent edges around a vertex should be equal. This is of course far from being true in actual meshes, which contain a variety of triangles of different sizes.

Treating all edges as if they had equal length has significant undesired consequences for the smoothing. For example, the Laplacian can be the same for two very different configurations, corresponding to different frequencies as depicted in Figure 3.6. This distorts the filtering significantly, as high frequencies may be considered as low ones, and vice versa. Nevertheless, the advantage of the umbrella operator is that it is normalized: the time step for integration is always 1, which is very convenient. But we want a more accurate diffusion process to smooth meshes consistently, in order to more carefully separate high from low frequencies.

We need to define a discrete Laplacian which is scale dependent, to better approximate diffusion. However, if we use explicit integration [Tau95b], we will suffer from a very restricted stability criterion. It is well known [PFTV94] that the time step for a parabolic PDE like Eq. (3.6) depends on the square of the smallest length scale (here, the smallest edge length $min(|e|)$):

$$dt \leq \frac{min(|e|)^2}{2\ \lambda}\ .$$

This limitation is a real concern for large meshes with small details, since an enormous number

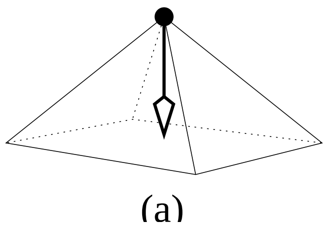


Figure 3.6: *Frequency confusion: the umbrella operator is evaluated as the vector joining the center vertex to the barycenter of its neighbors. Thus, cases (a) and (b) will have the same approximated Laplacian even if they represent different frequencies.*

of integration steps will have to be performed to obtain noticeable smoothing. This is *intractable* in practice.

Using the implicit integration described in Section 3.2, we can overcome this restriction and use a much larger time step while still achieving good smoothing, saving considerable computation. In the next two subsections we present one design of a good approximation for the Laplacian.

3.4.2 Simulation of the 1D Heat Equation

The 1D case of a diffusion equation corresponds to the heat equation $x_t = x_{uu}$. It is therefore worth considering this example as a test problem for higher dimensional filtering. To do so, we use Milne's test presented in [Mil95]. Milne compared two cases of the same initial problem: first, the problem is solved on a regular mesh on $[0, 1]$, and then on an irregular mesh, taken to consist of a uniform coarse grid of cells on $[0, 1]$ with each of the cells in $[\frac{1}{2}, 1]$ subdivided into two fine cells as depicted in Figure 3.7(a) and 3.7(b). With such a configuration, classical finite difference coefficients for second derivatives can be used on each cell, except for the middle one which does not have centered neighbors. Milne shows that if no particular care is taken for this "peripheral" cell, it introduces a *noise term* that creates large inaccuracies — larger than if the mesh was represented uniformly at the coarser resolution! But if we fit a quadratic spline at this cell to approximate the second derivative, then the noise source disappears and we get more accurate results than with a constant coarse resolution (see the errors created in each case in one iteration of the heat equation in Figure 3.7(c)).

This actually corresponds to the extension of finite difference computations for irregular meshes proposed by Fornberg [For88]: to compute the FD coefficients, just fit a quadratic function at the sample point and its two immediate neighbors, and then return the first and second derivative of that function as the approximate derivatives. For three points spaced Δ and δ apart (see Figure 3.7(d)), we get the 1D formula:

$$(x_{uu})_i = \frac{2}{\delta + \Delta}\left(\frac{x_{i-1} - x_i}{\delta} + \frac{x_{i+1} - x_i}{\Delta}\right).$$

Note that when $\Delta = \delta$, we find the usual finite difference formula.

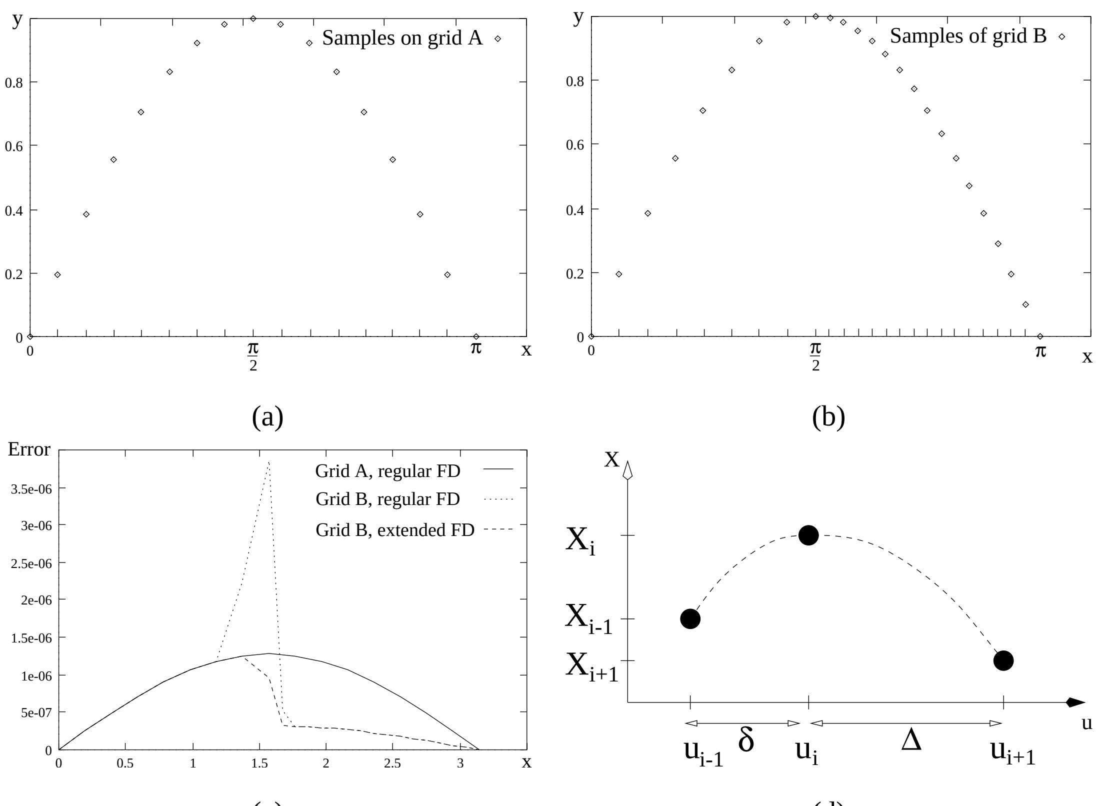


Figure 3.7: *Test on the heat equation: (a) regular sampling vs. (b) irregular sampling. Numerical errors in one step of integration (c): using the usual FD weight on an irregular grid to approximate second derivatives creates noise, and gives a worse solution than on the coarse grid, whereas extended FD weights offer the expected behavior. (d) Three unevenly spaced samples of a function and corresponding quadratic fitting for extended FD weights.*

3.4.3 Extension to 3D

The umbrella operator suffers from this problem of large inaccuracies for irregular meshes as the same assumed constant parameterization is used (Figure 3.8 shows such a behavior). Surprisingly, a simple generalization of the previous formula valid in 1D corresponds to a known approximation of the Laplacian. Indeed, Fujiwara [Fuj95] presents the following formula:

$$\mathcal{L}(\mathbf{x}_i) = \frac{1}{E} \sum_{j \in N_1(\mathbf{x}_i)} \frac{\mathbf{x}_j - \mathbf{x}_i}{|e_{ij}|}, \quad \text{with } E = \sum_{j \in N_1(\mathbf{x}_i)} |e_{ij}|. \tag{3.11}$$

where $|e_{ij}|$ is the length of the edge e_{ij}. Note that, when all edges are of size 1, this reduces to the umbrella operator (Eq. 3.7). We will then denote this new operator as the *scale-dependent umbrella operator*.

Unfortunately, the operator is no longer linear. But during a typical smoothing, the length of the edges does not change dramatically. We thus make the approximation that the coefficients of the matrix $\mathbf{A} = (I - \lambda dt\mathcal{L})$ stay constant during an integration step. We can compute them initially using the current edges' lengths and keep their values constant during the PBCG iterations. In practice, we have not noted any noticeable drawbacks from this linearization. We can even keep the same coefficients for a number of (or all) iterations: it will correspond to a filtering "relative" to the initial mesh instead if the current mesh. For the same reason as before, we also recommend the use of the second Laplacian for higher quality smoothing without significant increase in computation time. As demonstrated in Figure 3.8, the scale-dependent umbrella operator deals better with irregular meshes than the umbrella operator: no spurious artifacts are created. We also applied this operator to noisy data sets from 3D photography to obtain smooth meshes (see Figures 3.1 and 3.12).

The number of iterations needed for convergence depends heavily on the ratio between minimum and maximum edge lengths. For typical smoothing and for meshes over 50000 faces, the average number of iterations we get is 20. Nevertheless, we still observe undesired behavior on flat surfaces: vertices in flat areas still slide during smoothing. Even though this last formulation generally reduces this problem, we may want to keep a flat area *intact*. The next section tackles this problem with a new approach.

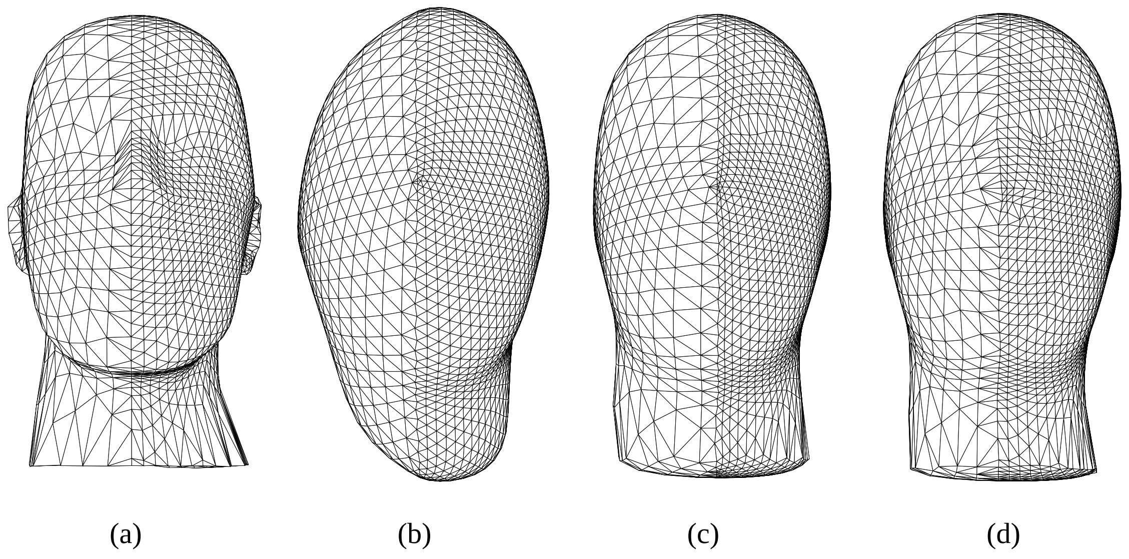


Figure 3.8: *Application of operators to a mesh: (a) mesh with different sampling rates, (b) the umbrella operator creates a significant distortion of the shape, but (c) with the scale-dependent umbrella operator, the same amount of smoothing does not create distortion or artifacts, almost like (d) when curvature flow is used. The small features such as the nose are smoothed but stay in place.*

3.5 Curvature Flow for Noise Removal

In terms of differential equations, diffusion is a close relative of curvature flow. In fact, the directions of Laplacian and curvature flows coincide for the *conformal* parameter space [DHKW92]. Thus we can interpret the mean curvature normal as a special laplacian: it is a laplacian for a parameter space naturally *induced by the surface itself*. In this section, we first explore the advantages of using curvature flow over diffusion, and then propose an efficient algorithm for noise removal using curvature flow.

3.5.1 Diffusion vs. Curvature Flow

The Laplacian of the surface at a vertex has both normal and tangential components. Even if the surface is locally flat, the Laplacian approximation will rarely be the zero vector [KCVS98]. This introduces undesirable drifting over the surface, depending on the parameterization we assume. We in effect fair the parameterization (or sampling) of the surface as well as the shape itself (see

Figure 3.10(b)).

We would prefer to have a noise removal procedure that does not depend on the parameterization. It should use only *intrinsic properties* of the surface. This is precisely what curvature flow does. Curvature flow smoothes the surface by moving along the surface normal **n** with a speed equal to the mean curvature: κ_H:

$$\frac{\partial \mathbf{x}_i}{\partial t} = -\kappa_H(\mathbf{x}_i)\ \mathbf{n}_i. \tag{3.12}$$

Using curvature flow, a sphere with different sampling rates should stay spherical under curvature flow as the curvature is constant. And we should also not get any vertex "sliding" when an area is flat as the mean curvature is then zero.

There are already different approaches using curvature flow [Set96], and even mixing both curvature flow and volume preservation [DCG98] to smooth object appearance, but mainly in the context of level-set methods. They are not usable on a mesh as is. However, we can use our discrete differential defined in section 2.4, repeated here for convenience:

$$\mathbf{K}(\mathbf{x}_i) = \ \frac{1}{2\mathcal{A}_{\text{Mixed}}} \sum_{j \in N_1(i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ \ (\mathbf{x}_i - \mathbf{x}_j). \tag{3.13}$$

Note that this equation for the mean curvature normal is equivalent to the gradient of surface area with respect to the position of $\mathbf{x}_i$.

$$\kappa_H\ \mathbf{n} = \frac{\nabla \mathcal{A}}{2\ \mathcal{A}}\,. \tag{3.14}$$

This discrete flow is thus an area-minimizing flow producing a minimal surface. Note the interesting similarity with [PP93]. We obtain almost the same equation, but with a completely different derivation than theirs, which was using energies of linear maps.

Using the area gradient property of our operator, it is easy to see that we will have a zero curvature normal vector for a flat area. As shown in Figure 3.9, we see that moving the center vertex x_i on a flat surface does not change the surface area. On the other hand, moving it above or below the plane will always increase the local area. Hence, we have the desired property of a null area gradient for a locally flat surface, whatever the valence, the aspect ratio of the adjacent faces, or the edge lengths around the vertex.

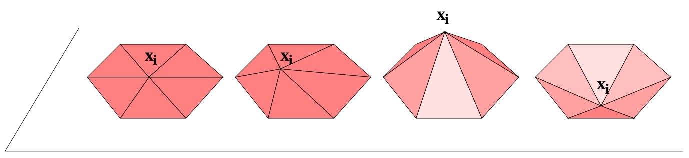


Figure 3.9: *The area around a vertex x_i lying in the same plane as its 1-ring neighbors does not change if the vertex moves within the plane, and can only increase otherwise. Being a local minimum, it thus proves that the derivative of the area with respect to the position of x_i is zero for flat regions.*

3.5.2 Boundaries

For non-closed surfaces or surfaces with holes, we can define a special treatment for vertices on boundaries. The notion of mean curvature does not make sense for such vertices. Instead, we would like to smooth the boundary, so that the shape of the hole itself gets rounder and rounder as iterations go. We can then use for instance Eq. (3.11) restricted to the two immediate neighbors which will smooth the boundary curve itself.

Another possible technique is to create a virtual vertex, stored but not displayed, initially placed at the barycenter of all the vertices placed on a closed boundary. A set of faces adjacent to this vertex and connecting the boundary vertices one after the other are also virtually created. We can then use the basic algorithm without any special treatment for the boundary as now, each vertex has a closed area around it.

3.5.3 Implementation

Similarly to Section 3.4, we have a non-linear expression defining the curvature normal. We can proceed in the same way, holding the operator constant over each time step, as the changes induced in a time step will be small. We simply compute the non-zero coefficients of the matrix $I - \lambda dt K$, where K represents the matrix of the curvature normals. We then successively solve the following linear system:

$$(I - \lambda dt K)\, \mathbf{X}^{n+1} = \mathbf{X}^n.$$

We can use preconditioning or constraints, just as before as everything is basically the same except for the local approximation of the speed of smoothing. As shown on Figure 3.10, a sphere with different triangle sizes will remain the same sphere thanks to both the curvature flow and the volume preservation technique.

In order for the algorithm to be robust, an important test must be performed while the matrix K is computed: if we encounter a face of zero area, we skip it. Mesh decimation to eliminate all degenerate triangles can also be used as suggested in [PP93].

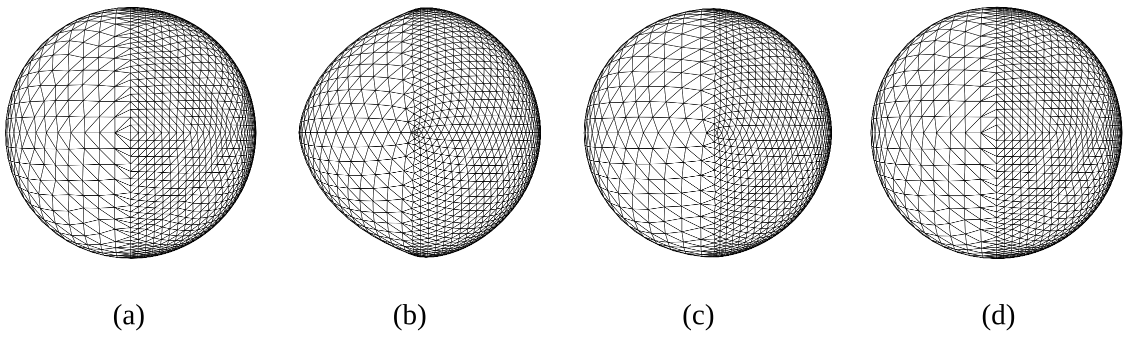


Figure 3.10: *Smoothing of spheres: (a) The original mesh containing two different discretization rates. (b) Smoothing with the umbrella operator introduces sliding of the mesh and unnatural deformation, which is largely attenuated when (c) the scale-dependent version is used, while (d) curvature flow maintains the sphere exactly.*

3.5.4 Comparison of Results

Figures 3.8, 3.10, and 3.11 compare the different operators we have used:

- For significant fairing, the umbrella operator changes the shape of the object substantially: triangles drift over the surface and tend to be uniformly distributed with an equal size.
- The scale-dependent umbrella operator allows the shape to stay closer to the original shape even after significant smoothing, and almost keeps the original distribution of triangle sizes.
- Finally, the curvature flow just described achieves the best smoothing with respect to the shape, as no drift happens and only geometric properties are used to define the motion.

Knowing these properties, the user can select the type of smoothing that fits best with the type of fairing that is desired. Diffusion will smooth the parameterization along with the shape, resulting in a more regular triangulation. If a parameterization independent smoothing is desired, then the curvature flow should be used. In the next section, we will show how to derive a smoothing technique that combines the advantages of both techniques.

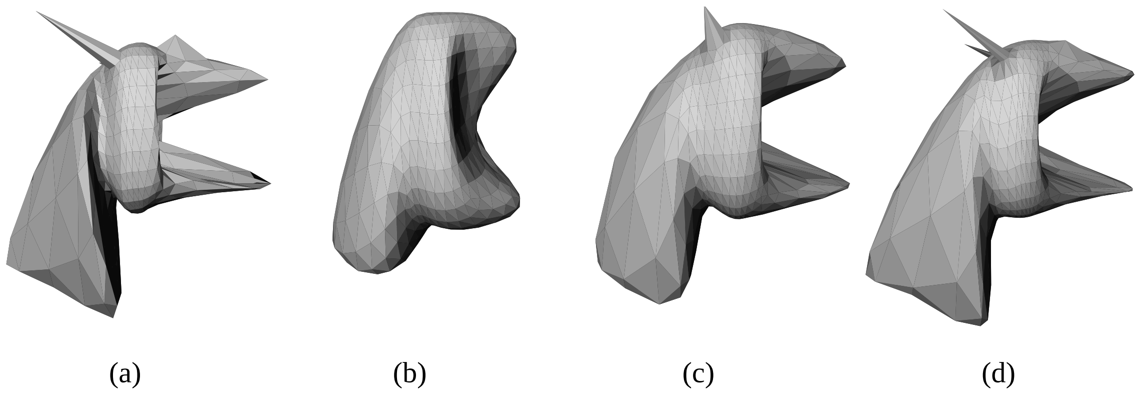


Figure 3.11: *Significant smoothing of a dragon: (a) original mesh, (b) implicit fairing using the umbrella operator, (c) using the scale-dependent umbrella operator, and (d) using curvature flow.*

3.6 Smoothing Shape and Sampling

As mentioned in the previous sections, the Laplacian contains a tangential component that smoothes the parameterization. While this parametric smoothing is not, in itself, a problem, the Laplacian does not contain the appropriate amount of shape smoothing in the normal component. On the other hand, the *parameterization-independent* curvature flow provides the appropriate shape smoothing while leaving the parameterization or sampling unchanged. In some applications, it may be desirable to have the sampling regularization of the laplacian as well as the correct shape smoothing of the curvature normal.

Graph Flow

We wish to produce a flow that is *geometrically equivalent* to curvature flow that allows us to simultaneously alter the sampling. This can be accomplished using a variant of a technique

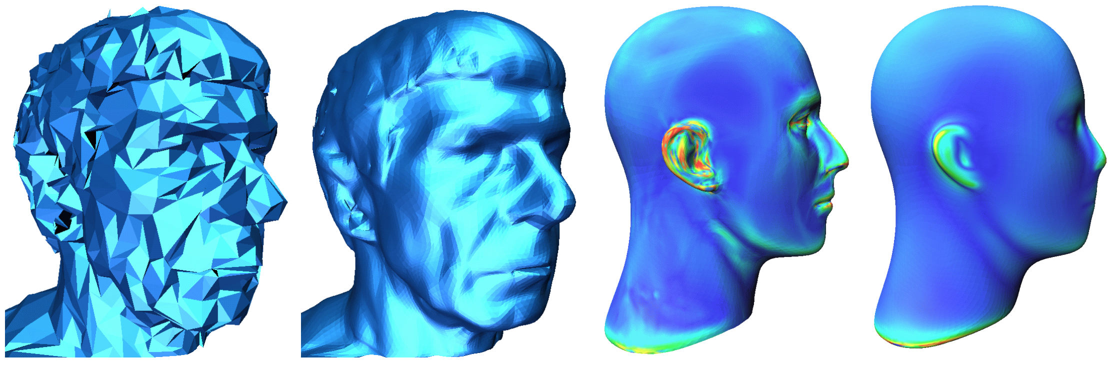


Figure 3.12: *Faces: (a) The original decimated Spock mesh has 12,000 vertices. (b) We linearly oversampled this initial mesh (every visible triangle on (a) was subdivided in 16 coplanar smaller ones) and applied the scale-dependent umbrella operator, observing significant smoothing. One integration step was used, $\lambda dt = 10$, converging in 12 iterations of the PBCG. Similar results were achieved using the curvature operator. (c) curvature plot for the mannequin head (obtained using our curvature operator), (d) curvature plot of the same mesh after a significant implicit integration of curvature flow (pseudo-colors).*

referred to as "graph flow."

Suppose we have a surface $S(t)$ evolving in time, starting with a shape S_0. Let us define a potential $f(\mathbf{x}(t), t)$ in space such that the zero isosurface of f corresponds to S at every time t. As the evolving potential characterizes a moving isosurface, we can derive a simple differential equation satisfied by f. The path of a point $\mathbf{x}(t)$ during the evolution of the surface satisfies $f(\mathbf{x}(t), t) = 0$ for any time t, yielding:

$$\frac{\partial f}{\partial t}(\mathbf{x}(t), t) + \nabla f(\mathbf{x}(t), t) \cdot \frac{d\mathbf{x}(t)}{dt} = 0\,. \tag{3.15}$$

Note that with this equation (the typical PDE used in the level-set literature) only the normal component of $d\mathbf{x}(t)/dt$ matters since it is dotted with the gradient of f, which is along the normal to the surface. An important consequence is that *only the normal component of a surface flow really affects the shape*: since any tangential component will not be accounted for in the PDE, the potential f will only evolve according to the normal component. Adding an arbitrary tangent component to a flow field will not perturb the evolution of a surface, just modify its parameterization (as mentioned in Section 3.6).

The preceding remark allows us to construct different particle paths that lie on the same surface family. Since we want to obtain a mean curvature flow, the graph flow needs to match the mean curvature flow after projection onto the normal, but we can use any tangential component we desire (though, when using sampled surfaces, care must be taken when adjusting the sampling since the resulting surface may undersampled or produce incorrect connectivity). One useful flow is to use the mean curvature normal plus the tangential component of the laplacian. Ohtake *et al.* [OBB00] use a similar flow to produce the results in Figures 3.13 and 3.14. Notice how the shape is smoothed correctly while the mesh sampling is regularized.

