Signal Processing on Textured Meshes

by

Fabian Andres Prada Niño

A dissertation submitted to Johns Hopkins University
in conformity with the requirements for the degree of
Doctor of Philosophy

Baltimore, Maryland

October, 2018

Abstract

In this thesis we extend signal processing techniques originally formulated in the context of image processing to techniques that can be applied to signals on arbitrary triangles meshes.

We develop methods for the two most common representations of signals on triangle meshes: signals sampled at the vertices of a finely tessellated mesh, and signals mapped to a coarsely tessellated mesh through texture maps.

Our first contribution is the combination of Lagrangian Integration and the Finite Elements Method in the formulation of two signal processing tasks: Shock Filters for texture and geometry sharpening, and Optical Flow for texture registration.

Our second contribution is the formulation of Gradient-Domain processing within the texture atlas. We define a function space that handles chart discontinuities, and linear operators that capture the metric distortion introduced by the parameterization.

Our third contribution is the construction of a spatiotemporal atlas parameterization for evolving meshes. Our method introduces localized remeshing operations and a compact parameterization that improves geometry and texture video compression. We show temporally coherent signal processing using partial correspondences.

Primary reader and advisor: Michael Kazhdan.
Secondary reader: Hugues Hoppe.
Auxiliary reader: Laurent Younes.

Acknowledgements

I feel privileged to have received mentoring from so many outstanding people in my life. They have enlighten my path with knowledge and experience, but more importantly, by showing me the passion and discipline that takes to conquer great goals.

I want to start thanking my advisor, **Michael Kazhdan**, for being the headlight of this journey. Misha is not just an incredible researcher but also an amazing human being. The research and personal skills I developed on my PhD are the result of his careful orientation.

Thanks to **Hugues Hoppe** for the opportunity of interring at Microsoft Research. His broad technical and practical expertise were fundamental for the successful completion of our projects. Thanks Hugues for showing me a practical approach to problem solving.

Thanks to **Ming Chuang** and **Alvaro Collet** for being my mentors and collaborators. Thanks for your creative ideas and constructive discussions on our projects. Thanks for opening so many doors in my career.

Thanks to the capture team at Microsoft for providing the amazing data I used on my research projects. Thanks to **Spencer Reynolds, Steve Sullivan, Pat Sweeney** and **Dennis Evseev** for developing such great technology.

Thanks to all the people that helped tracing the path leading to Hopkins. Thanks to **Diego Nehab** who was my advisor at IMPA and who introduced me to Misha. Thanks to **Luiz Velho** and **Luiz Figuredo**, to **IMPA** and to the government of **Brazil** for all the support provided at the start of my career in Computer Graphics.

Thanks to **Camilo Rivera** and **Jose Samper** for fruitful career advise. Thanks to **Maria Losada** and to the **Colombian Math Olympiad** for developing my interests and skills on problem solving since I was in elementary school.

Thanks to all my school and college **teachers** for their commitment on creating a better society. Thanks to my **friends** for all the good energy and support. Thanks to my **family** for their unconditional love.

Finally, thanks to the **Johns Hopkins University** and the **Department of Computer Science** for make me feel home.

To Luis Ernesto, Martha Eugenia and Luis Carlos. For all their love!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rapid integration of depth, motion, and georeference sensors on our conventional cameras is changing the nature of images and videos. These new devices allow us to record not just the light but also the geometric properties of the space being captured. This additional geometric data has opened a door to applications like image-based navigation systems, augmented reality, and improved facial recognition.

From the signal processing point view we can raise several questions regarding the representation and analysis of these multimodal signals: What kind of data structures should be used to fuse geometric and photometric measures? How should we process geometric and photometric data to generate new meaningful signals? While these questions have inspired a significant amount of research over the last decades, we believe that the increasing accessibility to devices that allow the creation and manipulation of this kind of data revives the discussion and encourages the development of new techniques like the ones described here.

1.2 Objective

The theory, methods, and algorithms introduced in this thesis are under the scope of the following question:

How should a signal defined on a parameterized surface be processed to produce a new signal in a manner consistent with the surface geometry?

To address this question, we start by looking back to traditional image processing tasks. In particular we choose three problems that have been well studied within the image processing community: Shock Filters, Optical Flow and Gradient-Domain Processing. For each of these problems we analyze the original formulation and proposed solutions in the context of image processing. We show how to formulate solutions to analogous problems for signals defined on arbitrary 3D surfaces.

1.3 Overview

This thesis is divided into five chapters. Chapter 2 introduces the mathematical tools that allow us to analyze signals on parameterized surfaces. We also introduce the data structures and notation used in subsequent chapters.

Chapter 3 proposes a Lagrangian formulation of one of the pioneering works in image sharpening. The original Shock Filters method [1] evolves a signal according to a PDE that preserves the critical points and accentuates concavity. We simplify this PDE in a way that admits a simple Lagrangian integration, without compromising the quality of the obtained solution. Our Lagrangian method works for signals defined on traditional images and extends to signals defined on triangles meshes. We show applications of our algorithms for sharpening colors and geometry. We highlight its simple implementation, efficiency and stability.

In Chapter 4 we study the classical formulation of the optical flow problem: computing a vector field that aligns a pair of images. Our solution to the optical flow problem for meshes builds on top of standard image-based optical flow [2]. Of particular interest in this chapter is the study of vector field regularization operators. We show applications of our mesh-based optical flow for signal interpolation and photometric tracking.

In Chapter 5 we discretize and solve the screened-Poisson equation directly in the texture atlas domain. We propose a method that pulls back the immersion metric to the parametric domain and produces results that are perceptually continuous across charts. The partial regularity of the texture atlas parameterization motivates the development of hybrid multigrid solvers for efficient solution to the screened-Poisson equation. We show applications of our technique to signal smoothing, sharpening and stitching. We demonstrate the robustness of our formulation by solving more challenging geometric problems like geodesic computation and line integral convolution.

In Chapter 6 we introduce the first texture atlas parameterization for meshes whose geometry, topology and surface attributes change over time. An evolving mesh is obtained

by applying local remeshing operations between successive frames to ensure precise reproduction of the captured geometry and maximize temporal coherence within the frame sequence. We extend the traditional notion of *atlas*, *chart*, and *texture maps*, from static to evolving meshes, and show how the new representation improves signal compression. We present preliminary results of signal processing within this spatiotemporal parametric domain, and motivate the exploration of further applications.

The work described in this thesis was introduced in the following publications:

- F. Prada, and M. Kazhdan. *Unconditionally Stable Shock Filters for Image and Geometry Processing.* SGP 2015.
- F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe. *Motion Graphs for Unstructured Textured Meshes.* SIGGRAPH 2016.
- F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe. *Spatiotemporal Atlas Parameterization for Evolving Meshes.* SIGGRAPH 2017.
- F. Prada, M. Kazhdan, M. Chuang, and H. Hoppe. *Gradient-Domain Processing within a Texture Atlas.* SIGGRAPH 2018.

Chapter 2

Preliminaries

2.1 Surfaces and signals in Computer Graphics

2.1.1 Surface discretization

In this thesis we will use triangle meshes to model surfaces. Triangle meshes are arguably the simplest and most versatile structures to represent 3D surfaces. We can use triangle meshes to *capture* complex models using very few triangles. For instance, the output of a 3D scanner is usually an unstructured collection of millions of points sampled from the model surface. By processing this point cloud, we can generate an adaptive mesh that covers regions of low curvature with very few triangles and use finer sampling in regions of geometric detail. This mesh provides an structured representation of our model that is both compact and accurate, and consequently it can be rapidly edited, stored, or analyzed.

Alternatively, we can use triangle meshes to *generate* complex models using very few triangles. From a small collection of geometric primitives and a set of subdivision rules, an artist can design a smooth surface.

Due to its versatility and easy manipulation, triangles meshes are ubiquitous in computer graphics applications: the simple modeling of extrinsic deformation makes them suitable for animation; the simple definition of ray intersection, spatial sorting and clipping operations makes them convenient for rasterization; the simple definition of discrete differential operators makes them appropriate for Finite Elements simulation.

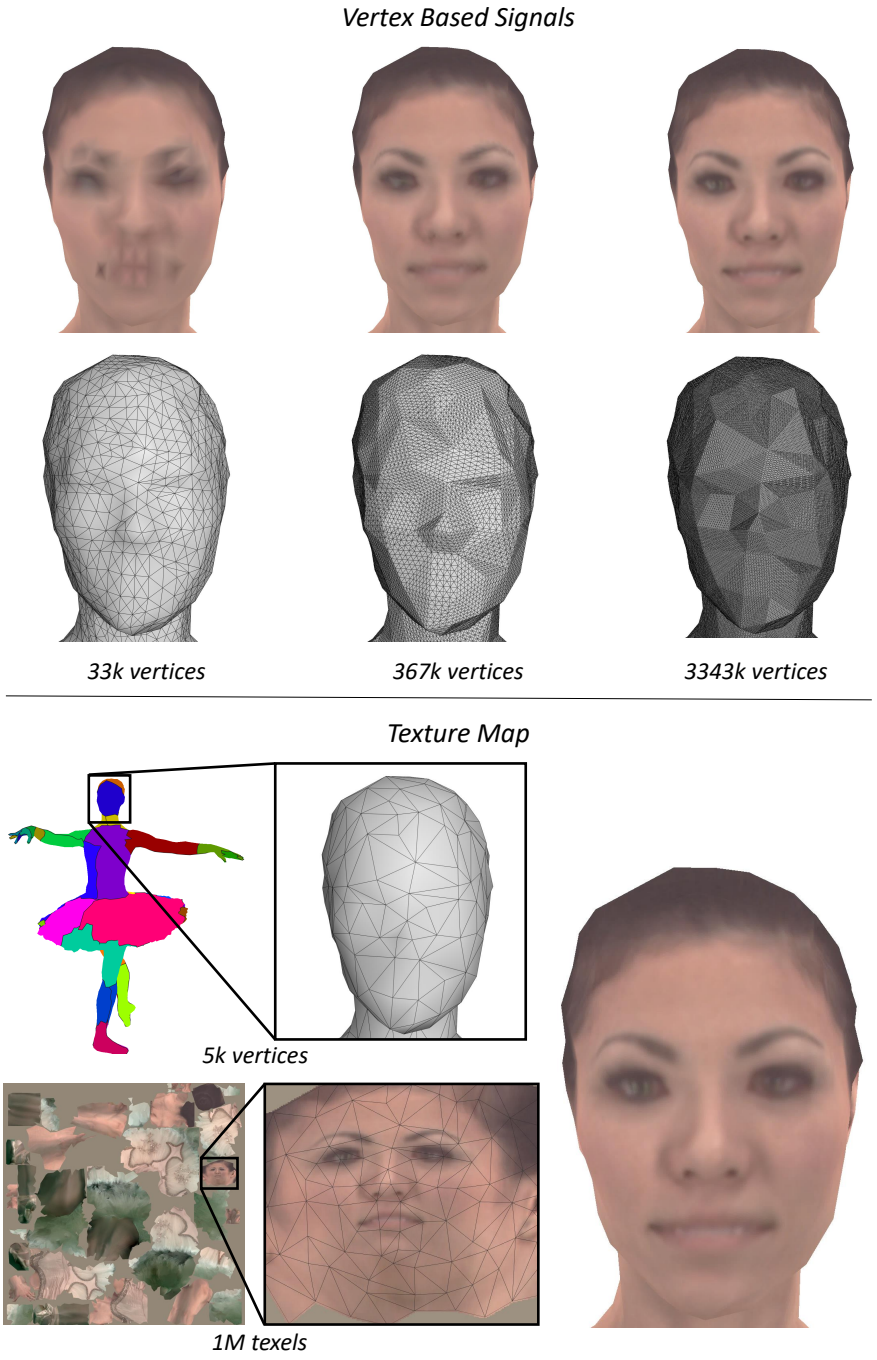


FIGURE 2.1: Signal representation on a triangle mesh.

2.1.2 Signal discretization

A signal is a function that assigns a property to each point in the surface. Signals are traditionally used on triangle meshes to represent colors, normals, displacements as well as other material properties.

In Figure 2.1 we show the two most common ways to represent signals on meshes. The

simplest and most intuitive way is using *vertex based signals* (top). Such signals are specified by values at the vertices, and are reconstructed in the triangle interior through barycentric interpolation. Since the resolution of the signal is defined by the density of vertices, we observe that reproducing signal detail requires a very fine tessellation. This is the case for facial features in the *Ballerina* model: a uniform triangulation with $33k$ vertices is insufficient to capture the details in the mouth and eyes, and those can only be captured using a finer triangulation.

At the bottom of Figure 2.1 we show an alternative approach to represent signals on surfaces which is called *texture mapping*. The idea of texture mapping is to specify signal values through a conventional image, and define an auxiliary function that tells each point in the surface from which location of the image to take its value from. The definition of this surface-to-image function involves the construction of an *atlas parameterization* [3]. Texture mapping can be used to define high resolution signals on very coarse meshes. For the *Ballerina* example, using texture mapping on a mesh with only $5k$ vertices and a texture with $1M$ texels, we obtain a result that has the same signal quality as a vertex based signal sampled on a mesh with above $1M$ vertices.

Texture mapping has significant storage and performance benefits over vertex-based signals. By moving the signal detail from an irregularly tessellated mesh to a regular grid, the gain in spatial coherence allows for better compression and efficient memory access.

In this thesis we will study signals that use a vertex-based representation or a texture map. It is worth mentioning that there are alternative ways to represent signals on meshes. Most notable are the extrinsic representations using 3D grids [4]. The advantage of the extrinsic representations is independence between signal and tessellation: the same signal can be sampled on any triangulation of the surface. The drawback of this representation is the requirement of excessive refinement of 3D space to reproduce signal detail, e.g. in regions of high curvature .

2.2 Notation

In this thesis we will use non-bold letters to represent entities in continuous domains like points (p), vector fields (X) and flows (Φ). We use bold letters to represent discrete entities like vectors ($\mathbf{b}$) and matrices ($\mathbf{M}$). In particular, the bold version of a non-bold symbol represent the array of coefficients with respect to a given basis. For instance if ϕ is a function then $\boldsymbol{\phi}$ correspond to its array of coefficients.

Symbol	Summary description
$S \subset \mathbb{R}^3$	surface
$M \subset \mathbb{R}^2$	parametric domain
$M_i \subset M$	chart
$p, q \in M$	points
$\phi, \psi \ : \ M \to \mathbb{R}$	functions
$X, Y \ : \ M \to TM$	vector fields
$\pi \ : \ M \to \mathbb{R}^3$	immersion map
$\Psi \ : \ M \to M$	diffeomorphism
$\Phi \ : \ M \times \mathbb{R} \to M$	flow
$\mu \ : \ TM \times TM \to \mathbb{R}^{\geq 0}$	Euclidean metric
$g \ : \ TM \times TM \to \mathbb{R}^{\geq 0}$	Riemannian metric
$\mathscr{B} \subset \mathscr{L}(M, \mathbb{R})$	function basis
$\phi_i \in \mathscr{B}$	basis function
$\mathbf{M} \in \mathbb{R}^{\lvert\mathscr{B}\rvert \times \lvert\mathscr{B}\rvert}$	mass matrix
$\mathbf{S} \in \mathbb{R}^{\lvert\mathscr{B}\rvert \times \lvert\mathscr{B}\rvert}$	stiffness matrix
$\mathscr{B}_1 \subset \mathscr{L}(M, TM)$	vector field basis
$X_i \in \mathscr{B}_1$	basis vector field
$\mathbf{M}_1 \in \mathbb{R}^{\lvert\mathscr{B}_1\rvert \times \lvert\mathscr{B}_1\rvert}$	vector field mass matrix
$\mathbf{S}_1 \in \mathbb{R}^{\lvert\mathscr{B}_1\rvert \times \lvert\mathscr{B}_1\rvert}$	vector field stiffness matrix

TABLE 2.1: Summary of notation.

Table 2.1 summarizes the symbols for the mathematical concepts introduced in this chapter and studied in more detail in subsequent chapters.

2.3 Mathematical background

2.3.1 Mathematics for a continuous world

2.3.1.1 Surfaces

We think of a *surface* as the interface between the interior and exterior of an object. To introduce a formal definition of surface we need to take a look to a broader class of mathematical objects known as *manifolds* [5]. An n-dimensional manifold is a set that locally resembles the euclidean space. More precisely, for any point $\boldsymbol{x}$ in the manifold, we can find a parametric map $\pi : M \subset \mathbb{R}^n \to S$ that establish a continuous and bijective association between a region in the euclidean space and a neighbourhood of the point. From this definition it follows that a 3D surface, $S \subset \mathbb{R}^3$, belongs to the class of 2-dimensional manifolds.

Parameterization The first step to compute distances, areas, and properties of functions over a surface is to construct a *parameterization*. The building blocks of a surface parameterization are the charts. A *chart* defines a continuous map, $\pi_i : M_i \to S_i$ from a region of the plane, $M_i \subset \mathbb{R}^2$, into a region of the surface, $S_i \subset S$. We say that a collection of charts $\mathcal{A} = \{(\pi_i, M_i, S_i)\}_i$ form an *atlas* when they provide a full covering of the surface, i.e., $\cup_i S_i = S$.

Tangent spaces Surfaces for which we can define a tangent plane TS that changes smoothly as we move along the surface are called *regular surfaces* [6].

Since the tangent plane at any point of a regular surface is a 2-dimensional vector space, we augment our parametric domain M with a tangent space TM, which is simply a copy of $\mathbb{R}^2$. The differential of the parametric map $d\pi \equiv [\frac{\partial \pi}{\partial u} \frac{\partial \pi}{\partial v}] : TM \to TS$ provides an identification between these vector spaces. Thus, for any point $p \in M$, the column vectors of the parametric map differential, $\{\frac{\partial \pi}{\partial u}|_p, \frac{\partial \pi}{\partial v}|_p\}$, form a basis for the tangent plane $T_{\pi(p)}S$.

Diffeomorphisms Given two parameterizations of the same surface patch, (π, M, S) and $(\tilde{\pi}, \tilde{M}, S)$, the map $\Psi := \tilde{\pi}^{-1} \circ \pi : M \to \tilde{M}$ is a differmorphism. This map provide an identification of points in the two parametric domains that match to a same point in the surface: by construction, any point $p \in M$ and its image $\tilde{p} = \Psi(p) \in \tilde{M}$ satisfy $\tilde{\pi}(\tilde{p}) = \tilde{\pi} \circ (\tilde{\pi}^{-1} \circ \pi)(p) = \pi(p)$.

Furthermore, the differential $d\Psi : TM \to T\tilde{M}$ provide an identification of vectors in T_pM and $T_{\tilde{p}}\tilde{M}$ that are matched to the same tangent direction in the surface tangent plane: for any vector $X_p \in T_pM$ and its image $\tilde{X}_{\tilde{p}} = d\Psi_p X_p \in T_{\tilde{p}}\tilde{M}$, we have $d\tilde{\pi}_{\tilde{p}}\tilde{X}_{\tilde{p}} = (d\tilde{\pi}_{\tilde{p}} \circ d\Psi_p)X_p = d\pi_p X_p \in T_{\pi(p)}S$.

2.3.1.2 Riemannian manifolds

To carry out computations of angles, lengths, and areas on the domain $M \subset \mathbb{R}^2$, we introduce an inner product on its tangent space. The inner product $g : TM \times TM \to \mathbb{R}^{\geq 0}$ is referred as the *metric*. The domain M augmented with the metric g belongs to a class of objects known as Riemannian manifolds.

Immersion metric When the domain M is associated to a surface patch $S \subset \mathbb{R}^3$ through the parametric map $\pi : M \to S$, we can define the *immersion metric*, that

matches the euclidean dot product of tangent directions to the surface:

$$\langle X, Y \rangle_\mu := (d\pi X)^\top (d\pi Y) = X^\top (d\pi^\top d\pi) Y \tag{2.1}$$

The immersion metric allow us to measure geometric properties of the surface on the parameter domain.

Lengths On the Riemannian manifold (M, g), the length of a curve $\gamma : I \subset \mathbb{R} \to M$ is defined as

$$\ell_{(M,g)}(\gamma) := \int_I \sqrt{\langle \gamma'(t), \gamma'(t) \rangle_g} dt \tag{2.2}$$

When $\pi : M \to S$ is a parameterization and $\mu = d\pi^\top d\pi$ is the immersion metric this definition matches the length of the curve $\pi \circ \gamma$ on the surface.

Areas On the Riemannian manifold (M, g), the area of a region $\Gamma \subset M$ is defined by,

$$A_{(M,g)}(\Gamma) := \int_\Gamma \sqrt{|g|} dA \tag{2.3}$$

On a parameterized surface the immersion metric satisfies $|\mu| = |\frac{\partial \pi}{\partial u} \times \frac{\partial \pi}{\partial v}|$, and we can verify this definition also matches the area of the patch $\pi(\Gamma)$ on the surface.

Isometries The fact that we can measure lengths of curves in a Riemannian manifold (M, g) allow us to define a notion of distance: the distance between two points, $d_{(M,g)}(p_0, p_1)$, is the minimal length of any curve in M passing through p_0 and p_1.

We say that two Riemannian spaces (M, g) and $(\tilde{M}, \tilde{g})$ are *isometric* when there is a map $\Psi : M \to \tilde{M}$ that preserve distances:

$$d_{(M,g)}(p_0, p_1) = d_{(\tilde{M},\tilde{g})}(\Psi(p_0), \Psi(p_1))$$

As expected, given a pair of parameterizations $\pi : M \to S$ and $\tilde{\pi} : \tilde{M} \to S$, the diffeomorphism $\Psi = \tilde{\pi}^{-1} \circ \pi$ is an isometry between the Riemannian manifolds M and $\tilde{M}$ with immersion metrics $\mu = d\pi^\top d\pi$ and $\tilde{\mu} = d\tilde{\pi}^\top d\tilde{\pi}$ respectively. Furthermore, we have the following identification of the immersion metrics:

$$\langle d\Psi_p X_p, d\Psi_p Y_p \rangle_{\tilde{\mu}_{\Psi(p)}} = \langle X_p, Y_p \rangle_{\mu_p} \tag{2.4}$$

2.3.1.3 Calculus

Gradient The gradient of a function $\phi : M \to \mathbb{R}$ on the Riemannian manifold (M, g) is a vector field $\nabla_g \phi : M \to TM$ that indicates the direction of fastest change of the function. The gradient can be explicitly computed as,

$$\nabla_g \phi = g^{-1} \begin{bmatrix} \frac{\partial \phi}{\partial u} \\ \frac{\partial \phi}{\partial v} \end{bmatrix}$$

For any curve $\gamma : I = [a, b] \subset \mathbb{R} \to M$ and $\phi : M \to \mathbb{R}$, the gradient satisfies,

$$\int_I \langle \gamma'(t), \nabla_g \phi(\gamma(t)) \rangle_g dt = \phi(\gamma(b)) - \phi(\gamma(a))$$

Curl The *curl* of a vector field X measures the limit ratio of rotation per unit area. Denote by Ω a neighbourhood of p, and $\gamma : I \to \partial\Omega$ a positively-oriented parameterization of its boundary. We define the curl by the limit:

$$(\nabla_g \times X)(p) := \lim_{\Omega \to p} \frac{\int_I \langle \gamma'(t), X(\gamma(t)) \rangle_g dt}{\int_\Omega \sqrt{|g|} dA} \tag{2.5}$$

For a vector field $X = (X^u(u, v), X^v(u, v))$ the curl can be expressed in the local coordinate basis as:

$$\nabla_g \times X = \frac{1}{\sqrt{|g|}} \Big(\frac{\partial}{\partial u} (gX)^v - \frac{\partial}{\partial v} (gX)^u \Big)$$

Divergence The *divergence* of a vector field X measures the limit ratio of outward flux per unit area:

$$(\nabla_g \cdot X)(p) := \lim_{\Omega \to p} \frac{\int_I \langle \gamma'(t), J_g X(\gamma(t)) \rangle_g dt}{\int_\Omega \sqrt{|g|} dA} \tag{2.6}$$

Here $J_g : TM \to TM$ denotes the orthogonal rotation operation on the tangent space of a 2-dimensional Riemannian manifold. This operator can be explicitly computed as,

$$J_g = \frac{Jg}{\sqrt{|g|}},$$

where $J : TM^* \to TM$ represents the 90 degree rotation, $J(u, v) = (-v, u)$.