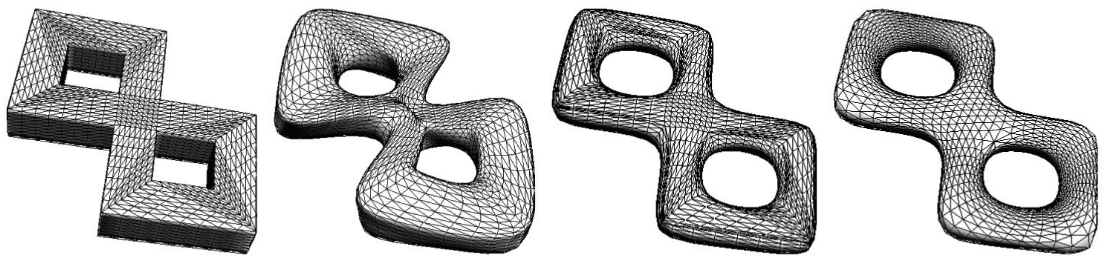


Figure 3.13: *From left: Original pretzel shape, smoothing using the Taubin λ/μ algorithm (notice the substantial shape deformation), mean curvature smoothing produces excellent shape smoothing, and the combined curvature flow plus laplacian produces a smoothed shape with regularized sampling (image from [OBB00]).*

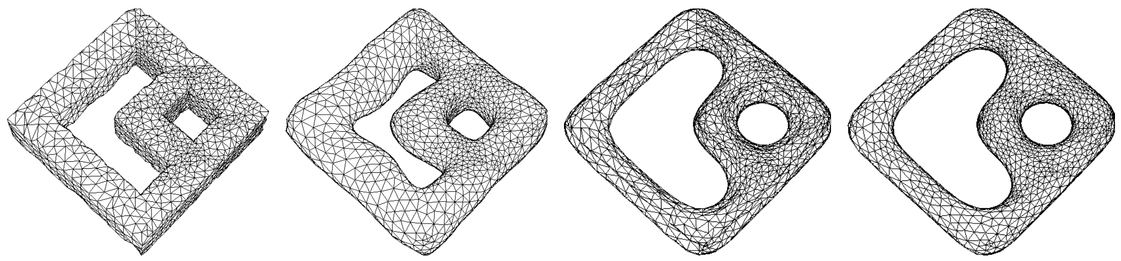


Figure 3.14: *From left: Original torus-like shape, smoothing using the Taubin λ/μ algorithm (notice the substantial shape deformation), mean curvature smoothing produces excellent shape smoothing, and the combined curvature flow plus laplacian produces a smoothed shape with regularized sampling (image from [OBB00]).*

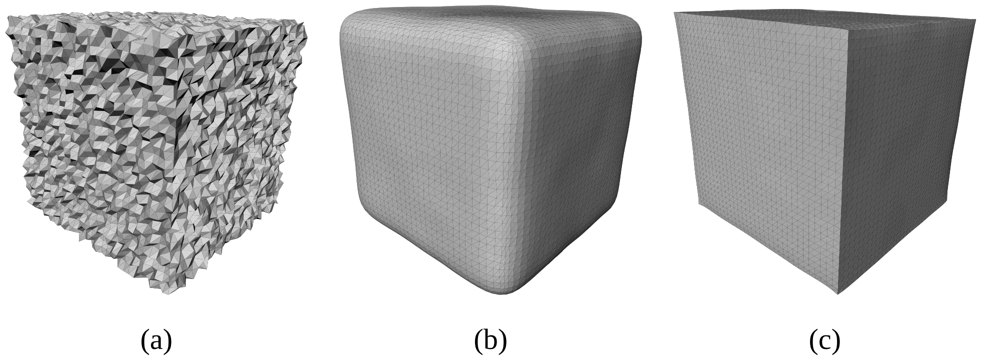


Figure 3.15: *Cube: (a) Original, noisy mesh (±3% uniform noise added along the normal direction). (b) Isotropic smoothing. (c) Anisotropic smoothing defined in Section 3.7.1.*

3.7 Anisotropic Smoothing

While the previous sections detail an impressive set of tools for denoising a mesh, they are all isotropic – smoothing equally in all directions. Since an input mesh may have many sharp features, we wish to get rid of the noise by smoothing the surface, while preserving clear features such as sharp edges. For example, we would like to smooth a noisy cube without turning it into the cushion-like shape in Figure 3.15(b).

Using anisotropic smoothing to solve this feature-preserving denoising problem has shown good results in image processing [PM90], in flow visualization [PR99], and recently on meshes [CDR00]. The underlying idea is to still diffuse the noise, but with an adaptive conductance over the domain in order to preserve edges. In Section 3.2.7, we described a way to control the smoothing by locally altering the parameter λ (possibly through a manual "spray-painting" of the mesh. While this technique could be used to manually define an anisotropic smoothing, it is a rather time-consuming task for big meshes, and it will leave ragged edges on the vertices forced to a low smoothing amount. Instead, we define an automatic weighting technique using the principal curvatures of the surface.

3.7.1 An Anisotropic Weighting Technique

In order to keep the sharp features of a mesh intact, we desire an isotropic implicit curvature flow on noisy regions, while directional diffusion should be applied to obvious edges and corners. The presence of such features can be determined using the principal curvatures of the surface. Indeed,

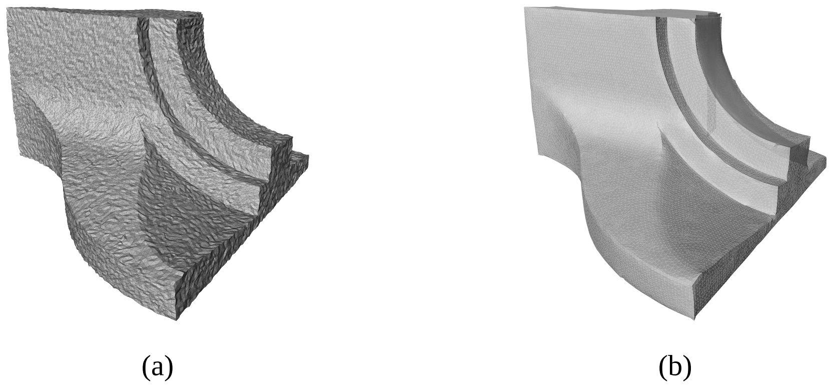


Figure 3.16: *Fandisk: (a) Original, noisy mesh. (b) Anisotropic smoothing is performed to maintain the mesh features while removing the noise.*

in the case of an edge between two faces of a cube mesh, the minimum curvature is zero along the edge, while the maximum curvature is perpendicular to this edge. An immediate idea is to perform a weighted mean curvature flow that penalizes vertices that have a large ratio between their two principal curvatures. This way, clear features like sharp edges will remain present while noise, more symmetric by nature, will be greatly reduced.

We define the smoothing weight at a vertex $\mathbf{x}_i$ as being:

$$w_i = \begin{cases} 1 & \text{if } |\kappa_1| \leq T \text{ and } |\kappa_2| \leq T \\ 0 & \text{if } |\kappa_1| > T \text{ and } |\kappa_2| > T \text{ and } \kappa_1\kappa_2 > 0 \\ \kappa_1/\kappa_H & \text{if } |\kappa_1| = \min(|\kappa_1|, |\kappa_2|, |\kappa_H|) \\ \kappa_2/\kappa_H & \text{if } |\kappa_2| = \min(|\kappa_1|, |\kappa_2|, |\kappa_H|) \\ 1 & \text{if } |\kappa_H| \ = \min(|\kappa_1|, |\kappa_2|, |\kappa_H|) \end{cases} .$$

The parameter T is a user defined value determining edges. The general smoothing flow is then: $\partial\mathbf{x}_i/\partial t = -w_i\ \kappa_H(\mathbf{x}_i)\ \mathbf{n}(\mathbf{x}_i)$. As we can see, uniformly noisy regions (cases 1 and 5 in the weight definition given above) will be smoothed isotropically, while corners (case 2) will not move. For edges (cases 3 and 4), we smooth with a speed proportional to the minimum curvature, to be assured not to smooth ridges. The caveat is that this smoothing is no longer well-posed: we try to enhance edges, and this is by definition a very unstable process. Pre-mollification techniques have been reported successful in [PR99], and should be used in such a process. However, we have had good results by simply thresholding the weights w_i to be no less than -0.1 to avoid strong inverse diffusion, and using implicit fairing to integrate the flow. As Figure 3.15 demonstrates, a noisy cube can be smoothed and enhanced into an almost perfect cube using our technique. For more complicated objects (see Figure 3.16), a pass of curve smoothing (also using implicit curvature flow) has been added to help straighten the edges.

3.8 Smoothing General Bivariate Data

Since the previous smoothing algorithms were constructed using our discrete differential operators, we can use extensions such as those described in section 2.8 to smooth non-surface data. In this section, we describe algorithms to smooth images, height fields, and vector/tensor fields.

3.8.1 Smoothing of Images and Height Fields

To reduce the noise in images, early research has advocated the use of the laplacian as a local differential operator. Diffusing the signal using laplacian smoothing will reduce high frequency noise. Unfortunately, an unintended consequence is that the noise is diffused uniformly in screen space. Sharp edges and other fundamental features of an image are then lost, blurred away by the uniform diffusion. Consequently, anisotropic operators have been proposed. They can diffuse the signal non-uniformly to better preserve edges, while reducing noise in the signal.

The first inhomogeneous diffusion model was introduced by Perona and Malik [PM90]. The idea was to vary the conduction spatially to favor noise removal in nearly homogeneous regions while avoiding any alteration of the signal along significant discontinuities (see [TT99] for an intuitive explanation of this technique). The change in intensity I over time was defined as:

$$I_t = div(\ g(\|\nabla I\|)\ \nabla I) \ \ \text{with: } g(x) = \frac{1}{1+\frac{x^2}{\alpha}}. \tag{3.16}$$

Many different variations on the conduction function g have been proposed [ROF92, ABBFC97, ALM92], and recently a higher-order PDE has been introduced by Tumblin [TT99] in the context of displaying high contrast computer graphics pictures. Similar techniques have been used to visualize complex flow fields, as in [PR99]. All of these approaches rely on isophotes of the image (see Figure 3.17(a)): the anisotropic diffusion equation can be interpreted as a diffusion mainly in the direction tangential to each isophote. Therefore, discontinuities present in the or-

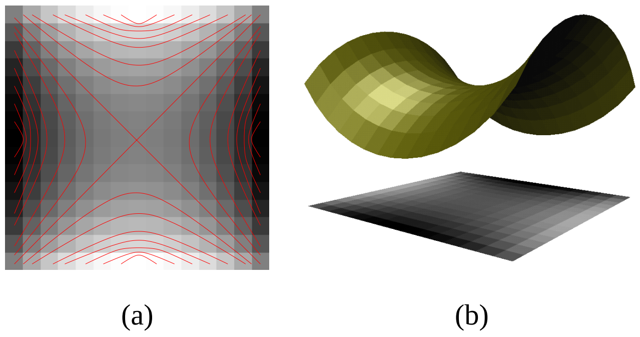


Figure 3.17: *The intensity map* $I(x, y)$ *of an image can be thought of as (a) a set of isophotes, or (b) a height field* $(x, y, z = I(x, y))$.

thogonal direction are not lost, as explained in [KDA97]. Typically, finite difference schemes are used to discretize the differential operators used. Some of these approaches also use an inverse diffusion process orthogonal to the isophotes to enhance edges; this process, being very unstable by nature, requires a pre-smoothing of the gradient for the well-posedness of the problem.

However, in general, relying only on isophotes to restore a noisy image is questionable: non-uniform lighting (glares, specularity effects) often enhance our understanding of a scene while significantly affecting isophotes in complex ways. Other anisotropic diffusion models are therefore desirable.

3.8.2 Intensity as a 2-Manifold

A number of approaches for denoising in image processing research consider an image as a 2-manifold embedded in 3D: the image $I(x, y)$ is regarded as a surface $(x, y, I(x, y))$ in a three dimensional space, as depicted in Figure 3.17(b). The surface $S = (x, y, I(x, y))$ is sometimes called a Monge surface, or simply a *height field* as the intensity represents an elevation along the z direction of the (x, y, z) space. Many algorithms also make use of the square root of the determinant of the first fundamental form of the surface [DHKW92, Gra98], denoted by $\mathcal{W}$. This quantity measures at a given point on the surface the area expansion between the parameter domain and the surface itself: a surface dA on the screen (parameter domain, also called screen space in our context) will then represent a surface area of $\mathcal{W}\ dA$ on the height field. Due to the simplicity of a height field, we can write:

$$\mathcal{W} = \sqrt{1 + I_x^2 + I_y^2}, \tag{3.17}$$

$$\mathbf{n} = \frac{1}{\mathcal{W}}(-I_x, -I_y, 1). \tag{3.18}$$

Now that we consider the image as a surface, it is natural to ask whether the mean curvature based surface smoothing techniques of previous sections can be used for images as well. In fact, since curvature flow is a natural generalization of diffusion (using a Laplacian parameterized by the natural mteric of the surface itself), several researchers have used the mean curvature normal for image smoothing:

- Malladi and Sethian [MS96] proposed: $I_t = -\mathcal{W}\kappa_H$ to implement the geometrically natural mean curvature flow. Contrary to the conventional laplacian filtering, it is an anisotropic flow more appropriate for a scale-space. They also derive a min/max flow, thresholding the curvature locally depending on local averages.

- Extending the Perona-Malik formulation for an intensity height field, Ford and El-Fallah [FEF98] proposed an inhomogeneous diffusion with a coefficient inversely proportional to the gradient magnitude:

  $$I_t = div\left(\frac{1}{\sqrt{1 + I_x^2 + I_y^2}}\,(-I_x, -I_y, 1)^t\right).$$

  Since this expression is actually the divergence of the unit normal **n** to the surface, we can reformulate it as:

  $$I_t = -2\,\kappa_H.$$

  They show how this flow provides good experimental results for noise removal with edge preservation, and give a FD (finite difference) algorithm to implement it using the Sobel operator for the evaluation of derivatives.

- Finally, Kimmel, Malladi and Sochen [KMS97, SKM98] proposed a framework for nonlinear diffusion where equations are derived by minimizing a functional. Using the extended Polyakov action, which reduces to the surface area functional for 2D greyscale images, they obtained the Laplace-Beltrami operator (Δ_g) as the associated parameterization-independent Euler-Lagrange equation. To introduce an edge preserving flow, they proposed the following technique, called Beltrami flow:

$$I_t = -\Delta_g S \cdot \mathbf{e}_z = -\frac{1}{\mathcal{W}}\kappa_H,$$

where $\mathbf{e}_z$ is the unit vector in the z (intensity) direction.

3.8.3 Denoising Greyscale Images

Using our differential operator based smoothing techinques, we can derive a general image smoothing algorithm. Directly applying our surface flow to images results in the equation:

$$\frac{\partial S}{\partial t} = -\kappa_H \mathbf{n}.$$

Although this flow minimizes the surface area, we often can not easily "move" the sample points along the normal direction as it is generally not aligned with the image parameter directions – the pixels would no longer be on a regular grid. We can, however, use the graph flow technique of Section 3.6 to create a geometrically-equivalent flow by only evolving the intensity field (therefore, constraining the sampling to remain the same). We require that the flow be only in the $\mathbf{e}_z$ direction and be equal to the mean curvature flow when projected back onto the normal:

$$I_t = -\mathcal{W}\kappa_H,$$

since $\mathbf{e}_z \cdot \mathbf{n} = 1/\mathcal{W}$ this gives the appropriate normal flow.

In the context of images, edges (i.e., sudden intensity changes) are fundamental. The above flow is isotropic and will smooth edges as well as the noise. To make the flow anisotropic and edge-preserving, we can add a smoothing weight, dependent on the metric of the surface, in order to penalize the edges more than the flat regions.

Consider the term $\mathcal{W}$ (square root of the determinant of the surface metric): it measures the

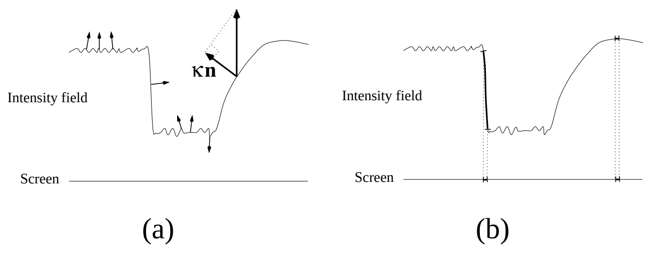


Figure 3.18: *(a): The left side indicates how normals are perpendicular to the screen in homogeneous, noisy areas, while parallel to the screen plane for edges. The right side shows how the graph flow is built out of the mean curvature flow by having the same magnitude once projected along the normal. (b): $\mathcal{W}$ measures the surface expansion between the parameter space (screen pixel) and the surface of the height field.*

surface expansion between the parameter space (screen) and the surface itself (intensity field considered as a height field). Therefore, this term will be infinite along edges, while equal to one in flat regions as depicted in Figure 3.18(b). Its inverse is therefore a good candidate for an edge "indicator". This holds for any positive power of $\mathcal{W}$ as well. Since $\mathcal{W}$ is unitless this edge indicator is also scale-invariant. The complete edge-preserving flow can now be expressed as:

$$I_t = -\frac{\kappa_H}{\mathcal{W}^\gamma}. \tag{3.19}$$

The coefficient $\gamma \geq -1$ determines the relative penalization of small jumps in intensity versus large jumps. Values less than one only penalize large jumps, while values larger than one penalize even small jumps. It controls the linearity of our edge-preservation metric: as such, γ can be described as an *edge contrast parameter*.

The flow derived above is quite general, and by varying the exponent γ we can derive many different flows. Setting $\gamma = -1$ results in the isotropic curvature flow of [MS96]. For $\gamma = 0$, we find the same flow used by El-Fallah and Ford [FEF98]. For $\gamma = 1$, our formulation leads to the Beltrami flow, mentioned in Section 3.8.2. Other values of γ offer a whole new family of denoising flows, all having the properties of parameterization-independence, scale-invariance, and feature-preservation.

3.8.4 Denoising of Arbitrary Bivariate Data

Two-dimensional data often has more than one channel of information. Color images for instance have three channels per pixel: red, green, and blue. Although a straightforward channel by channel smoothing is easily achieved by the previous method, it may not lead to optimal smoothing. Independent changes in the red, green, and blue channels result in perceptually-strong color variations in the smoothed image. Therefore, smoothing in color should be performed in a higher dimensional color space such as rgb where coupling between channels results in more natural color smoothing [Sha96]. Similarly, higher dimensional data should be smoothed in its respective space, not channel-by-channel. This section demonstrates that our previous approach can be extended easily to provide a denoising technique for higher dimensional data.

Graph Flow for Mean Curvature Smoothing

We now consider our bivariate multi-dimensional data as lying on 2-manifold embedded in nD. We can still define the Laplace-Beltrami operator as being the generalization of the mean curvature normal, or the generalization of the (parameterization-independent) surface area gradient. For the sake of simplicity, we will denote the Laplace-Beltrami operator as **B** from now on: $\Delta_g S = \mathbf{B}$. To make this flow a graph flow, we have to project this vector onto the sub-space of free parameters, such as r, g, b in the case of color images. The orthogonal projection of **B** onto this sub-space is the vector $\overline{\mathbf{B}}$. It consists of the same coordinates as **B**, except for the first two components (corresponding to the x and y axes of screen space) set to zero. Therefore, we need a vector in the direction opposite to $\overline{\mathbf{B}}$ to ensure a graph flow, but such that its projection onto **B** has the same magnitude as **B** to ensure the geometric equivalence:

$$-\frac{\mathbf{B} \cdot \mathbf{B}}{\overline{\mathbf{B}} \cdot \overline{\mathbf{B}}}\ \overline{\mathbf{B}}. \tag{3.20}$$

Applied to color images (5D space ($\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_r, \mathbf{e}_g, \mathbf{e}_b$)), the graph flow geometrically equivalent to a mean curvature flow is therefore:

$$\frac{d}{dt}\begin{pmatrix} r \\ g \\ b \end{pmatrix} = -\frac{\mathbf{B}\cdot\mathbf{B}}{\overline{\mathbf{B}}\cdot\overline{\mathbf{B}}}\begin{pmatrix} \overline{\mathbf{B}}\cdot\mathbf{e}_r \\ \overline{\mathbf{B}}\cdot\mathbf{e}_g \\ \overline{\mathbf{B}}\cdot\mathbf{e}_b \end{pmatrix}. \tag{3.21}$$

Edge-Preserving Flow

Following the same arguments as in Section 3.8.3, we now want to weight the features to favor smoothing of almost uniform regions. Thus, we need to find a way to measure discontinuities. Based on the same idea as in the greyscale case, we can use the ratio of surface expansion between the screen and the surface. It is directly measured by the ratio between the magnitudes of $\mathbf{B}$ and $\overline{\mathbf{B}}$, as cliffs are characterized by a normal parallel to the screen plane. Our multi-dimensional scale-invariant edge indicator can be written as: $||\overline{\mathbf{B}}||/||\mathbf{B}||$: the edge indicator will be valued 0 on sharp edges, and 1 in homogeneous regions. Adding an edge contrast parameter γ (slightly different than the previously defined γ, purely for aesthetic reasons), our feature-preserving flow becomes, for color pictures for instance:

$$\frac{d}{dt}\begin{pmatrix} r \\ g \\ b \end{pmatrix} = -\left(\frac{||\overline{\mathbf{B}}||}{||\mathbf{B}||}\right)^{\gamma}\begin{pmatrix} \overline{\mathbf{B}}\cdot\mathbf{e}_r \\ \overline{\mathbf{B}}\cdot\mathbf{e}_g \\ \overline{\mathbf{B}}\cdot\mathbf{e}_b \end{pmatrix}. \tag{3.22}$$

Notice that $\gamma = 0$ simplifies greatly to a Beltrami flow. The creation of higher dimension feature-preserving smoothing flows follows naturally.

Incorporating Perceptual Bias for Color

The (r, g, b) color space is not necessarily the most perceptually sound. Put simply, the human eye is not similarly sensitive to a change of red, green, or blue: what we visibly consider as a major color edge may not be considered as such in this color space, and vice versa. Therefore, smoothing a color image in such a space may not lead to the most pleasant visual results.

Instead, we use the (L^*, U^*, V^*) color space to take some of the human color perception biases into account. This model has the advantage of being almost perceptually uniform for the human eye, and therefore, will appropriately define edges. Note that any other model and/or linear combination of existing models is straightforward to implement in our framework as only the input has to be changed.

Tuning of Global Contrast

The framework defined so far has an additional degree of freedom: the scaling of intensity/colors. Colors are usually rescaled between 0 and 1, but the real color spectrum of the image is undetermined. Unless radiometric values of the image are available, we can arbitrarily choose a scale factor α to define the global contrast of the image. Note that our surface functional for a large value of α will be equivalent, for $\gamma = 0$, to a regularized version of the L_1 norm of the intensity: therefore, our flow will be equivalent to the total variation denoising approach of [ROF92]. On the other hand, a small scale factor will tend to create a flow based on the L_2 norm [Sha96] for the same γ [KMS97].

3.8.5 Discussion

We have defined a scale-invariant anisotropic flow to denoise any bivariate data while preserving features. It is based on surface area minimization, well-known in 3D to provide good denoising. As this method tends to minimize surface area in nD, the smoothing between data samples is treated in a non-linear way, significantly different from a channel-by-channel smoothing. In the special case of color images, color smoothing will induce an alignment of the gradient of each channel, which does not appear in a channel by channel smoothing. The integration of the flow can be performed using either an explicit or implicit Euler scheme. The user can stop the smoothing when the data is sufficiently denoised. The integration time step can either be user defined or computed using El-Fallah and Ford's technique based on the variation of the global area [FEF98]. If the area of the whole image changes significantly during a time step, a lot of noise was present in the image, and it is safe to take a larger time step. When the area change starts to decrease, the image structure may be significantly affected by too large a time step, thus the time step should be reduced.

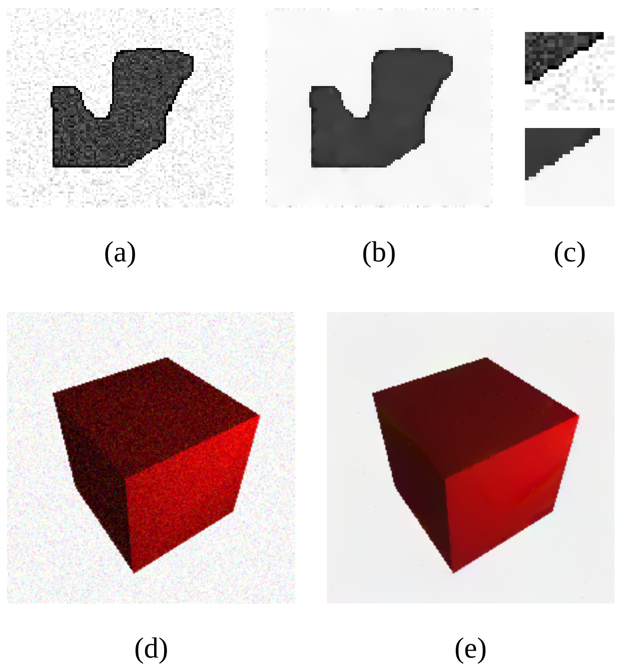


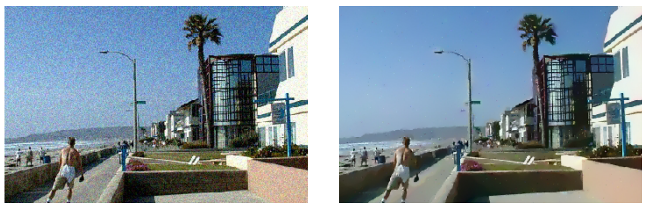
Figure 3.19: *Examples of denoising for computer-generated greyscale and color images (a and d: noisy images, b and e: denoised output, c: close-up of a and b).*

3.8.6 Results

We tested our method on several datasets. We first used computer generated images with artificially added noise. In Figures 3.19(a-c) we can see that our method removes the noise from a simple greyscale image while retaining the edges present in the original image. Similarly, Figures 3.19(d-e) shows a smoothing for a simple color picture in the presence of large amounts of noise.

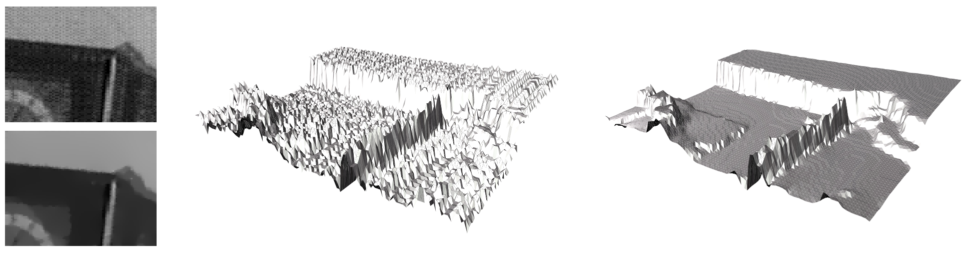
Next, we tested the method on “real-world” images. The denoising technique performs well on classical test images, as demonstrated for instance in Figure 3.20. In Figure 3.21, we display a noisy image of a clock and its restored version, along with the height field representation of the images.

We also tried our technique on different depth data. Rather than using a 3D smoothing as in Section 3.5, we can take advantage of the fact that the error is only in the z direction. While former methods [Tau95b, DMSB99] would make the assumption of an isotropic noise in space, our method applies better to this depth field as the noise (measurement error) mainly resides along the z axis. To demonstrate this advantage, we smoothed an elevation map of a section of



(a) (b)

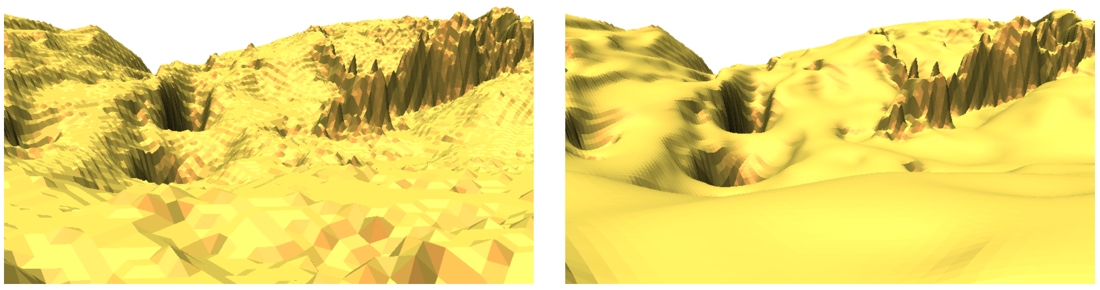
Figure 3.20: *(a) Noisy color image, (b) Denoising flow applied to (a), in 300 explicit iterations with $dt = 1$, $\gamma = 0$.*



(a) (b) (c)

Figure 3.21: *Clock example: The initial image (a, top) contains a significant amount of noise as its height field (b) shows. Our denoising technique significantly reduces this amount of noise (a, bottom) while keeping the features in place (c).*

Mars. Due to measurement errors and poor quantization of the original data, the height field is noisy as shown in Figure 3.22(a). After an anisotropic smoothing, we suppress the noise and most of the quantization effects, resulting in a smooth surface even with a flat-shaded rendering.



(a) (b)

Figure 3.22: *Mars elevation map: (a) raw data, (b) smooth version after anisotropic diffusion. Notice how, with our non-uniform diffusion, the aliasing due to poor quantization is suppressed without altering the general topography of the surface (both pictures are flat-shaded).*

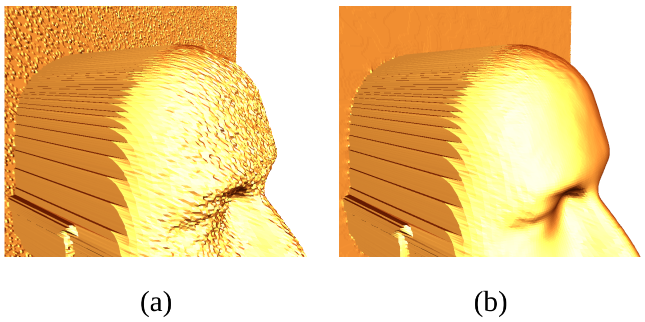


Figure 3.23: *(a) Head model obtained from a noisy depth image. (b) Reconstructed model after denoising (flat-shaded).*

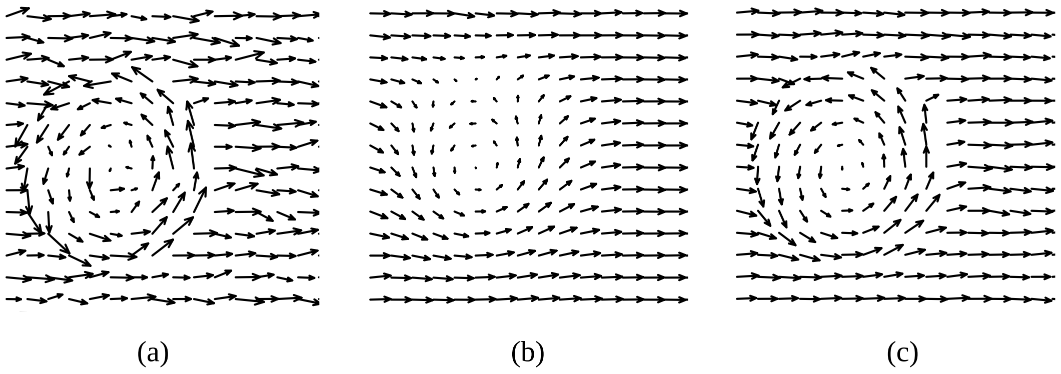


Figure 3.24: *Vector field denoising: (a) Original, noisy vector field; (b) Smoothed using Beltrami flow; (c) Smoothed using anisotropic weighted flow to automatically preserve the vortex region.*

Figure 3.23 demonstrates how our method behaves on range images. Given a noisy range image of a face, we can smooth the range image to reconstruct the face without visible noise while keeping the features in place. Once again, previous methods would have altered the shape since the assumption of isotropic noise in the data does not apply for range images.

The extension of our discrete differential operator to higher dimensional embedding spaces allows us to use the same smoothing technology even for vector fields or tensor images. As a final example demonstrating the practical accuracy of our operator, we performed different smoothings on higher dimensional spaces. For instance, Figure 3.24 demonstrates how our operators can smooth a vector field, with or without preservation of features. Anisotropic smoothing can indeed preserve significant discontinuities such as the boundary between the straight flow and the vortex, just as we preserved edges during mesh smoothing in 3D.

Chapter 4

Remeshing

In this chapter, we present a novel technique, both flexible and efficient, for interactive remeshing of irregular geometry [AMD02]. First, the original (arbitrary genus) mesh is replaced by a series of 2D maps in parameter space. Since these maps contain geometric quantities including our discrete differential operators, they provide a complete substitute for the 3D mesh. Using these maps, our algorithm is then able to take advantage of established signal processing and halftoning tools that offer real-time interaction and intricate control. The user can easily combine these maps to create a control map – a map which controls the sampling density over the surface patch. This map is then sampled at interactive rates allowing the user to easily design a tailored resampling. Once this sampling is complete, a Delaunay triangulation and fast optimization are performed to perfect the final mesh.

As a result, our remeshing technique is extremely versatile and general, being able to produce arbitrarily complex meshes with a variety of properties including: uniformity, regularity, semi-regularity, curvature sensitive resampling, and feature preservation. We provide a high level of control over the sampling distribution allowing the user to interactively custom design the mesh based on their requirements thereby increasing their productivity in creating a wide variety of meshes.

4.1 Introduction

As 3D geometry becomes a prevalent media, a proliferation of meshes are readily available, coming from a variety of sources including 3D scanners, modeling software, and output from computer vision algorithms. Although these meshes capture geometry accurately, their sampling quality is usually far from ideal for subsequent applications. For instance, these (sometimes highly) irregular meshes are not appropriate for computations using Finite Elements, or for rapid, textured display on low-end computers. Instead, meshes with nearly-equilateral triangles, a smooth gradation of sample density depending on curvatures, or even uniform sampling are preferable inputs to most existing geometry processing algorithms. *Remeshing*, i.e., modifying the sampling and connectivity of a geometry to generate a new mesh, is therefore a fundamental step for efficient mesh processing.

4.1.1 Background

Although studied in Computer Graphics for obvious reasons, surface remeshing has also received a lot of attention from various non-CG fields interested in mesh generation — mainly Computational Fluid Dynamics, Finite Element Methods, and Computational Geometry. However, the diverging goals resulted in vastly different, non-overlapping solutions as we now briefly review.

Mesh Generation Community Since the emphasis is generally on numerical accuracy, most of the tools developed in the non-CG communities focus on mesh quality. Remeshing procedures often use a parameter space to impose quantitative mesh properties such as local triangle sizes and shapes [dCS96, TOC98, GB98]. Others simply perform mesh simplification [PV97] or edge operations and vertex shifting [Bor98] to conform to a global mesh property. However, most techniques heavily rely on mesh optimization [Fre00, RVSS00] to satisfy common requirements like equal angles for FE computations [BGH$^+$97] or smooth gradation [BHF97]; *accuracy* is therefore obtained at the price of rather slow computations.

Computer Graphics Community In contrast to the quality requirements of the other fields, CG work has focused mainly on *efficiency*. The majority of previous work has proposed semi-regular remeshing techniques [LSS$^+$98, GSS99, GVSS00, HLG01], based on an initial phase of simplification which could be used in itself for remeshing [GH98, LT98] since it performs the aforementioned edge operations and vertex shifting. A noticeable body of work has also recently been proposed to accurately remesh sharp features [VRKS01, BK01]. However, none of these methods can offer flexibility on the quality of the remeshing obtained, since issues such as area distortion or triangle shape distortion are not even considered: tailored output can only be produced through extensive trial-and-error by a patient user.

A controllable mesh re-tiling technique was proposed by Turk [Tur92] to resample an input mesh using properties such as uniformity or curvature-based density, allowing a much more precise design of the output meshes. However, the algorithm requires the propagation of “particles” on the original mesh and a global relaxation of their positions until convergence, requiring heavy computation. Similarly, Bossen and Heckbert [BH96] proposed a 2D anisotropic mesh generation involving vertex insertions, vertex removals, and iterative relaxation. Again, output meshes conforming to various requirements can be generated but only after significant computational effort. Our goal is thus to attain accuracy, flexibility, and efficiency for resampling, as none of the techniques described above can offer such a combination.

4.1.2 Contributions and Overview

Our main contributions over previous remeshing techniques are in terms of **efficiency** as simple meshes can now be processed in real or interactive time through a novel resampling stage followed by an *output-sensitive* remeshing algorithm, and **flexibility** as we offer complete and precise control over the sampling rate and quality anywhere on the geometry. These two critical properties are obtained through the use of parameterization and conventional image processing tools such as filtering, transfer functions and error diffusion, in order to compute near-optimal resamplings in a matter of milliseconds. Previous approaches often worked directly on the mesh, resulting in either slow performance or little control over the remeshing quality.

The structure of this paper follows closely the overall algorithmic pipeline depicted at the bottom of Figure 4.1. We first describe the atlas of parameterization and geometry analysis we perform on the input mesh in Section 4.2, in order to generate a catalog of 2D maps as an alternate representation for the input mesh. We detail how these resulting maps are processed efficiently using standard signal processing tools to create a near-optimal resampling of the input

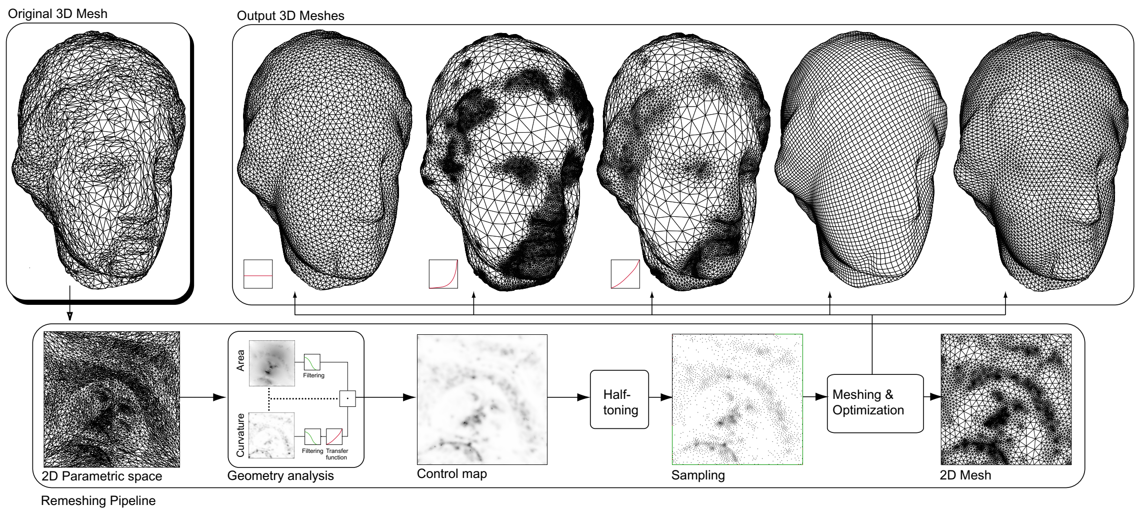


Figure 4.1: *A brief overview of our remeshing process: The input surface patch (top left) is first parameterized; Then geometric quantities are computed over the parameterization and stored in several 2D maps; These maps are combined to produce a control map, indicating the desired sampling distribution; The control map is then sampled using a halftoning technique, and the samples are triangulated, optimized and finally output as a new 3D mesh. A few examples of the various types of meshes our system can produce are shown (top, from left to right): uniform, increased sampling on higher curvature, the next with a smoother gradation, regular quads, and semi-regular triangles. After an initial pre-processing stage (∼1s), each of these meshes was produced in less than 2 seconds on a low-end PC.*

mesh in Section 4.3. A final, rapid phase of optimization can then be performed to get accurate results as described in Section 4.4. Finally, we present a number of results to demonstrate the wide range of possible resamplings we can interactively obtain in Section 4.5.

4.2 Geometry Analysis

In this section, we explain in detail how we build a complete set of *maps* from the raw, input geometry. This will construct an alternative representation of the surface and all of its intrinsic properties in the form of convenient 2D images, which are easy to process. We demonstrate how simple and efficient this process is when graphics hardware is used appropriately. We also show how to create a small set of tiling patches from a closed object of arbitrary genus, and give details on how to compute the geometry maps from these surface patches by flattening them onto isomorphic planar triangulations.

4.2.1 Creation of an Atlas of Parameterization

The first processing stage undergone by the input mesh consists in splitting the surface into disk-like patches, creating an *atlas of parameterization* [GH95]. A number of existing clustering algorithms such as [GWH01, PG01, LPRM02] could be used successfully to achieve such a partition. Unfortunately, they do not produce smooth patch boundaries on the geometry as demonstrated in Figure 4.4, and therefore lead to poor-quality stitching across the remeshed patches. Note that one could also make some cuts in the geometry to turn it into a single patch, as often proposed in the last two years [LPVV01, EHP02, She02, GGH02]. All of these methods are valid ways to deal with arbitrary genus surfaces, and the resampling technique presented in this paper is mostly independent of the cutting/unfolding method chosen.

In the remainder of this paper, we use a variant of the mesh partitioning proposed by Eck *et al.* [EDD$^+$95] (later improved by Guskov *et al.* [GVSS00]), that computes approximate Voronoï diagrams as an initial non-smooth partitioning of the mesh into genus-0 patches. This procedure, which we will extend in Section 4.2.4 to generate area-balanced patches, automatically produces a series of tiling patches from *input meshes of arbitrary genus*.

4.2.2 Parameterization

The second stage is to map each individual surface patch to an isomorphic planar triangulation. This operation, called *parameterization*, also has many solutions readily available ([EDD$^+$95, Lév01, LPRM02, DMA02] to name a few). Although most parameterization techniques would be adequate, one that guarantees visual smoothness of isoparametric lines and preserves the conformal structure of the input mesh is most preferable. We thus strongly advocate for the conformal parameterization as defined in [PP93, EDD$^+$95] since it behaves extremely well even on irregular triangulations [DMA02]. This technique requires solving a simple, sparse linear system with coefficients based on the geometry of the mesh, and is usually handled in a matter of seconds using a Conjugate Gradient solver with good preconditioning. We fix the boundary to be a square (see Figure 4.2) or any convenient rectangular region so that our maps can be efficiently stored and processed as regular floating point images.

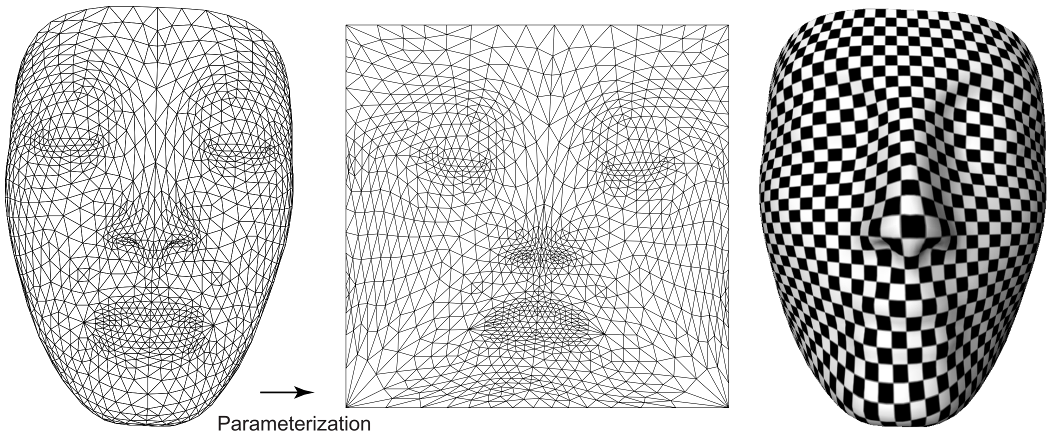


Figure 4.2: *Original mesh, conformal parameterization [EDD$^+$95] and texture mapping of a checker-board. Notice the inevitable area distortion on the nose, which we will automatically compensate for during the resampling process (see Section 4.3.1).*

Once a parameterization has been found, we compute several scalar maps to serve as a *complete substitute* for the input geometry. This will allow us to work almost solely on the 2D images instead of on the original 3D mesh.

Catalog of Maps For our application, we have identified the following geometrical values as being relevant:

- *Area distortion map $\mathcal{M}_\mathcal{A}$:* since no discrete parameterization can (in general) preserve the area of every triangle, we need a piecewise constant scalar map indicating how each triangle has been shrunk or expanded during the parameterization. This is easily computed using the ratio $\mathcal{A}_{3D}/\mathcal{A}_{2D}$ of each triangle's surface area in 3D and its corresponding area in the 2D parameterization. Note that this map will *compensate* for any area distortion inevitably introduced by the parameterization (as depicted in Figure 4.2).

- *Curvature maps $\mathcal{M}_K$ and $\mathcal{M}_H$:* since any differential quantity on a smooth surface can be expressed as a (possibly nonlinear) combination of three invariants: area $\mathcal{A}$, Gaussian curvature K, and mean curvature H [Gra98], we compute both a Gaussian curvature and a mean curvature map (in addition to the previously mentioned area distortion map). Using our discrete differential operators to compute these maps allows for accurate results even on very irregular input meshes. These two maps can then be combined to obtain other useful curvature maps: for instance, one can compute maps of minimum curvature κ_1, maximum curvature κ_2, or total curvature $\kappa_1^2 + \kappa_2^2$ by simple per-pixel operations on those two basic maps. Additional data, such as curvature tensors could also be computed on the surface and stored in maps, but we do not make use of them in this work.

- *Embedding Map $\mathcal{M}_{\mathbf{x}}$:* we also need the position $\mathbf{x} = (x, y, z)$ of each vertex, describing the exact geometry of the surface in 3D. These three maps (one per component) will provide a very efficient way of computing the mapping between a value $\mathbf{u} = (u_x, u_y)$ on the parameterization and its associated 3D point on the input mesh $\mathbf{x} = (x, y, z)$.

- *Face Index Map $\mathcal{M}_{index}$:* we also construct a face index map by assigning a color to each triangle in the parameterization corresponding to its face index in the mesh, as done by Botsch *et al.* [BRK00]. Such a map turns out to be efficient for locating in constant time the triangle in which a given parametric value lies, saving potentially costly searches.

- *Additional Maps:* finally, any attribute (normal, texture, color, etc.) can also be mapped onto the parameterization to complete the catalog of maps.

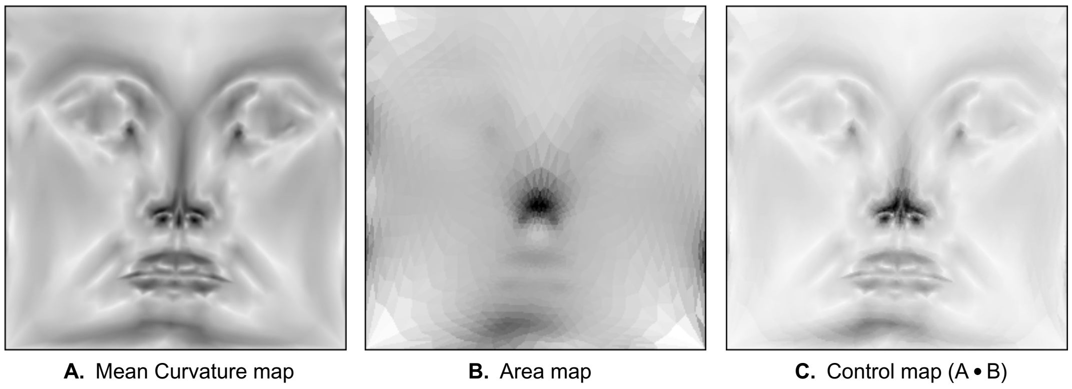


Figure 4.3: *Examples (in inverse mode for better visualization) of geometry maps for the mask in Figure 4.2. A. $\mathcal{M}_H$, the mean curvature map computed according to [MDSB02]. B. $\mathcal{M}_{\mathcal{A}}$, the area map; the nose has been compressed during the flattening process, while areas nearby the corners have been stretched. C. Sampling control map, using a per-pixel multiplication: $A \cdot B$.*

Hardware-Assisted Map Generation Piecewise-constant maps representing area distortions, face indices or per-face normals are efficiently generated using hardware accelerated OpenGL commands. Each floating-point or integer value is separated into the R, G, B, A color channels (similar to [BRK00]), and all the triangles are rendered using OpenGL flat shaded triangle primitives in a back buffer. We assign a depth proportional to the surface area of each triangle to reduce the aliasing of small triangles in the map.

For linearly interpolated maps representing curvature, positions, per-vertex normals or attributes, we use the face index map and standard barycentric coordinates to compute the linear interpolation between the vertices in the parametric space. Note that the map creation could be simplified and optimized even further now that graphics boards implement full 32-bit floating point buffers for rendering. Nonetheless, generating the maps naively using graphics hardware speeds up the map creation by *two orders of magnitude* compared to a naïve pixel-by-pixel implementation, and takes less than $100\ ms$ for large meshes with thousands of triangles. Figure 4.3 depicts both a curvature and an area map, as well as a compositing of the two.

4.2.3 Features and Constraints

In addition to the geometry maps, we sometimes need to define specific features and/or constraints that the user wishes to enforce during the remeshing process. Typically, we want sharp features (present in mechanical parts for instance, see top left of Figure 4.6) to be preserved. Similarly, some particular points of the input surface may need to be constrained to become vertices of the remeshed version, for animation purposes for example.

Features We first assume that feature edges are either extracted using a simple dihedral angle thresholding, or directly input by the user by tagging existing input edges or creating arbitrary piecewise-linear feature curves. From this set of feature edges (Figure 4.6, top middle) we classify vertices by their number of adjacent feature edges, leading to two categories: we call *crease* vertices any vertex connected to exactly two feature edges, and *corner* vertices all the other vertices, connected to one or more than two feature edges. These feature edges are then chained together into a feature graph. This is very similar to the feature skeleton composed of "*backbones*" as introduced by Kobbelt *et al.* in a series of papers concerning geometry resampling and feature remeshing [BRK00, VRKS01, BK01] (see Figure 4.6, top right, for an example). This

feature graph requires little memory and can be computed in a straightforward way. We should note the following details that need to be addressed during the implementation: i) the graph can have cycles, ii) each patch boundary or cutting path is *also* added to the feature graph as a closed cycle (as being either a *sharp*, *boundary* or *seaming* backbone), iii) some features may meet at corners living on the boundary, and iv) a crease vertex should be classified as a corner if an important change of direction is detected along the feature. The latter corresponds to a feature inflexion point and is a rare occurrence. Once the feature graph has been properly constructed, the specified piecewise linear features will be exactly preserved by our remeshing technique as explained in Section 4.3.2.

Constraints We also allow the user to define a list of (u,v) values for which (s)he desires to get corresponding vertices in the output mesh. These values can be defined by the user by simply clicking on the input mesh. We save a list of all the constraints for later use during resampling.

4.2.4 Making the Atlas Area-Balanced

As mentioned in Section 4.2.1, we mostly use an existing technique to construct the atlas of parameterization. We, however, make use of our novel maps to improve this procedure.

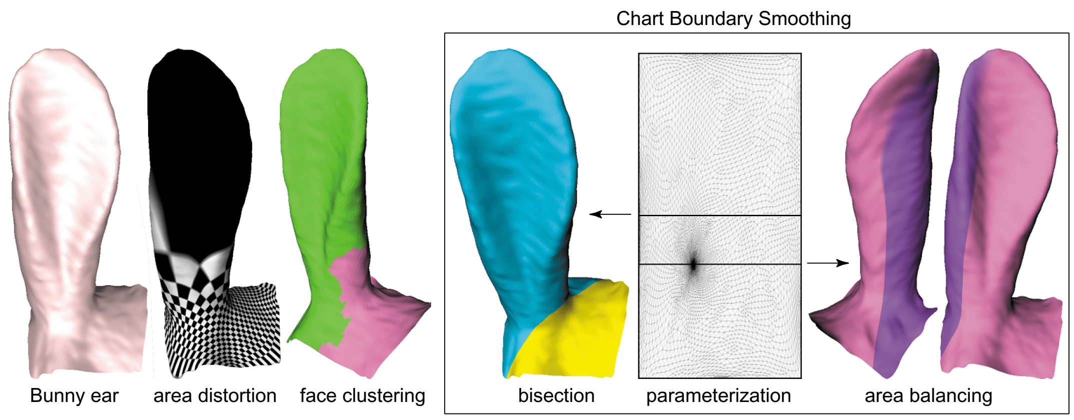


Figure 4.4: *Area-balanced atlas. From left to right: geometry of a Bunny ear; conformal parameterization and resulting area distortion visualized through a texture mapping of a checkerboard; face clustering obtained using [GWH01]; partitioning obtained by simple bisection [EDD$^+$95, GVSS00]; the conformal parameterization, with the two medians; area-balanced and smooth partitioning, using the median line of its area map $\mathcal{M}_{\mathcal{A}}$ (computed in $50\ ms$).*

Eck [EDD$^+$95] proposed to smooth patch boundaries iteratively by mapping two adjacent patches onto a 2×1 rectangular region using the discrete conformal mapping discussed in Section 4.2.2, and then re-defining the boundary between the two patches as the middle isoline in the parameterization (see Figure 4.4), which guarantees smoothness. However, this relaxation has a major inconvenience: it is *slippery* – since the parameterization does not have any guarantee on area distortion, the middle isoline often splits the two patches into patches of two very different sizes, with a tendency to slip away from very curved features. As depicted on Figure 4.4, this often leads to patches with highly variable surface areas (compare the left and right areas after splitting) and with large parameterization distortion (note that one patch contains the entire ear, while the other is relatively flat).