In local coordinates the divergence of a vector field is given by the expression:

$$\nabla_g \cdot X = \frac{1}{\sqrt{|g|}}\Big(\frac{\partial}{\partial u}(\sqrt{|g|}X^u) + \frac{\partial}{\partial v}(\sqrt{|g|}X^v)\Big)$$

Laplacian The Laplacian of a function ϕ is defined as the divergence of the functions gradient:

$$\Delta_g \phi := \nabla_g \cdot (\nabla_g \phi)$$

From the divergence product rule,

$$\nabla_g \cdot (\phi X) = \langle \nabla_g \phi, X \rangle_g + \phi(\nabla_g \cdot X),$$

and the Divergence Theorem,

$$\int_\Omega \nabla_g \cdot X \sqrt{|g|} dA = \int_{\partial\Omega} \langle \gamma'(t), J_g X(\gamma(t)) \rangle_g dt,$$

we deduce (by replacing $X = \nabla_g \phi$) that under natural or free boundary conditions, the Laplace operator satisfies:

$$\int_\Omega \langle \nabla_g \phi, \nabla_g \phi \rangle_g \sqrt{|g|} dA = -\int_\Omega \phi \Delta_g \phi \sqrt{|g|} dA$$

This last equation suggests that the Laplacian can be interpreted as a symmetric negative semi-definite operator that measures the smoothness of a function.

2.3.1.4 Finite Elements

Given a Riemannian manifold (M, g) we construct approximations to the space of functions and vector fields using a finite dimensional basis.

Basis A *basis*, $\mathscr{B} = \{\phi_i\}_{1 \le i \le n}$, is a set of linearly independent functions that span a subspace of functions we use to represent signals. By definition, any signal within

this subspace can be expressed as $\phi = \sum_i a_i \phi_i$, where the a_i are called the *coefficients* of the signal in the basis $\mathscr{B}$. We evaluate the signal at any point p on the surface by aggregating the scaled values of the basis function, i.e., $\phi(p) = \sum_i a_i \phi_i(p)$. In general, the basis functions are piece-wise polynomials with compact support, so the evaluation of a signal at any point on the surface only requires the evaluation of a few basis functions.

We say that the basis is *interpolatory* at a set of points $\{p_i\}_i \subset S$ when $\phi_i(p_j) = \delta_{ij}$. We say that the basis forms a *partition of unity* if $\sum_i \phi_i(p) = 1$ for all $p \in S$.

We denote by $\mathscr{B}_1 = \{X_i\}_i$ a basis for the subspace of vector fields. From a functional basis $\mathscr{B}$ we can generate two different basis of vector fields: the Gradient basis($\mathscr{B}_1^C$) and the Whitney basis($\mathscr{B}_1^W$). The Gradient basis is composed by gradients and rotated gradients of the basis functions:

$$\mathscr{B}_1^C = \{\nabla_g \phi_i\}_i \cup \{J_g \nabla_g \phi_i\}_i \quad (1 \leq i \leq n) \tag{2.7}$$

The Whitney basis corresponds to the anti-symmetric difference of the product between pairs of basis functions and their gradients:

$$\mathscr{B}_1^W = \{X_{ij}\}_{ij}, \text{ where } X_{ij} = \phi_i \nabla_g \phi_j - \phi_j \nabla_g \phi_i \quad (1 \leq i < j \leq n) \tag{2.8}$$

Operators The use of a basis allow us to evaluate inner products of functions and vector fields just in terms of their coefficients and discrete operators.

Given $\phi = \sum_i a_i \phi_i$ and $\psi = \sum_i b_i \phi_i$, we compute their inner product as,

$$\int \phi \psi \sqrt{|g|} dA = \sum_i \sum_j a_i b_i \int \phi_i \phi_j \sqrt{|g|} dA = \boldsymbol{\phi}^\top \mathbf{M} \boldsymbol{\psi} \tag{2.9}$$

where the matrix $\mathbf{M}_{ij} = \int \phi_i \phi_j \sqrt{|g|} dA$ is the *mass matrix*. Similarly, we can compute the inner product of the gradients of ϕ and ψ as,

$$\int \langle \nabla_g \phi, \nabla_g \psi \rangle_g \sqrt{|g|} dA = \sum_i \sum_j a_i b_i \int \langle \nabla_g \phi_i, \nabla_g \phi_j \rangle_g \sqrt{|g|} dA = \boldsymbol{\phi}^\top \mathbf{S} \boldsymbol{\psi} \tag{2.10}$$

where the matrix $\mathbf{S}_{ij} = \int \langle \nabla_g \phi_i, \nabla_g \phi_j \rangle_g \sqrt{|g|} dA$ is the *stiffness matrix*.

In the case of a pair of vector fields $X = \sum a_i X_i$ and $Y = \sum_i b_i X_i$, the inner product is given by,

$$\int \langle X, Y \rangle_g \sqrt{|g|} dA = \sum_i \sum_j a_i b_i \int \langle X_i, X_j \rangle_g \sqrt{|g|} dA = \boldsymbol{X}^\top \mathbf{M}_1 \boldsymbol{Y} \tag{2.11}$$

where the matrix $(\mathbf{M}_1)_{ij} = \int \langle X_i, X_j \rangle_g \sqrt{|g|} dA$ is the *vector field mass matrix*.

2.3.2 Mathematics for a discrete world

2.3.2.1 Simplicial complexes

From the combinatorial point of view a triangle mesh is an instance of a class of objects known as *homogeneous simplicial 2-complexes* [7]. This means that a triangle mesh can be represented by a hierarchical graph of *vertices* (V), *edges* (E) and *triangles* (T), where the edges correspond to a subset of vertex pairs $E \subset \binom{V}{2}$, and the triangles correspond to a subset of vertex triplets $T \subset \binom{V}{3}$. The *simplicial 2-complex* property requires that the intersection of any two triangles is a common edge, a vertex or the empty set, and the intersection of any two edges is either a vertex or the empty set. Additionally, it is called *homogeneous* since any edge (resp. vertex) strictly belongs to the boundary of at least one triangle (resp. edge). In Figure 2.2, the first structure (from left to right) is not simplicial since the intersection of the two triangles is not a common edge or a vertex, and the second structure is not homogeneous since it has an edge that does belong to any triangle.

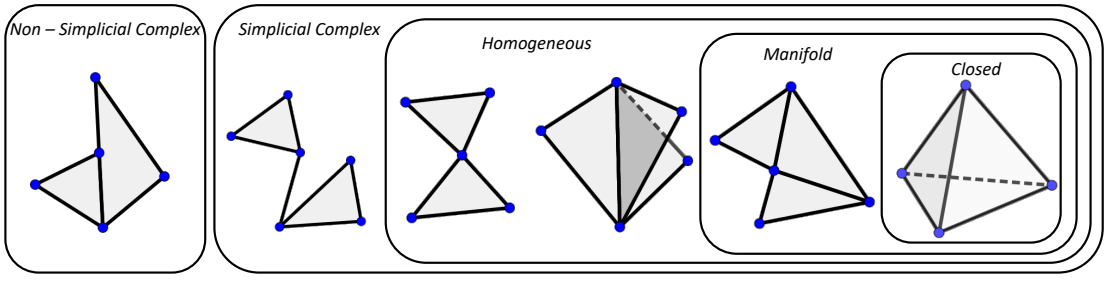


FIGURE 2.2: Classification of simplicial 2-complexes

When we associate a 3D position to each vertex of the simplicial 2-complex, we can reconstruct a surface in $\mathbb{R}^3$ by taking convex combinations of the vertices within each triplet. This piece-wise flat surface is what we call a *triangle mesh*. As we show in Figure 2.2, a triangle mesh is a 2-dimensional *manifold* as long as the set of edge-adjacent triangles around a vertex form a single connected component, and every edge is shared by at most two triangles. Furthermore, we say that a triangle mesh is *closed* when each edge is shared by exactly two triangles.

Though triangle meshes are not regular surfaces (a unique tangent plane cannot be defined at edges or vertices), we can extend differential concepts like curvature to triangle meshes, and preserve important properties like the Gauss-Bonnet Theorem. Please refer to [8] for an introduction to discrete differential geometry.

2.3.2.2 Triangle mesh parameterization

In this thesis we will use two different kinds of parameterizations of triangle meshes: the canonical parameterization and the texture parameterization.

Canonical parameterization The *canonical parameterization*, $\mathscr{A} = \{(\pi_t, \Pi, S_t)\}_t$, is an atlas that associates each triangle to a single chart. The map π_t is the unique affine transformation from the unit triangle $\Pi = \{(u, v) : u, v \geq 0, x + y \leq 1\}$, to the 3D triangle S_t spanned by vertices $\boldsymbol{x}_t^0$, $\boldsymbol{x}_t^1$, and $\boldsymbol{x}_t^2$:

$$\pi_t(u, v) = \boldsymbol{x}_t^0 + (\boldsymbol{x}_t^1 - \boldsymbol{x}_t^0 | \boldsymbol{x}_t^2 - \boldsymbol{x}_t^0) \begin{pmatrix} u \\ v \end{pmatrix}$$

The differential of the immersion map is constant within each triangle and is given by,

$$d\pi_t = (\boldsymbol{x}_t^1 - \boldsymbol{x}_t^0 | \boldsymbol{x}_t^2 - \boldsymbol{x}_t^0), \tag{2.12}$$

The immersion metric tensor corresponds to $\mu_t = d\pi_t^\top d\pi_t$.

Texture parameterization The *texture parameterization*, $\mathscr{A} = \{(\pi_i, M_i, S_i)\}_i$, is an atlas that maps disjoint polygonal patches in the unit square $[0, 1]^2$ into polygonal patches on the triangle mesh. Each π_i is a piecewise affine map on a triangulation of $M_i \subset \mathbb{R}^2$, that maps triangles from the Euclidean plane to triangles on the surface. By definition each map π_i is continuous, but is only guaranteed to be differentiable in the interior of the triangles that form M_i.

Let $M_t \subset M_i$ be the texture triangle spanned by 2D coordinates $\boldsymbol{p}_t^0$, $\boldsymbol{p}_t^1$, and $\boldsymbol{p}_t^2$, that parameterizes the 3D triangle S_t spanned by respective vertices $\boldsymbol{x}_t^0$, $\boldsymbol{x}_t^1$, and $\boldsymbol{x}_t^2$. The restriction of the immersion map on triangle t corresponds to,

$$\pi_i|_t(u, v) = \boldsymbol{x}_t^0 + (\boldsymbol{x}_t^1 - \boldsymbol{x}_t^0 | \boldsymbol{x}_t^2 - \boldsymbol{x}_t^0)(\boldsymbol{p}_t^1 - \boldsymbol{p}_t^0 | \boldsymbol{p}_t^2 - \boldsymbol{p}_t^0)^{-1} \left(\begin{pmatrix} u \\ v \end{pmatrix} - \boldsymbol{p}_t^0 \right)$$

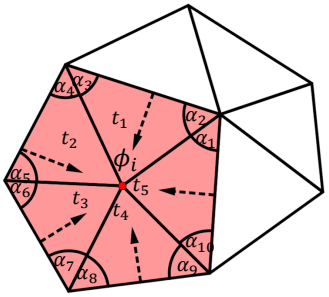
and its differential is given by,

$$d\pi_i|_t = (\boldsymbol{x}_t^1 - \boldsymbol{x}_t^0 | \boldsymbol{x}_t^2 - \boldsymbol{x}_t^0)(\boldsymbol{p}_t^1 - \boldsymbol{p}_t^0 | \boldsymbol{p}_t^2 - \boldsymbol{p}_t^0)^{-1} \tag{2.13}$$

The piece-wise constant immersion metric corresponds to $\mu|_{M_t} = d\pi_i|_t^\top d\pi_i|_t$.

2.3.2.3 Triangle Finite Elements

Functional basis Our function space on triangle meshes is spanned by the so called *hat* basis. As shown in the inset, each hat function ϕ_i is centered at a vertex of the mesh and has support on its adjacent triangles. The hat functions are linear within each triangle, interpolatory, and form a partition of unity. As a consequence, the only supported basis functions within a given triangle are the hat functions at its corners, and the evaluation of the basis at a point $p \in t_{ijk}$ give us the triplet $(\phi_i(p), \phi_j(p), \phi_k(p))$ of barycentric coordinates. Also shown in the inset is the gradient of a hat basis. The gradient is constant on each incident triangle, and within a triangle is orthogonal to the edge opposite to the vertex.



Since the basis functions are linear within each triangle, the computation of the mass and stiffness matrix coefficients are usually performed on a per-triangle basis using the canonical parameterization. To compute these coefficients we integrate and aggregate the product of functions and gradients over commonly supported triangles. These coefficients can be directly expressed in terms of triangles areas and angles as follow:

$$\mathbf{M}_{ij} = \begin{cases} \sum_{t_k \in N_i} \frac{|t_k|}{6} & \text{if } i = j \\ \sum_{t_k \in N_i \cap N_j} \frac{|t_k|}{12} & \text{otherwise.} \end{cases}, \quad \mathbf{S}_{ij} = \begin{cases} \sum_{\alpha_k \in \partial N_i} \frac{\cot \alpha_k}{2} & \text{if } i = j \\ \sum_{\alpha_k \in \partial N_i \cap \partial N_j} \frac{-\cot \alpha_k}{2} & \text{otherwise.} \end{cases}$$

where $|t|$ is the area of a triangle t and N_i (resp. ∂N_i) is the interior (resp. boundary) of the one-ring neighbourhood of vertex i. Additionally, α_k denotes an interior angle at a corner of ∂N_i.

Vector basis In this thesis we will consider three different representations of vector fields on triangle meshes: the *Triangle basis*, the *Gradient basis*[9], and the *Whitney basis*[10]. For a more extensive discussion of representation and processing of vector fields on triangle meshes please refer to de Goes et al. [11].

As shown in Figure 2.3 the Triangle basis ($\mathscr{B}_1^T$) specifies a direction in the tangent space of each triangle and generates a piece-wise constant vector field. This provide a very simple yet powerful representation that we explore in Chapters 3 and 4.

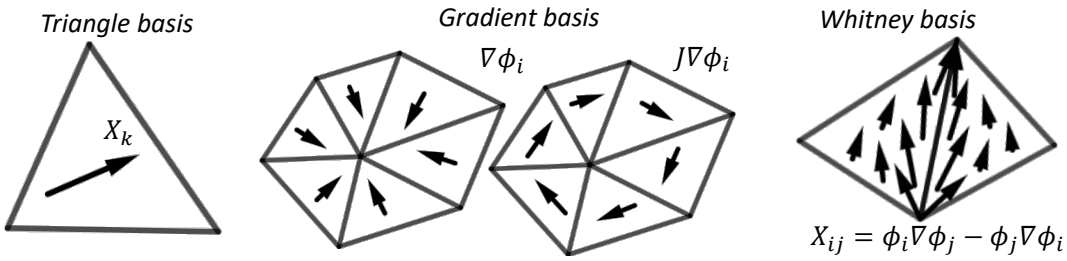


FIGURE 2.3: Discretization of vector fields on triangle meshes.

The Gradient basis ($\mathscr{B}_1^C$) is formed by gradients and rotated gradients of the hat functions. This vector space is a subspace of the Triangle basis and provides a decomposition between curl-free and divergence-free vector fields.

Finally, the Whitney basis ($\mathscr{B}_1^W$) is an edge-based representation where coefficients indicate the magnitude of the vector field in the direction parallel to the edge.

Vector representation and prolongation For the Triangle Basis, vector fields are represented using the coordinates of the per-triangle direction in the canonical parameterization $\pi_t : \Pi \to S_t$. The vector $X_t \in \Pi$ in parametric coordinates corresponds to the direction $d\pi_t X_t$ in the surface. We encode the entire vector field by concatenating the array of coordinates on each of the triangles : $\boldsymbol{X} = [X_1 X_2 \dots X_{|T|}] \in \mathbb{R}^{2|T|}$.

For the Gradient basis, vector fields are represented by the coefficients of the gradients and rotated gradients at each vertex. The vector field $X = \sum_i \kappa_i \nabla\phi_i + \sum_i \xi_i J\nabla\phi_i$, is represented by the concatenated array $\boldsymbol{X} = [\boldsymbol{\kappa}, \boldsymbol{\xi}] \in \mathbb{R}^{2|V|}$.

For the Whitney basis vectors field are represented by the coefficients at each edge. We use the notation ω_{ij} for the coefficient at edge $\vec{e}_{ij}$ to remind us that this coefficient can be interpreted as an integrated 1-form (i.e. the integral of the vector field along the edge). The entire vector field is encoded by an array $\boldsymbol{X} = \boldsymbol{\omega} = [\omega_{ij}] \in \mathbb{R}^{|E|}$.

For some applications we require an explicit representation of the vector fields. Thus, we introduce prolongation operators $\mathbf{P}^{C\to T}$ and $\mathbf{P}^{W\to T}$ mapping from the Gradient and Whitney basis to the Triangle Basis.

From the properties introduced in Section 2.3.1.3, and using the canonical parameterization on a triangle t_{ijk}, it follows that,

$$(\mathbf{P}^{C\to T}[\boldsymbol{\kappa\xi}])_{t_{ijk}} = g^{-1}_{t_{ijk}} \begin{pmatrix} \kappa_j - \kappa_i \\ \kappa_k - \kappa_i \end{pmatrix} + J_{g_{t_{ijk}}} g^{-1}_{t_{ijk}} \begin{pmatrix} \xi_j - \xi \\ \xi_k - \xi_i \end{pmatrix}. \tag{2.14}$$

For the Whitney basis, which is not constant per triangle, we take as representative direction its value at the triangle barycenter,

$$(\mathbf{P}^{C\to W}\boldsymbol{\omega})_{t_{ijk}} = \frac{1}{3} g_{t_{ijk}}^{-1} \begin{pmatrix} 2\omega_{ij} - \omega_{jk} - \omega_{ki} \\ \omega_{ij} + \omega_{jk} - 2\omega_{ki} \end{pmatrix} \tag{2.15}$$

Vector mass operator The mass operator for vector fields (Equation 2.11) in the Triangle basis, $M_1^T \in \mathbb{R}^{2|T|\times 2|T|}$, is given by the block diagonal operator:

$$\mathbf{M}_1^T := \begin{pmatrix} \frac{1}{2}\sqrt{|g_1|}g_1 & & & 0 \\ & \frac{1}{2}\sqrt{|g_2|}g_2 & & \\ & & \ddots & \\ 0 & & & \frac{1}{2}\sqrt{|g_{|T|}|}g_{|T|} \end{pmatrix} \tag{2.16}$$

The mass operators for the Gradient and Whitney basis are constructed by composing with the prolongation: $\mathbf{M}_1^C = (\mathbf{P}_1^{C\to T})^\top \mathbf{M}_1^T \mathbf{P}_1^{C\to T}$, and $\mathbf{M}_1^W = (\mathbf{P}_1^{W\to T})^\top \mathbf{M}_1^T \mathbf{P}_1^{W\to T}$.

2.3.2.4 Discrete Exterior Calculus

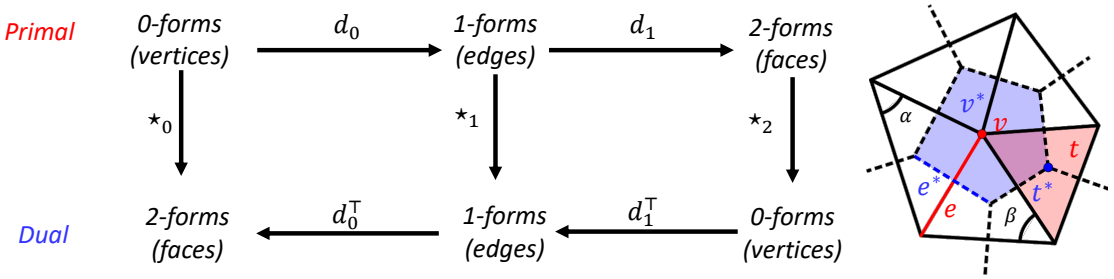


FIGURE 2.4: Mesh duality and discrete exterior calculus operators.

Discrete Exterior Calculus (DEC) [12] associates integrated differential forms to elements of the mesh and provides a discretization of the exterior derivative and Hodge star operators.

As shown in the diagram of Figure 2.4, 0-forms are sampled at vertices, 1-forms are sampled at (oriented) edges, and 2-forms are sampled at (oriented) triangles. DEC defines exterior derivative operators $\boldsymbol{d_0} : \mathbb{R}^{|V|} \to \mathbb{R}^{|E|}$ and $\boldsymbol{d_1} : \mathbb{R}^{|E|} \to \mathbb{R}^{|T|}$ mapping k-forms to $k+1$-forms as follows:

$$(\boldsymbol{d_0})_{e,v} = \begin{cases} 1 & \text{if } v \text{ is the source vertex of } e. \\ -1 & \text{if } v \text{ is the target vertex of } e. \\ 0 & \text{otherwise.} \end{cases}$$

$$(\boldsymbol{d_1})_{t,e} = \begin{cases} 1 & \text{if } e \text{ is in the boundary of } t \text{ and has same orientation.} \\ -1 & \text{if } e \text{ is in the boundary of } t \text{ and has opposite orientation.} \\ 0 & \text{otherwise.} \end{cases}$$

This definition satisfies a discrete version of the Stokes' Theorem. Formally, given a discrete 0-form $\boldsymbol{\omega_0} \in \mathbb{R}^{|V|}$ and a path $\gamma \subset E$, we can verify,

$$\sum_{v \in \partial\gamma} (\boldsymbol{\omega_0})_v = \sum_{e \in \gamma} (\boldsymbol{d_0}\boldsymbol{\omega_0})_e.$$

Similarly, given a discrete 1-form $\boldsymbol{\omega_1} \in \mathbb{R}^{|E|}$ and a region Ω, we obtain,

$$\sum_{e \in \partial\Omega} (\boldsymbol{\omega_1})_e = \sum_{t \in \Omega} (\boldsymbol{d_1}\boldsymbol{\omega_1})_t$$

DEC also introduces Hodge star operators that map primal k-forms to dual $2-k$-forms by scaling according to the ratio between between primal and dual elements of the triangulation. The Hodge stars $\boldsymbol{\star_0} \in \mathbb{R}^{|V|\times|V|}$, $\boldsymbol{\star_1} \in \mathbb{R}^{|E|\times|E|}$, and $\boldsymbol{\star_2} \in \mathbb{R}^{|T|\times|T|}$ are diagonal matrices satisfying:

$$(\boldsymbol{\star_0})_v = \frac{|v^*|}{|v|} = |v^*|, \qquad (\boldsymbol{\star_1})_e = \frac{|e^*|}{|e|} = \frac{\cot\alpha + \cot\beta}{2}, \qquad (\boldsymbol{\star_2})_t = \frac{|t^*|}{|t|} = \frac{1}{|t|}.$$

Chapter 3

Shock Filters

FIGURE 3.1: Sharpening of texture and geometry with Shock Filters.