Instead, we propose to construct the area distortion map of the 2×1 mapping as described in the previous section, and use it to find a good splitting line that creates equal sized patches. This is done by finding the median vertical line such that the sum of all pixel values on one side of the line is equal to the sum of the pixel values on the other side. Since a single sweep of the picture is sufficient to find the median, this operation takes little time – about $50\ ms$ for a 512×512 image. As demonstrated in Figure 4.4, this change in the original algorithm significantly enhances the quality of the partitioning, as no slipping occurs and each patch has the same surface area. Note that the dividing line is smooth thanks to the angle-preserving parameterization (*i.e.*, a straight line in parametric space corresponds to a smooth line on the surface).

Once the partitioning is done, we can compute the maps for each of the created patches as aforementioned. We use a lazy evaluation, computing a map only if needed to save both memory and time. We show in the next section the main contribution of this chapter, *i.e.*, how these maps alone are used to resample the surface geometry at interactive rate.

4.3 Real-Time Geometry Resampling

Now that the input geometry has been preprocessed and replaced by an equivalent series of maps, we can use these maps to design a proper resampling. In this section we propose a real-time technique to resample the geometry. This is achieved in two stages: first, the user designs a *control map* by combining different geometry maps to define the desired *density of samples*;

then a simple *halftoning* technique is used to discretize this map and generate the exact, requested number of vertices. We show that this resampling is near optimal, and only a quick optimization will be needed to obtain a high quality mesh as output.

4.3.1 Designing the Control Map

To allow for a vast range of possible remeshings, we let the user design a control map that denotes the vertex density for the remeshing.

Area Map as Sampling Space Resampling the parameterization uniformly would not result in a regular 3D resampling of the geometry, due to the area distortion introduced during flattening. However, the area map $\mathcal{M}_{\mathcal{A}}$ does indicate the density of sampling needed on the parameterization to obtain a uniform sampling on the surface itself. The area map is therefore the sampling space we will use as *reference sampling density*.

Modulating the Sampling Density The final control map is obtained by multiplying the sampling space map by the **importance map** – a map denoting the desired sampling density across the patch. Many different maps can be used to tailor the sampling to the user's requirements, though we have mainly used curvature related maps in this work. To demonstrate the diversity of possible remeshing, we mention some canonical examples of importance maps that we have tried:

- constant, we will obtain a uniform vertex density on the 3D surface (see Figure 4.8),
- related to an estimation of curvature using $\mathcal{M}_K$ and $\mathcal{M}_H$, we will adapt the sampling rate to the local curvature (see Figure 4.11),
- any user-defined map, we will obtain a map with user specified sampling (useful for animation and displacement maps). See Figure 4.9 for such an example.

The resulting map is then rescaled to the unit interval, and inverted ($x \to 1 - x$) so that darker areas on the picture correspond to regions which require higher sampling. A simple example is depicted in Figure 4.3(c), where the area map is modulated with a mean curvature map (very light (white) areas correspond to flat and/or highly stretched regions of the mesh due to the flattening, and require few samples).

4.3.2 Halftoning the Control Map

Once the control map has been decided upon, we need to resample it with a local density of vertices in accordance with the control map, and with the exact number of samples the user requests. In other words, we need to transform the control map into a *binary image*, indicating the presence or absence of a vertex on the parameterization. In essence, our problem is directly related to the technique of *halftoning* grey-level images. Halftoning has been carefully studied for decades [Uli88] and is still an active research field [Ost01], mainly trying to improve the quality of dithering and printing. Different methods have been proposed to sample a continuous image with an adequate density, and to best statistically simulate an optimal blue noise signal in a single rasterization pass [Uli88].

Discretizing the Control Map We use a recent error diffusion algorithm developed by Ostromoukhov [Ost01], which samples an image using a *serpentine rasterization* (left to right on even lines, right to left on odd lines) with near-optimal quality. We add the following modifications to suit our purposes:

- while the original technique works on 8-bit images, we use 32-bit images to increase the range of densities;
- to avoid the well-known “dead zone” problem in error diffusion (large empty areas at the start of an error diffusion), we concatenate a vertically flipped copy of the control map above the control map and perform the halftoning for the total image, retaining only the bottom half of the image as the result;
- we also test for features and constraints (see Section 4.2.3), forcing a pixel to be black if it falls on one of the constraints, or forcing a pixel to be white if it falls on one of the features (as they will be sampled separately). The error diffusion accommodates for these forced selections by diffusing the error into nearby pixels.

The user simply chooses a given number of samples (which will be the final number of vertices) since an *exact number of vertices* can easily be reached by a simple linear scaling of the intensity

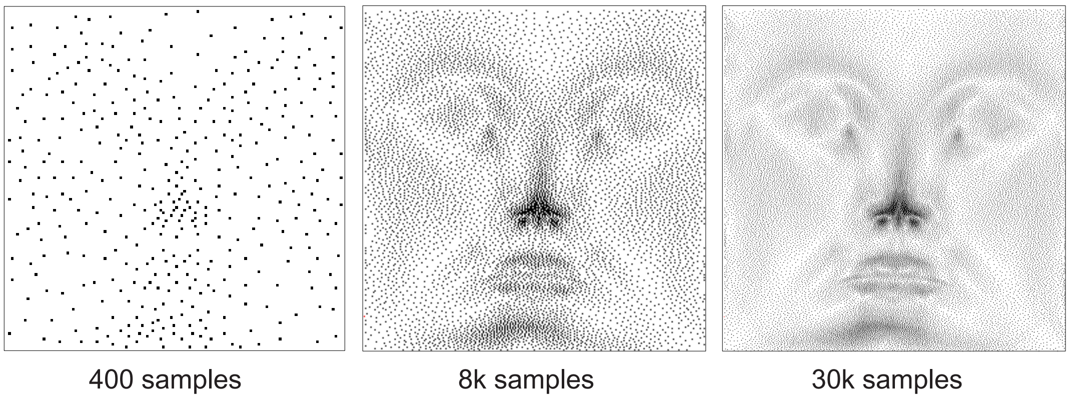


Figure 4.5: *Sampling of the map from Figure 4.3(c) using error diffusion with various numbers of requested samples ($40\ ms$ each).*

of the control map [Ost01] that preserves the ratio between the number of black pixels (*i.e.*, number of samples) and the image area. Note that the size of the maps determines the maximum number of samples (there cannot be more samples than there are pixels in the map). Therefore, we allow the user to select an appropriate image size having enough space for the sampler to work properly (though the choice of image size can easily be made automatically if desired). Such a technique turns out to be extremely efficient: a $512{\times}512$ image is sampled in only $40\ ms$ on a 1 GHz PIII. Examples of error diffusion are given in Figure 4.5.

Discretizing the feature graph A separate 1D error diffusion is performed along the boundaries and features in order to guarantee a consistent mesh density between the boundary and inner regions, as well as good feature preservation. After the initial sampling, we: $i)$ gather all the pixels of the feature graph in a 1D array using Bresenham's line algorithm, $ii)$ normalize their intensity according to the following law: $x \rightarrow 1 - \sqrt{(1-x)}$ (intuitively, the square root appears since if we want the fraction x of the samples to be black in 2D, it means we need the fraction $\sqrt{x}$ of the samples to be black in any 1D cross-section), $iii)$ apply a 1D error diffusion, and finally $iv)$ put the resulting samples into the sampled image. This guarantees an adequate feature sampling conforming to the control map, as demonstrated in Figure 4.8. The seams across patches are dealt with similarly to ensure an easy stitching.

4.3.3 User Control

Since our resampler runs at interactive rates, we can provide the user with a preview of the new mesh and allow for real-time editing of the control map to tailor the sampling to specific needs. An extremely powerful feature of our map based technique is that we can take advantage of many well-known signal processing tools for images. As a consequence, we can offer a multitude of tools still with real-time performance; for example:

- **Transfer Function -** Besides combinations obtained from filtering, scaling and shifting of the maps, we found it particularly useful to allow editing of a general transfer function over the importance map, or even direct editing of the importance map itself. For instance, a simple gamma function $f(x, \gamma) = x^\gamma$ over the curvature map gives the user control over the sampling with respect to the curvature. The user can also use pass-band filters or even a general transfer function to produce meshes with arbitrary sampling. Notice that the generality of this approach allows our system to simulate virtually any remeshing by choosing the maps and transfer functions appropriately (such as the $\mathcal{L}^2$*-optimal sampling* derived in [Sim94]).
- **Smooth Gradation** [BGH$^+$97] of the vertex density can be achieved by *low pass filtering* of the importance map, using an optimized Gaussian filter routine. Changes over the global size of the filter kernel allow a fine and interactive tuning of the gradation. Note that in the ideal case, the local size of the filter kernel should be driven by the area map, making it a non-linear diffusion of the importance map.
- **Minimum Sampling -** A guaranteed minimum density of samples can be obtained by shifting the intensity of the importance map so that its minimum corresponds to the requested minimum sampling (*i.e.*, a minimum grey level).

Interactive Preview The error diffusion is fast enough ($40\ ms$ including the transfer function computation) to provide a real-time feedback of the sampling. Additionally, we provide the option of using the dithered map as a texture directly on the 3D original model since we already have the (u, v) parameterization. The samples thus appear on the mesh instantaneously, leading to a good preview of the current sampling.

4.4 Mesh Creation and Optimization

At this point, we are already able to *interactively* produce a resampling of an input mesh with a density proven to be statistically in agreement with the user's request. However, connectivity has not yet been computed. Additionally, the halftoning implies *quantized positions* for the vertices. Therefore, we now explain how to generate an initial connectivity and how a post-process optimization can greatly improve both connectivity and geometry in mere seconds. We emphasize that, contrary to [BH96] and most other remeshing techniques, we neither add, nor remove any vertex during the optimization since, in essence, the blue noise property already spreads "just enough" vertices everywhere. Consequently, the optimization is extremely efficient and consists of only a few edge swaps and local vertex displacements.

4.4.1 Mesh Creation

Once the control map has been sampled, we perform a 2D constrained Delaunay triangulation [CGA, She96] over the points sampled in the parametric space. Constrained edges correspond to an ordered list of points sampled using 1D error diffusion along backbones of the feature skeleton (see Section 4.2.3), as can be seen in Figure 4.6, bottom middle. The vertex coordinates are then mapped into 3D using the face index map (see Section 4.2.2) and barycentric coordinates within the triangle to find the accurate 3D position[1]. The constrained Delaunay triangulation and the reprojection onto the original 2-manifold typically take a total of 200 ms for 3000 vertices generated. Notice that the connectivity generated by a Delaunay triangulation in the parameter plane may not be the most relevant one. However, since all triangulations with a given number of vertices are all isomorphic to each other through edge swapping, we use this triangulation as an initial "guess," and will perform connectivity optimization as necessary.

4.4.2 Connectivity Optimization

For a fixed set of vertices obtained by resampling, the connectivity can be arbitrarily modified by simple edge swapping. Many optimizations can be easily implemented (see, for instance, tightest

[1]Although using $\mathcal{M}_{\mathbf{X}}$ would be faster, it is usually not accurate enough for small maps, and could therefore result in small noise in the reprojection.

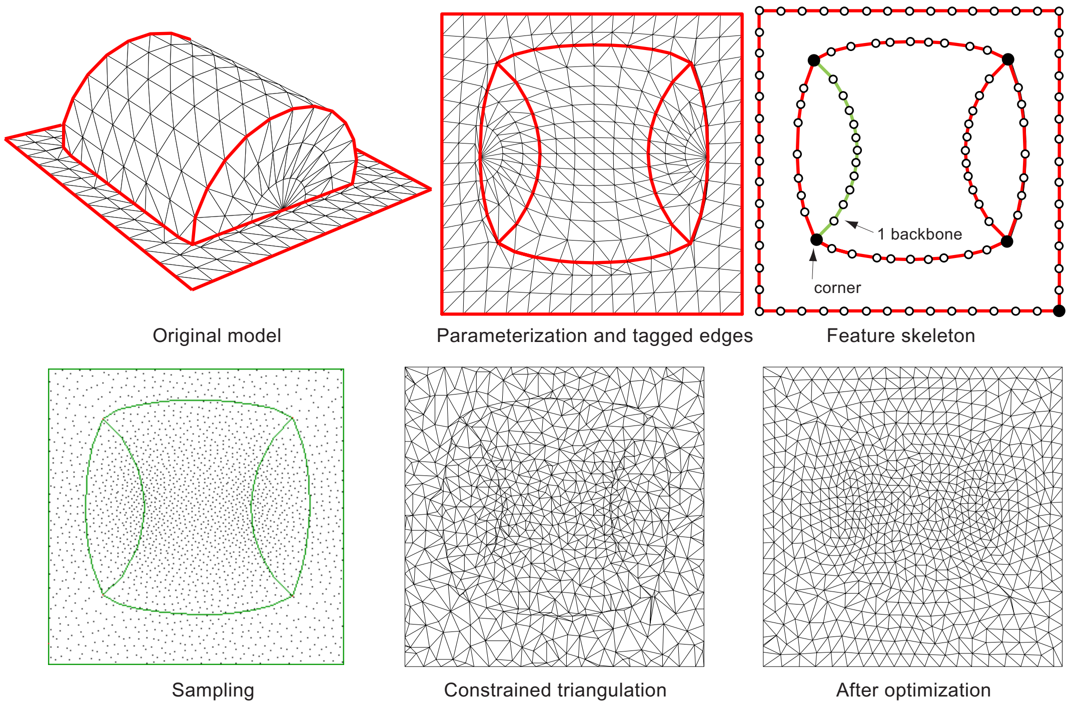


Figure 4.6: *Simple example of features: the feature edges (in red) are chained together to create the feature graph; a 1D error diffusion is then performed along the graph followed by a constrained Delaunay triangulation of the whole sampling; after a constrained mesh optimization, the feature edges are perfectly preserved, while blended in the new mesh.*

triangulation [vDA95], minimum curvature [DHKL01]). We also used the following two simpler criteria:

Regularity Edge swaps can be performed in order to favor valence 6 for interior vertices, and valence 4 on boundary vertices. This is implemented by randomly picking a non-feature edge and performing an edge swap only if it reduces the valence dispersion. A few additional constraints can be added in order to prevent face flipping in the parameterization, or large geometric distortions for instance. Note that we can also balance the valences on both sides of each inner backbone. The “rib effect” [BK01] can therefore be obtained by forcing exactly two neighbors on each side of a sharp edge whenever possible, as demonstrated in Figure 4.8.

Face Aspect Ratio Similarly, edge swaps can be performed to improve the aspect ratio of the triangles. In practice, we swap an edge between two triangles if it improves their *surface* *area/perimeter*^{2} ratio (computed in 3D). This simple test often results in dramatic improvements, since the connectivity is now dependent on the embedding, and not solely on the parameterization.

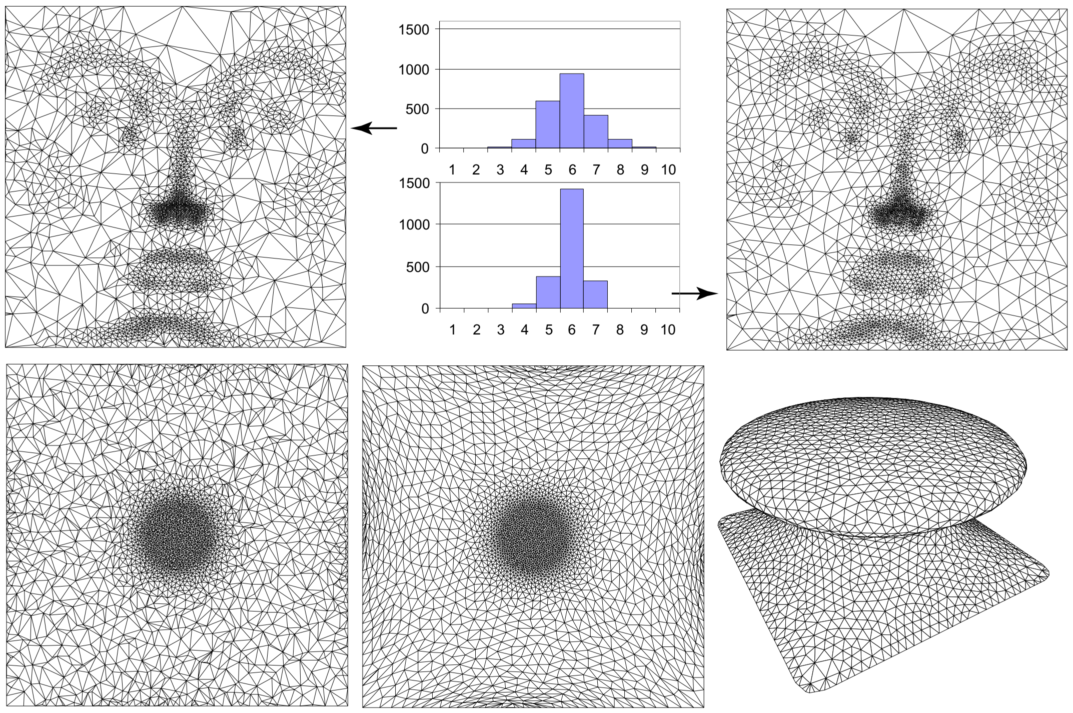


Figure 4.7: *Top: Left, Delaunay triangulation over the sampling. Right, after connectivity and geometry optimization. Middle, comparison of valence dispersion. Bottom: Left, Delaunay triangulation of a sampling performed upon the area map (leading to uniform mesh) of a mushroom-shape model. Middle, after minimization of local area dispersion. Right, the remeshed model. Note the uniformity obtained despite the strong area distortion due to the flattening process.*

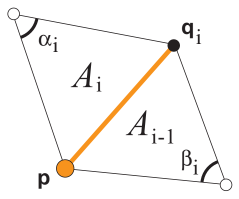
4.4.3 Geometry Optimization

In addition to the connectivity optimization, we also perform a small geometry optimization to improve the geometric quality of the mesh. We perform a weighted Laplacian flow in the parameterization by moving every vertex $\mathbf{p}$ that does not belong to the feature graph by:

$$\Delta \mathbf{p} = \Delta t \sum_{i \in \mathcal{N}_\infty(\mathbf{p})} w_i(\mathbf{q}_i - \mathbf{p})$$

where Δt is a step chosen sufficiently small (*e.g.*, 0.1), $\mathcal{N}_\infty(\mathbf{p})$ is the set of adjacent vertex indices to vertex $\mathbf{p}$, and $\mathbf{q}_i$ corresponds to the i^{th} adjacent vertex to $\mathbf{p}$.

Depending on the choice of remeshing that the user made when selecting the control map, we perform an adequate optimization by choosing the weights w_i so as to minimize an appropriate quantity. For example, if the users require a uniformly resampled mesh, we can minimize the local area dispersion by using the following weighting: $w_i = \frac{(A_i^{3D} \cdot cot(\alpha_i) + A_{i-1}^{3D} \cdot cot(\beta_i))}{\sum_{j=1}^{n} A_j^{3D}}$, where α_i and β_i are the opposite angles in the parameterization as depicted, and A_i^{3D} and A_{i-1}^{3D} are the 3D face areas to the left and right of $\mathbf{pq}_i$.



This novel weighting has the quality of inducing no changes if the triangles are already of equal sizes, while producing a Laplacian smoothing ([MDSB02]) to iteratively improve the quality otherwise. The result of such an optimization can be seen in the bottom of Figure 4.8. The area distortion minimization is only a particular instance of the more general mesh optimization we offer. The area terms in the previous weights can be substituted by other values, based on the control map used. For a curvature-based map for instance, we replace the area terms by integrals of the control map over the associated triangles. Indeed, a single pass over the control map suffices to collect the integral of the map over each triangle. These integrals, measuring the ”amount of curvature” (or amount of anything the control map measures) contained in a triangle, are therefore appropriate weighting values if one wants to guarantee a triangulation adapted to the control map. This efficient smoothing generally happens in a matter of seconds, leading for instance to the results on Figure 4.11.

4.4.4 Combined Optimization

Our system can create a variety of optimizations by alternating between connectivity and geometry optimization stages. For instance, uniform meshes can be obtained by alternating edge swaps favoring regularity with geometry optimization iterations minimizing area dispersion (see

Figure 4.7 bottom). If the user wishes to create the “rib” effect [BK01], she can simply alternate edge swaps which favor regularity and a univariate Laplacian smoothing of the feature vertices (Figure 4.6, bottom right). Additional results are given in the following section.

4.5 Remeshing Results

Our current implementation is written in C++ using a sparse matrix structure, biconjugate gradient and SSOR preconditioning for computing the conformal parameterization. All operations on the maps are performed using a standard image processing library, using OpenGL hardware whenever possible. The sampling previews use standard OpenGL texture mapping. All result timings are given for a 1 GHz PIII with 256 MBytes of memory. Figure 4.8 illustrates uniform remeshing of the *fandisk* at various resolutions using a 800×800 control map. Note how the 1D error diffusion performs well all the way from 200 vertices to higher complexity along the

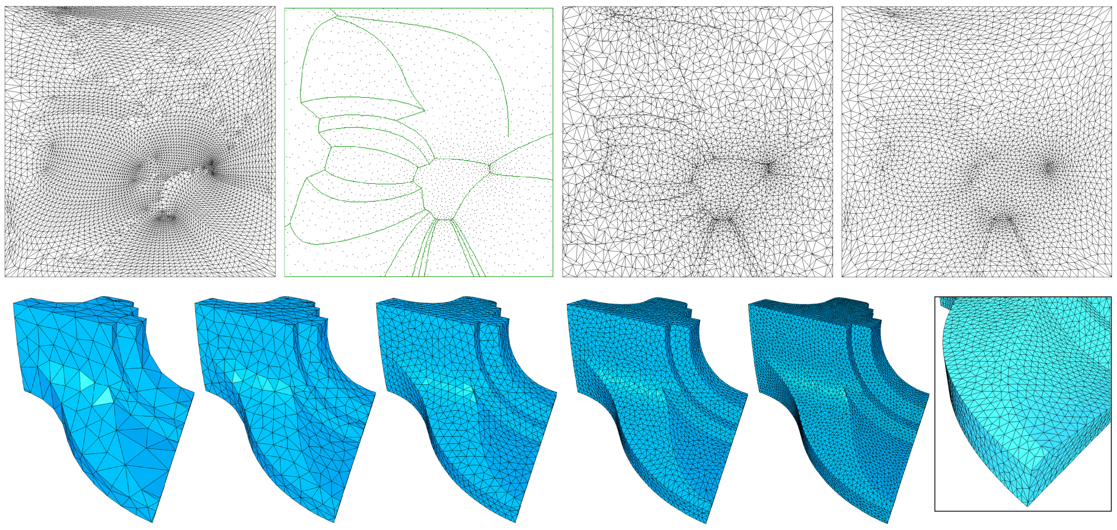


Figure 4.8: *Uniform remeshing of the fandisk. Top: conformal parameterization, and sampling obtained by error diffusion with 2.5k vertices with superimposed feature skeleton. Middle: result of constrained Delaunay triangulation before and after uniformity optimization. Bottom: several uniform remeshings with 0.2, 0.6, 1.4, 2.5 and 50k vertices respectively. Note the excellent behavior of the 1D error diffusion along the backbones, leading to consistent density between sharp edges and planar areas.*

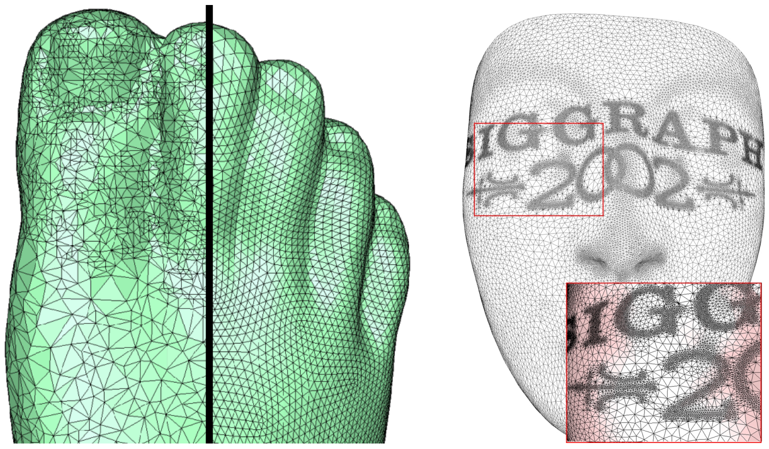


Figure 4.9: *Left: Semi-regular remeshing of a foot model. Right: Mesh created by pasting an image on the importance map (useful for animations and displacement maps).*

backbones of the feature skeleton. The conformal mapping is performed in $3.1\ s$ using SSOR with over-relaxation, and all the maps are computed in $1.2\ s$ total, while each sampling is done at an interactive rate in $160\ ms$. For the 2.5k vertex version the constrained Delaunay triangulation [She96] takes $190\ ms$, and the optimization stage takes $5\ s$ overall. The final 3D mapping takes $250\ ms$. Note that our goal of sampling at interactive rates is achieved, greatly increasing the user's productivity and workflow. Figure 4.10 illustrates an example of uniform geometry remeshing of the *MaxPlanck* model using a 3 patch atlas. The original mesh (23kV) is uniformly remeshed to the requested 8.3kV. Notice that the final, remeshed model shows no signs that it was created using 3 independent patches. In Figure 4.11 the *MaxPlanck* model is remeshed with various transfer functions over the curvature map. The first example is uniform (*i.e.*, flat transfer function) with 15kV, and the three following examples are generated using a progressively increasing gamma function over the curvature map. All intermediate meshes ranging from uniform to adapted sampling can be obtained easily just by increasing the gamma or other custom transfer functions. Figure 4.9 (left) shows a semi-regular remeshing of the *foot* model by applying regular subdivision in parametric space over a uniform base mesh. Figure 4.9 (right) illustrates a custom-tailored sampling.