In this chapter we propose a new Lagrangian formulation for Shock Filters and show its applications in sharpening signals defined over triangle meshes. As shown in Figure 3.1, when applied to color textures, our mesh-based Shock Filters produces a new signal with sharper edges and regions of constant color. When applied to geometry, we obtain enhanced contours and piece-wise flat surfaces.

3.1 Introduction

Introduced more than two decades ago, Shock Filters [1] formulates image processing as a PDE which evolves the image towards a steady-state solution which is piecewise smooth with sharp discontinuities (shocks) forming along edges. The PDE holds extrema fixed and evolves concave-up (resp. concave-down) regions towards their local minima (resp. maxima).

Shock Filters have been made more robust in recent works which have included an unconditionally stable implementation that uses an implicit time-integrator to solve the

PDE coupled with anisotropic diffusion [13], as well as regularized implementations that are more stable in the presence of signal noise [14].

In addition to Shock Filters, a variety of methods for edge-aware filtering have been proposed. Anisotropic diffusion [15, 16] smooths an image while constraining the diffusion not to cross edges. Bilateral filtering [17] replaces a pixel with the weighted average of its neighbors, adapting the weights so that more smoothing occurs between pixels on the "same side" of an edge. Laplacian sharpening [18] amplifies high-frequency content. And, L_0 gradient minimization [19] solves for the image which matches the input but has sparse gradients.

Though initially proposed for image-processing, many of these approaches have since been adapted to editing surface geometry, including anisotropic diffusion of geometry [20, 21] and normals [22], bilateral mesh denoising [23], and Laplacian/spectral sharpening [4, 24].

There has also been a significant body of work that leverages priors, learned either from the image itself, frames of a video, or a large database of images, to perform edge-aware processing [25–28].

Though unconditionally stable solutions for PDEs have been proposed in numerous image-processing applications, these are often obtained through the solution of a large linear system. In contrast, our approach only requires tracing values along flow-lines. Thus, much like Stam's Unconditionally Stable Fluids [29, 30], our method can use arbitrarily large time-steps and easily generalizes to meshes.

3.2 Osher-Rudin formulation

The original Shock Filters work [1] proposes a method to sharpen a signal while preserving critical points. The most distinctive characteristic of Shock Filters is its preservation of the range of the input signal. Other techniques like Laplacian sharpening expand the signal range to increase contrast. This range amplification is effective for creating perceptually sharper signals, but has some drawbacks like numerical overflow, loss of signal fidelity, and perceptual artifacts like haloing, as shown in the left of Figure 3.2.

Shock Filters evolves convex and concave regions to their locals minima and maxima, respectively, and pushes the entire signal variation to the inflection points. This produces sharp edges and piece-wise constant regions as shown in the right of Figure 3.2.

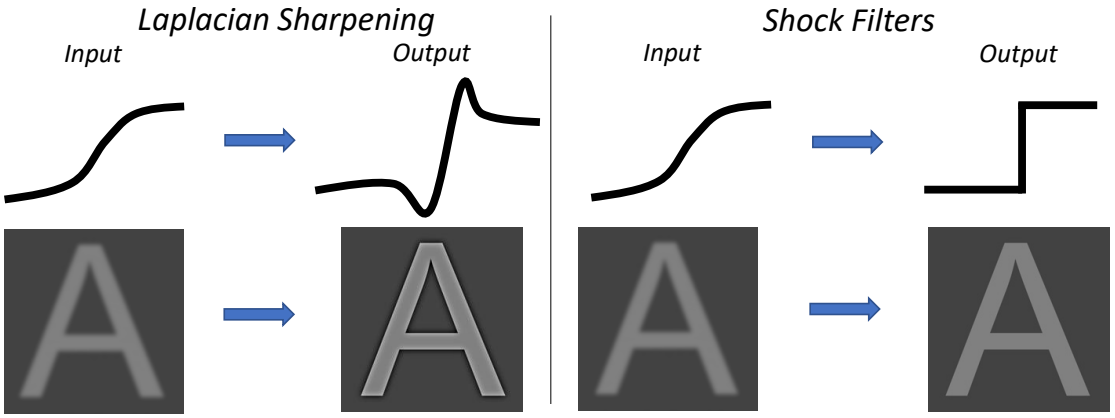


FIGURE 3.2: Comparison of Laplacian sharpening and Shock Filters.

3.2.1 1D signals

Given an input signal $\phi_0 : \mathbb{R} \to \mathbb{R}$, the shock equation solves for a time evolving signal $\phi : \mathbb{R}^{\geq 0} \times \mathbb{R} \to \mathbb{R}$ that transforms the input into a signal with sharper edges and piece-wise constant regions. Given the initial condition $\phi(0, x) = \phi_0$, the shock equation transforms the signal as,

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi|^2 F(\mathscr{L}(\phi)) \tag{3.1}$$

where $\mathscr{L}(\phi)$ is an operator that capture the local concavity of the signal and $F : \mathbb{R} \to \mathbb{R}$ is a sign-preserving modulation function. In the original work the authors suggest using $\mathscr{L}(\phi) = \Delta\phi = \frac{\partial^2 \phi}{\partial^2 x}$ and setting F to be the identity function.

To understand the behaviour of this equation, we start by analyzing it's fixed points, i.e., those points where $\frac{\partial \phi}{\partial t} = 0$. These points either satisfy $\nabla \phi = 0$ or $\frac{\partial^2 \phi}{\partial^2 x} = 0$. The first case corresponds to the critical points of the signal, i.e., the local maximum and minimum. The seconds case corresponds to inflection points, i.e., the positions where the function change concavity, which can also be identified as the precise location of the edges.

The evolution of non-fixed points is dictated by the concavity of it's surrounding region. Points where $\frac{\partial^2 \phi}{\partial^2 x} > 0$ belong to convex regions and evolve towards the local minimum. Points where $\frac{\partial^2 \phi}{\partial^2 x} < 0$ belong to concave regions and evolve towards the local maximum.

As we show in Section 3.2.3, a careful discretization of the 1D Shock equation evolves the signal while preserving monotonicity, total variation and critical points.

3.2.2 2D signals

The shock equation in 2D is defined analogously to the 1D case, using as concavity term $\mathscr{L}(\phi) = \Delta\phi = \frac{\partial^2 \phi}{\partial^2 x} + \frac{\partial^2 \phi}{\partial^2 y}$, and F as the identity function.

As expected, this concavity term ensures that convex regions ($\nabla^2\phi \succ 0$) are evolved towards the local minima and concave regions ($\nabla^2\phi \prec 0$) are evolved toward the maxima. Saddle regions are evolved either to a maxima or minima according to the dominant sign of the second derivatives.

The effect of Shock Filters in 2D is a signal with piece-wise constant regions and sharp transitions. As we will see next, the discretization of the 2D equation still guarantees preservation of local maxima and minima, but monotonicity and total variation are not preserved anymore.

3.2.3 Eulerian implementation

Osher and Rudin evolve the discrete input signal $\boldsymbol{\phi}_0$ using an explicit Eulerian approach. For the 1D case, the authors update the value at each node according to the minmod of forward and backward differences. Finite differences are denoted by,

$$\boldsymbol{d}_+\boldsymbol{\phi}_t[i] := \boldsymbol{\phi}_t[i+1] - \boldsymbol{\phi}_t[i] \text{ and } \boldsymbol{d}_-\boldsymbol{\phi}_t[i] := \boldsymbol{\phi}_t[i] - \boldsymbol{\phi}_t[i-1],$$

and the minmod function is defined by $m(x,y) = \text{sign}(x)\min(|x|,|y|)$ if $xy > 0$ and $m(x,y) = 0$ otherwise. The signal update rule proposed by Osher and Rudin is given by:

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t - \epsilon_t |m(\boldsymbol{d}_-\boldsymbol{\phi}_t, \boldsymbol{d}_+\boldsymbol{\phi}_t)| F(\boldsymbol{d}_-\boldsymbol{d}_+\boldsymbol{\phi}_t)$$

For a stable evolution of the signal, the maximum step that can be taken is given by the Courant-Friedrichs-Lewy(CFL) condition:

$$\epsilon_t \leq \frac{1}{2\max_i |F(\boldsymbol{d}_-\boldsymbol{d}_+\boldsymbol{\phi}_t[i])|}$$

When the CFL condition is met, the evolving signal preserves local minima and maxima, total variation, and monotonicity.

The update rule in the 2D case is a direct extension of the one dimensional case. Denoting $\boldsymbol{d}^{\boldsymbol{u/v}}_{+/-}$ the forward/backward derivative along the respective coordinate direction, the update rule in the 2D case is given by

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t - \epsilon_t \sqrt{(m(\boldsymbol{d}^{\boldsymbol{x}}_-\boldsymbol{\phi}_t, \boldsymbol{d}^{\boldsymbol{x}}_+\boldsymbol{\phi}_t))^2 + (m(\boldsymbol{d}^{\boldsymbol{y}}_-\boldsymbol{\phi}_t, \boldsymbol{d}^{\boldsymbol{y}}_+\boldsymbol{\phi}_t))^2} F((\boldsymbol{d}^{\boldsymbol{x}}_-\boldsymbol{d}^{\boldsymbol{x}}_+ + \boldsymbol{d}^{\boldsymbol{y}}_-\boldsymbol{d}^{\boldsymbol{y}}_+)\boldsymbol{\phi}_t)$$

The respective CFL condition is given by,

$$\epsilon_t \leq \frac{1}{4 \max_{i,j} |F((\boldsymbol{d}_-^{\boldsymbol{x}} \boldsymbol{d}_+^{\boldsymbol{x}} + \boldsymbol{d}_-^{\boldsymbol{y}} \boldsymbol{d}_+^{\boldsymbol{y}}) \boldsymbol{\phi}_t[i,j])|}$$

In the 2D case the CFL condition preserves local minima and maxima, but it does not preserve total variation or monotonicity.

The Eulerian formulation of Shock Filters, as proposed by Osher and Rudin, has a simple implementation and a single update iteration is computationally efficient. However, the CFL condition (which guarantees stability of the signal evolution) also imposes a constraint on the convergence speed. In practice, multiple update iterations are required, introducing signal dissipation. Our Lagrangian formulation overcomes this limitation allowing us to take a single step of arbitrary size while still guaranteeing a stable solution.

3.3 Lagrangian formulation

To derive the Lagrangian formulation we start by writing equation 3.1 in the form:

$$\frac{\partial \phi}{\partial t} = -\langle F(\mathscr{L}(\phi)) \nabla \phi, \nabla \phi \rangle, \tag{3.2}$$

Substituting, $X := F(\mathscr{L}(\phi))\nabla\phi$, we have:

$$\frac{\partial \phi}{\partial t} = -\langle X, \nabla \phi \rangle. \tag{3.3}$$

What does equation say?. From a finite-difference point of view this equation gives,

$$\phi_{t+1}(p) \approx \phi_t(p) - \langle X_t(p), \nabla \phi_t(p) \rangle.$$

On the other hand, from the Taylor series point of view we know that,

$$\phi_t(p - X_t(p)) \approx \phi_t(p) - \langle X_t(p), \nabla \phi_t(p) \rangle$$

Putting these together we conclude,

$$\phi_{t+1}(p) \approx \phi_t(p - X_t(p)).$$

In other words, we can compute the signal at time $t+1$ by resampling the signal at time t at the offset positions given by the vector field X_t. We make this more precise in the following proposition.

Proposition 3.1. *Denote by Φ_X the flow induced by the temporally evolving vector field $X : \mathbb{R}^{\geq 0} \times M \to TM$. In other words, $\Phi_X : \mathbb{R}^{\geq 0} \times M \to M$ is defined by:*

$$\begin{cases} \Phi_X(0, p) = p \\ \frac{d\Phi_X}{dt} = X \circ \Phi_X \end{cases} \tag{3.4}$$

If the flow at each time step is a diffeomorphism, i.e. $\Phi_{X,t} := \Phi_X(t, \cdot)$ is bijective and differentiable, then the signal $\phi_t := \phi_0 \circ \Phi_{X,t}^{-1}$ satisfies

$$\frac{\partial \phi}{\partial t} = -\langle X, \nabla \phi \rangle$$

Proof. By definition $\phi_t \circ \Phi_{X,t} = \phi_0$. Taking derivative respect to t we get,

$$\begin{aligned} 0 &= \frac{d}{dt}(\phi_t \circ \Phi_{X,t})\Big|_{(t,p)} \\ &= \frac{\partial \phi_t}{\partial t}\Big|_{(t,\Phi_{X,t}(p))} + \Big\langle \nabla \phi_t\Big|_{\Phi_{X,t}(p)}, \frac{d\Phi}{dt}\Big|_{(t,p)} \Big\rangle \\ &= \frac{\partial \phi_t}{\partial t}\Big|_{(t,\Phi_{X,t}(p))} + \langle X, \nabla \phi_t \rangle\Big|_{(t,\Phi_{X,t}(p))} \end{aligned}$$

Since $\Phi_{X,t}$ is a bijection, we conclude, $\frac{\partial \phi}{\partial t}\Big|_{(t,p)} = -\langle X, \nabla \phi \rangle\Big|_{(t,p)}$ as desired. □

This interpretation of computing the solution to the Shock Filters equation by resampling the input signal at the position given by the negated flow is the key to our implementation in Section 3.3.1.

Our second major distinction from the original Osher and Rudin formulation is in the choice of the concavity indicator function. If we think of the second directional derivative as a measure of concavity for a specific direction, then the Laplace operator, Δ, is precisely the average measure of concavity along all directions. From our experience, a more useful measure is given by the second derivative along the gradient direction, i.e., the concavity along the perpendicular direction to edges. More precisely, our concavity indicator function is given by,

$$F(\mathscr{L}(\phi))\Big|_p := \frac{d^2}{d^2\tau}\phi\Big(p + \tau\frac{\nabla\phi}{|\nabla\phi|}\Big)\Big|_{\tau=0} = \frac{\langle \nabla^2\phi \cdot \nabla\phi, \nabla\phi\rangle}{|\nabla\phi|^2}\Big|_p = \frac{\langle \nabla|\nabla\phi|^2, \nabla\phi\rangle}{2|\nabla\phi|^2}\Big|_p \tag{3.5}$$

Replacing the expression above into Equation 3.2, the Shock Filters equation reduces to,

$$\frac{\partial \phi}{\partial t} = -\langle \frac{1}{2}\nabla|\nabla\phi|^2, \nabla\phi\rangle \tag{3.6}$$

This is the equation that dictates the evolution of the signal in our Shock Filters implementation. The function $P = \frac{1}{2}|\nabla\phi|^2$ plays a fundamental role in our formulation and will be called the *potential*.

3.3.1 Lagrangian implementation

Our implementation of Lagrangian Shock Filters is based on two major routines: **ComputeFlow** and **IntegrateFlow**.

ComputeFlow takes as input a signal (ϕ) sampled at the nodes of a grid and returns a vector field that is constant per grid face. We use parameters α_ϕ and α_P to control the smoothness of the vector field. This facilitates processing noisy input.

ComputeFlow$(\phi, \alpha_\phi, \alpha_P)$

1 $\phi \leftarrow$ **Smooth**(ϕ, α_ϕ) — *smooth input signal*
2 $P \leftarrow \frac{1}{2}|\nabla\phi|^2$ — *compute per-face potential*
3 $P \leftarrow \frac{\sum_{f \in N(v)} A(f)P}{\sum_{f \in N(v)} A(f)}$ — *sample per-node potential*
4 $P \leftarrow$**Smooth**(P, α_P) — *smooth potential*
5 $X \leftarrow \nabla P$ — *compute per-face vector field*
6 `return` X

So far we have discussed application of Shock Filters to single-channel functions, but the approach for multi-channel signals is similar. Rather than defining an independent potential and vector field for each channel, we create a common one that makes the sampling process more coherent. For multi-channel signals the face potential corresponds to the sum of the squared norms of each channel's gradient: if $\phi = (\phi^r, \phi^g, \phi^b)$, then $P = \frac{1}{2}\left(|\nabla\phi^r|^2 + |\nabla\phi^g|^2 + |\nabla\phi^b|^2\right)$.

IntegrateFlow takes as input a position in the image (p), a vector field (X), and an integration time (t), and returns a new position given by flowing along the vector field. We use the parameter ϵ to define the maximum step size.

IntegrateFlow(X, t, p, ϵ)

1 `while` $t > 0$:
2 $\quad\vec{d} \leftarrow X(p)$ — *sample vector field*
3 $\quad s \leftarrow \min(t, \epsilon/|\vec{d}|)$ — *compute maximum time-step*
4 $\quad p \leftarrow p + s\vec{d}$ — *advance step*
5 $\quad t \leftarrow t - s$ — *decrease time*
5 `return` p

Composing the **ComputeFlow** and **IntegrateFlow** routines provides two different approaches to signal sharpening. The first strategy, which we refer to as the **Iterative**

Sampling, updates the signal at the k-th iteration by setting the new flow from ϕ_{k-1} and sampling from ϕ_{k-1}.

Shock Filters - Iterative Sampling(ϕ_0, t, N)

```
1  for k = [1 : N]
2    X ← ComputeFlow(φ_{k-1})
3      for i, j = [1 : H] × [1 : W]
4        (i', j') ← IntegrateFlow(−X, t/N, (i, j))
5        φ_k(i, j) ← φ_{k-1}(i', j')                    resample signal
6  return φ_N
```

Our second approach, which we refer to as the **Flow Composition**, updates the signal at the k-th iteration by updating the current flow according to ϕ_{k-1} and sampling from ϕ_0.

Shock Filters - Flow Composition(ϕ_0, t, N)

```
1  for i, j = [1 : H] × [1 : W]
2    p^0_ij ← (i, j)                                  initalize flow position
3  for k = [1 : N]
4    X ← ComputeFlow(φ_{k-1})
5      for i, j = [1 : H] × [1 : W]
6        p^k_ij ← IntegrateFlow(−X, t/N, p^{k-1}_ij)    update flow position
7        φ_k(i, j) ← φ_0(p^k_ij)                        resample signal
8  return φ_N
```

In the second and third columns of Figure 3.3 we compare both implementation strategies for a short ($t = 4, N = 4$) and a large ($t = 128, N = 128$) evolution period. The **Iterative Sampling** strategy exhibits significant loss of detail when a large number of integration steps is used. This is particularly noticeable in thin structures like the eyebrows. In contrast, **Flow Composition** produces a result that effectively sharpens the original signal, and preserves fine features after many iterations.

The **Flow Composition** approach resembles the statement of Proposition 3.1: the output signal can be obtained by sampling the input signal at a position given by the flow of an evolving vector field. However, there is a major difference between Proposition 3.1 and our implementation of **Flow Composition**: we approximate the inverse of the flow induced by a dynamic vector field X as the flow induced by the negated vector field $-X$. In other words, we approximate $\Phi^{-1}_{X,t}$ by $\Phi_{-X,t}$. The computation of the inverse map $\Phi^{-1}_{X,t}$ might require an intricate image space discretization and might not be well defined on the entire domain. Instead, the map $\Phi_{-X,t}$ can be easily computed using our flow integration routine and is well defined on the entire domain.

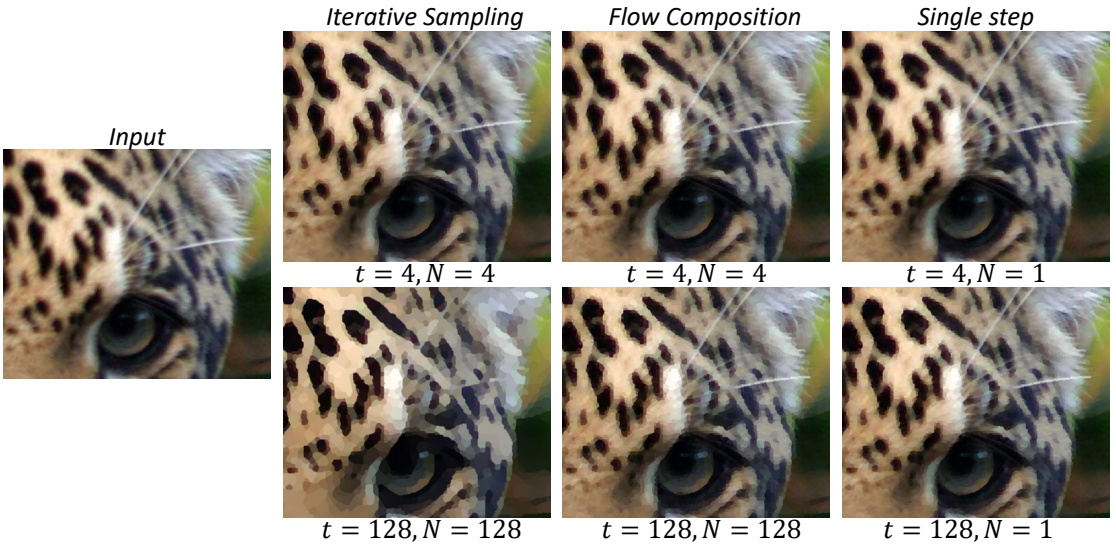


FIGURE 3.3: Comparison of Iterative Sampling, Flow Composition and Transport-Based implementation of Lagrangian Shock Filters.

3.3.2 Transport-Based Shock Filters

So far we have reinterpreted the solution to the Shock Filters equation as the integration and inversion of the flow generated by an evolving vector field (Proposition 3.1). We simplify this problem by considering a flow generated from a static vector field instead. In other words, we assume $X_t = X_0 = \frac{1}{2}\nabla|\nabla\phi_0|^2$ at all time values t. Equation 3.3 with a static vector field is traditionally referred as the Transport Equation [31].

Having a static vector field simplifies the inversion of the flow. Indeed, if Φ_X in Equation 3.4 is computed from a static vector field X_0, it's inverse flow is obtained by integrating the negated vector field. More precisely, if $\Psi_{X_0} : \mathbb{R}^{\geq 0} \times M \to M$ is the solution to

$$\begin{cases} \Psi_{X_0}(0, p) = p \\ \frac{d\Psi_{X_0}}{dt} = -X_0 \circ \Psi_{X_0} \end{cases} \tag{3.7}$$

then $\Psi_{X_0,t} = \Phi^{-1}_{X_0,t}$. From this condition, we compute the evolved signal by sampling the original one at the position given by $\Psi_{X_0,t}$, i.e., $\phi_t = \phi_0 \circ \Psi_{X_0,t}$.

This new approach that sharpens the signal by integrating a static vector field will be referred as the **Transport-Based** implementation. This is a particular instance of both **Iterative Sampling** and **Flow Composition** for the case $N = 1$. For simplicity, we define an auxiliary **Advect** routine that outputs the signal transported by a vector field.

In the fourth column of Figure 3.3 we present the result of the **Transport-Based** Shock Filters. Despite its simplicity, the result is indistinguishable from the iterative **Flow Composition**.

Advect(ϕ, t, X)

```
1  for i, j = [1 : H] × [1 : W]
2    (i', j') ← IntegrateFlow(X, t, (i, j))        path integration
3    φ'(i, j) ← φ(i', j')                          resample signal
4  return φ'
```

Shock Filters - Transport-Based$(\phi, t, \alpha_\phi, \alpha_P)$

```
1  X ← ComputeFlow(φ, α_φ, α_P)                    compute flow field
2  φ ← Advect (φ, −X, t)                           transport signal
3  return φ
```

In Figure 3.4 we compare our Lagrangian **Transport-Based** implementation of Shock Filters to the Eulerian implementation of Osher and Rudin. A close-up on the Osher and Rudin approach, exhibit jagged patterns due to the Eulerian update rule. These artifacts are accentuated when many update iterations are applied to the signal ($N = 100$ for these examples). Our **Transport-Based** approach enhance edges without introducing significant artifacts even for a large integration time ($t = 100$).