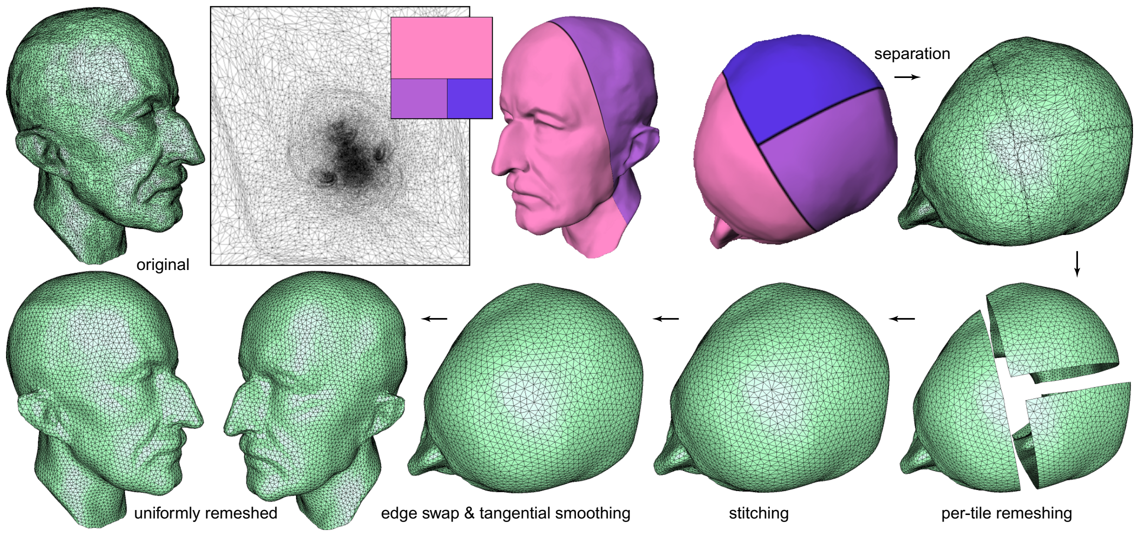


Figure 4.10: *Uniform remeshing of the MaxPlanck model. In clockwise order: Original mesh; Conformal parameterization; Parameterization-driven tiling with tree tiles requested; Three tiles meet at a corner; Mesh separation from the tiling; Burst view of the three tiles after independent uniform remeshing; The tiles put together require vertex stitching at the boundaries; A post-process swaps some edges and performs tangential smoothing along the stitching line; and the new model after uniform remeshing.*

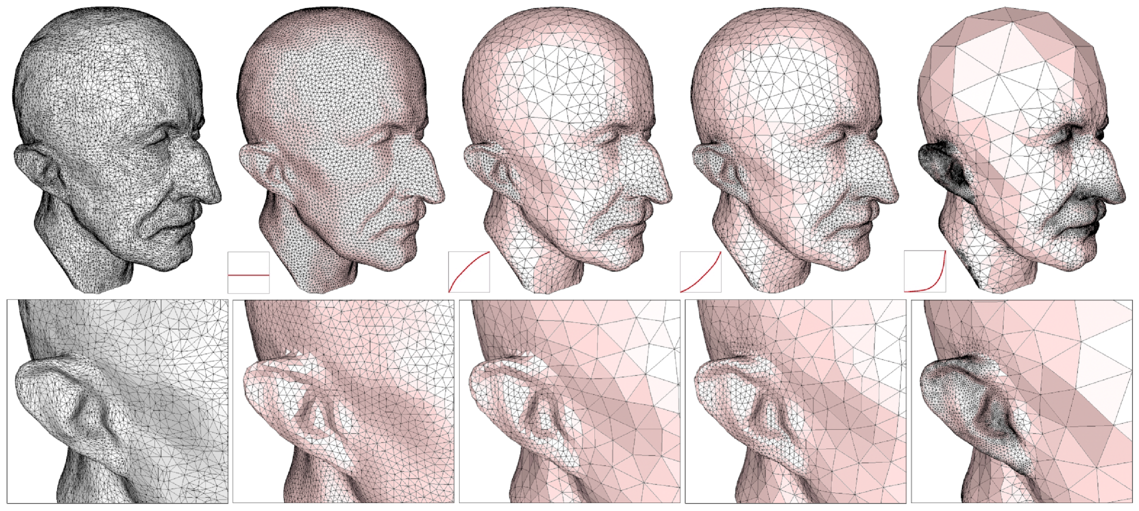


Figure 4.11: *Remeshing of the MaxPlanck model with various distribution of the sampling with respect to the curvature. The original model (left) is remeshed uniformly and with an increasing importance placed on highly curved areas (left to right) as the magnified area shows.*

Chapter 5

Parameterization

Parameterization of discrete surfaces is a fundamental and widely used operation in graphics, required, for instance, for texture mapping or remeshing. As 3D data becomes more and more detailed, there is an increased need for fast and robust techniques to automatically compute least-distorted parameterizations of large meshes. In this chapter, we present new theoretical and practical results on the parameterization of triangulated surface patches. Given a few desirable properties such as rotation and translation invariance, we show that the only admissible parameterizations form a two-dimensional set and each parameterization in this set can be computed using a simple, sparse, linear system. Since these parameterizations minimize the distortion of different intrinsic measures of the original mesh, we call them *Intrinsic Parameterizations*. In addition to this partial theoretical analysis, we propose robust, efficient and tunable tools to obtain least-distorted parameterizations automatically. In particular, we give details on a novel, fast technique to provide an optimal mapping without fixing the boundary positions, thus providing a unique *Natural Intrinsic Parameterization*. Other techniques based on this parameterization family, designed to ease the rapid design of parameterizations, are also proposed.

5.1 Introduction

Parameterization is a central issue in graphics. Parameterizing a 3D mesh amounts to computing a correspondence between a discrete surface patch and an isomorphic planar mesh through a piecewise linear function or *mapping*. In practice, this piecewise linear mapping is simply

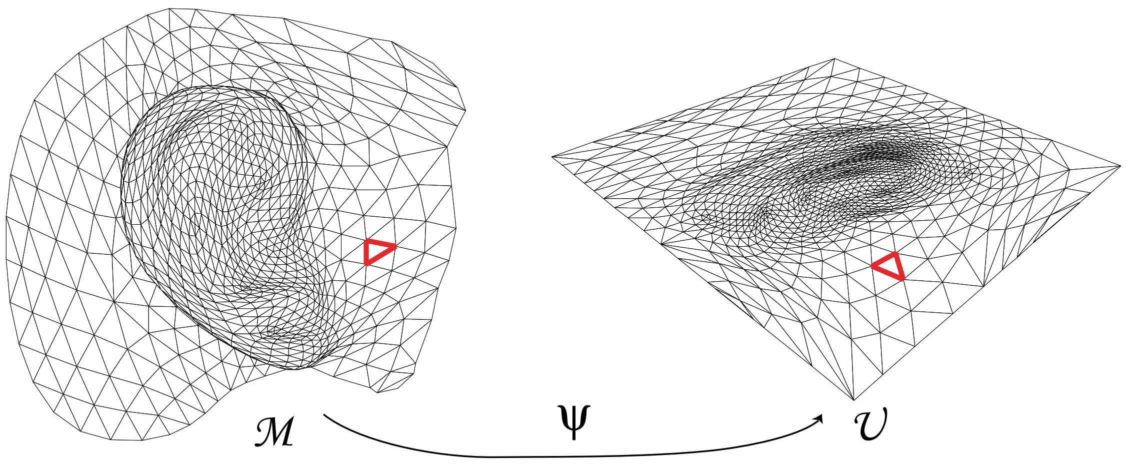


Figure 5.1: *A piecewise linear mapping between a 3D mesh $\mathcal{M}$ and an isomorphic flat mesh $\mathcal{U}$, where a triangle on the mesh is mapped to a triangle in the parameterization.*

defined by assigning each mesh node a pair of coordinates (u, v) referring to its position on the planar region. Such a one-to-one mapping provides a flat parametric space, allowing one to perform any complex operation directly on the flat domain rather than on the curved surface. This facilitates most forms of mesh processing, such as surface fitting, remeshing, or texture mapping. This last application, for instance, is widely used in Graphics as it dramatically enhances the visual richness of a 3D surface, both for overly simplified character meshes in game engines as well as for incredibly detailed complex surfaces in computer-generated feature films. Unfortunately despite numerous existing parameterization techniques [Flo97, Lév01] and commercial applications (Maya, Softimage, Flesh), it usually takes several hours of tedious (u, v) adjustments for a talented user to map a texture correctly (i.e., with acceptable distortions) onto an arbitrary surface.

This failure can be partially blamed on the intrinsic difficulty of the problem at hand. Since we are basically trying to flatten a surface from 3D down to 2D, there is in general *no perfect way* to perform such a flattening without introducing some form of distortion. However, most existing techniques, supposed to minimize distortion, do not result in a visually "smooth parameterization", even more so for very irregular meshes issued from scanners as demonstrated in Figure 5.2.

5.1.1 Problem Statement and Conventions

In this chapter, we will deal with the following problem: *Given a piecewise linear mesh patch $\mathcal{M}$, possibly with holes but non-closed, construct a piecewise linear mapping ψ between $\mathcal{M}$ and an isomorphic planar triangulation $\mathcal{U} \in I\!R^2$ that best preserves the original, intrinsic characteristics of $\mathcal{M}$.* Throughout the chapter, we will denote by $\mathbf{x}_i$ the 3D position of the i^{th} node in the original mesh $\mathcal{M}$, and by $\mathbf{u}_i$ the 2D position (parameter value) of the corresponding node in the 2D mesh $\mathcal{U}$. We will also use the self-explanatory notation: $\mathbf{x}_i = (x_i, y_i, z_i)^t$, $\mathbf{u}_i = (u_i, v_i)^t$. Parameterizing a mesh is therefore providing the piecewise linear mapping ψ (see Figure 5.1) such as:

$$\begin{aligned} \psi : \quad & \mathcal{M} \rightarrow \mathcal{U} \\ & \mathbf{x}_i \rightarrow \mathbf{u}_i . \end{aligned}$$

Texturing the mesh $\mathcal{M}$ will then be as simple as pasting a picture onto the parameter domain, and mapping each triangle of the original mesh $\mathcal{M}$ with the part of the picture present within the associated triangle in the parameter plane.

5.1.2 Background

Due to its primary importance for any subsequent mesh manipulation, the subject of mesh parameterization has been researched for a number of years, and not only in Computer Graphics.

Computer Graphics: A significant body of work on parameterization has been published over the last ten years in Computer Graphics. Almost all techniques explicitly aim at producing least-distorted parameterizations, and vary only by the distortions considered and the minimization processes used. Early work used the notion of flattening to obtain an isomorphic planar triangulation [BVIG91, MYV93, SF96], often minimizing discrete variables in the process, such as the ratio of angles between the 3D triangles and their associated 2D versions [CCFM99, SdS00, CCM01]. Others considered spring-like energies [KCP92, PM93, CS98, KL96, Flo97, LSS$^+$98, PFH00, FR01, LM98] that can be minimized quickly by a linear system solver when the boundary has been fixed to an arbitrary contour (with the noticeable exception of [Lév01] where only a few internal points need to be fixed by the user).

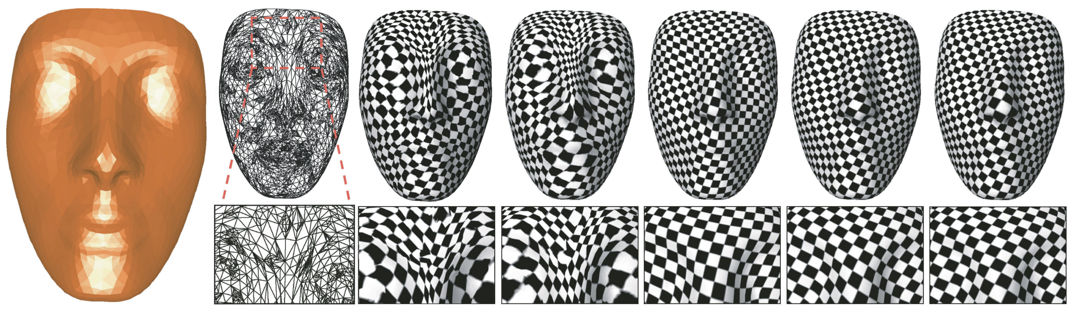


Figure 5.2: *Intrinsic Parameterizations: Most previous parameterization techniques (b-c) are not robust to mesh irregularity, exhibiting large distortions for highly irregular, yet geometrically smooth meshes such as in (a). Non-linear techniques (d) can achieve much better results, but often require several minutes of computational time. In comparison, with the exact same boundary conditions, our technique quickly generates very smooth parameterizations, regardless of the mesh irregularity (sampling quality) as demonstrated by the two texture-mapped members (e-f) of the novel parameterization family (denoted* Intrinsic Parameterizations*) that we introduce in this chapter.*

The Discrete Conformal Parameterization (DCP) has been proposed independently by a number of authors [PP93, EDD$^+$95, HAT$^+$00] who derived the same linear condition for conformality either using Differential Geometry, harmonic maps, or Finite Elements. Here again, a boundary condition is needed to induce a conformal mapping.

Finally, one can use nonlinear formulations to define an optimal parameterization [HG00, PSS01]. The MIPS method for instance finds a “natural boundary” that minimizes their highly non-linear energy [HG00]. Unfortunately, this requires quite a computational effort (even if hierarchical solvers can be used [HGC98, SGSH02]) for a result visually very close to the DCP. Sander and co-authors [SSGH01] proposed yet another nonlinear energy for the specific problem of texture stretch distortion.

Most of these techniques proposed to minimize a continuous energy over a piecewise linear surface. However, the choice of the energy sometimes seems arbitrary, and most of them may visually result in non-smooth parameterizations and therefore non-smooth textured meshes, as demonstrated in Figures 5.2(b-c). Note that using [SSGH01] results in a smooth parameterization, but takes more than six minutes to converge since it requires a non-linear minimization. In our experience, the only parameterizations that consistently provide visually smooth parameterizations are Floater's [Flo97, Flo02].

Cartography: Concurrently, cartographers have been dealing with the parameterization of non-flat surfaces for centuries, in order to represent our spherical earth as flat maps. Their work has mainly focused on differential parameterization, and is therefore only marginally relevant in Computer Graphics in practice. It is however interesting to mention that it is well known in this field that a mapping of a curved surface can either be *authalic* (i.e., area-preserving), or *conformal* (i.e., angle-preserving). No mapping of the earth can be *isometric* (i.e., distance-preserving): as it would have to be *both* authalic and conformal, and this is strictly impossible for non-developable surfaces like a sphere and most other geometries. This chapter establishes similar results, but for discrete surfaces, extending the known continuous differential geometry results as well as providing insights for novel notions.

5.1.3 Overview

In this chapter, we restrict ourselves to the parameterization of non-closed triangulated surfaces since many existing papers already describe different techniques to split a closed object into a series of patches (also called atlas of charts [GH95, GWH01, SSGH01, LPRM02]). We demonstrate that the set of desirable mappings for such patches form a simple low-dimensional space (Section 5.2). Moreover, the two generative parameterizations of this space are the existing discrete conformal mapping and a novel discrete authalic mapping, and all other valid *intrinsic* parameterizations can be found by simply solving a sparse linear system as detailed in Sections 5.3 and 5.4. We also demonstrate that they generate smooth texture mappings even for highly irregular meshes. We then show how easily one can find an optimal parameterization without fixing boundary points, providing a **natural** parameterization, by simply adding natural boundary conditions. Finally, we quickly review the possible immediate extensions that one could do with this new parameterization family before concluding.

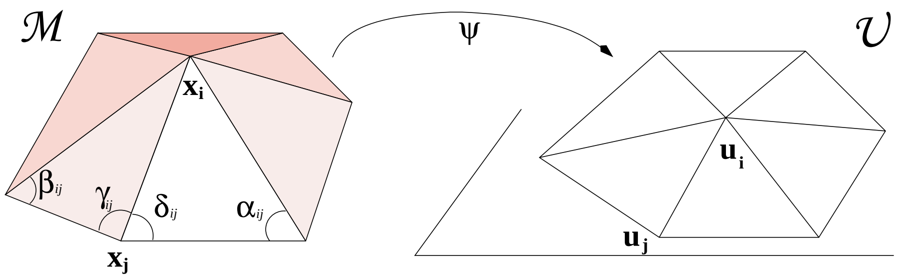


Figure 5.3: *A 3D 1-ring, and its associated flattened version.*

5.2 Distortion Measures for 1-Rings

We want to preserve as much of the intrinsic qualities of a surface as we possibly can during its parameterization, i.e., its flattening. This implies that we need to first define what these intrinsic qualities are for a discrete surface: minimal distortion will then mean best preservation of these qualities. In this section, we restrict our investigation to the distortion measures between *simple 1-ring neighborhoods*, and demonstrate that the appropriate measures actually form a low dimensional space.

5.2.1 Notion of Distortion Measure

As in the problem statement, let $\mathcal{M}$ be a simple mesh embedded in 3D consisting of a 1-ring neighborhood (i.e., a vertex and all its adjacent triangles), and let $\mathcal{U}$ be an isomorphic mesh: $\mathcal{U} \sim \mathcal{M}$ (we use the symbol $\sim$ to indicate isomorphy). Figure 5.3 shows the mapping between two simple 1-ring neighborhoods. We define a *distortion measure* between $\mathcal{M}$ and $\mathcal{U}$ as a functional E taking two isomorphic triangulations ($\mathcal{T}$) as inputs, and returning a real value:

$$\begin{aligned} E: \quad & \mathcal{T} \times \mathcal{T} \rightarrow \mathbb{R} \\ & (\mathcal{M},\mathcal{U}) \rightarrow E(\mathcal{M},\mathcal{U}). \end{aligned}$$

This kind of functional is sometimes referred to as a *mutual energy*, as it can be seen as a measure of the energy required to distort one into the other. By the very definition of a distortion measure, $E(\mathcal{M},\cdot)$ must be minimum for $\mathcal{M}$, as there is no mesh less distorted compared to $\mathcal{M}$

than itself.[1] We therefore have the following inequality for every $\mathcal{U}$ such that $\mathcal{U} \sim \mathcal{M}$:

$$E(\mathcal{M},\mathcal{M}) \leq E(\mathcal{M},\mathcal{U}). \tag{5.1}$$

For convenience, we will denote ϕ the distortion of a 1-ring with itself: $E(\mathcal{M},\mathcal{M}) = \phi(\mathcal{M})$. Thus, ϕ is a *measure* (sometimes called *energy*) of the triangulated surface. In order to further investigate what the appropriate distortion measures are for 1-rings, we now explore what the possible measures of a mesh are, since it will restrict the possible set of distortion measures.

5.2.2 Properties of Intrinsic Measures

A measure of a mesh is a *functional* ϕ which, given a piecewise-linear surface patch $\mathcal{M}$, basically returns a "score" $\phi(\mathcal{M})$. This functional must satisfy a few *basic properties*, that we now go over.

- **Rotation and Translation Invariance:** Obviously, we want the functional to be *invariant* to any translation or rotation of the mesh. Since these affine transformations do not affect the geometry of the mesh, the measure should remain identical. This will consequently render the parameterization independent of rotation and translation of the input mesh.

- **Continuity:** We also want the functional to be a discrete version of a continuous measure, consistent with the continuous, differential case. Thus, the functional needs to converge to a continuous measure as we get a finer and finer triangulation, under some possible additional conditions (such as bounded fatness, or more generally, non-degenerated triangulations). This is called *conditional continuity*, and is usually stated as:

  $$\phi(\mathcal{M}_n) \rightarrow \phi(\mathcal{M}) \text{ if } \mathcal{M}_n \rightarrow \mathcal{M} \text{ as } n \rightarrow \infty.$$

  Here again, this will induce a very natural property for our parameterizations.

- **Additivity:** A measure should also be *additive*, i.e.:

  $$\phi(\mathcal{M}_1 \cup \mathcal{M}_2) + \phi(\mathcal{M}_1 \cap \mathcal{M}_2) = \phi(\mathcal{M}_1) + \phi(\mathcal{M}_2).$$

[1]Note however that there generally exist other meshes, different from $\mathcal{M}$, that also achieve the same energy minimum.

The measure with such a property has the desirable quality of being *intrinsic*, that is to say, it *only depends on the surface itself*, not on its sampling. To illustrate this fact, consider the addition of one or several vertices onto the existing surface (along the boundary or inside a triangle for instance); it is easy to verify that the functional will still return the same measurement, since the real geometry of the surface is not affected – only its discretization, hence the term intrinsic. This sampling-independent property will be particularly attractive when dealing with large meshes, since hierarchical solvers will prove particularly efficient in solving for the parameterization.

5.2.3 Admissible Intrinsic Measures

Although the restrictions imposed on the notion of measure seem to be loose and natural, there are, surprisingly, only a small family of functionals that meet the requirements.

Minkowski Functionals of 2-Manifolds: A set of well-known functionals satisfies all the previous conditions. These are called the Minkowski functionals. For 2D surfaces, there are three such functionals: the **Area** $\phi^{\mathcal{A}}$, the **Euler characteristic** ϕ^{χ}, and the **Perimeter** $\phi^{\mathcal{P}}$ (length of the boundary) of a triangle mesh. It is straightforward to check that each of these functionals meet the three conditions we just listed. It is also interesting to notice that the two first ones (the perimeter being only a boundary measure) correspond respectively to the integrals of the determinants of the *first*— area element — and of the *second*— Gaussian curvature —*fundamental forms* [Gra98]. These are well-known to be *intrinsic* in the differential geometry sense, meaning that they could be computed by “inhabitants” of the surface having no knowledge of the actual embedding of the surface.

Admissible Functionals: Since we are looking for measures over a triangulation, a result dating back to the previous century explicitly states the set of all admissible functionals. A triangulation (considered as a 2-manifold, and disregarding its embedding) belongs to the *convex ring*, since it is the union of a set of triangles, therefore a union of convex bodies (which does *not* mean that the triangulation itself has to be convex). On this convex ring, Hadwiger [Had57] has proven that the only functionals, defined over the convex ring, matching the three conditions we mentioned above are *linear combinations of the Minkowski functionals* [2]. Therefore, the only admissible functionals fitting the three previous properties are *linear combinations of Area, Euler characteristic, and Perimeter*. The set of all admissible functionals is therefore a 3-dimensional space, and for any admissible measure ϕ, there exists a *unique* triplet of constants c_1, c_2, c_3 such that:

$$\phi = c_1 \; \phi^{\mathcal{A}} + c_2 \; \phi^{\chi} + c_3 \; \phi^{\mathcal{P}}. \tag{5.2}$$

Valid Distortion Measures Between 1-rings

Let's go back to our measures of distortion between two isomorphic 1-rings. Since the distortion measures must match the intrinsic measures, this restrains the admissible set to a special subset of the general case proven in [Rum96], because of the additional additivity and continuity conditions. We show in the next section that the *simplest* relevant distortion measures form a two-dimensional space.

5.3 Optimal 1-Ring Flattening

We now introduce the only quadratic distortion measures that fit the requirements that we derived in the previous sections. We show that their critical points, when a boundary condition is imposed, can be found by solving a simple linear equation. We start by developing the two most representative optimal mappings, the *discrete conformal parameterization* (DCP) and the novel *discrete authalic parameterization* (DAP), and demonstrate how all the others can be deduced from their formulation. We also point to some of the similarities between the differential and the discrete case.

5.3.1 Notion of Optimal Vertex Placement

We call an *optimal 1-ring parameterization* any mapping from a given 3D 1-ring $\mathcal{M}$ to an isomorphic 2D 1-ring $\mathcal{U}$ that is the minimum of a distortion measure (as previously defined) for a

[2]Hadwiger's book has never been translated in English. There are however several books [San76] and papers [SK97, KM00] that clearly state the aforementioned theorem.

fixed, given boundary mapping $\psi(\partial\mathcal{M})$. Therefore, if a distortion measure E is known and if each boundary vertex $\mathbf{x}_j$ has a given parameter value $\mathbf{u}_j$, the condition for the 2D 1-ring to be minimally distorted (i.e., optimal) is simply that $E(\mathcal{M},\mathcal{U})$ is minimum over all $\mathcal{U} \sim \mathcal{M}$, which yields this simple condition for the center node $\mathbf{u}_i$ of the 1-ring in the parameter plane:

$$\frac{\partial E}{\partial \mathbf{u}_i} = 0.$$

We shall now describe what the appropriate energies E are that define a distortion measure between two meshes. The first one is known under the name of Dirichlet energy.

5.3.2 Discrete Conformal Mapping

The first optimal mapping is actually already known. We now recall a bit of history and background to demonstrate the connection to our problem, and later build upon it.

Conformality on Differential Surfaces: While working on the area minimization problem introduced by the Belgian physicist Plateau, Mrs. Rado, Douglas, and later Courant proposed the use of the Dirichlet energy of a mapping instead of the highly nonlinear area functional previously used (see [PP93] for a good overview). The simple idea behind this functional [Gra98] is that, in differential geometry, the area of a patch $\mathcal{M}$ is:

$$\begin{aligned} \text{Area} &= \tfrac{1}{2}\int_{\mathcal{M}} |f_u \times f_v| \ dudv \leq \tfrac{1}{2}\int_{\mathcal{M}} \ |f_u|\,|f_v| \ dudv \\ &\leq \tfrac{1}{4}\int_{\mathcal{M}} (f_u^2 + f_v^2)\ dudv = \text{Dirichlet energy.} \end{aligned}$$

It is simple to verify that the first inequality becomes an equality iff $f_u \cdot f_v = 0$ everywhere, while the second does iff $|f_u| = |f_v|$ (deriving from the positivity of $(f_u - f_v)^2$). A further analysis [Gra98] shows that the minimum of this energy (quadratic in the parameterization) is the area, and is only attained for *conformal mappings*, i.e., mappings where the two previously introduced conditions on f hold. Conformality of the map equivalently means *angle preservation* since these conditions imply that any angle between two vectors on the parameter plane will be preserved through the mapping. In other words, in the differential case, it is known that a conformal map will result from the minimization of the Dirichlet energy.

Dirichlet Energy on Triangulations: Pinkall and Polthier provided a formal derivation of the Dirichlet energy between two triangles in [PP93] for piecewise linear parameterizations. Summing the energies over the whole 1-ring, they found:

$$E_{\mathcal{A}} = \sum_{\text{oriented edges}(i,j)} \cot\alpha_{ij}\ |\mathbf{u}_i - \mathbf{u}_j|^2, \tag{5.3}$$

where $|\mathbf{u}_i - \mathbf{u}_j|$ is the length of the edge (i, j) in the parameter domain, and α_{ij} is the opposite left angle in 3D as shown in Figure 5.3. This nicely complements the differential case since this is also a quadratic energy in the parameterization, and that this discrete energy depends *only on the angles* of the original surface. Indeed, the only term depending on the original surface is the cotangent term. This energy is also equal, at its minimum, to the total surface area $\phi^{\mathcal{A}}(\mathcal{M})$ when applied on the identity map (i.e., when $\mathbf{u}_i - \mathbf{u}_j$ is taken to be the actual 3D edge), and is therefore the exact equivalent of Equation 5.1: $E_{\mathcal{A}}$ represents a distortion measure that fits our requirements defined in Section 5.2.

Critical Point of $E_{\mathcal{A}}$: Mimicking the differential case, Pinkall and Polthier [PP93] proposed to define the *discrete conformal map* to be the critical point (a.k.a. the minimum) of the Dirichlet energy. Since this energy is quadratic, the derivation results in a simple linear system, that has a provably unique solution which is easy to compute once we fix the boundaries in the parameter domain. Notice that we can not formally claim this mapping to be angle-preserving, since there is generally no way to flatten a curved, discrete surface with a one-to-one correspondence of the 3D angles to the 2D angles. However, since the Dirichlet energy depends only on the 3D angles, and that in the differential case, the minimum of the Dirichlet energy is indeed conformal, this definition results in a visually satisfying parameterization as depicted in Figures 5.2(f) and 5.6(a). This explains the success (and the name) of this discrete conformal parameterization (DCP). Due to the simple formulation of this energy, deriving its critical point is rather simple, yielding the linear equation for the central node i:

$$\frac{\partial E_{\mathcal{A}}}{\partial \mathbf{u}_i} = \sum_{j \in \mathcal{N}_1(\mathbf{u}_i)} (\cot\alpha_{ij} + \cot\beta_{ij})\ (\mathbf{u}_i - \mathbf{u}_j) = 0. \tag{5.4}$$

Again, we can note that the linear coefficients are functions of only the angles of the original surface. We will describe in detail the computations required to numerically solve for the parameterization in Section 5.4, as well as an extension to natural boundary conditions.

2D Analogy: Consider a mesh vertex in flatland (2D) and its immediate neighboring vertices. Any motion of this vertex $\mathbf{u}_i$ in the plane will preserve the *1-ring area*. Therefore, computing the gradient of the 1-ring area with respect to $\mathbf{u}_i$ will provide a nontrivial equation (the gradient for each triangle being nonzero) that does always sum to zero (since the total area is constant). Not surprisingly, Appendix A.2 shows that we find the same coefficients as in Equation 5.4. Indeed, $\phi^{\mathcal{A}}$ is the area of mesh, and therefore, the coefficients of $\partial E_{\mathcal{A}}(\mathcal{M}, \mathcal{M})/\partial \mathbf{x}_i$ must match those of $\partial E_{\mathcal{A}}(\mathcal{M}, \mathcal{U})/\partial \mathbf{u}_i$ — giving an alternate, simple derivation of the conformality condition.

5.3.3 Discrete Authalic Mapping

Similarly to $E_{\mathcal{A}}(\mathcal{M}, \cdot)$ matching $\phi^{\mathcal{A}}$ on the identity map of $\mathcal{M}$, we now discuss the existence of a novel quadratic energy E_χ that matches the Euler characteristic χ on the identity map while being a valid distortion measure. Despite the relative simplicity of the optimality condition, we did not find any mention of it for the differential or the discrete case in the vast literature available. We will show, however, that this new condition has smoothness qualities similar to those of the conformality condition.

Hands-on Derivation: Remember that the Euler characteristic is the integral of the Gaussian curvature. Our differential operators show that the Gaussian curvature, hence the determinant of the second fundamental form, is equal to $2\pi - \sum_j \theta_j$ where the θ_j's are the tip angles around $\mathbf{u}_i$. Therefore, a similar gradient computation can be done for the *sum of the tip angles* around $\mathbf{u}_i$. Indeed, for a flat triangulation, this sum also remains constant (and is equal to 2π) as $\mathbf{u}_i$ moves within the plane. This time, we get new coefficients as proven in Appendix A.6. From this simple derivation, we now derive an appropriate energy E_χ in the next paragraph.

Chi Energy on Triangulations: Guided by the previous derivation, we introduce the following quadratic energy:

$$E_\chi = \sum_{j \in \mathcal{N}_1(\mathbf{x}_i)} \frac{(\cot \gamma_{ij} + \cot \delta_{ij})}{|\mathbf{x}_i - \mathbf{x}_j|^2} \, (\mathbf{u}_i - \mathbf{u}_j)^2 \, , \tag{5.5}$$

where the angles γ_{ij} and δ_{ij} are defined in Figure 5.3. This energy is constant for a given 1-ring when evaluated on the identity map, and therefore can always be scaled and shifted to be equal to χ (1 for a closed 1-ring). Additionally, since it is quadratic, we can show that it is greater than (or equal to) χ (after the above scaling and shifting) for any other map with the same boundary. This energy therefore satisfies all the properties we required in Section 5.2.

Critical Point of $\boldsymbol{E_\chi}$: Once again, the optimal parameterization deriving from E_χ is easily obtained when the center node $\mathbf{u}_i$ satisfies:

$$\frac{\partial E_\chi}{\partial \mathbf{u}_i} = \sum_{j \in \mathcal{N}(i)} \frac{(\cot \gamma_{ij} + \cot \delta_{ij})}{|\mathbf{x}_i - \mathbf{x}_j|^2} \, (\mathbf{u}_i - \mathbf{u}_j) = 0. \tag{5.6}$$

Duality of $\boldsymbol{E_\mathcal{A}}$ and $\boldsymbol{E_\chi}$: Now, the coefficients of both this optimality condition and of E_χ are shown (also in the appendix) to be only functions of local areas of the 3D mesh. This should not come as a complete surprise: remember that the Dirichlet energy, which derives from the determinant of the first fundamental form, a measure of the local **area extension** [Gra98], depends only on local **angles** and provides an **angle-preserving** mapping when minimized. Since χ is (up to a constant) the integral of the determinant of the second fundamental form, which is a measure of the local **angle excess** [Gra98], we have a dual situation here. The energy E_χ is now depending only on local **areas**, and we therefore denote it *Discrete Authalic Parameterization* (DAP) for the same reasons as the DCP. Solving for the optimality of E_χ, using numerical methods described in Section 5.4, results in smooth parameterizations as shown on Figures 5.2(e) and 5.6(b). Just like the DCP does for the angles, the DAP tries to preserve the area structure of the original 1-ring.

5.3.4 General Discrete Parameterization

As mentioned in Section 5.2.3, the set of all admissible measures are linear combinations of the Minkowski functionals. For fixed boundary conditions, the only distortion measures possible are linear combinations of the area and the angle distortion measures (note: the perimeter distortion does not induce a particular position for the center vertex being a lower-order (1D) distortion measure for the boundary only). Therefore, it results that the family of admissible, simple distortion measures of a 1-ring is reduced to *linear combinations of the two discrete distortion measures* defined above. A general distortion measure E as we defined can thus always be written:

$$E = \lambda E_{\mathcal{A}} + \mu E_{\chi},$$

where λ and μ are two arbitrary real constants. The optimality condition will simply be a linear combination of the two optimality condition we have described above. We call this 2-dimensional space of optimal discretizations *Intrinsic Parameterizations*, since they naturally derive from intrinsic measures of the input mesh. As demonstrated in Figure 5.2, they provide smooth parameterizations even on highly irregular meshes since they minimize *intrinsic distortions*. Caveat: Although the DCP can easily be proven to be *globally* optimal (and therefore, angle-preserving when the triangulation is fine enough), the DAP is, as far as we know, only *locally* optimal. This means that we should not expect the DAP to perfectly preserve the area distortion across the mesh, but only as best as possible between each 1-ring.

5.3.5 Connection to Barycentric Coordinates

There exists a direct connection between barycentric coordinates and parameterization. It was already noted in [Flo97, GS01] that the coefficients of the usual linear systems used to parameterize meshes can be interpreted as barycentric coordinates of each internal vertex within its 1-ring. If the linear system for parameterization really represents barycentric coordinates, then *any flat mesh will be its own parameterization*, since each vertex will not move from its original position within its 1-ring. Although this condition seems to be an obvious quality for a

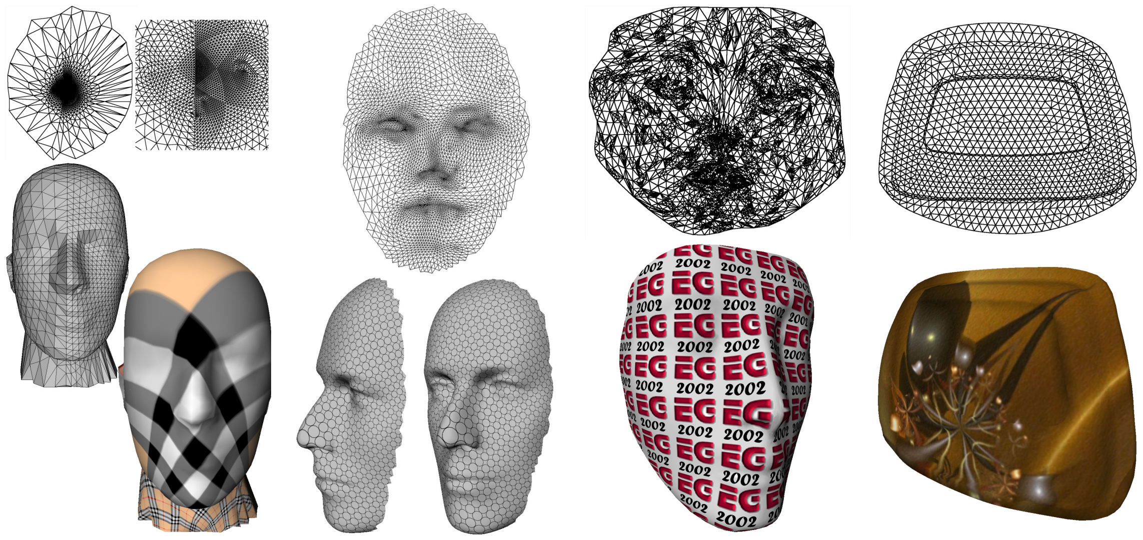


Figure 5.4: *Other Examples of Natural Conformal Maps: to demonstrate the conformality of the maps we obtain, we use an irregularly sampled mesh and observe that the symmetry is preserved despite the drastic change in sampling rate. The third natural parameterization uses the same mesh as in Figure 5.2. These four parameterizations were obtained in 0.8s, 0.5s, 1.8s, and 0.3s, respectively.*

“good” parameterization, only a few previous techniques satisfy this simple criterion. On the other hand, any linear combinations of the coefficients in Equations 5.4 and 5.6 defines perfectly valid barycentric coordinates [MLBD02]. We additionally proved that there is no other possible barycentric coordinates with the same properties, due to Hadwiger’s theorem.

5.4 Parameterizing Meshes

We now discuss the practical implementation of the theoretical results presented above, along with convenient improvements to further aid in the design of good parameterizations. We first give details on how to solve for the least-distorted parameterizations with a fixed boundary, then show how to interactively move the boundaries to further reduce the burden of designing texture mapped surfaces, and finally present a natural parameterization that automatically finds an optimal boundary.

5.4.1 Computing an Intrinsic Parameterization

Since the gradients of the energies introduced in Section 5.3.4 are linear, computing a parameterization reduces to solving a sparse linear system:

$$\mathbf{MU} = \begin{bmatrix} \lambda\mathbf{M}^{\mathcal{A}} + \mu\mathbf{M}^{\mathcal{X}} \\ \mathbf{0} \quad\quad\quad \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U}^{\text{internal}} \\ \mathbf{U}^{\text{boundary}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{C}^{\text{boundary}} \end{bmatrix} = \mathbf{C}\,,$$

where **U** is the vector of 2D-coordinates to solve for (separated for convenience into the internal vertices and the boundary vertices); **C** is a vector of boundary conditions that contains the positions where the boundary vertices are placed; and $\mathbf{M}^{\mathcal{A}}$ and $\mathbf{M}^{\mathcal{X}}$ are sparse matrices whose coefficients are given respectively by:

$$\mathbf{M}^{\mathcal{A}}_{ij} = \begin{cases} \cot(\alpha_{ij}) + \cot(\beta_{ij}) & \text{if } j \in \mathcal{N}_1(\mathbf{x}_i) \\ -\sum_{k\in\mathcal{N}_1(\mathbf{x}_i)} \mathbf{M}^{\mathcal{A}}_{ik} & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathbf{M}^{\mathcal{X}}_{ij} = \begin{cases} \left(\cot(\gamma_{ij}) + \cot(\delta_{ij})\right)/|\mathbf{x}_i - \mathbf{x}_j|^2 & \text{if } j \in \mathcal{N}_1(\mathbf{x}_i) \\ -\sum_{k\in\mathcal{N}_1(\mathbf{x}_i)} \mathbf{M}^{\mathcal{X}}_{ik} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Note that this technique can handle an arbitrary number of boundary curves (they are simply additional boundary vertices) and therefore easily parameterize patches containing holes. Once the boundary points have been chosen (either automatically or by the user), the sparse system is efficiently solved using Conjugate Gradient with an appropriate preconditioning (we recommend SSOR or inverse diagonal preconditioning — see [PFTV94]).

Constraints The user may possibly want to constrain certain points to given parameter values. This can be easily achieved using Lagrange multipliers. Each point constraint creates a linear equation relating the parametric values of the vertices of the enclosing triangle (using triangular barycentric coordinates) to the constrained position. We then add these additional constraints to the linear system using standard Lagrange multipliers. The previous linear system is then augmented to the following system:

$$\begin{bmatrix} \mathbf{M} & (\mathbf{M}^{\eta})^T \\ \mathbf{M}^{\eta} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \eta \end{bmatrix} = \begin{bmatrix} \mathbf{C} \\ \mathbf{C}^{\eta} \end{bmatrix},$$

where $\mathbf{M}^{\eta}_{ij}$ is nonzero only if the j^{th} constraint constrains the i^{th} vertex and 0 otherwise, and $\mathbf{C}^{\eta}_j$ is set to the j^{th} constrained position. Note that constraining a line is also possible by simply constraining the endpoints as well as the intersections of the line with the edges of the mesh.

5.4.2 Modifying Boundaries

In addition to simple fixed-boundary conditions, we can allow the user to interactively modify the positions of boundary points while updating the parameterization in *realtime*. This efficiency is achieved by taking advantage of the linear nature of our solution and precomputing how the parameterization responds to the movement of a boundary point.

Impulse response We first precompute the parameterizations that result from placing one boundary point at $(1, 1)$ and all others at $(0, 0)$ (these correspond to the Green functions of our parameterization equation):

$$\mathbf{M}\mathbf{b}^i = \mathbf{e}^i, \quad \forall i \in \text{boundary},$$

where $\mathbf{e}^i$ is a 1D vector (i.e., a vector of scalars) containing 1 in i^{th} position and 0 elsewhere, while $\mathbf{b}^i$ is the unknown 1D vector.

Real-time Boundary Manipulation By solving this system once for every boundary point, we construct a set of “basis parameterizations” that describe how the parameterization is altered by a change in a single boundary position. Indeed, the parameterization can then be efficiently updated as the user manipulates the boundary by noting that:

$$\mathbf{C} = \left[\sum_{i \in \text{boundary}} (\mathbf{u}_i^{\text{boundary}})^T \mathbf{e}^i \right] = \mathbf{M} \left[\sum_{i \in \text{boundary}} (\mathbf{u}_i^{\text{boundary}})^T \mathbf{b}^i \right],$$

where $\mathbf{u}_i^{\text{boundary}}$ is the position of i^{th} boundary point. Therefore, the parameterization for a given set of boundary points can easily be reconstructed in real-time, allowing real-time boundary manipulation, as:

$$\mathbf{U} = \sum_{i \in \text{boundary}} (\mathbf{u}_i^{\text{boundary}})^T \mathbf{b}^i.$$

This novel feature provides an easy tool for a user to optimize the design of a texture mapping on arbitrary surface patches.

Natural Boundaries / Natural Conformal Map

Interestingly, we can also solve a similar linear system while letting the computer pick the "best" boundaries. Earlier, we showed how to get a parameterization once a boundary was given, but we can also solve for an optimal conformal mapping by imposing *natural boundaries* (also called Neumann boundaries). This requires only minor modifications to the prior algorithm, and due to the quadratic nature of the energy, we will also obtain a unique solution. As demonstrated in Appendix A.2, we show that the derivative of the area energy on a triangle with respect to one of its vertices is equal to the opposite edge rotated by 90 degrees (such that $(x, y) \rightarrow (-y, x)$). A natural boundary condition is therefore to have the same property at the boundary. Summing over all adjacent triangles, the equation for the boundary point i (which we place into the matrix **M**) becomes:

$$\sum_{\Delta_{ijk}} \cot\alpha(\mathbf{u}_i - \mathbf{u}_j) + \cot\beta(\mathbf{u}_i - \mathbf{u}_k) = \sum_{\Delta_{ijk}} (\mathbf{u}_k - \mathbf{u}_j)^{\perp}, \qquad (5.7)$$

where α and β are the angles at k and j, and $\perp$ is a rotation by 90°. Note that this property also holds for interior vertices as the terms on the left become the conformal condition (Equation 5.4) and the terms on the right sum to zero.

To complete the minimization, we need to constrain two vertices to fix the rotation and translation of the resulting minimum parameterization. In our implementation, we constrain the two boundary vertices the farthest from each other to two arbitrary positions in the parameterization

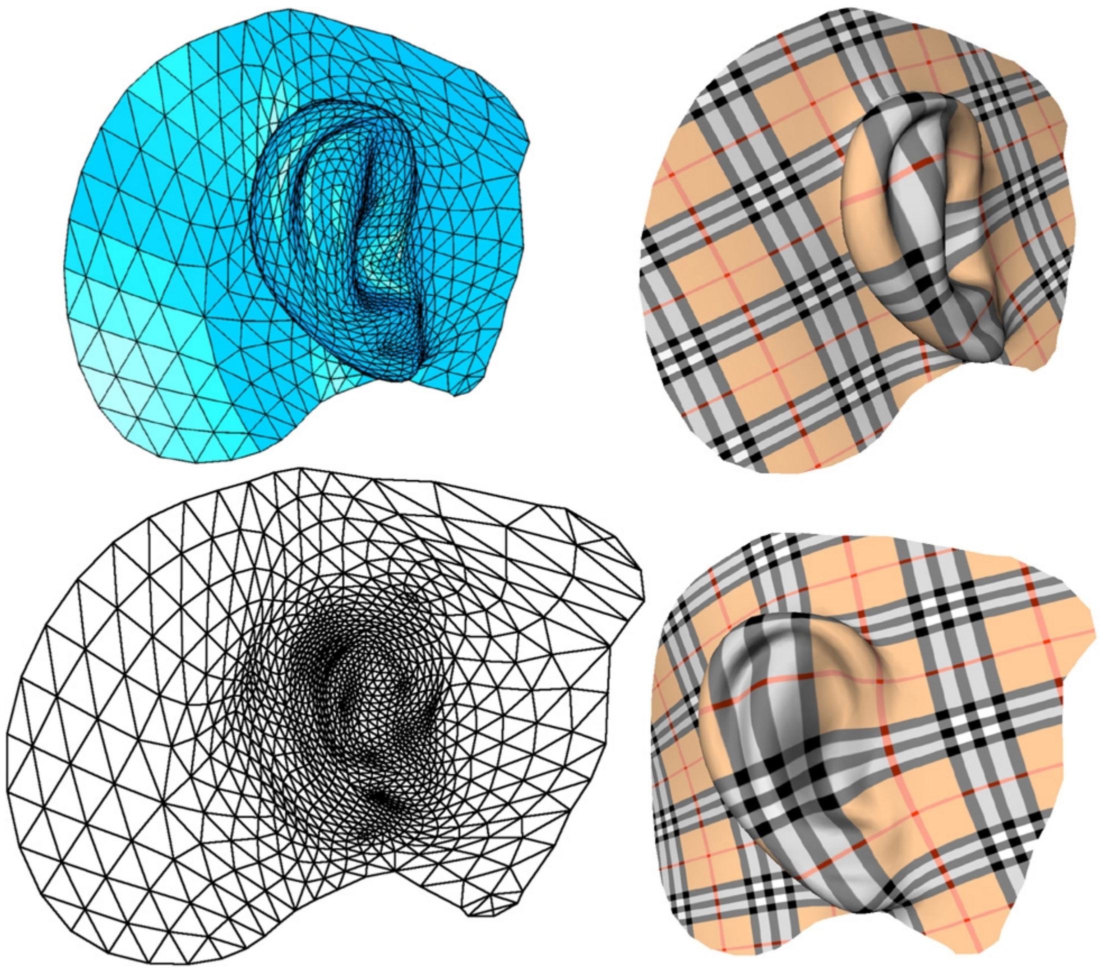


Figure 5.5: *Left: A 3D surface (top) and its natural conformal parameterization (bottom). Right: Views of the textured 3D surface.*

plane. This simple modification results in *Natural Conformal Maps*, such as those depicted on Figures 5.4 and 5.5. Notice that these parameterizations take the same amount of time to compute as the fixed-boundary ones, offering a very nice tool for initial flattening before minor editing of the boundaries if necessary. Note also that if the authalic coefficients are proven, in the future, to derive from a global energy, a similar treatment can be used to find natural boundaries.

5.5 Nonlinear Optimization of Maps

The theoretical and practical work introduced in this chapter opens many avenues. Aside from the natural conformal map and the entire family of intrinsic parameterizations by varying the parameters λ and μ, we can also compute parameterizations sufficiently close to these optimal ones to be visually smooth, but potentially more appropriate for a given application. As a final addition to our parameterization toolbox, we propose two simple algorithms to compute good parameterizations that minimize other types of functionals.

5.5.1 Near-Optimal Maps

We sometimes wish to minimize highly nonlinear energies while remaining within the space of the aforementioned intrinsic parameterizations. In order to make this tractable, we can linearize the solution space by assuming that all solutions can be expressed as a linear combination of the two base intrinsic parameterizations:

$$\mathbf{U} = \lambda \mathbf{U}^{\mathcal{A}} + (1-\lambda)\mathbf{U}^{\mathcal{X}}. \tag{5.8}$$

Note that $\mu = 1-\lambda$ to ensure that the solution always interpolates the same boundary as we vary λ. Since we restrained the vector space of solutions to only linear combinations of the intrinsic parameterizations, many nonlinear functionals can be minimized by a simple low-order polynomial minimization (often in real-time). Below are two simple examples of such functionals.