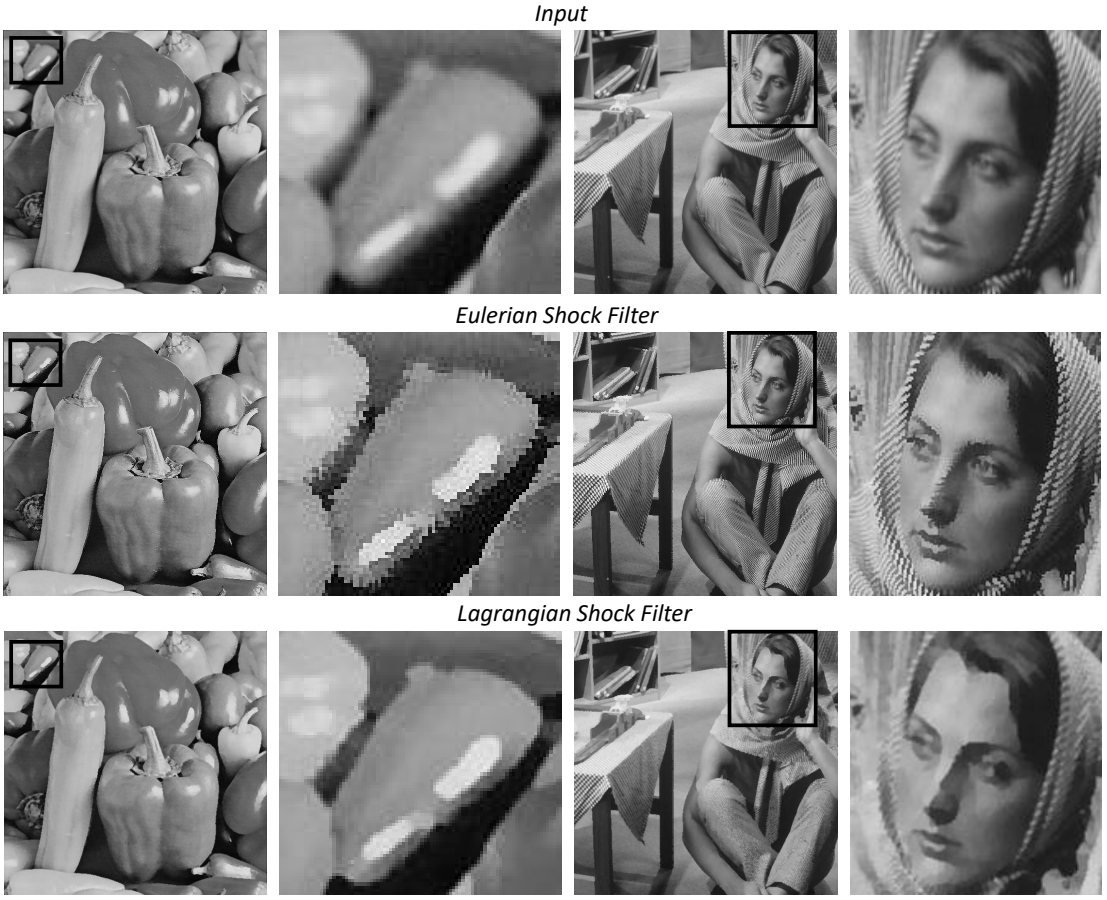


FIGURE 3.4: Comparison of Eulerian and Lagrangian (Transport-Based) Shock Filters

3.4 Extension to meshes

In this section we explore the extension of the **Transport-Based** Shock Filters algorithm to sharpening signals on triangle meshes. We evaluate its convergence, noise robustness, and performance.

3.4.1 Implementation

The pseudo-code for the implementation of **Transport-Based** Shock Filters in triangle meshes is identical to the one described for images. Signals are still sampled at vertices and vector fields are constant per face (in this case triangles rather than cells). Due to the distinct domain discretization, the are subtle differences in the computation of gradients (bilinear gradient vs. linear gradient), face to vertex prolongation (uniform weighting vs. area weighting), and signal sampling (bilinear interpolation vs. barycentric interpolation).

Integrate Flow The first major difference is the integration of vector fields. In particular, given a position p and a direction $\vec{d} \in T_pM$ we compute the new position $p + s\vec{d}$ by successively unfolding triangles. As shown in Figure 3.5, once the partially integrated vector field hits a triangle edge the trajectory is continued by unfolding the adjacent triangle. We continue traversing (unfolding) triangles along the current direction until we reach the max step distance ϵ or we complete the total integration time. We set the max step distance ϵ to be the average edge length of the mesh.

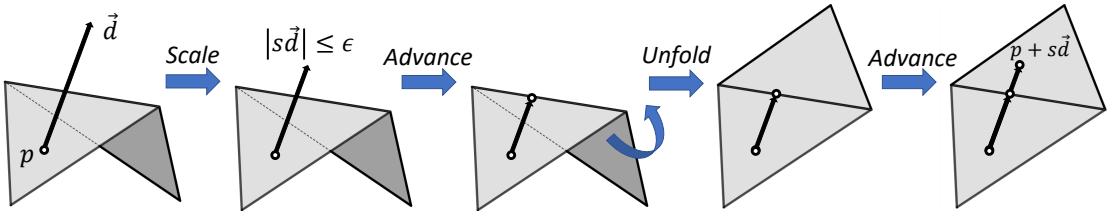


FIGURE 3.5: Path integration in triangle meshes.

Smooth Signals The second major difference between the image and mesh Shock Filters algorithms is the implementation of the smoothing operators. In the image domain, we can smooth the signal by convolving with a Gaussian. On meshes, the irregular connectivity and the non-homogeneous sampling make this problem harder. We smooth a signal on a mesh following the gradient-domain approach, i.e., given an input signal ϕ we obtain a smoothed signal ψ by solving a least squares problem that

trades off between gradient attenuation and signal fidelity. The solution to this problem is the output of our **Smooth** operator on meshes:

$$\textbf{Smooth}(\psi, \alpha) := \underset{\phi}{\operatorname{argmin}} \, ||\psi - \phi||^2 + \alpha ||\nabla \phi||^2 \tag{3.8}$$

The optimal solution to this problem is given by $\boldsymbol{\phi} = (\mathbf{M} + \alpha \mathbf{S})^{-1} \mathbf{M} \boldsymbol{\psi}$ where $\mathbf{M}$ and $\mathbf{S}$ are the mass and stiffness matrices (Section 2.3.1.4).

Images vs Meshes In Figure 3.6 we compare side-by-side the results of our Lagrangian Shock Filters on a regular image and on a triangle mesh. The procedure is analogous for both domains and the differences come from the discretization. In the middle rows we visualize the two vector fields that we compute with our formulation: the gradient of the input signal and the flow field (which is the gradient of the potential). For both domains, the vector fields are constant per face (cells and triangles respectively). Note how the flow field points towards the signal edges. The new signal value at each node is obtained by flowing in the opposite direction and sampling the input signal. Thus, for nodes on opposite sides of an edge, we flow in opposite directions and sample far apart values, thereby increasing the edge contrast.

3.4.2 Geometry sharpening

The Gauss map, $N : M \to S^2$ assigns a normal direction to each point on the surface. On triangle meshes, the normal field is usually represented at each vertex by a 3D vector (n_x, n_y, n_z), and it is extended to the interior of the triangles using barycentric interpolation.

We run our Lagrangian Shock Filters method on the Gauss Map to compute a sharper normal field. In this case the potential $P = \frac{1}{2}|\nabla N|^2 = \kappa_1^2 + \kappa_2^2$ is a measure of the total curvature of the surface. The local minimum of the potential corresponds to regions of low curvature, while the local maximum corresponds to geometric features such as edges and corners. Our Shock Filters method flows along the negated gradient of the potential and samples new normals from low curvature regions. This creates a new normal field with larger flat regions and increased contrast around the features of the initial geometry.

We compute new geometry that closely match the sharpened normal field by following the approach of Yu et al. [32]. Specifically, we compute new vertex positions p by solving

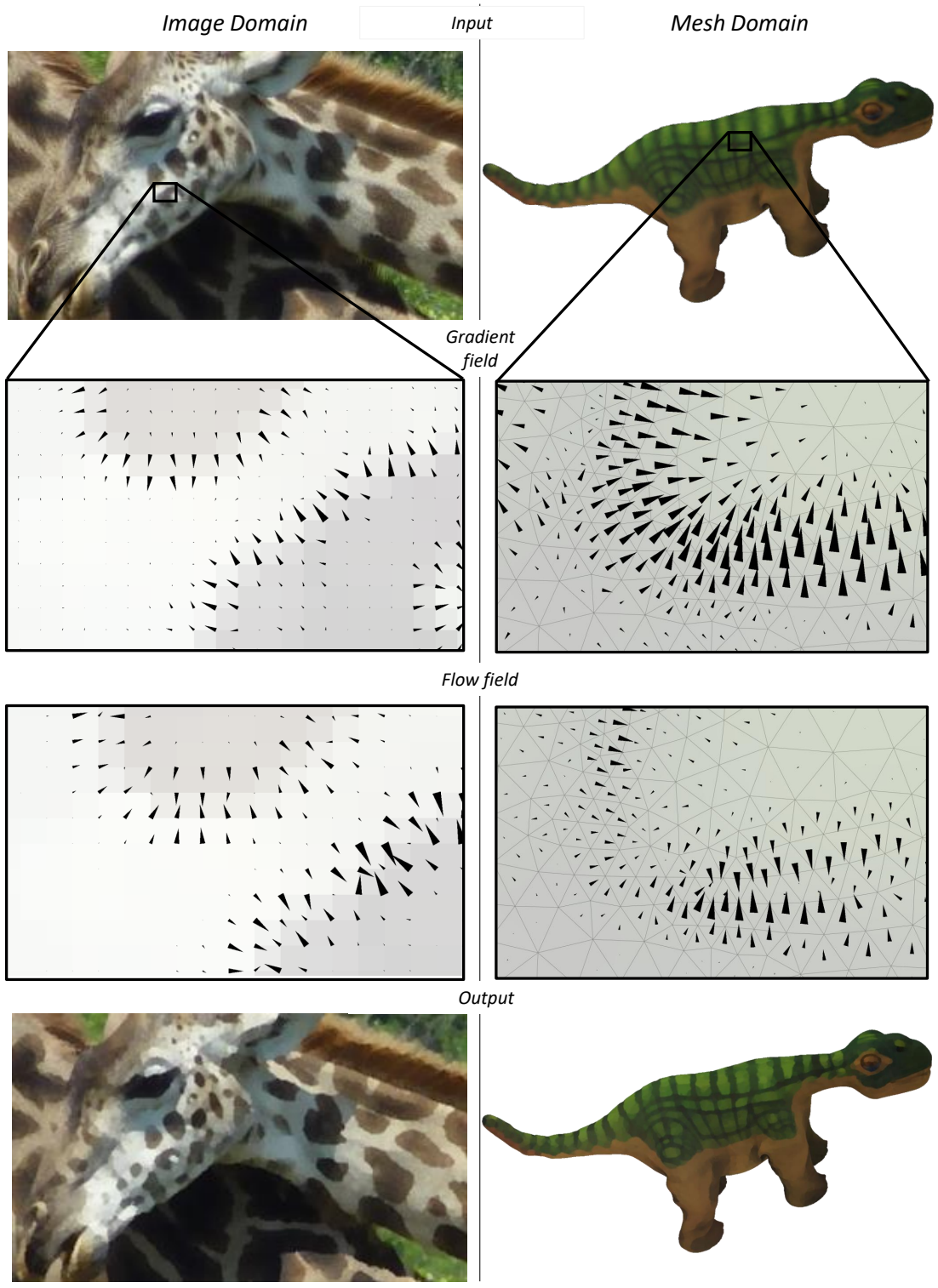


FIGURE 3.6: Comparison of the discretization of Lagrangian Shock Filters on images and triangles meshes.

the screened-Poisson equation,

$$\min_p ||p_0 - p||^2 + \alpha||(\nabla p_0 - \langle \nabla p_0, N\rangle N) - \nabla p||^2, \tag{3.9}$$

where N denotes the sharpened normal signal and p_0 the original vertex positions. The first term of this energy (the screening term) is a regularization term that encourages the reconstructed positions to remain close to the input. The second term of the energy (the gradient term) encourages the gradient of the new positions to be close to the projection of the input positions gradient onto the plane perpendicular to the computed normal.

3.4.2.1 Results

In this section we compare the effect of the two main parameters that guide the sharpening process: the flow integration time (t) and the flow field smoothing (α_ϕ, α_P).

In Figure 3.7 we compare results for the *Gargoyle* model (872K vertices) using different flow times. For this example we compute a common flow field (we set $\alpha_\phi = \alpha_P = 0.01$) and show the reconstructed surface from the transported normal field for times $t = 1, 4, 10$, and 10^6. Since our flow lines attract each sample in the surface to a critical point of the potential, as we increase the flow time, the target normal field gets sampled from smaller regions (eventually just the normal at the critical points). Thus, as the time increases, the target normal field becomes piece-wise constant, as is effectively captured by the reconstruction. For $t = 1$ we obtain a sharper geometry without sacrificing the details of the input model. For larger flow times we accentuate large scale features at the cost of losing fine detail. We observe the stability of our approach by showing the result for $t = 10^6$, where the flow of all the samples have converged to a critical point. Under each result we also report the running time (in seconds) of the **IntegrateFlow** routine. Even for very large flow times our method terminates quickly. A more detailed analysis of performance is presented in Section 3.4.3.1.

FIGURE 3.7: Comparison of geometry sharpening for different flow integration times.

In Figure 3.8 we compare results for the *Chinese Dragon* model (1.3M vertices) using different smoothing parameters for the flow field. For this example we compute flow fields by setting $\alpha_\phi = \alpha_P = 0, 10^{-4}, 10^{-3}$, and 10^{-2} and show the reconstructed surface from the transported normal at the full convergence time $t = 10^6$. By increasing the smoothing weight we remove the local critical points in the potential, and produce a target normal field with fewer and larger constant patches. As observed from Figure

3.8, when no smoothing is enabled the local flow is sufficient to sharpen the crests of the model (see the contour of the ear, mouth and eyebrows) without losing much detail. For larger smoothing weights we preserve the dominant critical features and reconstruct a model with stronger contours and piece-wise flat regions.

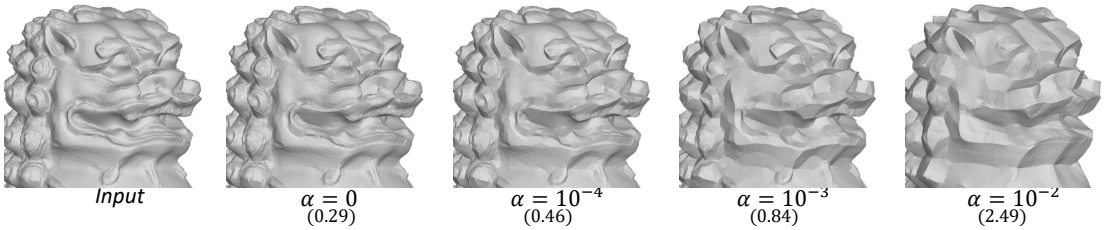


FIGURE 3.8: Comparison of geometry sharpening for different magnitude of smoothing of the flow field.

3.4.3 Evaluation

3.4.3.1 Performance and convergence

To evaluate the performance and convergence of our method, we consider the simple scenario of sharpening a blurred step function on a sphere. In Figure 3.9 we report the running time (in seconds) and visualize the sharpened signal for different combinations of flow time (increasing from left to right) and mesh resolution (increasing from top to bottom). For all the results we use the common set of smoothing parameters $\alpha_\phi = \alpha_P = 10^{-4}$. First, it is interesting to observe that for a fixed flow time (i.e., any column in Figure 3.9) the quality of the reconstruction is similar across all resolutions. This is an expected result, which demonstrates that the construction of our flow field depends on the input signal and the intrinsic geometry of the surface, but not on the tessellation.

Second, we highlight that the increase in flow time produces a sub-linear increase in running time. This situation can be understood from the early flow termination of some of the samples. Since our flow field is the gradient of a potential, each flow line has a termination point (due to the compactness of the surface), which can be reached in finite time (at least in the discrete setting). Once a sample reaches its termination point, flowing for larger time has no effect[1]. Comparing the quality of the reconstructed signal and the running time between $t = 10$ and $t = 10^6$, we can conclude that by $t = 10$ almost all samples have reached their termination point (in this case, the poles of the sphere).

[1]In practice, when our vector field does not vanish, the flow jitters within an ϵ-radius of the termination point

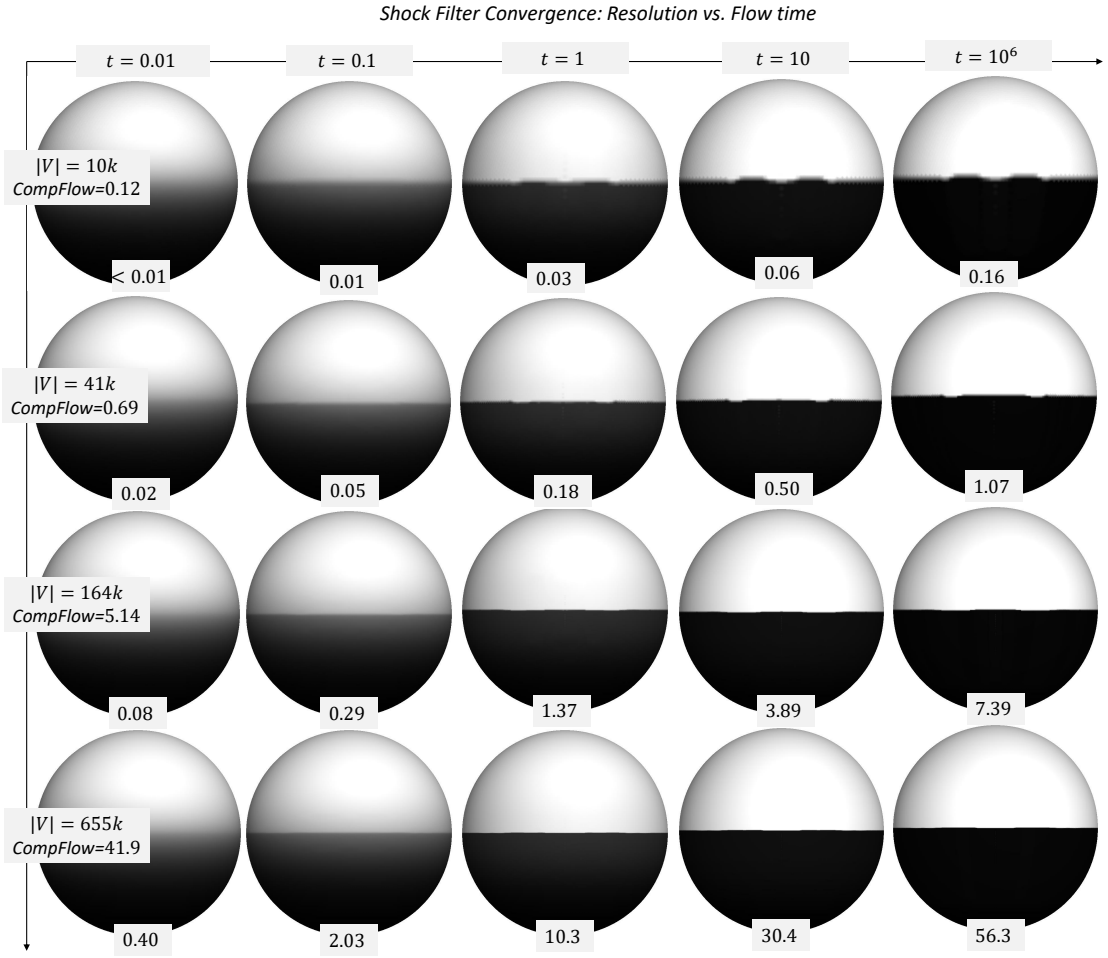


FIGURE 3.9: Evaluation of Shock Filters convergence as a function of resolution and flow integration time.

Third, we notice a super-linear increase of running time with respect to resolution. In our experiment, the number of samples (i.e., mesh vertices) increases by a factor of 4 between consecutive rows, but the running time increases by a factor closer to 8. Observe that increasing resolution increases the number of paths to be integrated by a factor of 4. If the cost of integrating each path were independent of resolution we would have linear scaling of running time. However that is not the case: as we increase resolution the number of triangles that a sample path traverse increases by a factor of two. The 4-fold increase in the number of paths combined with the 2-fold increase in the number of triangles along a path together explain the 8-fold increase in running time.

3.4.3.2 Robustness to noise

In Figure 3.10 we compare our approach to the methods of Solomon et al. [33] on two noisy *Frog* models (10K vertices). We show the results of bilateral filtering and mean-shift in the second and third column resp., and the result of our method with smoothing

disabled and enabled in the fourth and fifth column resp. For low amplitude noise (top row), all three successfully clean the data, and Shock Filters additionally sharpens the edges. For larger amplitude noise (bottom row), bilateral filtering and Shock Filters (with signal smoothing disabled) fail to clean the data. When enabling the smoothing of the normal field (we use two passes of Laplacian smoothing) we get a signal that is successfully sharpened with Shock Filters.

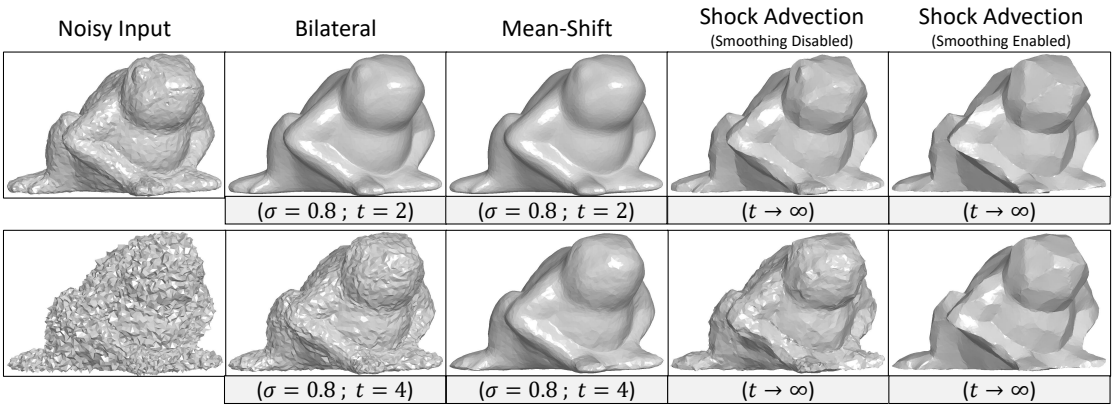


FIGURE 3.10: Comparison of Shock Filters robustness to noise.