Edge-Length Distortion Minimization is achieved by minimizing the nonlinear energy:

$$E = \sum_{ij\in\text{Edges}} \left(\frac{|\mathbf{u}_i - \mathbf{u}_j|^2}{|\mathbf{x}_i - \mathbf{x}_j|^2} - 1\right)^2.$$

By substituting the values for $\mathbf{u}_i$ and $\mathbf{u}_j$ from Equation 5.8, the energy becomes a quartic polynomial in λ. This energy can then be minimized in real-time using a 3^{rd} order polynomial root finder to solve for: $\frac{dE}{d\lambda} = \sum_{ij\in\text{Edges}} 4\left(\frac{|\mathbf{u}_i-\mathbf{u}_j|^2}{|\mathbf{x}_i-\mathbf{x}_j|^2} - 1\right) s\frac{(\mathbf{u}_i-\mathbf{u}_j)\cdot[(\mathbf{u}_i^{\mathcal{A}}-\mathbf{u}_i^{\mathcal{X}})-(\mathbf{u}_j^{\mathcal{A}}-\mathbf{u}_j^{\mathcal{X}})]}{|\mathbf{x}_i-\mathbf{x}_j|^2} = 0.$

Area Distortion Minimization can be achieved by minimizing: $E = \sum_{ijk \in \text{Faces}} \left(\left(\frac{\mathcal{A}^{\text{param}}_{ijk}}{\mathcal{A}^{\text{3D}}_{ijk}} \right)^2 - 1 \right)^2$. As in the edge length distortion minimization, this results in a quartic polynomial in lambda and can be efficiently solved using a simple root finder. An example resulting from this technique is depicted in Figure 5.6

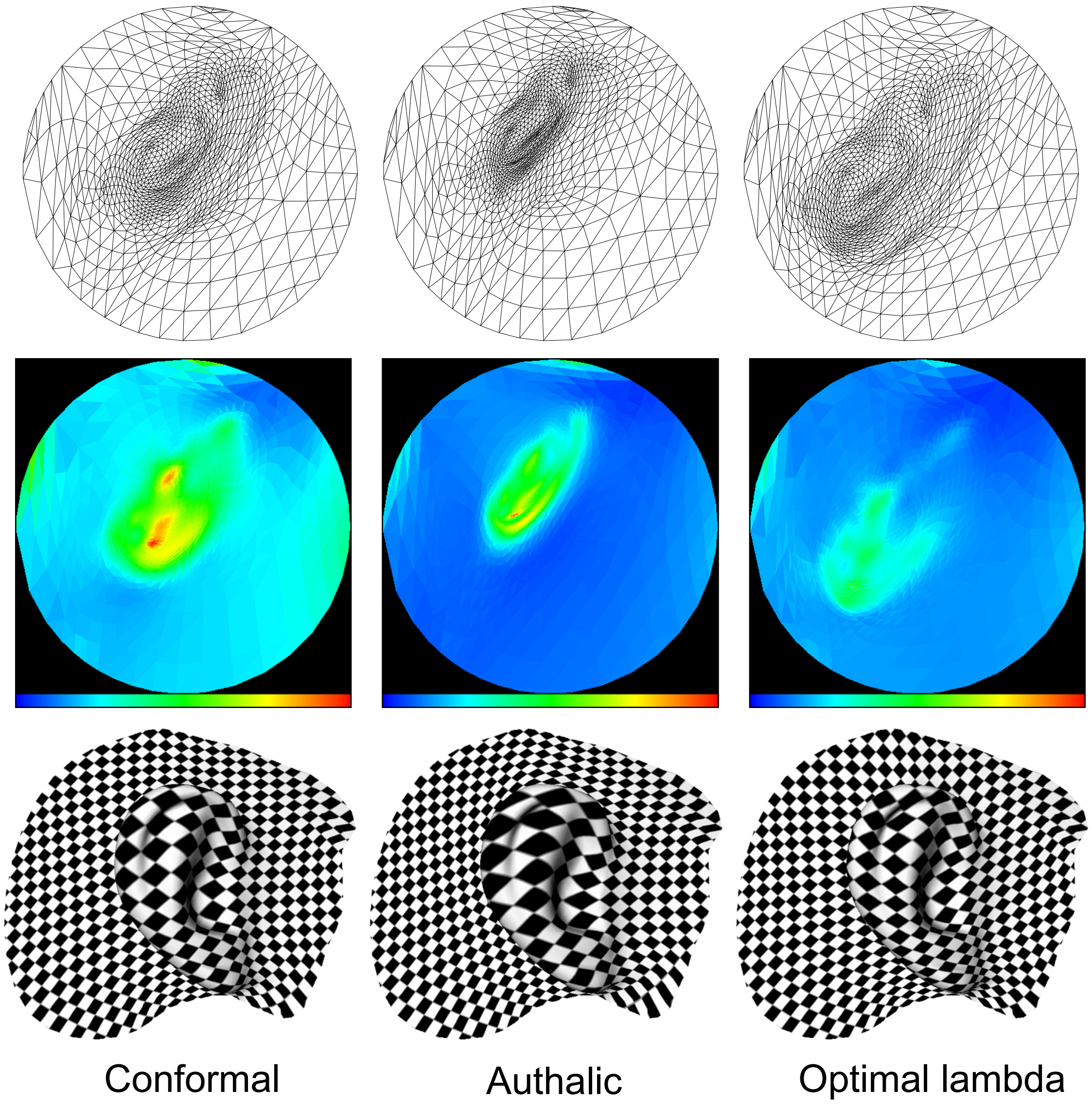


Figure 5.6: *Area Distortion Minimization can be achieved by optimizing the linear combination* $\lambda \mathbf{U}^{\mathcal{A}} + (1 - \lambda)\mathbf{U}^{\chi}$ *of the conformal and authalic parameterizations. The parameterizations (top) and the area distortion pseudo-coloring (middle) demonstrate the quality of the optimization.*

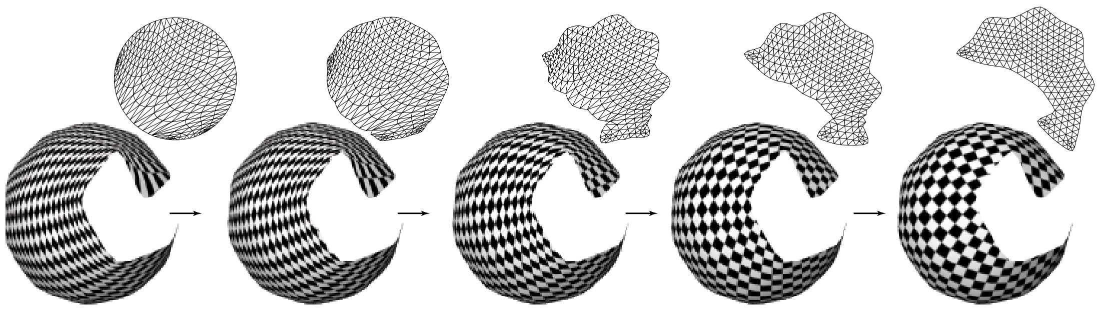


Figure 5.7: *Boundary Optimization: after choosing a (non)linear functional to minimize over the parameterization, we can move the boundary points to perform a gradient descent and optimize the parameterization. Here, an initial irregular spherical strip is mapped to a circle, then evolves towards an optimized parameterization (1.5 s) minimizing edge-length distortion.*

5.5.2 Boundary Optimization

Similar to the way we minimized an energy by modifying λ, we can, alternatively, minimize the energy by modifying the boundary of the parameterization. We first choose an appropriate energy (edge length distortion, area distortion, etc.), and then take its derivative with respect to each of the boundary points. Note that the terms of the form $\partial \mathbf{u}_i / \partial \mathbf{u}_p^{\text{boundary}}$ can be (pre)computed using the impulse response technique described in section 5.4.2. These derivatives are then used to perform a gradient descent to find a local minimum of the specified energy. Since the gradient descent is performed in terms of boundary points only (much fewer than the total number of points), this process is very efficient, and takes generally less than 10 seconds for several hundreds boundary vertices. A sequence of boundary optimizations using this method is depicted in Figure 5.7.

All the results we obtained were computed in less than 5 seconds for fixed-boundary and natural parameterizations, and less than 15 seconds for boundary-optimized maps. These techniques are thus very well suited for interactive parameterization design and form an extensive and powerful parameterization toolbox.

Chapter 6

Conclusions and Future Work

6.1 Contributions

This thesis has presented a family of operators for computing differential quantities on triangulated surfaces commonly found in computer graphics. We have derived operators for computing normal vectors and mean curvatures, Gaussian curvatures, principal curvatures and principal directions. These operators were derived using area averaging and a mixed finite element - finite volume technique resulting in a robust and consistent operator family. Since these operators were derived using properties of continuous differential geometry they preserve many of the properties used in the continuous setting. This allows one to use these operators, leveraging work in the continuous setting, to build an extensive and powerful mesh processing toolbox. In order to demonstrate the practical use of these operators, we have designed several novel digital geometry processing algorithms that are contributions in their own right:

Surface Smoothing: Chapter 3 presented an algorithm for smoothing triangulated surfaces. We presented an *implicit fairing* method, using implicit integration of a diffusion process that allows for both efficiency, quality, and stability. Since the umbrella operator used in the literature appears to have serious drawbacks, we defined a new scale-dependent umbrella operator to overcome undesired effects such as large distortions on irregular meshes. Additionally, by using the proposed mean curvature normal operator, we derived a curvature flow for surface smoothing that only depends only on the surface itself, not its parameterization/sampling.

This isotropic smoothing algorithm was then made anisotropic to account for, preserve and enhance the features of the mesh. A graph flow based technique was also presented that allows for independent shape and sampling smoothing. Additionally, since our differential operators generalize to arbitrary dimensional data, the above techniques were easily applied to other data types including images, height fields / range images, vector and tensor fields, and even volume data. By taking into account the way these datasets were acquired, we show that more accurate smoothing can be achieved.

Surface Remeshing: Chapter 4 presented a versatile technique for interactive geometry resampling that allows a very fine and intuitive control over the desired quality of the mesh. We substitute the original geometry by one or more 2D maps on which numerous surface quantities, including our differential operators, have been computed. We then use these maps to carry out the sampling. This has the advantage in that it allows us to use many image processing techniques as well as halftoning to both control the sampling as well as offer interactive updates. Once the initial, near-optimal resampling has been designed, a fast optimization is performed to perfect the resulting mesh. This map based remeshing pipeline allows the user to custom design the mesh based on their requirements at interactive rates thereby increasing their productivity in creating a wide variety of meshes.

Surface Parameteriztion: Chapter 5 presented a novel family of discrete surface parameterizations that we denote Intrinsic Parameterizations. We showed that they are the only parameterizations satisfying the proper conditions to make them easy to compute and robust to arbitrary meshes with guaranteed smoothness qualities. We have also proposed several algorithms using these parameterizations and automatically design optimal maps, with or without boundary conditions.

Thus, we have shown that our differential operators along with the presented algorithms create a versatile set of mesh processing tools. Hopefully, this work, combined with the efforts of other researchers, will form a basis for digital geometry processing and bring all the benefits that signal processing offers to audio, images and movies to the realm of 3D geometry.

6.2 Future Research

Future work on the discrete operators will try to answer some of the open questions. For instance, we are trying to determine what would be the minimum sampling rate of a continuous surface to guarantee that our discrete estimates are accurate within a given ϵ – laying the foundations for an irregular sampling theory. More generally, we would like to extend well-known digital signal processing tools and theorems to digital geometry. Additionally, we would like to test their use in other applications including geometry based subdivision schemes, and mesh simplification [HG99].

Although the implicit smoothing offers a performance jump over the explicit schemes, we believe that the computational time for the integration can still be improved on. We expect that multigrid preconditioning for the PBCG in the case of the scale-dependent operator for diffusion and for curvature flow would speed up the integration process. This multigrid aspect of mesh fairing has already been mentioned in [KCVS98], and could be easily extended to our method. Likewise, subdivision techniques can be directly incorporated into our method to refine or simplify regions according to curvature for instance. Additionally, wish to further study smoothing of volumetric data.

Many additional features can be added to our remeshing framework. We are investigating error diffusion in a quadtree data structure (to avoid the possibly large memory requirement of our current approach), anisotropic remeshing (possibly using ellipse packing) on a tensor control map of the principal curvatures, and hierarchical solving to accelerate the potentially slow parameterization stage. Finally, we plan to use our remeshing engine for other projects such as compression (how to remesh a surface to obtain the best rate/distortion tradeoff), as well as better geometric approximation.

Future work for the parameterization algorithm includes defining additional energies to optimize using the linear combination of our Intrinsic Parameterizations. Since these combinations of our basis parameterizations will be visually smooth, this provides us with a great deal of freedom in building other algorithms for parameterizations if more complex constraints must be enforced. Additional future work will focus on clarifying the relationship between our results and the existing body of work in Circle Packing, a technique which also provides the same kind of discrete mapping, but at the cost of a computationally expensive iterative process. Developing a good hierarchical solver as in [DCDS97] and in [SGSH02] could also speed up the process, making parameterization of extremely large meshes tractable. Finding optimal charts on a closed surface to locally parameterize a whole geometry is also of interest.

6.3 Subsequent Developments

Following our initial publication of the techniques described in this thesis, and preceding the final thesis publication, several researchers have begun to build upon this work.

Additional Discrete Operators: Recently, several researchers have continued to derive additional discrete differential operators. For example, Polthier [PP03], Tong [TLHD03] and their colleagues have developed operators to decompose vector fields into divergence-free, curl-free, and harmonic components. Additionally, Hildebrandt and Polthier [HP04] developed a discrete Shape Operator on edges and demonstrated its use for surface smoothing. Notice also recent work by Cohen-Steiner [CSM03] on curvature tensor estimation, also defined on edges.

Initial Work on Convergence and Error Analysis: Although the convergence and error analysis of many of the discrete operators is still an open research question, various researchers have begun to examine these topics. Cazals *et al.* [BCM03] have examined the pointwise convergence properties of several discrete operators including the Gaussian curvature defined in this thesis. Meanwhile, Dziuk and coauthors [DD03] have published several interesting papers on the accuracy of discrete surface flows, this time using Sobolev spaces to prove convergence.

Towards a Complete Discrete Differential Calculus: Recently, Desbrun and colleagues [DHLM04] have revisited the foundations of calculus (namely exterior calculus) in order to develop a complete discrete exterior calculus. The foundations of exterior calculus have been defined by many well-known mathematicians over the last two centuries (Poincaré, Lagrange, etc.). They established the basis of all differential and integral computations on smooth manifolds through a small set of operators (exterior derivative, Hodge star, etc.). By redefining these basic operators ab initio on discrete manifolds, Desbrun *et al.* can define any complex calculation on the discrete manifold by simply combining these basic building blocks. Therefore, these calculations will automatically satisfy the important differential properties (invariance to rotation, integration by parts, etc.). The link with our geometric operators remains to be determined, as a notion of discrete connection needs to be added to this DEC work in order to derive notions such as curvatures.

Appendix A

Additional Proofs

A.1 Mean Curvature Normal on a Triangulated Domain

In this appendix, we derive the integral of the mean curvature normal over a triangulated domain. We begin by computing the integral of the Laplacian of the surface point **x** with respect to the conformal parameter space. Using Gauss's theorem, we can turn the integral of a Laplacian over a region into a line integral over the boundary of the region:

$$\iint_{\mathcal{A}_\mathrm{M}} \Delta_{u,v}\mathbf{x}\; du\; dv = \int_{\partial\mathcal{A}_\mathrm{M}} \nabla_{u,v}\mathbf{x}\cdot\mathbf{n}_{u,v}\;\; dl, \tag{A.1}$$

where the subscript u, v indicates that the operator or vector must be with respect to the parameter space.

Since we assumed our surface to be piecewise linear, its gradient $\nabla_{u,v}\mathbf{x}$ is constant over each triangle of the mesh. As a consequence, whatever the type of finite-volume discretization we use, the integral of the normal vector along the border $\partial\mathcal{A}_\mathrm{M}$ within a triangle will result in the same expression since the border of both regions passes through the edge midpoints as sketched in Figure A.1(b). Inside a triangle $T = (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$, we can write:

$$\int_{\partial\mathcal{A}_\mathrm{M}\cap T} \nabla_{u,v}\mathbf{x}\cdot\mathbf{n}_{u,v}\; dl = \nabla_{u,v}\mathbf{x}\cdot[\mathbf{a}-\mathbf{b}]^{\perp}_{u,v} = \frac{1}{2}\nabla_{u,v}\mathbf{x}\cdot[\mathbf{x}_j-\mathbf{x}_k]^{\perp}_{u,v}\,,$$

where $^{\perp}$ denotes a counterclockwise rotation of 90 degrees.

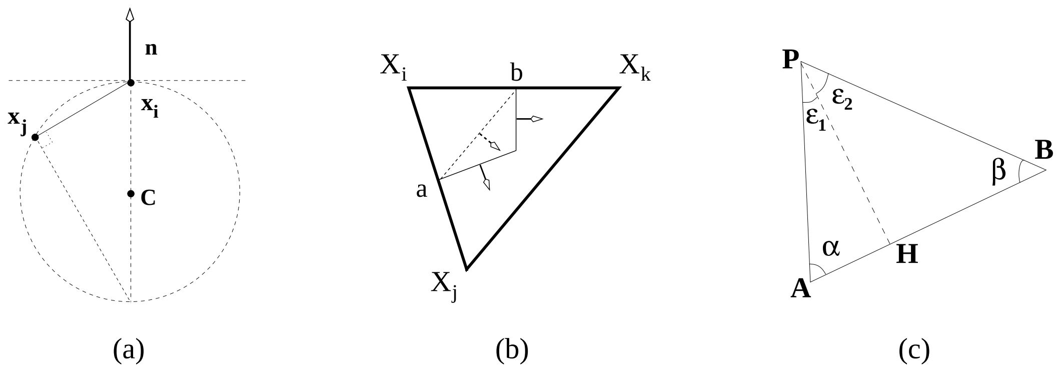


Figure A.1: *(a) Osculating circle for edge $\mathbf{x}_i\mathbf{x}_j$. (b) The integration of the surface gradient dotted with the normal of the region contour does not depend on the finite volume discretization used. (c) The area and angle gradients of triangle* **PAB** *can be computed from the edges and angles shown here.*

Since the function **x** is linear over any triangle T, using the linear basis functions B_l over the triangle, it follows:

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_i\ B_i(u,v) + \mathbf{x}_j\ B_j(u,v) + \mathbf{x}_k\ B_k(u,v) \\ \nabla_{u,v}\mathbf{x} &= \mathbf{x}_i\ \nabla_{u,v}B_i(u,v) + \mathbf{x}_j\ \nabla_{u,v}B_j(u,v) + \mathbf{x}_k\ \nabla_{u,v}B_k(u,v). \end{aligned}$$

Using the fact that the gradients of the 3 basis functions of any triangle T sum to zero and rearranging terms, the gradient of **x** over the triangle can be expressed as:

$$\nabla_{u,v}\mathbf{x} = \frac{1}{2\mathcal{A}_T}[(\mathbf{x}_j - \mathbf{x}_i)\left([\mathbf{x}_i - \mathbf{x}_k]^{\perp}_{u,v}\right)^T + (\mathbf{x}_k - \mathbf{x}_i)\left([\mathbf{x}_j - \mathbf{x}_i]^{\perp}_{u,v}\right)^T],$$

where ^{T} denotes the transpose. Note that $\nabla_{u,v}\mathbf{x}$ is an n x 2 matrix - n for the dimension of the embedding of **x** and 2 for the (u,v) space. The previous integral can then be rewritten as:

$$\begin{aligned} \int_{\partial\mathcal{A}\cap T} \nabla_{u,v}\mathbf{x}\cdot\mathbf{n}_{u,v}\ dl \quad &= \quad \frac{1}{4\mathcal{A}_T}\left[([\mathbf{x}_i - \mathbf{x}_k]\cdot[\mathbf{x}_j - \mathbf{x}_k])_{u,v}\ (\mathbf{x}_j - \mathbf{x}_i)\right. \\ &\quad \left.+([\mathbf{x}_j - \mathbf{x}_i]\cdot[\mathbf{x}_j - \mathbf{x}_k])_{u,v}\ (\mathbf{x}_k - \mathbf{x}_i)\right]. \end{aligned}$$

Moreover, the area $\mathcal{A}_T$ is proportional to the sine of any angle of the triangle. Therefore, we can use the cotangent of the 2 angles opposite to $\mathbf{x}_i$ to simplify the parameter space coefficients and write:

$$\int_{\partial \mathcal{A}_{\mathrm{M}} \cap T} \nabla_{u,v}\mathbf{x} \cdot \mathbf{n}_{u,v}\; dl = \frac{1}{2}[cot_{u,v}\ \angle(\mathbf{x}_k)(\mathbf{x}_j - \mathbf{x}_i) + cot_{u,v}\ \angle(\mathbf{x}_j)\ (\mathbf{x}_k - \mathbf{x}_i)].$$

Combining the previous equation with Eq. (2.4) and (A.1), using the current surface discretization as the conformal parameter space, and reorganizing terms by edge contribution, we obtain:

$$\iint_{\mathcal{A}_{\mathrm{M}}} \mathbf{K}(\mathbf{x}) dA = \frac{1}{2} \sum_{j \in N_1(i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ (\mathbf{x}_i - \mathbf{x}_j)\,,$$

where α_{ij} and β_{ij} are the two angles opposite to the edge in the two triangles sharing the edge $(\mathbf{x}_j, \mathbf{x}_i)$ as depicted in Figure 2.2(a).

A.2 Gradient of Area

In this section, we show that the mean curvature normal defined in this thesis (see Appendix A.1) has the property of being equivalent to the gradient of the surface area. In this and the following sections, we will make heavy use of Einstein summation notation for conciseness. For an introduction, see [Bar89].

Let's consider a point P of the mesh. Its neighbors, in counterclockwise order around P, and the points $\{A, B, ...\}$. We will denote the edge vector going, for example, from P to A as:

$$PA = A - P.$$

The only triangles affected by the movement of P are the adjacent faces. Let us consider a single face PAB whose area is thus: $\mathcal{A} = \frac{1}{2}\|PA \times PB\|$. So, using Einstein summation notation [Bar89], we have:

$$\mathcal{A}^2 = \frac{1}{4}\epsilon_{ijk}\ PA_j\ PB_k\ \epsilon_{ilm}\ PA_l\ PB_m,$$

where ϵ_{ijk} is the permutation symbol. Using the Kronecker delta δ_{ij}, and the ϵ-δ rule stating

$\epsilon_{ijk}\epsilon_{ilm} = \delta_{jl}\delta_{km} - \delta_{jm}\delta_{kl}$, we obtain:

$$\mathcal{A}^2 = \frac{1}{4}\left(PA_iPA_iPB_jPB_j - PA_iPB_iPA_jPB_j\right).$$

Straighforward differentiation (using $\frac{\partial P_i}{\partial P_q} = \delta_{iq}$) term by term yields:

$$\begin{aligned}
4\frac{\partial \mathcal{A}^2}{\partial A_q} &= -\delta_{iq}PA_iPB_jPB_j - \delta_{iq}PA_iPB_jPB_j \\
&\quad -\delta_{jq}PA_iPA_iPB_j - \delta_{jq}PA_iPA_iPB_j \\
&\quad +\delta_{iq}PB_iPA_jPB_j + \delta_{iq}PA_iPA_jPB_j \\
&\quad +\delta_{jq}PA_iPB_iPB_j + \delta_{jq}PA_iPB_iPA_j \\
&= -2PA_qPB_jPB_j - 2PA_iPA_iPB_q + PB_qPA_jPB_j \\
&\quad +PA_qPA_jPB_j + PA_iPB_iPB_q + PA_iPB_iPA_q \\
&= 2\left[PA_q(PA\cdot PB - PB\cdot PB) + PB_q(PA\cdot PB - PA\cdot PA)\right] \\
&= 2\left[AP_q(PB\cdot AB) + BP_q(PA\cdot BA)\right].
\end{aligned}$$

Using the fact:

$$\frac{\partial \mathcal{A}^2}{\partial P_q} = 2\mathcal{A}\frac{\partial \mathcal{A}}{\partial P_q},$$

we can solve for $\frac{\partial \mathcal{A}}{\partial P}$ as:

$$\frac{\partial \mathcal{A}}{\partial P} = \frac{1}{4\,\mathcal{A}}\left(AP(PB\cdot AB) + BP(PA\cdot BA)\right).$$

Noting that the dot product divided by the area is simply a cotangent produces:

$$\frac{\partial \mathcal{A}}{\partial P} = \frac{1}{4}\left(AP\cot\beta + BP\cot\alpha\right).$$

Performing this gradient on each of the neighboring triangles, summing, and reorganizing terms by edge contribution shows that the mean curvature operator defined in the previous section is equivalent to the gradient of the surface area:

$$\iint_{\mathcal{A}_{\mathrm{M}}} \mathbf{K}(\mathbf{x})dA = 2\nabla\mathcal{A}.$$

Additionally, since the area is equal to $\frac{1}{2}\|AB\|$ times the height $\|PH\|$, we have another simple expression for the gradient of the area of a triangle:

$$\nabla\mathcal{A} = |AB|\nabla(|PH|) = |AB|\frac{\mathbf{PH}}{|PH|} = \mathbf{AB}^{\perp}\,,$$

where “$\perp$” indicates a 90 degrees counterclockwise rotation about the triangle’s normal.

A.3 Area Gradient in nD

Notice that most of the calculations in the previous section did not rely on the dimensionality of the vertex data. Therefore, we can easily extend the area gradient to nD. In fact, the only quantity that must be extended to nD is the cross product, as the rest of the derivation does not rely on the dimension of the data. If we define the nD area (and similarly the nD cross product) as:

$$\begin{aligned}\mathcal{A} &= \frac{1}{2}||\mathbf{u}||||\mathbf{v}||sin(\mathbf{u},\mathbf{v}) = \frac{1}{2}||\mathbf{u}||||\mathbf{v}||\sqrt{1 - cos^2(\mathbf{u},\mathbf{v})} \\ &= \frac{1}{2}\sqrt{||\mathbf{u}||^2||\mathbf{v}||^2 - (\mathbf{u}\cdot\mathbf{v})^2}.\end{aligned}$$

and the cotangent then becomes:

$$cot(\mathbf{u},\mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{\sqrt{||\mathbf{u}||^2||\mathbf{v}||^2 - (\mathbf{u}\cdot\mathbf{v})^2}}\,.$$

Therefore the gradient of the surface area defined in the previous section naturally generalizes to any dimension.

A.4 Volume Gradient in nD

In this section, we generalize the notion of the area gradient to volumes, defining a volume gradient operator that can be used, for instance, in volume smoothing applications.

Let P, A, B, and C be four n-dimensional points. We can calculate the volume of the region (tetrahedron in 3D) formed by the three vectors originating at A:

$$a = PA \qquad b = PB \qquad c = PC.$$

We define a transformation of a 3D unit cube with axes u, v, w: $T(u, v, w) = au + bv + cw$. The Jacobian matrix J of this transformation is composed of three columns, a, b, and c:

$$J = \big(a|b|c\big) \tag{A.2}$$

The volume of the transformed unit cube is: $\iiint \sqrt{det\ G}\ \ dudvdw$, where $G_{ij} = J_{im}J_{jm}$ is the 3x3 metric tensor of the transformation.