Chapter 4

Optical Flow

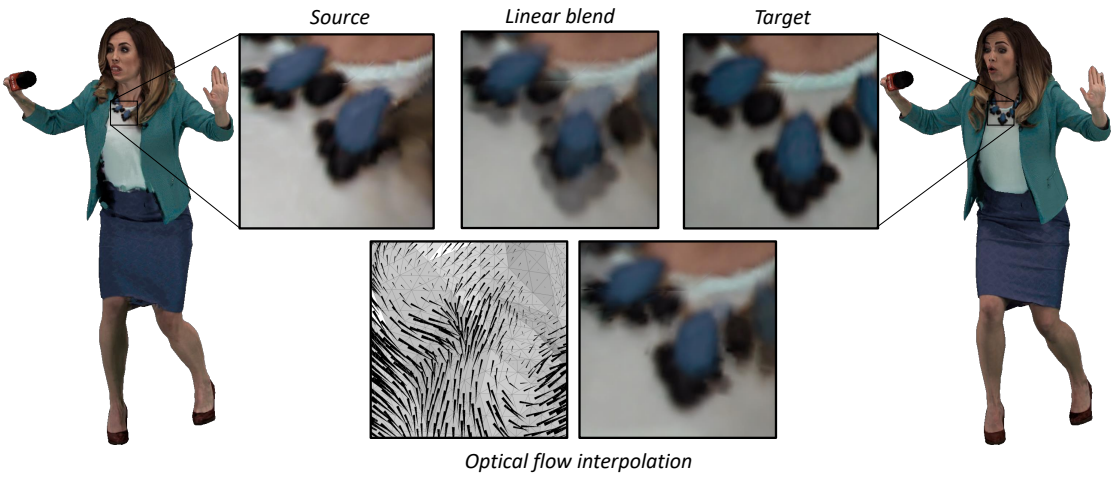


FIGURE 4.1: The alignment of a pair of textures using our mesh-based optical flow produce ghosting-free interpolation, as shown by the *Reporter* model.

Optical flow is the problem of estimating the motion of features captured in a sequence of images. The most simple formulation involves the alignment of a pair of images, where the goal is to compute a vector field that associates points from the source image to corresponding points in the target. As shown in Figure 4.1, we study the pairwise alignment problem for signals sampled at the vertices of a triangle mesh and describe how the techniques originally developed for images can be extended to meshes.

4.1 Introduction

In this chapter we introduce the variational formulation of optical flow proposed by Horn and Schunck [2]. This contains the fundamental principles of optical flow: the brightness constancy constraint and motion regularization. These principles have been adopted by most of the work in the area, and we also follow them in our mesh-based formulation.

Due to its locality, the Horn-Schunck method provides poor motion estimation on scenarios with large displacement. Subsequent works on optical flow introduced multi-scale decomposition to compensate for this. Of particular interest is the linear scale space approach of Alvarez et al. [34] which we extend to signals on meshes.

Optical Flow has been an active research area in Computer Vision for over 35 years and state-of-the-art results in flow estimation are far beyond the capabilities of the work by Horn and Schunck. State-of-the art methods [35] relay on segmentation, feature matching, and even learning techniques. Adapting these methods to irregular and non-homogeneous domains like triangle meshes seems a promising research direction, and is not discussed in our current work.

Despite its simplicity, our extension of the Horn-Schunck method to optical flow on triangle meshes provides robust results. We demonstrate this with applications in texture interpolation and photometric tracking.

4.2 Horn-Schunck formulation

The variational formulation of Horn-Schunck represents the time varying signal as a function $\phi : \mathbb{R}^{\geq 0} \times \mathbb{R}^2 \to \mathbb{R}$, and solves for a flow $\Phi : \mathbb{R}^{\geq 0} \times \mathbb{R}^2 \to \mathbb{R}^2$ that tracks the motions of each point in the signal domain. For simplicity, we denote the signal and flow at a fixed time t by ϕ_t and Φ_t resp.

The Horn-Schunck formulation has two major components: the characterization of valid deformations through the brightness constancy constraint, and the introduction of a regularization term to compensate for the Aperture Problem.

4.2.1 Brightness constancy

The brightness constancy constraint states that the intensity of each point in the scene is invariant across all temporal instances of the signal. More formally, for each point p in the scene and any time t we have:

$$\frac{d}{dt}(\phi_t \circ \Phi_t)\Big|_{(t,p)} = 0 \tag{4.1}$$

Following Proposition 3.1, the total derivative of this equation at time $t = 0$ gives us the brightness constancy constraint:

$$\Big(\frac{\partial \phi}{\partial t} + \Big\langle \frac{d\Phi}{dt}, \nabla \phi \Big\rangle \Big)\Big|_{(0,p)} = 0 \tag{4.2}$$

This equation establishes that the temporal change of intensity at any point of the domain ($\frac{\partial \phi}{\partial t}$) must be explained by the local intensity model of the signal ($\nabla \phi$) and the direction of motion ($\frac{d\Phi}{dt}$). Both $\frac{\partial \phi}{\partial t}$ and $\nabla \phi$ are known values derived from the input and $X \equiv \frac{d\Phi}{dt}$ is the motion differential we wish to solve for.

Equation 4.2 provides a necessary but not sufficient condition to solve for the motion differential X. If $\nabla \phi$ is non zero, then $X = -\frac{\partial \phi}{\partial t} \frac{\nabla \phi}{|\nabla \phi|^2}$ is a solution to the equation, but so is any offset in the direction orthogonal to the gradient. The fact that we cannot solve for the motion by just looking at local (spatial and temporal) changes in intensity is known as the Aperture Problem [36]. In the next section we describe how to overcome it.

4.2.2 Motion regularization

Intuitively, the motion differential X should be piecewise smooth, i.e., neighbouring points belonging to the same object should be moving at roughly the same speed in the image plane. Following this intuition, Horn and Schunck add a regularization term that enforces smoothness of the motion differential. Denoting by ∇X the Jacobian of the motion differential, Horn and Schunck estimate the motion by minimizing the energy:

$$E(X) := \underbrace{\int_\Omega \Big(\frac{\partial \phi}{\partial t} + \langle X, \nabla \phi \rangle\Big)^2 dA}_{\text{Data Term}} + \underbrace{\epsilon \int_\Omega |\nabla X|_F^2 dA}_{\text{Smoothness Term}} \tag{4.3}$$

By adding this regularization term, the computation of the alignment vector field becomes a global optimization problem. The Taylor expansion of the energy gives us the condition:

$$E(X+Y) = E(X) + 2\int_\Omega \Big\langle \frac{\partial \phi}{\partial t} \nabla \phi + (\nabla \phi \nabla \phi^\top - \epsilon \Delta) X, Y \Big\rangle dA + \underbrace{\int_\Omega \langle (\nabla \phi \nabla \phi^\top - \epsilon \Delta) Y, Y \rangle dA}_{\geq 0} \tag{4.4}$$

Since the third term of equation 4.4 is non-negative, a sufficient condition for the optimal solution X^* of equation 4.3 is:

$$\frac{\partial \phi}{\partial t} \nabla \phi + (\nabla \phi \nabla \phi^\top - \epsilon \Delta) X^* = 0 \tag{4.5}$$

Inverting, the optimal solution is given by:

$$X^* = -(\nabla\phi\nabla\phi^\top - \epsilon\Delta)^{-1}\frac{\partial\phi}{\partial t}\nabla\phi \tag{4.6}$$

Scale correction In practice, the regularization term $|\nabla X|_F^2$ induces shrinkage in the alignment vector field X^*. We compute an appropriate scale by solving for the factor α that minimizes the Data Term of Equation 4.3,

$$\int_\Omega \Big(\frac{\partial\phi}{\partial t} + \langle \alpha X^*, \nabla\phi\rangle\Big)^2 dA = \Big(\int_\Omega \langle \nabla\phi\nabla\phi^\top X^*, X^*\rangle dA\Big)\alpha^2 + 2\Big(\int_\Omega \langle \frac{\partial\phi}{\partial t}\nabla\phi, X^*\rangle dA\Big)\alpha + O(1) \tag{4.7}$$

The optimal scale factor is given by,

$$\alpha^* = -\Big(\int_\Omega \langle \nabla\phi\nabla\phi^\top X^*, X^*\rangle dA\Big)^{-1}\Big(\int_\Omega \langle \frac{\partial\phi}{\partial t}\nabla\phi, X^*\rangle dA\Big) \tag{4.8}$$

Finally we reassign the optimal motion differential as $X^* \leftarrow \alpha^* X^*$

4.3 Pairwise alignment

The simplest instance of the optical flow problem consist of the alignment of a pair of images ϕ_0 and ϕ_1, which we call *source* and *target* respectively. In this section we compare the two main alternatives to represent motion between a pair of images, we described the naive iterative flow correction algorithm, and its improved version using multiresolution techniques.

4.3.1 Motion representation

The traditional goal of optical flow algorithms is to compute a *forward* vector field X that matches each point in the source image to its respective position in the target:

$$\phi_0(p) \approx \phi_1(p + X) \tag{4.9}$$

This alignment formulation is very intuitive due to the temporal nature of the motion and it is the one used for evaluation in optical flow benchmarks [37].

An alternative formulation is given by the *halfway* alignment approach [38, 39] where the computed vector field should satisfy:

$$\phi_0(p - X/2) \approx \phi_1(p + X/2) \tag{4.10}$$

In the next sections we describe the implementation of our optical flow algorithm using the halfway alignment approach. In contrast to the forward approach, the halfway approach can represent motions with partially or totally occluded regions using continuous vector fields. Additionally, the symmetric role of source and target signals make the halfway representation more convenient for applications like image interpolation [38].

4.3.2 Iterative flow correction

Our approach iterates between warping source and target signals to a half way alignment, and updating the halfway vector field to decrease the alignment error. More precisely, given signals ϕ_0, ϕ_1 and a current estimate of the alignment vector field X, we define partially aligned signals $\tilde{\phi}_0(p) = \phi_0(p - X/2)$ and $\tilde{\phi}_1(p) = \phi_1(p + X/2)$. We compute a smooth vector field that provides a better alignment between the partially aligned signals by taking a Taylor expansion of the signal difference:

$$\tilde{\phi}_1(p + Y/2) - \tilde{\phi}_0(p - Y/2) \approx (\tilde{\phi}_1 - \tilde{\phi}_0)(p) + \langle \nabla(\frac{\tilde{\phi}_0 + \tilde{\phi}_1}{2})(p), Y \rangle$$

Then, we solve for he correction field Y that minimizes:

$$\int_\Omega \Big((\tilde{\phi}_1 - \tilde{\phi}_0) + \langle \nabla(\frac{\tilde{\phi}_0 + \tilde{\phi}_1}{2}), Y \rangle \Big)^2 + \epsilon ||\nabla Y||^2 dA \tag{4.11}$$

This is precisely the Horn and Schunck regularized optical flow energy 4.3 for the halfway alignment formulation: $\frac{\partial \phi}{\partial t} = \tilde{\phi}_1 - \tilde{\phi}_0$, and $\nabla \phi = \nabla(\frac{\tilde{\phi}_0 + \tilde{\phi}_1}{2})$.

We compute the regularized correction vector field X^* as in Equation 4.6, and estimate the optimal scale α^* given by Equation 4.8. Finally, we update our estimation of the halfway alignment field by setting $X \leftarrow X + \alpha^* X^*$. The entire process is summarized in the **IterativeFlowCorrection** algorithm.

IterativeFlowCorrection$(\phi_0, \phi_1; \epsilon, N)$

$X \leftarrow 0$
`for` $i = 0, \ldots, N-1$:
  $\tilde{\phi}_0 \leftarrow \phi_0(p - X/2)$ — *source halfway alignment*
  $\tilde{\phi}_1 \leftarrow \phi_1(p + X/2)$ — *target halfway alignment*
  $X^* \leftarrow$ **EstimateFlow**$(\tilde{\phi}_0, \tilde{\phi}_1, \epsilon)$ — *local correction(Equation 4.6)*
  $\alpha^* \leftarrow$ **GetScale**$(\tilde{\phi}_0, \tilde{\phi}_1, X^*)$ — *correction scale (Equation 4.8)*
  $X \leftarrow X + \alpha^* X^*$ — *update vector field*
`return` X

The **IterativeFlowCorrection** algorithm for the forward alignment formulation is almost identical. In the next table we summarize the minor implementation differences between these two approaches:

Method	Forward Alignment	Halfway Alignment
Objective	$\phi_0(p) \approx \phi_1(p+X)$	$\phi_0(p-X/2) \approx \phi_1(p+X/2)$
Warping	$\tilde{\phi}_0 \leftarrow \phi_0$, $\tilde{\phi}_0 \leftarrow \phi_0(p+X)$	$\tilde{\phi}_0 \leftarrow \phi_0(p-X/2)$, $\tilde{\phi}_1 \leftarrow \phi_1(p+X/2)$
Discretization	$\frac{\partial \phi}{\partial t} = \tilde{\phi}_1 - \tilde{\phi}_0, \nabla\phi = \nabla\tilde{\phi}_1$	$\frac{\partial \phi}{\partial t} = \tilde{\phi}_1 - \tilde{\phi}_0, \nabla\phi = \nabla(\frac{\tilde{\phi}_0+\tilde{\phi}_1}{2})$

TABLE 4.1: Alignment implementation comparison

4.3.3 Multiscale

Due to its local nature, the **IterativeFlowCorrection** algorithm has very slow convergence and is unable to provide correct alignment in the case of large motion. This weakness of single-resolution optical flow estimation was already pointed in the seminal work of Lucas and Kanade [40].

Several multiscale approaches have been proposed to compensate for the locality of the optical flow estimation. The work of Glazer [41] was the first to suggest the use of image pyramids within the Horn-Schunck model. This was realized in the works of Enk [42] and Anandan [43] which develop a complete system for optical flow estimation in natural images using image pyramids.

The image pyramid approach downsamples the source and target images to a coarse resolution where local displacements corresponds to large motions at the input resolution. Starting from the coarsest level, the alignment vector field is iteratively improved using a similar strategy to the one described in **IterativeFlowCorrection**. Once a satisfactory alignment is obtained at a given level, the alignment vector field is upsampled and corrected in the next finer level.

Alvarez et al. [34] proposed an alternative multiscale approach to optical flow based on linear scale-space theory. Rather than downsampling source and target images to coarser resolutions, each scale level is defined by applying a low pass filter to the input images. Formally, given a collection of Gaussian filters $\{G_{\sigma_i}\}_{i=0}^{L}$, with standard deviations $\sigma_0 > \sigma_1 > \ldots > \sigma_L = 0$, a hierarchical representation of the signals is obtained by convolving with these filter. In other words, $\{G_{\sigma_i} * \phi_0\}_{i=0}^{L}$ and $\{G_{\sigma_i} * \phi_1\}_{i=0}^{L}$ are the hierarchical representation of source and target signals. As in the pyramid case, the computation of the alignment vector is done using an iterative correction approach, starting from the coarsest representation of the signals and progressively moving to finer representations.

Our mesh-based optical flow algorithm follows a similar linear scale-space approach to the one described in [34]. For completeness we present the pseudo-code for the **ScaleSpace-IterativeFlowCorrection** algorithm, which extends the single-resolution IterativeFlowCorrection algorithm. Observe that the only modification is the specification of a family of smoothing filters $\{G_{\sigma_i}\}_{i=0}^{L}$ that are successively applied to source and target signals:

ScaleSpace-IterativeFlowCorrection$(\phi_0, \phi_1; \epsilon, \{G_{\sigma_l}\}_{l=0}^{L})$

$X \leftarrow 0$
`for` $l = 0, \ldots, L-1$:
  $\tilde{\phi}_0 \leftarrow (G_{\sigma_i} * \phi_0)(p - X/2)$ *source halfway alignment*
  $\tilde{\phi}_1 \leftarrow (G_{\sigma_i} * \phi_1)(p + X/2)$ *target halfway alignment*
  $X^* \leftarrow$ **EstimateFlow**$(\tilde{\phi}_0, \tilde{\phi}_1, \epsilon)$ *local correction(Equation 4.6)*
  $\alpha^* \leftarrow$ **GetScale**$(\tilde{\phi}_0, \tilde{\phi}_1, X^*)$ *correction scale (Equation 4.8)*
  $X \leftarrow X + \alpha^* X^*$ *update vector field*
`return` X

4.4 Extension to meshes

In this section we describe our extension of the Horn-Schunck algorithm to compute a halfway alignment between a pair of signals ϕ_0 and ϕ_1 sampled at the vertices of mesh. We start with an overview of the algorithm and then proceed with a detailed discussion of its components.

4.4.1 Overview

Our **MeshOpticalFlow** algorithm has an analogous formulation to the **ScaleSpace-IterativeFlowCorrection** algorithm introduced for images:

MeshOpticalFlow$(\phi_0, \phi_1; \alpha_g, \alpha_s, \epsilon, L)$

$\phi_0, \phi_1 \leftarrow$ **SignalPreprocessing**$(\phi_0, \phi_1, \alpha_g)$ *remove lighting bias*
$X \leftarrow 0$
`for` $l = 0, \ldots, L-1$:
  $\tilde{\phi}_0 \leftarrow$ **Advect**(**Smooth**$(\phi_0, \alpha_s/4^l), -X, 1/2)$ *source halfway alignment*
  $\tilde{\phi}_1 \leftarrow$ **Advect**(**Smooth**$(\phi_1, \alpha_s/4^l), X, 1/2)$ *target halfway alignment*
  $X^* \leftarrow$ **EstimateFlow**$(\tilde{\phi}_0, \tilde{\phi}_1, \epsilon)$ *local flow correction*
  $\alpha^{\leftarrow}$ **GetScale**$(\tilde{\phi}_0, \tilde{\phi}_1, X^*)$ *correction flow scale*
  $X \leftarrow X + \alpha^* X^*$ *update vector field*
`return` X

Our algorithm adds a preprocessing step to compensate for the lighting variations on the input signals. For simplicity, this step was not included on the image-based algorithms, where the changes of lighting or viewpoint can be more subtle, but can also be included for a more robust algorithm.

We compute the halfway warping and the scale-space representation of signals using the flow integration and smoothing operators already introduced for Shock Filters sharpening.

From a theoretical point of view the most interesting component of extending optical flow to meshes is the definition of a vector field regularization term. On triangle meshes, vector fields can be represented using per-vertex, per-edge or per-triangle discretizations. Section 4.5 explains the constructions of the smoothing operators and the construction of the mesh-based **EstimateFlow** routine.

In Section 4.6 we compare the properties, and evaluate the performance of the different vector field representations in the context of optical flow. We conclude by showing applications of our mesh-based optical flow.

Source code of our **MeshOpticalFlow** can be found at `https://github.com/fabianprada/MeshOpticalFlow`.

4.4.2 Signal preprocessing

The brightness constancy constraint 4.1 is approximately satisfied when there are low illumination changes in the signal acquisition process. However, for signals on meshes this is not generally the case : the object is scanned from multiple view points and each might have different lighting conditions. To compensate for illumination changes, we replace the input signals by new signals that vanish on regions of low intensity variation. This is done by subtracting from each signal a smoothed version of itself. The **Smooth** operator we apply on the input signal is the solution to the screened-Poisson equation already introduced in Equation 3.8.

SignalPreprocessing$(\phi; \alpha_g)$

$\phi \leftarrow \phi - \textbf{Smooth}(\phi, \alpha_g)$

`return` ϕ

4.4.3 Halfway alignment

In the context of images, halfway alignment can be directly represented though a vector field: given a point p in the image plane and an vector X_p, both $p - X_p/2$ and $p + X_p/2$

correspond to well defined positions on the image domain. However, when $p \in M$ is a point in a curved surface and $X_p \in T_pM$ is a tangent vector, which points in M are associated to $p - X/2$ and $p + X/2$? A natural answer is to use the exponential map, i.e., we could define $p \pm X_p/2 := \exp(p, \pm X_p/2)$. The exponential map give us the endpoint of the unique geodesic starting at p with direction X_p and length $|X_p|$.

An alternative approach, which we prefer instead, is to use the flow induced by X. We denote this flow by $\Phi_X : M \times \mathbb{R} \to M$, with $\Phi_X(p,t) \in M$ the point obtained by integrating the vector field X starting from position p, for time t. Using the flow notation, the action of the halfway alignment transformation of vector field X is given by $p \pm X/2 := \Phi_{\pm X}(p, 1/2)$.

Our preference for using the flow over the exponential map as alignment transformation is twofold. First, flow along smooth vector fields define local diffeomorphisms on surfaces:

$$\Phi_{-X}(\Phi_X(p,t),t) = p \tag{4.12}$$

This property does not hold on exponential maps unless the vector field is aligned with geodesic curves.

The second reason is the sensitivity of the exponential map to noise. In surfaces with large negative curvature a small perturbation in the tangent direction may produce large variation in the position given by the exponential map. Vector field regularization is more effective for attenuating this phenomena in the case of flows.

We transform the vector field representation to be constant per triangle (see Section 2.3.2.3) and integrate paths using the sequential unfolding procedure introduced in Section 3.4.1.

4.4.4 Linear scale space

In order to support alignment under large motion, we use a direct extension of the linear scale-space proposed by Alvarez et al. [34]. We use the **Smooth** operator described in Equation 3.8. Since the smoothing operators are applied to the source and target signals, the computation of the entire multiscale representation can be performed before the flow estimation. The magnitude of smoothing at the coarsest level is defined by the parameter α_s. In our implementation we relax this weight by a factor of $\frac{1}{4}$ as we move to finer levels.

Multiresolution vector fields The extension of the linear scale space approach to signals on meshes is achieved by specifying a parametric family of smoothing operators. In contrast, the extension of the image pyramid approach (i.e., the construction of a "mesh pyramid") is more challenging due to the complex topology and non-homogeneous sampling of triangle meshes. Still, multiresolution methods on meshes have been explored for applications such as geometric modeling [44, 45], mesh editing [46], and signal processing [47]. To our knowledge multiresolution methods for vector fields have not been explored before. Adapting the mesh based optical flow algorithm to multiresolution meshes is left as an open problem which we believe to be a promising approach for improving the run-time performance.

4.5 Vector field representation

Horn and Schunck use the Jacobian of the vector field to measure the smoothness of the alignment transformation. While this is a natural choice for vector fields defined on the plane, how we can extend this notion of smoothness to vector fields on curved surfaces?

The Jacobian of a vector field on the plane is a linear operator that describes how the vector field changes in the neighbourhood of a point. An analogous definition in the case of surfaces is the covariant derivative [6]. Given a vector field, $X : M \to TM$, the covariant derivative at a point $p \in M$ is a linear transformation on its tangent space, $(\nabla_{(\cdot)}X)_p : T_pM \to T_pM$, that indicates the tangential change of X as we move in a particular direction.

The covariant derivative of a vector field X at a particular point p and direction Y_p, can be extrinsically computed in two steps: first, compute the differential of the vector field X for a curve passing through p with direction Y_p, then, project the differential back to the tangent plane. More precisely, if $\gamma : [-\epsilon, \epsilon] \to M$ is a curve with $\gamma(0) = p$ and $\gamma'(0) = Y_p$, then,

$$(\nabla_{Y_p}X)_p \equiv \Pi_{N_p^\perp}\Big((X \circ \gamma)'(0)\Big) \tag{4.13}$$

While the previous computation is extrinsic, the covariant derivative is an intrinsic operator, i.e., it only depends on the first fundamental form.

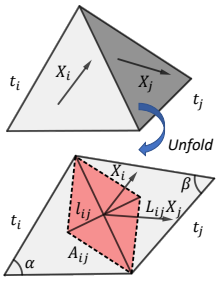
Discretizing the covariant derivative in simplicial surfaces is a challenging task. Knöppel et al. [48] provide a discretization of this operator using piecewise linear vector fields associated to mesh vertices.

Instead of discretizing the covariant derivative we rely on simpler operators that are defined over more traditional representation of vector fields and are easier to construct.

As we show in our experiments, these operators capture the smoothness of a vector field and produce good results when incorporated in the optical flow framework.

We start by introducing an operator that measures smoothness by computing vector differences between adjacent triangles. Then, we introduce operators that measure smoothness from the divergence and curl of the vector field.

4.5.1 Vector differences



Our first smoothing operator aggregates the squared differences between vectors in adjacent triangles. As shown in the inset, we unfold adjacent triangles to compare vectors in a common tangent spaces. We associate to the mesh a dual structure, and normalize the vector difference by the geodesic distance between the dual centers.

Formally, our measure of smoothness on the Triangle basis is given by the energy,

$$\textbf{VectorDifferences}(\boldsymbol{X}) := \sum_{t_i \in T} \sum_{t_j \in N(t_i)} A_{ij} \frac{|X_i - L_{ij} X_j|^2}{|l_{ij}|^2} \tag{4.14}$$

where L_{ij} is the linear transformation mapping vectors from triangle t_j to triangle t_i, and l_{ij} is the distance between dual centers. We modulate the error by A_{ij} which is the dual area associated to the edge between triangles t_i and t_j,

An intuitive choice for the dual structure is the circumcentric dual, which satisfies orthogonality between dual and primal edges. For the circumcentric dual the ratio $\frac{A_{ij}}{|l_{ij}|^2}$ reduces to the inverse of the traditional cotangent weights:

$$\frac{A_{ij}}{|l_{ij}|^2} = \frac{1}{\cot \alpha + \cot \beta}$$

However, the use of circumcentric duals has a major drawback: when a triangle and its unflipped neighbour are concyclic we get $|l_{ij}| = 0$ which makes the energy 4.14 undefined. Furthermore, when the edge is not Delaunay, the weight $\frac{A_{ij}}{|l_{ij}|^2}$ is negative. As an alternative we use the barycentric duals. On the barycentric dual, dual and primal edges are not orthogonal anymore, but the ratio $\frac{A_{ij}}{|l_{ij}|^2}$ is well defined.

We denote by $\mathbf{S}_1^T$ the bilinear operator that measures smoothness of a vector field using **VectorDifferences**. More formally, if $\boldsymbol{X}$ are the coefficients of the vector field, then $\mathbf{S}_1^T$ is the symmetric matrix that satisfies $\textbf{VectorDifferences}(\boldsymbol{X}) = \boldsymbol{X}^\top \mathbf{S}_1^T \boldsymbol{X}$.

4.5.2 Curl and divergence

An alternative approach to measure the smoothness of a vector field is through its curl and divergence. To compute the curl and divergence of a discrete vector field we will rely on the limit approximation introduced in Section 2.3.1.3. From equation 4.17, a point-wise approximation to the vector field curl is given by,

$$\nabla \times X\Big|_{\Omega} \approx \frac{\int_{\delta\Omega} X \cdot ds}{|\Omega|} \tag{4.15}$$

and we can approximate the integral of its squared value by,

$$\int_{\Omega} |\nabla \times X|^2 dA \approx \Big(\int_{\delta\Omega} X \cdot ds\Big)^2 / |\Omega| \tag{4.16}$$

Partitioning the surface, $M = \cup\Omega_i$, we define the curl energy over the surface as

$$\int_{M} |\nabla \times X|^2 dA \approx \sum_{\Omega_i} \Big(\int_{\delta\Omega} X \cdot ds\Big)^2 / |\Omega_i| \tag{4.17}$$

An analogous expression gives the vector field divergence energy,

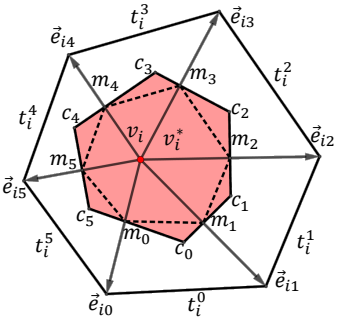
$$\int_{M} |\nabla \cdot X|^2 dA \approx \sum_{\Omega_i} \Big(\int_{\delta\Omega} JX \cdot ds\Big)^2 / |\Omega_i| \tag{4.18}$$

We use these expressions for the computation of the squared divergence and curl on two different representation of vector fields: the Gradient basis and the Whitney basis.

4.5.2.1 Gradient basis

The Gradient basis is the space of vector fields generated by gradients and rotated gradients of the hat basis function, $\mathscr{B}_1^C := \text{span}\{\nabla\phi_i, J\nabla\phi_i\}_{i\in V}$, with J the rotation by 90 degrees in the tangent plane. Since the hat basis functions are linear within each triangle, the Gradient basis is a subspace of the vector fields that are constant per triangle, $\mathscr{B}_1^C \subset \mathscr{B}_1^T$.

In order to compute the curl energy we partition the mesh into per-vertex Voronoi regions. As shown in the inset, we denote by $m_0, c_0, m_1, \ldots, m_{n-1}, c_{n-1}, m_n$ the set of edge midpoints and triangle circumcenters on the boundary of the Voronoi region v_i^* in positive orientation. Given $X \in \mathscr{B}_1^T$, integrating the vector field along the Voronoi boundary we obtain:



$$\int_{v_i^*} \nabla \times X d\mu = \sum_k \langle \overrightarrow{m_k c_k}, X_k \rangle + \langle \overrightarrow{c_k m_{k+1}}, X_k \rangle = \sum_k \langle \overrightarrow{m_k m_{k+1}}, X_k \rangle \tag{4.19}$$

We denote by $\nabla \phi_i^k$ the gradient of the hat basis at vertex i restricted to the k-th incident triangle t_i^k. It is known that this vector can be expressed as $\nabla \phi_i^k = \frac{1}{|t_i^k|} J \overrightarrow{m_k m_{k+1}}$, or equivalently that $\overrightarrow{m_k m_{k+1}} = -|t_i^k| J \nabla \phi_i^k$. Using this identity we conclude,

$$\sum_k \langle \overrightarrow{m_k m_{k+1}}, X_k \rangle = -\sum_k \langle J \nabla \phi_i^k, X_k \rangle |t_i^k| = -\int_M \langle J \nabla \phi_i, X \rangle dA$$

Expressing the vector field in the Gradient basis, $X = \sum_j \kappa_j \nabla \phi_j + \sum_j \xi_j \nabla J \phi_j$, and using the fact that $\text{span}\{J \nabla \phi_j\}_j \perp \text{span}\{\nabla \phi_j\}_j$, we get,

$$\begin{aligned}
-\int_M \langle J \nabla \phi_i, X \rangle dA &= -\sum_j \kappa_j \cancel{\int_M \langle J \nabla \phi_i, \nabla \phi_j \rangle dA} - \sum_j \xi_j \int_M \langle J \nabla \phi_i, J \nabla \phi_j \rangle dA \\
&= -\sum_j \xi_j \int_M \langle \nabla \phi_i, \nabla \phi_j \rangle dA \\
&= -\sum_j \xi_j \mathbf{S}_{ij} \\
&= -(\mathbf{S}\boldsymbol{\xi})_i
\end{aligned}$$

Letting $\mathbf{M}$ be the lumped mass matrix, with per-vertex Voronoi areas on its diagonal, and applying the discretization given in Equation 4.17, we conclude,

$$\int_M |\nabla \times X|^2 dA \approx \boldsymbol{\xi}^\top (\mathbf{S}^\top \mathbf{M}^{-1} \mathbf{S}) \boldsymbol{\xi}$$

The operator $\mathbf{S}^\top \mathbf{M}^{-1} \mathbf{S}$ is traditionally referred to as the bilaplacian [49]. Following an analogous procedure we arrive to an identical result for the energy of the vector field divergence:

$$\int_M |\nabla \cdot X|^2 dA \approx \boldsymbol{\kappa}^\top (\mathbf{S}^\top \mathbf{M}^{-1} \mathbf{S}) \boldsymbol{\kappa}$$

Finally, our measure of smoothness in the Gradient basis is given by the operator $\mathbf{S}_1^C$:

$$\textbf{CurlAndDiv}(\boldsymbol{X} = [\boldsymbol{\kappa}, \boldsymbol{\xi}]) := \boldsymbol{\kappa}^\top (\mathbf{S}^\top \mathbf{M}^{-1} \mathbf{S}) \boldsymbol{\kappa} + \boldsymbol{\xi}^\top (\mathbf{S}^\top \mathbf{M}^{-1} \mathbf{S}) \boldsymbol{\xi}$$

The Gradient basis provide an explicit decomposition between divergence free and curl free vector fields: the first corresponds to $\text{span}\{J \nabla \phi_i\}_i$, and the second to $\text{span}\{\nabla \phi_i\}_i$. In particular the only vector field that is simultaneously divergence and curl (these are called *harmonic*) is the zero vector field.

However, it is known that the dimension of the harmonic field of a surface is $2g$, where g its is genus [10]. Thus, the Gradient basis cannot contain the harmonic vector fields for non genus-zero surfaces. The Whitney discretization that we describe next overcomes this limitation.

4.5.2.2 Whitney basis

Discrete exterior calculus (DEC) provides an alternative way to compute the divergence and curl of a vector field using differential forms and the so called Hodge Laplacian [10]. Instead of considering an explicit representation of the vector field, DEC uses an implicit representation by storing the integral of the vector field along the edges of the mesh. More precisely, to each oriented edge $\vec{e}_{ij}$, we associate a value ω_{ij}, such that

$$\omega_{ij} = \int_{\vec{e}_{ij}} X \cdot ds$$

This definition makes the computation of the integrated curl on a triangle straightforward. When the triangle t_{ijk} is positively oriented we deduce:

$$\int_{t_{ijk}} (\nabla \times X) dA = \int_{\vec{e}_{ij}} X \cdot ds + \int_{\vec{e}_{jk}} X \cdot ds + \int_{\vec{e}_{ki}} X \cdot ds = \omega_{ij} + \omega_{jk} + \omega_{ki}$$

Following Equation 4.17, the squared integrated curl over the entire mesh is given by,

$$\int_M |\nabla \times X|^2 dA \approx \sum_{t_{ijk}} (\omega_{ij} + \omega_{jk} + \omega_{ki})^2 / |t_{ijk}|$$

Denoting by $\boldsymbol{\omega}$ the array of integrated coefficients, and using DEC notation (Section 2.3.2.4) the expression above corresponds to,

$$\int_M |\nabla \times X|^2 dA \approx \boldsymbol{\omega}^\top (d_1^\top \star_2 d_1) \boldsymbol{\omega}$$

We compute the integrated divergence at each vertex, by estimating the integral of the rotated vector field along the boundary of its Voronoi region. We decompose the boundary of the Voronoi region into segments $c_{k-1}c_k$, and denote by $\gamma_k : I_k \to c_{k-1}c_k$ their arc-length parameterization (see the inset in 4.5.2.1). Since $J\overrightarrow{c_{k-1}c_k}$ and $\vec{e}_{ik}$ are

parallel, we can estimate

$$\begin{aligned}
\int_{v_i^*} \nabla \cdot X dA &= \sum_k \int_{\overrightarrow{c_{k-1}c_k}} JX \cdot ds \\
&= \sum_k \int_{I_k} \Big\langle \frac{-J\overrightarrow{c_{k-1}c_k}}{|c_{k-1}c_k|}, X(\gamma_k(t)) \Big\rangle dt \\
&= \sum_k \Big\langle \frac{-J\overrightarrow{c_{k-1}c_k}}{|c_{k-1}c_k|}, \frac{\vec{e}_{ik}}{|e_{ik}|} \Big\rangle \int_{I_k} \Big\langle \frac{\vec{e}_{ik}}{|e_{ik}|}, X(\gamma_k(t)) \Big\rangle dt \\
&\approx \sum_k \Big\langle \frac{-J\overrightarrow{c_{k-1}c_k}}{|c_{k-1}c_k|}, \frac{\vec{e}_{ik}}{|e_{ik}|} \Big\rangle \frac{|c_{k-1}c_k|}{|e_{ik}|} \int_{\vec{e}_{ij}} X \cdot ds \\
&= \sum_k \langle -J\overrightarrow{c_{k-1}c_k}, \frac{\vec{e}_{ik}}{|e_{ik}|^2} \rangle \omega_{ik}
\end{aligned}$$

Using the fact that $\langle -J\overrightarrow{c_{k-1}c_k}, \frac{\vec{e}_{ik}}{|e_{ik}|^2} \rangle = \int_M \langle \nabla\phi_i, \nabla\phi_k \rangle dA = \mathbf{S}_{ij}$, we obtain,

$$\int_{v_i^*} \nabla \cdot X dA = \sum_k \omega_{ik} \mathbf{S}_{ik}$$

and following Equation 4.18 we conclude,

$$\int_M |\nabla \cdot X|^2 dA \approx \sum_i \Big(\sum_k \omega_{ik} \mathbf{S}_{ik} \Big)^2 / |v_i^*|$$

From the DEC notation this corresponds to,

$$\int_M |\nabla \cdot X|^2 dA \approx \boldsymbol{\omega}^\top (\star_1 d_0 \star_0^{-1} d_0^\top \star_1) \boldsymbol{\omega}$$

We denote by $\mathbf{S}_1^W$ the discrete operator that measures the smoothness of a vector field in the DEC representation. Summarizing the previous results, this corresponds to:

$$\textbf{HodgeLaplacian}(\boldsymbol{X} = \boldsymbol{\omega}) := \boldsymbol{\omega}^\top \mathbf{S}_1^W \boldsymbol{\omega} = \boldsymbol{\omega}^\top (d_1^\top \star_2 d_1 + \star_1 d_0 \star_0^{-1} d_0^\top \star_1) \boldsymbol{\omega}$$

4.5.3 Estimate flow

The **EstimateFlow** routine of our **MeshOpticalFlow** algorithm computes the solution to the mesh-based discretization of Equation 4.3.

As we described in Section 4.3.2, for the halfway alignment formulation, the gradient and temporal derivative of the signal are discretized by $\nabla\phi = \nabla(\frac{\phi_0+\phi_1}{2})$ and $\frac{\partial\phi}{\partial t} = \phi_1 - \phi_0$

respectively. In practice, we discretize the data term of Equation 4.3 as an area-weighted sum of the correction field error evaluated at the triangle barycenter[1]:

$$E_{\text{Data}}(X) := \int_M \Big(\frac{\partial \phi}{\partial t} + \langle X, \nabla \phi \rangle\Big)^2 \approx \sum_{t \in T} |t| \Big((\phi_1 - \phi_0) + \langle X_t, \nabla(\frac{\phi_0 + \phi_1}{2})\rangle\Big)^2 \quad (4.20)$$

To represent the previous energy in matrix form, we introduce the following auxiliary operators:

- $\mathbf{M}_0^T \in \mathbb{R}^{|T| \times |T|}$: diagonal triangle mass operator, $(\mathbf{M}_0^T)_{ii} = |t_i|$.
- $\boldsymbol{K}_1^T(\boldsymbol{Y}) \in \mathbb{R}^{|T| \times 2|T|}$: point-wise inner product with a reference vector field $\boldsymbol{Y}$,

$$\boldsymbol{K}_1^T(\boldsymbol{Y}) = \begin{pmatrix} Y_1^\top g_1 & & & 0 \\ & Y_2^\top g_2 & & \\ & & \ddots & \\ 0 & & & Y_{|T|}^\top g_{|T|} \end{pmatrix} \quad (4.21)$$

- $\mathbf{P}_0^{V \to T} \in \mathbb{R}^{|T| \times |V|}$: vertex-to-triangle signal prolongation.
- $\boldsymbol{D}_0 \in \mathbb{R}^{2|T| \times |V|}$: gradient of a signal in the Triangle basis.

Denoting by $\boldsymbol{\delta} := \boldsymbol{\phi}_1 - \boldsymbol{\phi}_0$ and $\boldsymbol{\mu} = \frac{1}{2}(\boldsymbol{\phi}_0 + \boldsymbol{\phi}_1)$ the difference and mean of the signal coefficients, the discrete data term in the Triangle basis can be written as:

$$E_{\text{Data}}(\boldsymbol{X}) = (\mathbf{P}_0^{V \to T}\boldsymbol{\delta} + \boldsymbol{K}_1^T(\boldsymbol{D}_0\boldsymbol{\mu})\boldsymbol{X})^\top \mathbf{M}_0^T(\mathbf{P}_0^{V \to T}\boldsymbol{\delta} + \boldsymbol{K}_1^T(\boldsymbol{D}_0\boldsymbol{\mu})\boldsymbol{X}) \quad (4.22)$$

The respective data terms for the Gradient and Whitney basis are obtained by transforming them to a per-triangle vector field representation using the prolongation operators introduced in Section 2.3.1.4. Finally, the flow correction energy we solve at each inner iteration of **MeshOpticalFlow** algorithm is given by,

$$E_{\text{Flow}}(\boldsymbol{X}) = E_{\text{Data}}(\mathbf{P}_1^{\gamma \to T}\boldsymbol{X}) + \epsilon \boldsymbol{X}\mathbf{S}_1^\gamma \boldsymbol{X} \quad (4.23)$$

where $\boldsymbol{X}$ is the array of vector field coefficients in the basis $\mathscr{B}_1^\gamma$ and $\mathbf{P}_1^{\gamma \to T}$ and $\mathbf{S}_1^\gamma$ are the respective prolongation and smoothness operators. The energy in Equation 4.23 is quadratic in $\boldsymbol{X}$ and its optima can be obtained as the solution to a linear system. We summarize the procedure through the pseudo-code for the mesh-based **EstimateFlow** routine:

[1]This 1-point quadrature estimation of the error has proven to be sufficient in our applications.

EstimateFlow$(\phi_0, \phi_1, \epsilon)$

$\boldsymbol{\delta} \leftarrow \boldsymbol{\phi}_1 - \boldsymbol{\phi}_0$
$\boldsymbol{\mu} \leftarrow \frac{1}{2}(\boldsymbol{\phi}_0 + \boldsymbol{\phi}_1)$
$\boldsymbol{G} \leftarrow \boldsymbol{K}_1^T(\boldsymbol{D}_0\boldsymbol{\mu})\mathbf{P}_1^{\gamma\to T}$
$\boldsymbol{A} \leftarrow \boldsymbol{G}^\top \mathbf{M}_0^T \boldsymbol{G} + \epsilon \mathbf{S}_1^\gamma$
$\boldsymbol{b} \leftarrow (\mathbf{P}_0^{V\to T}\boldsymbol{\delta})^\top \mathbf{M}_0^T \boldsymbol{G}$
$\boldsymbol{X} \leftarrow \boldsymbol{A}^{-1}\boldsymbol{b}$
`return` $\boldsymbol{X}$

Since the system matrix $\boldsymbol{A}$ depends on the warped signal values, this matrix needs to be recomputed at each call of **ExtimateFlow**. Furthermore, since we use a Cholesky decomposition [50] of $\boldsymbol{A}$ to solve for the correction field, we also need to update the numerical factorization. Summary of the computation costs of matrix construction, numerical factorization and substitution for the different vector field representations are presented in Section 4.7.2.

4.6 Vector field evaluation

In the previous sections we described three different representations of vector fields on meshes and their respective smoothness operators: the **VectorDifferences** operator ($\mathbf{S}_1^T$) for the Triangle basis, the **CurlAndDiv** operator ($\mathbf{S}_1^C$) for the Gradient basis, and the **HodgeLaplacian** operator ($\mathbf{S}_1^W$) for the Whitney basis. In this section we compare the spectral properties of theses smoothness operators and their performance within the optical flow setup.

4.6.1 Spectrum

The spectrum of the smoothness operators are obtained by solving the generalized eigenvalue problem,

$$\mathbf{S}_1^\gamma \boldsymbol{\phi} = \lambda \mathbf{M}_1^\gamma \boldsymbol{\phi} \tag{4.24}$$

with $\mathbf{S}_1^\gamma$ and $\mathbf{M}_1^\gamma$ the respective smoothness and mass operator (see Section 2.3.2.3).

4.6.1.1 Flat surfaces

In Figure 4.2 we compare the spectrum of our smoothness operators to the analytic spectrum of the covariant derivative for a *Flat Torus* (i.e., the $[0, 2\pi]^2$ domain with

periodic boundary conditions). On a *Flat Torus*, the squared Frobenius norm of the covariant derivative of a vector field $X = (u(x,y), v(x,y))$ is simply given by,

$$\int_{\mathbb{T}^2} |\nabla X|^2 dA = \int_{\mathbb{T}^2} \langle \nabla u, \nabla u \rangle + \langle \nabla v, \nabla v \rangle dA = \int_{\mathbb{T}^2} \langle u, -\Delta u \rangle + \langle v, -\Delta v \rangle dA \qquad (4.25)$$

where the second equality follow from the Divergence Theorem.

From Equation 4.25, it follows that the Fourier basis,

$$\mathscr{B}_1^F := \Big\{ \begin{pmatrix} e^{i(kx+ly)} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ e^{i(kx+ly)} \end{pmatrix} \Big\}_{k,l \in \mathbb{Z}},$$

provides an eigen decomposition for the *Flat Torus* spectrum, where vector fields $\left(\begin{smallmatrix} e^{i(kx+ly)} \\ 0 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 0 \\ e^{i(kx+ly)} \end{smallmatrix}\right)$ have associated eigenvalue $k^2 + l^2$. Thus for each $n \in \mathbb{Z}$, the dimension of the eigenspace associated to $\lambda = n$ is given by $2|\{(k,l) \in \mathbb{Z} \times \mathbb{Z} : n = k^2 + l^2\}|$. For instance, the dimension of the eigenspace associated to $\lambda = 0$ is $2 = 2|\{(0,0)\}|$ and for $\lambda = 1$ it is $8 = 2|\{(-1,0),(1,0),(0,-1),(0,1)\}|$.

In Figure 4.2 we plot the numerical spectrum of our smoothness operators for an irregular triangulation of a *Flat Torus* against its analytic spectrum.

We can think of **VectorDifferences** as a unbiased estimator of the squared Frobenius norm of the covariant derivative on flat surfaces. To prove this observe that at any point $p \in \mathbb{T}^2$ we can verify,

$$\lim_{r \to 0} \mathop{\mathbb{E}}_{|Y|=r} \Big(\frac{|X_{p+Y} - X_p|^2}{r^2} \Big) = \mathop{\mathbb{E}}_{|Y|=1} (|(\nabla_Y X)_p|_2^2) = \frac{1}{2} |(\nabla X)_p|_F^2$$

Thus, given a dense sampling $\{p_i\}_{i \to \infty} \subset \mathbb{T}^2$, and directions $\{Y_i\}_{i \to \infty} \to 0$, we have that

$$\begin{aligned} \mathbb{E}\Big(\sum_{i \to \infty} |p_i| \frac{|X_{p_i+Y_i} - X_{p_i}|^2}{|Y_i|^2} \Big) = \sum_{i \to \infty} |p_i| \mathop{\mathbb{E}}_{Y_i \to 0} \Big(\frac{|X_{p_i+Y_i} - X_{p_i}|^2}{|Y_i|^2} \Big) &= \frac{1}{2} \sum_{i \to \infty} |p_i| |(\nabla X)_{p_i}|_F^2 \\ &= \frac{1}{2} \int_{\mathbb{T}^2} |\nabla X|^2 dA \end{aligned}$$

VectorDifferences is an estimator of this form where we take $\{p_i\}$ to be the edge midpoints of a triangulation and $\{Y_i\}$ the dual edges. Using circumcentric weights we reproduce the correct spectrum as observed in Figure 4.2. Instead, barycentric weights distort the precision of the spectrum. This loss of precision might be a consequence of the lack of orthogonality between the primal triangulation and its barycentric dual [51].