The volume $\mathcal{V}$ we are looking for is therefore $\frac{1}{6}$ of the square root of the determinant of G (ratio between the untransformed and transformed cube). We can obtain this latter term through the standard formulation:

$$det\ G = \epsilon_{ijk} J_{1u} J_{2v} J_{3w} J_{iu} J_{jv} J_{kw}.$$

Expanding this expression, we find the following terms involving dot products:

$$\begin{aligned} det\ G = 2(a\cdot b)(a\cdot c)(b\cdot c) + (a\cdot a)(b\cdot b)(c\cdot c) \\ -(a\cdot b)^2(c\cdot c) - (a\cdot a)(b\cdot c)^2 - (a\cdot c)^2(b\cdot b). \end{aligned}$$

The remainder of the derivation is very similar to the surface area minimization in nD, detailed in the previous section. So, using the fact that:

$$\frac{\partial \mathcal{V}^2}{\partial A_q} = 2\mathcal{V}\frac{\partial \mathcal{V}}{\partial A_q}\,,$$

and that we have, as a consequence of Eq. (A.2): $\frac{\partial J_{ij}}{\partial A_q} = -\delta_{iq}$, we finally get the following terms for the gradient:

$$\begin{aligned} \tfrac{\partial \mathcal{V}}{\partial A_q} = \tfrac{1}{\mathcal{V}}(\quad & a_q && ((a\cdot c)(b\cdot b) + (b\cdot c)^2 + (a\cdot b)(c\cdot c) \\ & && -(b\cdot b)(c\cdot c) - (a\cdot c)(b\cdot c) - (a\cdot b)(b\cdot c)) \\ & +b_q && ((b\cdot c)(a\cdot a) + (a\cdot c)^2 + (a\cdot b)(c\cdot c) \\ & && -(a\cdot a)(c\cdot c) - (b\cdot c)(a\cdot c) - (a\cdot b)(a\cdot c)) \\ & +c_q && ((a\cdot c)(b\cdot b) + (a\cdot b)^2 + (a\cdot a)(b\cdot c) \\ & && -(a\cdot a)(b\cdot b) - (a\cdot b)(b\cdot c) - (a\cdot b)(a\cdot c))). \end{aligned}$$

Although we can use this expression to compute the gradient of the volume, it turns out we can simplify it using Lagrange's identity to get a better insight of what these terms are. Lagrange's identity in 3D can be written as:

$$(s \cdot u)(t \cdot v) - (s \cdot v)(t \cdot u) = (s \wedge t) \cdot (u \wedge v).$$

The multiplicative term in front of c_q is then $(PA \wedge PB) \cdot (CB \wedge CA)$, representing (up to the product of the norm of these vectors) the cosine of the dihedral angle between the two opposite faces to the edge c. As the volume is proportional to the sine of this angle, we can see that we once again have the same formula as Eq. (2.5), this time with cotangents of the dihedral angles opposite to the edge. Note that there are generally more than two tetrahedra sharing the same edge.

A.5 Preconditioned Bi-Conjugate Gradient for Smoothing

In this section, we describe the different implementation choices we made for the PBCG linear solver. With this simple configuration, we obtain an efficient linear solver for the implicit integration in the smoothing application.

Preconditioning

A good preconditioning, and particularly a multigrid preconditioning, can drastically improve the convergence rate of conjugate gradient solver. The umbrella operator (Eq. 3.7) has all its eigenvalues in $[-1, 0]$: in turn, the matrix A is always well conditioned for typical values of λdt. In practice, the simpler the conditioning the better. In our examples, we used the usual diagonal preconditioner $\tilde{A}$ with: $\tilde{A}_{ii} = 1/A_{ii}$, which provides a significant speedup with almost no overhead.

Convergence Criterion

Different criteria can be used to test whether or not further iterations are needed to get a more accurate solution of the linear system. We opted for the following stopping criterion after several tests: $||AX^{n+1} - X^n|| < \epsilon||X^n||$, where $||.||$ can be either the L_2 norm, or, if high accuracy is needed, the L_∞ norm.

Memory Requirements

An interesting remark is that we don't even need to store the matrix A in a dedicated data structure. The mesh itself provides a sparse matrix representation, as the vertex x_i and its neighbors are the only non-zero locations in A for row i. Computations can thus be carried directly within the mesh structure. Computing AX can be implemented by gathering values from the 1-ring neighbors of each vertex, while $A^T X$ can be achieved by "shooting" a value to the 1-ring neighbors.

A.6 Gradient of Angle

Despite an extensive literature search, we have not found any published derivation for the gradient of one of a triangle's angles with respect to its associated vertex. We therefore describe our derivation here.

Let $\mathcal{T} = (P, A, B)$ be a triangle (as shown in Figure A.1 (c)), and let H be the orthogonal projection of P onto the segment AB. We denote by ϵ_1 the angle $\widehat{APH}$, ϵ_2 the angle $\widehat{HPB}$, and ϵ the angle of $\mathcal{T}$ at P. Finally, we denote by α the angle at A and β the angle at B.

The gradient of ϵ with respect to **P** can be decomposed into the sum of the gradients of ϵ_1 and ϵ_2. Using the relation $cos(\epsilon_1) = |PH|/|PA|$, the gradient can be computed as:

$$\begin{aligned} \nabla\epsilon_1 &= \nabla \arccos(|PH|/|PA|) = -\frac{|PA|}{|AH|}\nabla(\frac{|PH|}{|PA|}) && \text{(A.3)} \\ &= -\frac{|PA|}{|AH|}\frac{\nabla(|PH|)\,|PA| - \nabla(|PA|)\,|PH|}{|PA|^2}\,. && \text{(A.4)} \end{aligned}$$

From the following identities: $\nabla|PA| = \mathbf{AP}/|PA|$, $\nabla|PH| = \mathbf{HP}/|PH|$, $\mathbf{HP} = \mathbf{HA} + \mathbf{AP}$, and $\cot\alpha = |AH|/|PH|$, we obtain:

$$\begin{aligned}\nabla\epsilon_1 \quad &= -\tfrac{|PA|}{|AH|}\left[\tfrac{\mathbf{HP}}{|PA|\,|PH|} - \tfrac{|PH|}{|PA|^3}\mathbf{AP}\right] \\ &= \tfrac{\mathbf{PA}}{|PA|^2}\left(\tfrac{|PA|^2}{|AH|\,|PH|} - \tfrac{|PH|}{|AH|}\right) + \tfrac{\mathbf{AH}}{|PH|\,|AH|} \\ &= \tfrac{\cot\alpha}{|PA|^2}\,\mathbf{PA} + \tfrac{\mathbf{AB}}{|PH|\,|AB|}\,. \end{aligned} \tag{A.5}$$

The gradient of ϵ_2 will cancel out the last term, leading to the simple formula:

$$\nabla\epsilon = \frac{\cot\alpha}{|PA|^2}\,\mathbf{PA} + \frac{\cot\beta}{|PB|^2}\,\mathbf{PB}\,.$$

Notice that the vector weights can be expressed only in terms of local areas: if K is the orthogonal projection of B onto PA, then $\cot\alpha/|PA|^2$ is equal to the area of the triangle (A, B, K) divided by twice the square of the total area of triangle $\mathcal{T}$.

Summing the contribution due to each triangle of a 1-ring, we obtain, with θ the total angle around a vertex $\mathbf{x}_i$ (in the notation of figure 5.3):

$$\nabla\theta = \sum_{j\in N_1(\mathbf{x}_i)} \frac{(\cot\ \gamma_{ij} + \cot\ \delta_{ij})}{||\mathbf{x}_i - \mathbf{x}_j||^2}\,(\mathbf{x}_j - \mathbf{x}_i).$$

Bibliography

[ABBFC97] G. Aubert, M. Barlaud, L. Blanc-Feraud, and P. Charbonnier. Deterministic edge-preserving regularization in computed imaging. *IEEE Trans. Imag. Process.*, 5(12), February 1997.

[ALM92] L. Alvarez, P-L. Lions, and J-M. Morel. Image selective smoothing and edge detection by nonlinear diffusion (II). *SIAM Journal of Numerical Analysis*, 29:845–866, 1992.

[AMD02] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 347–354. ACM Press, 2002.

[AvD95] L. Alboul and R. van Damme. Polyhedral metrics in surface reconstruction: Tight triangulations. *Mathematical Methods for Curves and Surfaces*, pages 309–336, 1995.

[AZ67] A. D. Aleksandrov and V. A. Zalgaller. *Intrinsic Geometry of Surfaces*. AMS, Rhode Island, USA, 1967.

[Bar89] Alan H. Barr. The Einstein summation notation: Introduction and extensions. In *SIGGRAPH 89 Course notes #30 on Topics in Physically-Based Modeling*, pages J1–J12, 1989.

[BCM03] V. Borrelli, F. Cazals, and J.-M. Morvan. On the angular defect of triangulations and the pointwise approximation of curvatures. *Comput. Aided Geom. Des.*, 20(6):319–341, 2003.

[BGH$^+$97] H. Borouchaki, P. L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specifications. *Finite Elements in Analysis and Design*, 25:61–83, 1997.

[BH96] Frank Bossen and Paul Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–76, oct 1996.

[BHF97] Houman Borouchaki, Frédéric Hecht, and Pascal J. Frey. Mesh gradation control. In *Proceedings of 6th International Meshing Roundtable, Sandia National Labs*, pages 131–141, oct 1997.

[BK01] Mario Botsch and Leif Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In *Eurographics Proceedings*, pages 402–410, sep 2001.

[Bor98] Houman Borouchaki. Geometric surface mesh. In *2nd International Conference on Integrated and Manufacturing in Mechanical Engineering*, pages 343–350, may 1998.

[BRK00] Mario Botsch, Christian Rössl, and Leif Kobbelt. Feature sensitive sampling for interactive remeshing. In *Vision, Modeling and Visualization Proceedings*, pages 129–136, 2000.

[BVIG91] Chakib Bennis, Jean-Marc Vézien, Gérard Iglésias, and André Gagalowicz. Piecewise surface flattening for non-distorted texture mapping. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):237–246, July 1991.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, pages 43–54, July 1998.

[CCFM99] Charlie C.L.Wang, Shiang-Fong Chen, Jin Fan, and Matthew M.F.Yuen. Two-dimensional trimmed surface development using a physics-based model. *Proceedings of the 25th Design Automation Conference*, Sept. 1999. Paper No. DETC99/DAC-8634.

[CCM01] Charlie C.L.Wang, Shiang-Fong Chen, and Matthew M.F.Yuen. Surface flattening based on energy model. *Computer Aided Design*, 2001.

[CDR00] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *IEEE Visualization*, pages 397–405, 2000.

[CGA] www.cgal.org: Computational Geometry Algorithms Library.

[CL96] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH 96 Conference Proceedings*, pages 303–312, 1996.

[CS92] Xin Chen and Francis Schmitt. Intrinsic surface properties from surface triangulation. In *European Conference on Computer Vision*, pages 739–743, May 1992.

[CS98] S. Campagna and H.-P. Seidel. Parameterizing meshes with arbitrary topology. In B. Girod H. Niemann, H.-P. Seidel, editor, *Image and Multidimensional Digital Signal Processing*, pages 287–290, 1998.

[CSM03] David Cohen-Steiner and Jean-Marie Morvan. Restricted delaunay triangulations and normal cycle. In *Proceedings of the Nineteenth Conference on Computational Geometry*, pages 312–321. ACM Press, 2003.

[dC76] Manfredo do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[DCDS97] Tom Duchamp, Andrew Certian, Tony DeRose, and Werner Stuetzle. Hierarchical Computation of PL Harmonic Embeddings". *Technical Report*, July 1997.

[DCG98] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Active implicit surface for computer animation. In *Graphics Interface (GI'98) Proceedings*, pages 143–150, Vancouver, Canada, 1998.

[dCS96] H. L. de Cougny and M. S. Shephard. Surface meshing using vertex insertion. In *Proceedings of 5th International Meshing Roundtable, Sandia National Labs*, pages 243–256, oct 1996.

[DD03] Klaus Deckelnick and Gerhard Dziuk. Mean curvature flow and related topics. In T. Shardlow J.F. Blowey, A.W. Craig, editor, *Frontiers in Numerical Analysis*, pages 63–108. Springer, 2003.

[DFG99] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tesselations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[DFRS03] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003.

[DHKL01] N. Dyn, K. Hormann, S.J. Kim, and D. Levin. Optimizing 3d triangulations using discrete curvature analysis. *Mathematical Methods for Curves and Surfaces, Oslo 2000*, pages 135–146, 2001.

[DHKW92] Ulrich Dierkes, Stefan Hildebrandt, Albrecht Küster, and Ortwin Wohlrab. *Minimal Surfaces (I)*. Springer-Verlag, 1992.

[DHLM04] Mathieu Desbrun, Anil Hirani, Melvin Leok, and Jerrold Marsden. Discrete exterior calculus. in preparation, 2004.

[DMA02] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. In *Proceedings of Eurographics*, 2002.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[Dzi91] G. Dziuk. An algorithm for evolutionary surfaces. *Numer. Math.*, 58, 1991.

[EDD$^+$95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH 95*, pages 173–182, August 1995.

[EHP02] Jeff Erikson and Sariel Har-Peled. Optimally cutting a surface into a disk. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry*, 2002. to appear.

[FEF98] G. Ford and A. El-Fallah. On mean curvature in non-linear image filtering. *Pattern Recognition Letters*, 19:433–437, 1998.

[Flo97] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. ISSN 0167-8396.

[Flo02] Michael Floater. Mean value coordinates. *Preprint*, 2002.

[For88] Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput.*, 51:699–706, 1988.

[FR01] Michael S. Floater and Martin Reimers. Meshless parameterization and surface reconstruction. *Computer Aided Geometric Design*, 18(2):77–92, March 2001.

[Fre00] Pascal J. Frey. About surface remeshing. In *Proceedings of the 9th Int. Meshing Roundtable*, pages 123–136, 2000.

[Fu93] J. Fu. Convergence of curvatures in secant approximations. *Journal of Differential Geometry*, 37:177–190, 1993.

[Fuj95] Koji Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. In *Proceedings of the AMS 123*, pages 2585–2594, 1995.

[GB98] P. L. George and H. Borouchaki, editors. *Delaunay Triangulation and Meshing Application to Finite Elements*. HERMES, Paris, 1998.

[GGH02] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *Proceedings of SIGGRAPH*, 2002.

[GH95] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Proceedings of SIGGRAPH 95*, pages 359–368, August 1995.

[GH98] Michael Garland and Paul Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization Conference Proceedings*, pages 263–269, 1998.

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 6(20):623–629, 1971.

[Gra98] Alfred Gray, editor. *Modern Differential Geometry of Curves and Surfaces*. Second edition. CRC Press, 1998.

[GS01] Craig Gotsman and Victor Surahhsky. Guaranteed intersection-free polygon morphing. *Computer and Graphics*, 25(1):67–75, 2001.

[GSS99] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH 99 Conference Proceedings*, pages 325–334, 1999.

[GVSS00] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In *Proceedings of SIGGRAPH*, pages 95–102, 2000.

[GWH01] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal meshes. *2001 ACM Symposium on Interactive 3D Graphics*, pages 49–58, March 2001.

[Had57] H. Hadwiger. *Vorlesungen Über Inhalt, Oberfläche und Isoperimetrie*. Springer-Verlag, 1957.

[Ham93] Bernd Hamann. Curvature approximation for triangulated surfaces. In G. Farin *et al.*, editor, *Geometric Modelling*, pages 139–153. Springer Verlag, 1993.

[HAT$^+$00] Steven Haker, Sigurd Angenent, Allen Tannenbaum, Ron Kikinis, Guillermo Sapiro, and Michael Halle. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):181–189, April-June 2000.

[HG99] Paul S. Heckbert and Michael Garland. Optimal triangulation and quadric-based surface simplification. *Journal of Computational Geometry: Theory and Applications*, November 1999.

[HG00] K. Hormann and G. Greiner. Mips: An efficient global parametrization method. In P.-J. Laurent, P. Sablonnière, and L. L. Schumaker, editors, *Curve and Surface Design: Saint-Malo 1999*, pages 153–162. Vanderbilt University Press, 2000.

[HGC98] K. Hormann, G. Greiner, and S. Campagna. Hierarchical parametrization of triangulated surfaces. In H.-P. Seidel B. Girod, H. Niemann, editor, *Proceedings of Vision, Modeling and Visualization*, pages 219–226, 1998.

[HLG01] K. Hormann, U. Labsik, and G. Greiner. Remeshing triangulated surfaces with optimal parameterizations. *Computer-Aided Design*, 33:779–788, 2001.

[HP04] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. In *Proceedings of Eurographics*, 2004.

[HS97] J. M. Hyman and M. Shashkov. Natural discretizations for the divergence, gradient and curl on logically rectangular grids. *Applied Numerical Mathematics*, 25:413–442, 1997.

[HSS97] J. M. Hyman, M. Shashkov, and S. Steinberg. The numerical solution of diffusion problems in strongly heterogenous non-isotropic materials. *Journal of Computational Physics*, 132:130–148, 1997.

[KCP92] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. *Proceedings of SIGGRAPH 92*, pages 47–54, July 1992.

[KCVS98] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multiresolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, July 1998.

[KDA97] P. Kornprobst, R. Deriche, and G. Aubert. Nonlinear operators in image restoration. In *CVPR'97*, pages 325–331, Puerto-Rico, 1997.

[KL96] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96*, pages 313–324, August 1996.

[KLM98] P. Kresk, G. Lukacs, and R. Martin. Algorithms for computing curvatures from range data. *The Mathematics of Surfaces VIII*, pages 1–16, 1998.

[KM00] J.G.E.M. Fraaije K. Michielsen, H. De Raedt. Morphological characterization of spatial patterns. *Prog. Theor/ Phys. Suppl.*, 138:453–548, 2000.

[KMS97] R. Kimmel, R. Malladi, and N. Sochen. Images as embedding maps and minimal surfaces: Movies, color, and volumetric medical images. In *IEEE CVPR'97*, pages 350–355, 1997.

[Kob97] Leif Kobbelt. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces '97*, pages 101–131, 1997.

[Lév01] Bruno Lévy. Constrained texture mapping for polygonal meshes. *Proceedings of SIGGRAPH 2001*, pages 417–424, August 2001.

[LK84] S. Lien and J. Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE CG&A*, 4(9), October 1984.

[LM98] Bruno Lévy and Jean-Laurent Mallet. Non-distorted texture mapping for sheared triangulated meshes. *Proceedings of SIGGRAPH 98*, pages 343–352, July 1998.

[LPRM02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM SIGGRAPH Proceedings*, July 2002.

[LPVV01] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proceedings of 17th Annu. ACM Sympos. Comput. Geom.*, pages 80–89, 2001.

[LSS$^+$98] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, July 1998.

[LT98] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization Proceedings*, pages 279–286, 1998.

[Max99] Nelson Max. Weights for computing vertex normals from facet normals. *Journal of Graphics Tools*, 4(2):1–6, 1999.

[MDSB02] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *VisMath ’02 Proceedings*, 2002.

[Mil95] Roger B. Milne. An adaptive level-set method. *PhD Thesis*, University of California, Berkeley, December 1995.

[MLBD02] Mark Meyer, Haeyoung Lee, Alan Barr, and Mathieu Desbrun. Generalized barycentric coordinates on irregular polygons. *Journal of Graphics Tools*, 7(1):13–22, 2002.

[Mor01] J.M. Morvan. On generalized curvatures. *Preprint*, 2001.

[MS92] Henry P. Moreton and Carlo H. Séquin. Functional minimization for fair surface design. In *SIGGRAPH 92 Conference Proceedings*, pages 167–176, July 1992.

[MS96] R. Malladi and J.A. Sethian. Image processing: Flows under min/max curvature and mean curvature. *Graphical Models and Image Processing*, 58(2):127–141, March 1996.

[MV97] Alan M. McIvor and Robert J. Valkenburg. A comparison of local surface geometry estimation methods. *Mach. Vision Appl.*, 10(1):17–26, 1997.

[MYV93] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. *Proceedings of SIGGRAPH 93*, pages 27–34, August 1993.

[OBB00] Yutaka Ohtake, Alexander G. Belyaev, and Ilia A. Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In *Proceedings of the Geometric Modeling and Processing 2000*, page 229. IEEE Computer Society, 2000.

[Ost01] Victor Ostromoukhov. A simple and efficient error-diffusion algorithm. In *Proceedings of SIGGRAPH*, pages 567–572, 2001.

[PFH00] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000.

[PFTV94] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: the Art of Scientific Computing*. Cambridge University Press, New York, USA, 2nd edition, 1994.

[PG01] Mark Pauly and Markus Gross. Spectral processing of point-sampled geometry. In *Proceedings of SIGGRAPH*, pages 379–386, 2001.

[PM90] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[PM93] Laxmi Parida and S.P. Mudur. Constraint-satisfying planar development of complex surfaces. *Computer Aided Design*, 25(4):225–232, April 1993.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PP03] Konrad Polthier and Eike Preuss. Identifying vector fields singularities using a discrete hodge decomposition. In H.C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 113–134. Springer Verlag, 2003.

[PR99] T. Preußer and M. Rumpf. Anisotropic nonlinear diffusion in flow visualization. In *IEEE Visualization*, pages 323–332, 1999.

[PS98] Konrad Polthier and Markus Schmies. Straightest geodesics on polyhedral surfaces. In H.C. Hege and K. Polthier, editors, *Mathematical Visualization*. Springer Verlag, 1998.

[PSS01] Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parameterizations. *Proceedings of SIGGRAPH 2001*, pages 179–184, August 2001.

[PV97] J-C. Léon P. Véron. Static polyhedron simplification using error measurements. *Computer-Aided Design*, 29(4):287–298, 1997.

[ROF92] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[Rum96] Martin Rumpf. A variational approach to optimal meshes. *Numer. Math.*, (72):523–540, 1996.

[RVSS00] A. Rassineux, P. Villon, J-M. Savignat, and O. Stab. Surface remeshing by local hermite diffuse interpolation. *International Journal for Numerical Methods in Engineering*, 49:31–49, 2000.

[San76] Luis A. Santalo. *Integral Geometry and Geometric Probability*. Addison-Wesley, 1976.

[SdS00] Alla Sheffer and E. de Struler. Surface parameterization for meshing by triangulation flattening. In *Proceedings of the 9th International Meshing Roundtable, Sandia National Laboratories*, pages 161–172, Oct. 2000.

[Set96] James A. Sethian. *Level-Set Methods: Evolving Interfaces in Geometry, Fluid Dynamics, Computer Vision, and Material Science*. Cambridge Monographs on Applied and Computational Mathematics, 1996.

[SF96] Meng Sun and Eugiene Fiume. A technique for constructing developable surfaces. *Graphics Interface '96*, pages 176–185, May 1996.

[SGSH02] Pedro Sander, Steven Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parameterization. *MSR Technical Report MSR-TR-2002-27*, 2002.

[Sha96] J. Shah. Curve evolution and segmentation functionals: Applications to color images. In *IEEE ICIP'96*, pages 461–464, 1996.

[She96] Jonathan R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Proceedings of the First workshop on Applied Computational Geometry, Philadelphia, Pennsylvania*, pages 123–133, 1996.

[She02] A. Sheffer. Spanning tree seams for reducing parameterization distortion of triangulated surfaces. In *Proceedings of Shape Modeling International*, 2002. to appear.

[Sim94] R. Bruce Simpson. Anisotropic mesh transformations and optimal error control. *Appl. Num. Math.*, 14(1-3):183–198, 1994.

[SK97] Jens Schmalzing and Martin Kerscher. Minkowsky functionals in cosmology. *Generation of Large-Scale Structure in Cosmology*, pages 255–260, 1997.

[SKM98] N. Sochen, R. Kimmel, and R. Malladi. A geometrical framework for low level vision. *IEEE Trans. on Image Processing*, 17(3):310–318, 1998.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. *Proceedings of SIGGRAPH 2001*, pages 409–416, August 2001.

[Tau95a] Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. 5th Intl. Conf. on Computer Vision (ICCV'95)*, pages 902–907, June 1995.

[Tau95b] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358, August 1995.

[Ter88] Demetri Terzopoulos. The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4), July 1988.

[TLHD03] Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, 2003.

[TM02] B. Thibert and J.M. Morvan. Approximations of a smooth surface with a triangulated mesh. *Preprint*, 2002.

[TOC98] Joseph R. Tristano, Steven J. Owen, and Scott A. Canann. Advancing front surface mesh generation in parametric space using a riemannian surface definition. In *Proceedings of 7th International Meshing Roundtable, Sandia National Labs*, pages 429–445, oct 1998.

[TT99] Jack Tumblin and Greg Turk. Lcis: A boundary hierarchy for detail-preserving contrast reduction. In *SIGGRAPH 98 Conference Proceedings*, pages 83–90, 1999.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH*, pages 55–64, 1992.

[Tut63] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, (13):743–768, 1963.

[TW98] Grit Thürmer and Charles Wüthrich. Computing vertex normals from polygonal facets. *Journal of Graphics Tools*, 3(1):43–46, 1998.

[Uli88] Robert A. Ulichney. Dithering with blue noise. In *Proceedings of the IEEE*, volume 76(1), pages 56–79, 1988.

[vDA95] R.M.J. van Damme and L. Aboul. Tight triangulations. *Mathematical Methods for Curves and Surfaces*, 1995.

[VRKS01] Jens Vorsatz, Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Feature sensitive remeshing. In *Eurographics Proceedings*, pages 393–401, sep 2001.

[WW94] Willian Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings*, pages 247–256, July 1994.