On the other hand, the sum of squared integrated curl and divergence of a vector filed match the Frobenius norm of its covariant derivative on flat surfaces. Writing $X =$

$(X^u(u,v), X^v(u,v))$, we get that,

$$\begin{aligned}\int_{\mathbb{T}^2} |\nabla \times X|^2 + |\nabla \cdot X|^2 dA &= \int_{\mathbb{T}^2} \left(\frac{\partial X^u}{\partial u} + \frac{\partial X^v}{\partial v}\right)^2 + \left(\frac{\partial X^v}{\partial u} - \frac{\partial X^u}{\partial v}\right)^2 dudv \\ &= \int_{\mathbb{T}^2} |\nabla X|^2 dA + 2\int_{\mathbb{T}^2} \frac{\partial X^u}{\partial u}\frac{\partial X^v}{\partial v} dudv - 2\int_{\mathbb{T}^2} \frac{\partial X^v}{\partial u}\frac{\partial X^u}{\partial v} dudv\end{aligned}$$

Form the chain rule we deduce,

$$\int_{\mathbb{T}^2} \frac{\partial X^u}{\partial u}\frac{\partial X^v}{\partial v} dudv = \cancel{\int_{\mathbb{T}^2} \frac{\partial}{\partial u}(X^u \frac{\partial X^v}{\partial v}) dudv} - \int_{\mathbb{T}^2} X^u \frac{\partial^2 X^v}{\partial u \partial v} dudv = -\int_{\mathbb{T}^2} X^u \frac{\partial^2 X^v}{\partial u \partial v} dudv$$

and,

$$\int_{\mathbb{T}^2} \frac{\partial X^v}{\partial u}\frac{\partial X^u}{\partial v} dudv = -\int_{\mathbb{T}^2} X^u \frac{\partial^2 X^v}{\partial v \partial u} dudv$$

Since $\frac{\partial^2 X^v}{\partial u \partial v} = \frac{\partial^2 X^v}{\partial v \partial u}$, we conclude,

$$\int_{\mathbb{T}^2} |\nabla \times X|^2 + |\nabla \cdot X|^2 dA = \int_{\mathbb{T}^2} |\nabla X|^2 dA$$

In the Gradient basis, the harmonic vector fields, $\text{span}(\binom{1}{0}, \binom{0}{1})$, cannot be represented as gradients or rotated gradients of periodic functions. Thus, **CurlAndDiv** produce a shifted spectrum of the covariant derivative operator in the *Flat Torus*.

The Whitney basis supports the representation of harmonics fields on the *Flat Torus*: taking dot product between a constant direction $\vec{v}$ and each edge of the triangulation to we get the coefficients of the harmonic field parallel to $\vec{v}$. As observed in Figure 4.2 **HodgeLaplacian** reproduces the spectrum of the covariant derivative.

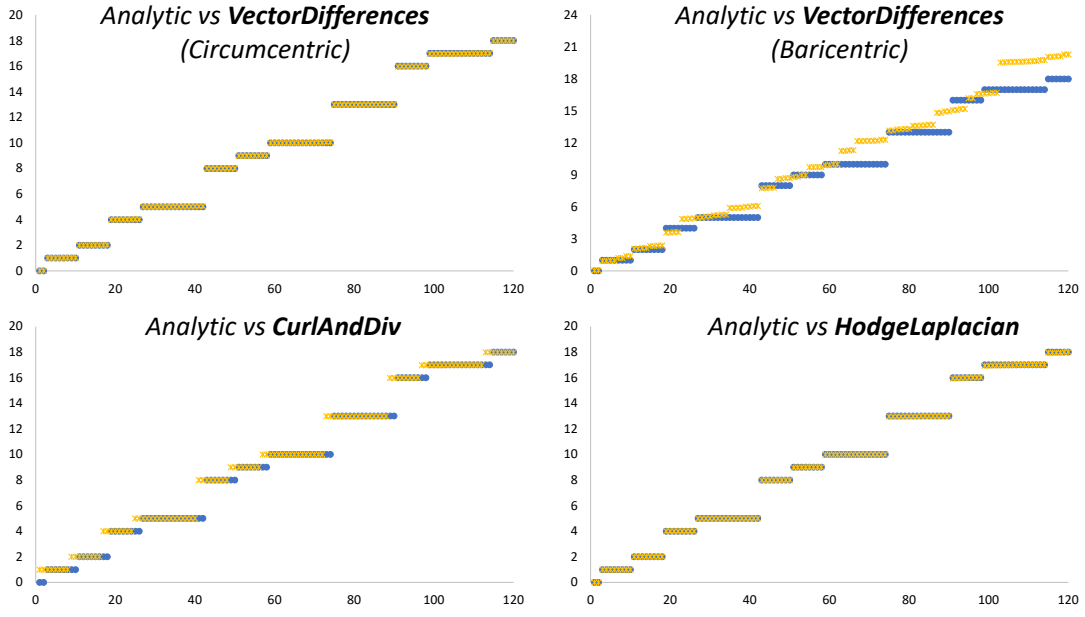


FIGURE 4.2: Spectral comparison of the smoothness operators in a *Flat Torus*.

VectorDifferences Circumcentric	**VectorDifferences** barycentric	**CurlAndDiv**	**HodgeLaplacian**
-321.364	17.863	4.885	0.000
-321.364	17.863	4.885	0.000
15.212	33.685	12.170	0.000
15.212	33.685	12.170	0.000
29.432	41.425	18.894	0.000
29.432	41.425	18.894	0.000
36.583	43.651	24.300	0.000
36.583	43.651	24.300	0.000
39.471	49.222	30.859	4.880
39.471	49.222	30.859	4.885
43.599	54.888	32.441	12.165
43.599	54.888	32.441	12.177
48.652	59.250	41.433	18.871
48.652	59.250	41.433	18.892
52.497	65.515	50.526	24.269
52.497	65.515	50.526	24.299
59.962	68.860	52.400	30.823
59.962	68.860	52.400	30.858
63.708	74.638	64.312	32.405
63.708	74.638	64.312	32.437

TABLE 4.2: Spectrum of the vector field smoothing operators on the *Fertility* model.

4.6.1.2 Curved surface

In Table 4.2 we list the first 20 eigenvalues in the spectrum of the *Fertility* model depicted in the top left corner of Figure 4.3. *Fertility* is a genus 4 surface and has 8 linearly independent harmonic vector fields (i.e., vector fields with zero divergence and curl). Only the **HodgeLaplacian** on the Whitney basis capture the harmonics. **CurlAndDiv** on the Gradient basis produces a shifted spectrum compared to **HodgeLaplacian**, and provides perfect pairing of eigen-vector fields (related by an orthogonal rotation on the tangent plane). **VectorDifferences** on the Triangle basis does not seem to correlate with the other methods and does not define a positive system in the cases of circumcentric weights.

Figure 4.3 shows the visualization of the harmonics vector fields of the *Fertility* model, computed from the spectrum of the **HodgeLaplacian** on the Whitney basis. This visualization is computed using the Line Integral Convolution technique introduced in Section 5.9.4.

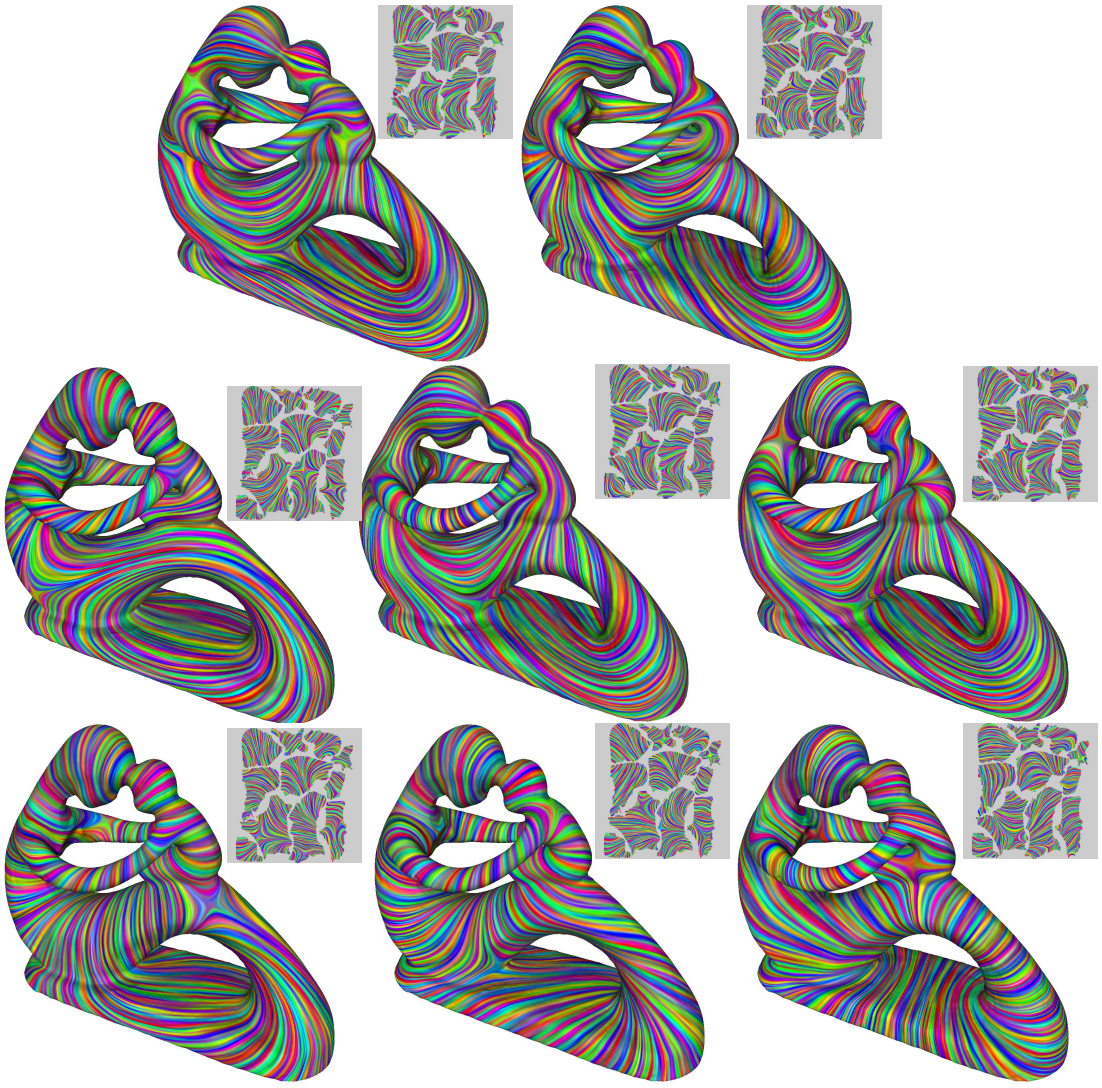


FIGURE 4.3: Eigen-vector fields of the HodgeLaplacian on the *Fertility* model.

4.6.2 Optical flow quality

We compare the quality of the **MeshOpticalFlow** algorithm using **VectorDifferences** (barycentric), **CurlAndDiv** and **HodgeLaplacian** for vector field smoothing.

Given a pair of signals ϕ_0, ϕ_1 and alignment vector field X, we define the interpolated signal at time t as,

$$\textbf{Interpolation}(\phi_0, \phi_1, X, t) := (1-t)\textbf{Advect}(\phi_0, -X, t) + t\textbf{Advect}(\phi_1, X, 1-t) \quad (4.26)$$

and the alignment error at time t as,

$$\textbf{AlignmentError}(\phi_0, \phi_1, X, t) := |\textbf{Advect}(\phi_0, -X, t) - \textbf{Advect}(\phi_1, X, 1-t)|^2 \quad (4.27)$$

For evaluation purposes we are particularly interested in the interpolated signal and alignment error at time $t = \frac{1}{2}$. These will be referred as the halfway alignment signal, and halfway alignment error, respectively.

In Figure 4.4 we compare optical flow for signals over the *Torus* model using a mesh with uniform longitude-latitude triangulation. In Figure 4.5 we compare optical flow for textures over the *Breakers* model, which has non uniform triangulation.

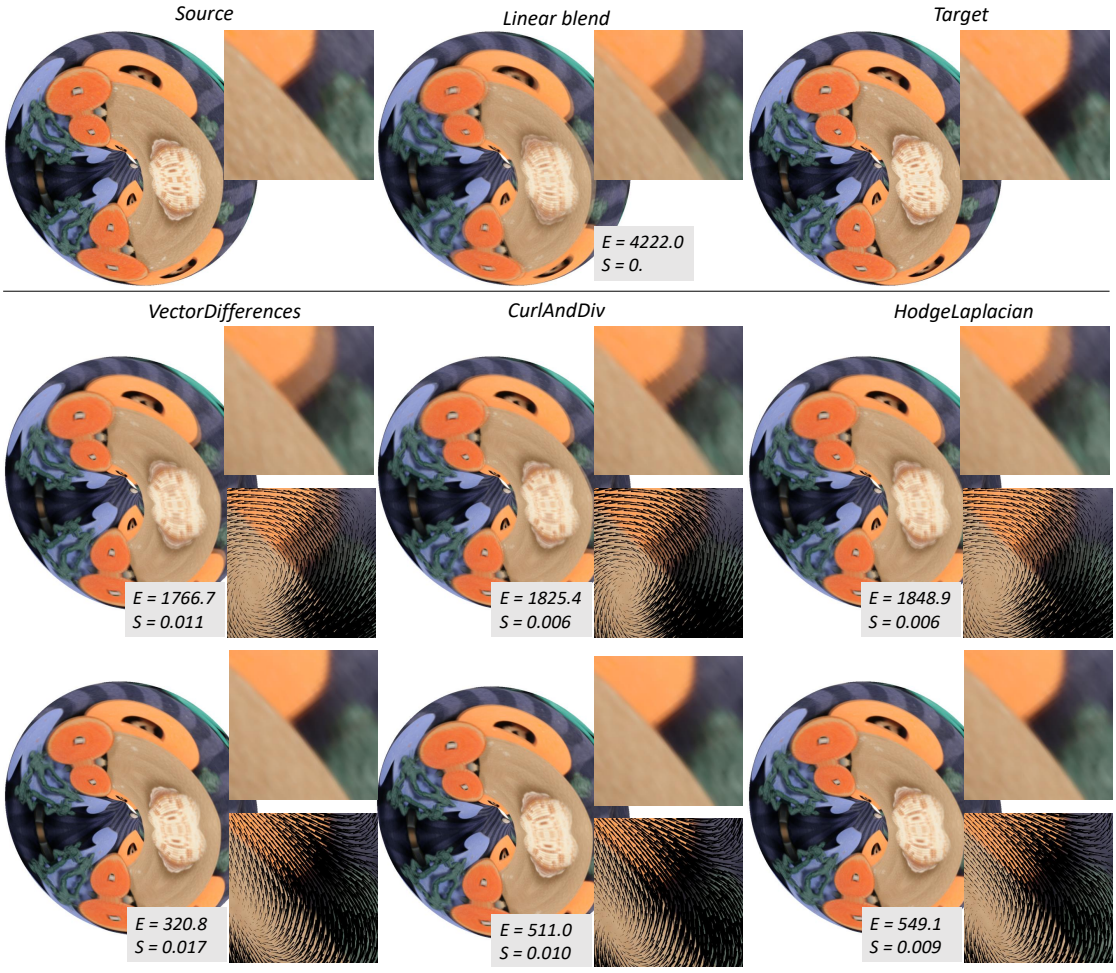


FIGURE 4.4: Comparison of vector field smoothing operators on the *Torus* model.

In the top part of Figures 4.4 and 4.5, we show the input of our algorithm: the source signal (left), target signal (right) and the linear blend[2](middle) to highlight the misalignment. In the bottom part of these figures we visualize the output of our algorithm through the halfway alignment signal after one (first row) and seven hierarchical iterations (second row) using the **VectorDifferences** (left), **CurlAndDiv** (middle) and **HodgeLaplacian** (right) operators. Additionally, we report the halfway alignment error (E) and vector field smoothness (S) for each configuration, and visualize the halfway alignment vector field in the zoom-in.

[2]The linear blend is the halfway alignment signal for $X = 0$.

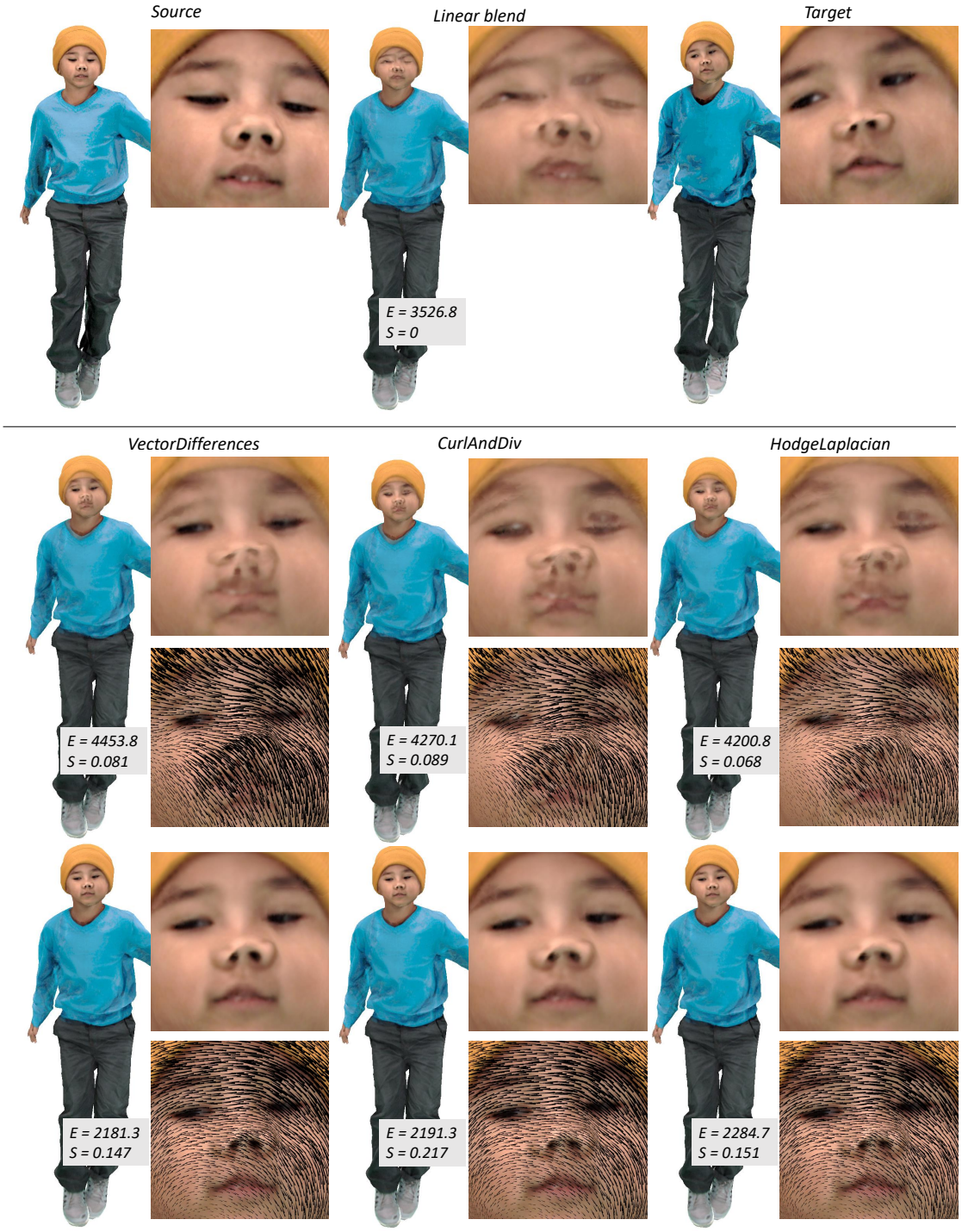


FIGURE 4.5: Comparison of vector field smoothing operators on the *Breakers* model.

The results in Figures 4.4 and 4.5 were generated using manually selected regularization weights : $\epsilon_T = 10^4$, $\epsilon_C = 5 \times 10^{-7}$, and $\epsilon_W = 3 \times 10^{-6}$ for the Triangle, Gradient and Whitney basis respectively. Our results show the effectiveness of all the three vector field representation to model the halfway alignment transformation, but are inconclusive about the qualitative superiority of one representation over the other. After seven iteration both the halfway alignment signal and the halfway alignment vector field generated

by the three methods are indistinguishable.

It's interesting to observe that after one iteration of optical flow on the *Breakers* model (Fig. 4.5) the halfway alignment error (E) is larger than the original alignment error given by the linear blend. Though this is the error we want to minimize, the optical flow algorithm does not guarantee a monotonic decrease of this energy: the alignment error depends non-linearly on the vector field coefficients and what we solve for at each iteration is only a linear approximation. We also notice that as we increase the number of iterations the smoothness of the vector field decreases. This is a consequence of our scale-space coarse-to-fine approach.

4.7 Performance

4.7.1 Flow correction

In Table 4.3 we report the running time (in seconds) of the major steps involved in the computation of the correction flow field. These tests were run on an Intel i7-5700HQ with 4 cores and 16GB of memory.

Smooth and **EstimateFlow** dominate the running time because they construct and solve global linear systems. **Smooth** solves a common linear system for both the source and target signals, and **EstimateFlow** solves a linear system for the correction flow field. Every call to these routines require recomputing a global matrix and updating the numerical factorization of their Cholesky decomposition. Since the **Smooth** routine constructs and solves a smaller linear system[3], the running time is shorter than for **EstimateFlow**. In Table 4.3, the reported running time of **EstimateFlow** is given for the Whitney discretization. A detailed analysis of **EstimateFlow** for the different vector field discretizations is presented in Section 4.7.2.

Model	$\|V\|$	Smooth	Advect	EstimateFlow	GetScale
Reporter (Fig 4.1)	56k	0.69	0.042	0.84	0.003
Torus (Fig 4.4)	108k	1.34	0.081	1.68	0.006
Breakers (Fig 4.5)	56k	0.67	0.068	0.85	0.003
Slick (Fig 4.6)	64k	0.78	0.074	0.96	0.004

TABLE 4.3: Decomposition of the run time of a flow field correction pass.

The **Advect** routine computes the halfway aligned signals by integrating the flow field and sampling from the input signals. These tasks are executed concurrently on all

[3]Non-zero entries of the screened Poisson equation is $7|V|$ on regular meshes.

vertices of the mesh. Finally, the **GetScale** method involves a few matrix-vector multiplications that are also executed concurrently.

We observe a linear scaling for the running time of the **Smooth**, **EstimateFlow**, and **GetScale** routines respect to the size of the model. Instead, the **Advect** routine is data dependent and its running time varies according to the magnitude of the vector field.

4.7.2 Estimate flow

We analyze the computational efficiency of each vector field representation in solving for the correction vector field at each iteration of **EstimateFlow**. Table 4.4 reports the size of the linear system (number of non-zero coefficients) as a function of the number of vertices in the input model[4].

Method	Triangle basis	Gradient basis	Whitney Basis
System DoF	$4\|V\|$	$2\|V\|$	$3\|V\|$
Data sparsity	2	14	5
Smoothness sparsity	8	≥ 19	≥ 11
System sparsity	8	≥ 26	≥ 11
System size	$32\|V\|$	$\geq 52\|V\|$	$\geq 33\|V\|$

TABLE 4.4: System sparsity on a regular triangulation.

Despite having the smallest number of degrees of freedom, the Gradient basis produces denser systems that are roughly 66% larger compared to the generated with the Triangle and Whitney basis. This behavior was verified on the test models as reported in Table 4.5.

	Smoothness sparsity			System size		
Model / Basis	Tri.	Grad.	Whit.	Tri.	Grad.	Whit.
Reporter (Fig 4.1)	8	21.84	11.94	1.81M	3.27M	2.03M
Torus (Fig 4.4)	8	22.58	12.19	3.47M	6.40M	3.96M
Breakers (Fig 4.5)	8	21.81	11.93	1.79M	3.22M	2.00M
Slick (Fig 4.6)	8	22.01	12.01	2.07M	3.76M	2.33M

TABLE 4.5: Smoothness term sparsity and system size in the test models.

In Table 4.6 we report the execution time (in seconds) of the three major steps involved in the solution of the vector field in the **EstimateFlow** routine: matrix construction, numerical factorization and back substitution within the Eigen-PARDISO library [50].

The fastest construction of the linear system and numerical factorization are provided by the Triangle basis. This is expected since this basis produces the linear system with

[4]The minimum sparsity and system size are computed from a triangulation where the degree of each vertex is 6. For irregular triangulations the size of the system can be $\Omega(|V|^2)$ for the Gradient and Whitney basis, and its still $32|V|$ for the Triangle basis.

Model	Triangle Basis	Gradient basis	Whitney Basis
Reporter (Fig 4.1)	**0.16** : **0.19** : 0.32	0.24 : 0.47 : **0.23**	0.18 : 0.24 : 0.25
Torus (Fig 4.4)	**0.30** : **0.41** : 0.57	0.46 : 1.97 : 0.52	0.31 : 0.58 : **0.48**
Breakers (Fig 4.5)	**0.12** : **0.18** : 0.31	0.23 : 0.47 : **0.23**	0.14 : 0.22 : 0.24
Slick (Fig 4.6)	**0.14** : **0.21** : 0.37	0.29 : 0.56 : **0.25**	0.16 : 0.26 : 0.28

TABLE 4.6: System solution. Report of the matrix construction, numerical factorization and substitution.

the smallest number of non-zeros. On the other hand, the back substitution is fastest on the Gradient basis. This is a surprising result and might be consequence of both the smaller number of degrees of freedom and the solver parallelism.

4.8 Applications

4.8.1 Texture interpolation

A natural application of optical flow is the synthesis of intermediate frames that interpolate the source and target signals. We start by computing the halfway alignment vector as described by the **MeshOpticalFlow** method. Then, we use the **Interpolation** routine introduced in Equation 4.26 to synthesize the intermediate frames for any time t.

In Figure 4.6 we compare interpolation of a pair of textures on the *Slick* model using direct linear blend (i.e., without signal alignment) and using our optical flow approach. The zoom-ins reveal severe ghosting artifacts on the linear blend result. Instead, our optical flow result produces a visually smooth transition between the two signals.

4.8.2 Photometric tracking

In Chapter 6 we describe our approach to construct a spatiotemporally coherent parameterization of a surface in motion. One of the main challenges in this process is deforming a template mesh to match the surface at each time step. In order to deform the template mesh to match the target surface we need to identify corresponding points. This is traditionally done by assigning each point in the template to its closest point in the target with consistent orientation [52]. However, geometric correspondences alone introduce drifting when tracking the surface for prolonged time periods, this means, that the vertices on the template move tangentially on the surface. This situation can be observed in the top row of Figure 4.7, where averaging the texture of three consecutive frames of the *Ballerina* produces a blurred result. In this example the motion of the

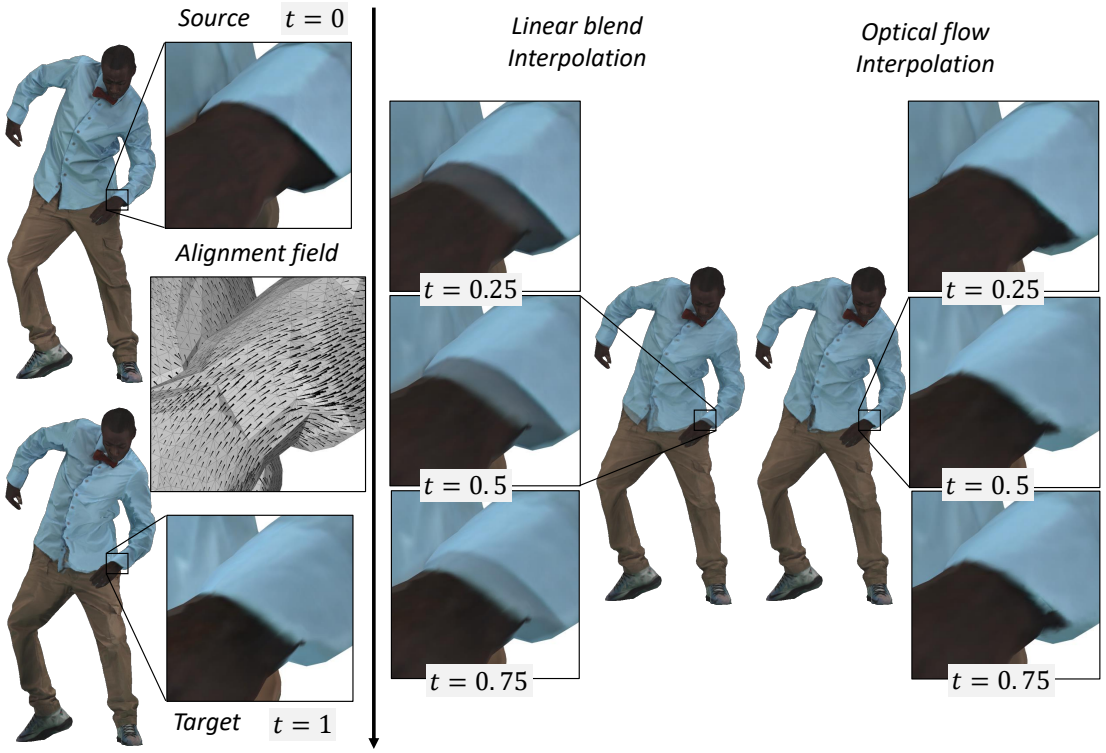


FIGURE 4.6: Comparison of texture interpolation on the *Slick* model.

Ballerina around its rotational symmetry axis makes the closest point method to provide poor correspondences: points on the dress and the face are matched to the current position rather than the rotated one.

To attenuate drifting we compute color correspondences using **MeshOpticalFlow**. Our tracking algorithm proceeds as follows:

1. Geometry registration: Given the template mesh, M_s, and target mesh, M_t, we use iterative closest point correspondences to compute an initial deformation where the template embedding matches the target embedding. We denote by $\Phi_G : M_s \to M_t$ the closest point map from the template to the target mesh.

2. Texture sampling: We project the texture of the deformed template onto the target mesh. We denote by I_s the projected template texture, and I_t the target texture.

3. Texture alignment: We run **MeshOpticalFlow** to compute a vector field X that aligns the template texture I_s to the target texture I_t.

4. Color correspondences: We compute color correspondences,$\Phi_C : M_s \to M_t$, by transporting the closest point correspondences along the alignment vector field X, $\Phi_C(p) := \textbf{IntegrateFlow}(X, 1, \Phi_G(p))$.

5. Drift correction: We use the color correspondences to update the template deformation.

The bottom row of Figure 4.7 shows the result of tracking using color correspondences. For this example we run two cycles of steps 2-5 to improve the quality of the registration. Despite the fast motion of the *Ballerina* our method successfully attenuates drifting artifacts. Having a correct registration is important for texture video compression and temporal filtering. In this example, the texture video generated by enabling optical flow improved MP4 compression by 10% over the result with optical flow disabled.

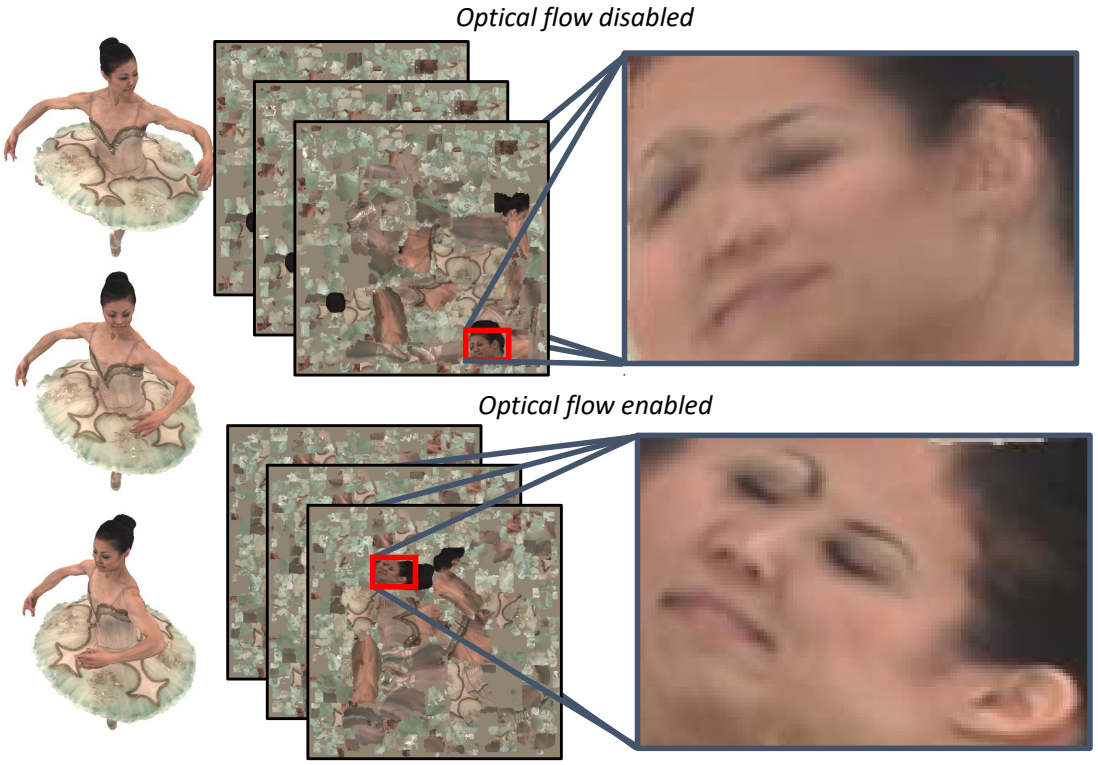


FIGURE 4.7: Correction of drift artifacts on mesh tracking using mesh-based optical flow.

Chapter 5

Gradient-Domain Processing

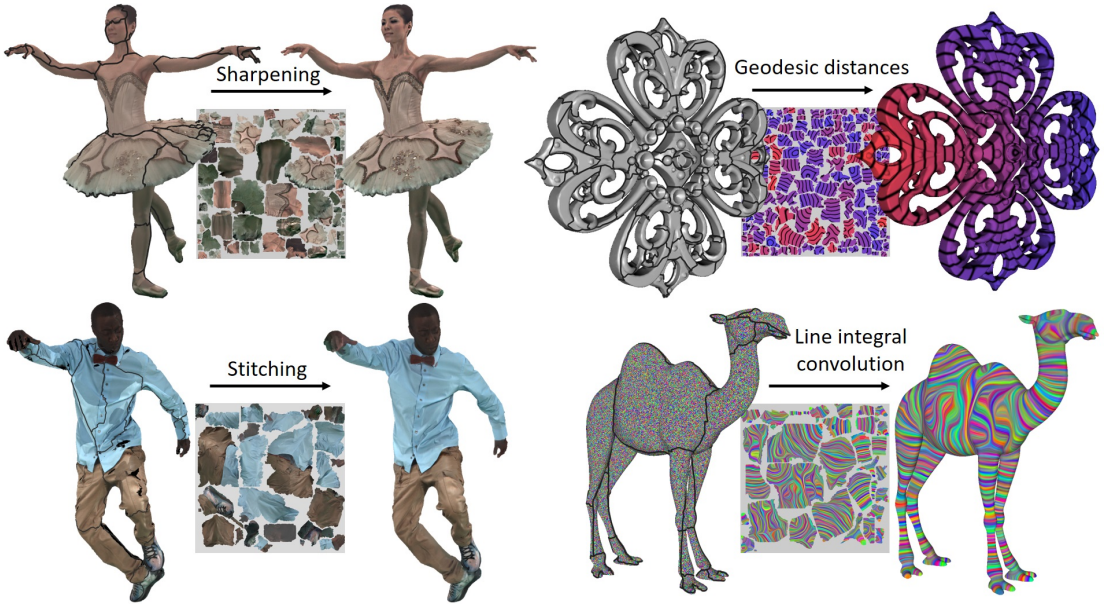


FIGURE 5.1: Applications of gradient-domain processing in texture sharpening, stitching, geodesics computation, and line integral convolution.

In this chapter we introduce a framework to do gradient-domain processing of signals sampled on a texture atlas. Our contribution is a method that produces results that are consistent with the surface metric and are seamless across chart boundaries.

We demonstrate the versatility of our approach through texture editing applications like texture sharpening, smoothing, and stitching. We prove its robustness by solving challenging geometric problems like computing geodesics distance to a source point and performing line integral convolution.

5.1 Introduction

The work developed in this chapter is at the intersection of two well-studied subjects in computer graphics: gradient-domain processing and seamless texture representation. In this section we describe how previous works in both subjects motivate and relate to our method.

5.1.1 Gradient-domain processing

The objective of gradient-domain processing is to solve for a signal ϕ that balances between matching its values to a prescribed signal ψ, and matching its gradients to a prescribed vector field X. More formally, our output signal ϕ is the solution to the screened-Poisson equation,

$$E(\phi; \psi, X, \alpha, g) = \|\phi - \psi\|_g^2 + \alpha \|\nabla\phi - X\|_g^2 \tag{5.1}$$

where α trades between fidelity to the input signal ψ and target vector field X, and g is a Riemannian metric on the parameterization domain.

We have already introduced two variations of this equation. In Equation 3.8, we set $X = 0$ to defined our metric-aware smoothing operator on meshes. In Equation 3.9 we set $X = \nabla p_0 - \langle \nabla p_0, N \rangle N$ to solve for vertex positions that match a target normal field.

Applications of gradient domain processing in images and geometry have been broadly explored. Applications in image processing include smoothing and sharpening [18], dynamic range compression [53, 54], and image stitching [55–57]. Applications in geometry processing include surface fairing [58, 59], deformation [60], detail transfer [32], and parameterization [61, 62].

In this chapter we describe a discretization of Equation 5.1 in the texture atlas. As we will see, this has two major challenges: capturing the metric properties of the surface in the texture domain and supporting chart discontinuities.

5.1.2 Seamless texture representation

A major limitation of texture mapping is the introduction of discontinuities across chart boundaries. This is an inevitable artifact of the traditional rendering pipeline, since values on opposite side of a boundary edge are reconstructed as linear combinations of unrelated texels.

In practice, three different approaches have been suggested to address inter-chart discontinuity: (1) modifying the rendering pipeline [63, 64], (2) optimizing the chart parameterization [65–67], and (3) compute values at boundaries texels to produce a perceptually seamless transition [68, 69]. Our method belongs to the third class: we solve for a signal on a fixed texture atlas that is sampled on the surface using the standard bilinear interpolation provided by the graphics hardware.

Our work is similar to González and Patow [63] which defines a function space that is continuous across chart boundaries. The authors zipper seams by adding a thin fillet of triangles over which standard bilinear sampling is replaced with linear sampling. Our approach also introduces a triangulation but we use refinement rather than zippering, allowing us to represent the signal using all active texels (including those immediately outside the chart). In contrast to [63], our intermediate triangulation is created to assist signal processing and does not redefine the rendering representation.

Carr et al.[65, 66] guarantee exact inter-chart continuity by decomposing the mesh into charts that are parameterized as axis-aligned rectangular patches with matching numbers of texels across boundaries. Our approach does not guarantee exact continuity but can be applied to arbitrary atlas parameterization.

To perform texture synthesis on multi-chart atlases, Lefebvre and Hoppe [68] pad the chart boundaries with pointers to "communicate" to texels on the opposite side of a seam. We also define some notion of adjacency between boundary texels, but we do it in the more global context of the Finite Elements Method.

The work of Liu et al. [69] defines an inter-chart continuity energy to generate seamless textures. This inter-chart continuity energy produces visible smearing artifacts when the signal has a large gradient parallel to a chart boundary, as discussed in Section 5.5.1. Our approach enforces continuity by construction (rather than by penalization) and it is free of the aforementioned artifacts.

5.2 Gradient-domain fundamentals

5.2.1 Screened-Poisson equation

The optimal solution to equation 5.1 can be derived through variational analysis. Expanding the screened-Poisson equation at $\phi + \zeta$ and rearranging terms we obtain

$$E(\phi+\zeta;\psi,X,\alpha) = \int_M ((\phi+\zeta)-\psi)^2 + \alpha\langle\nabla(\phi+\zeta)-X, \nabla(\phi+\zeta)-X\rangle dA \quad (5.2)$$

$$= E(\phi;\psi,X,\alpha) + 2\int_M (\phi-\psi)\zeta + \alpha\langle\nabla\phi - X, \nabla\zeta\rangle dA + \underbrace{\int_M \zeta^2 + \alpha\langle\nabla\zeta,\nabla\zeta\rangle dA}_{\geq 0} \quad (5.3)$$

Since the third term of equation 5.3 is non negative, we conclude that a sufficient condition for ϕ to be the energy minimizer is to satisfy

$$\int_M (\phi-\psi)\zeta + \alpha\langle\nabla\phi - X, \nabla\zeta\rangle dA = 0 \quad (5.4)$$

for any signal ζ. From the Divergence Theorem[1] it follows that

$$\int_M \langle\nabla\phi - X, \nabla\zeta\rangle dA = \int_M (-\Delta\phi - \nabla\cdot X)\zeta dA$$

Thus, the optimal condition on ϕ can be rewritten as,

$$\int_M \Big((1-\alpha\Delta)\phi - \psi - \alpha\nabla\cdot X\Big)\zeta dA = 0 \quad (5.5)$$

for any signal ζ. This is the case when $(1-\alpha\Delta)\phi - \psi - \alpha\nabla\cdot X \equiv 0$, or equivalently,

$$\phi = (1-\alpha\Delta)^{-1}(\psi - \alpha\nabla\cdot X) \quad (5.6)$$

5.2.2 Spectral analysis

Equation 5.6 allows us to compute the analytic solution to the screened-Poisson problem in terms of the spectrum of the Laplace operator Δ. Letting $\{(\lambda_k,\phi_k)\}_k$ be the set of eigen-value and eigen-vectors of $-\Delta$, and expressing $\psi - \alpha\nabla\cdot X \equiv \sum_k r_k\phi_k$, then we can write the solution to the screened-Poisson problem as,

$$\phi = \sum_k \frac{r_k}{1+\alpha\lambda_k}\phi_k \quad (5.7)$$

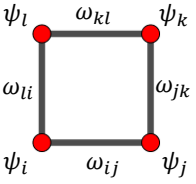
[1] Assuming free boundary conditions, $\langle\nabla\phi, n\rangle = 0$ on ∂M.

Bhat et al. [18] studied the particular case where $X = \beta\nabla\psi$. For this particular case $\psi - \alpha\nabla \cdot X = (1 - \alpha\beta\Delta)\psi$. Letting $\psi \equiv \sum_k s_k\phi_k$ be the spectral decomposition of the input signal, the solution can be expressed as,

$$\phi = \sum_k \frac{(1 + \alpha\beta\lambda_k)}{1 + \alpha\lambda_k} s_k\phi_k \tag{5.8}$$

When $\beta < 1$ all frequencies are attenuated and the the output signal ϕ is a smoothed version of the input ψ. When $\beta > 1$ all frequencies are amplified and the output is a sharper signal.

5.2.3 Images discretization



For images, the discretization of equation 5.1 is traditionally done using staggered grids. As show in the inset, signal values ψ and ϕ are stored at the nodes of the grid (N), and differentials at its edges (E). Due to the homogeneity of the metric, the problem can be simply formulated as solving for $\boldsymbol{\phi}$ minimizing

$$\sum_{i \in N} (\psi_j - \phi_i)^2 + \alpha \sum_{ij \in E} (\omega_{ij} - (\phi_j - \phi_i))^2 \tag{5.9}$$

The regularity of the image domain has motivated the use of multigrid methods to solve equation 5.9. These methods alternate between updating the solution at different resolutions. At each resolution the solution update is done using efficient relaxation techniques that exploit parallelism and memory coherence. In Section 5.8 we show the extension of these techniques to multi-chart atlases.

5.2.4 Mesh discretization

To compute the discretization of 5.1 on meshes we follow the Finite Elements approach (see Section 2.3.1.4) associating a hat basis function to each mesh vertex. Assuming the target vector field X is constant within each triangle, we can write energy 5.1 as,

$$(\boldsymbol{\psi} - \boldsymbol{\phi})^\top \mathbf{M} (\boldsymbol{\psi} - \boldsymbol{\phi}) + \alpha (\boldsymbol{D_0}\boldsymbol{\phi} - \boldsymbol{X})^\top \mathbf{M}_1^T (\boldsymbol{D_0}\boldsymbol{\phi} - \boldsymbol{X}) \tag{5.10}$$

where $\boldsymbol{D_0}$ is the discrete gradient operator (introduced in 4.5.3), and $\mathbf{M}_1^T$ is the vector field mass matrix (introduced in 4.6.1). Since, $\mathbf{S} = \boldsymbol{D_0}^\top \mathbf{M}_1^T \boldsymbol{D_0}$, the optimal value to

5.10 is given by,

$$\boldsymbol{\phi} = (\mathbf{M} + \alpha\mathbf{S})^{-1}(\mathbf{M}\boldsymbol{\psi} + \alpha\boldsymbol{D}_{\mathbf{0}}^{\top}\mathbf{M}_1^T\boldsymbol{X}) \tag{5.11}$$

Due to the irregular connectivity and non homogeneous sampling of triangle meshes, the extensions of multigrid methods to meshes have been limited. The construction of multiresolution meshes and the efficient relaxation at each resolution are challenging tasks. Instead, general methods like Cholesky factorization are commonly used to solve these linear systems.

5.2.5 Anisotropy

While most applications of gradient-domain processing use the immersion metric of the surface, alternate metrics can be incorporated in the formulation. The seminal work by Perona and Malik [15] demonstrates the power of incorporating anisotropy in the context of edge-aware image processing. In geometry processing, anisotropic filtering has also been used for feature-preserving smoothing [20, 22, 70, 71].

In Section 5.9.4 we present an application of anisotropic gradient-domain processing for line integral convolution, which enable visualization of vector fields on surfaces.

5.3 Overview

In the following sections we introduce our technique for performing gradient-domain processing directly in the texture atlas. We combine properties of both the image-domain discretization and the mesh-domain discretizations to produce a method that is metric-aware, continuous across chart boundaries and computationally efficient.

Discretizing and solving equation 5.1 in a texture atlas poses several challenges:

- Using standard bilinear interpolation, texture maps represent functions that do not (in general) align across chart boundaries. As a result, continuity can only be enforced by constraining the texture signal to have constant value along the seams.
- Evaluating the texture near chart boundaries requires the use of both *interior* and *exterior* texels. Because exterior texels are not associated with positions on the surface, defining discrete derivatives across chart boundaries is non-trivial.
- Although texels lie on a uniform grid, their corresponding locations on the surface are distorted by the parameterization. The nonuniform metric must be taken into account.

To address these challenges, we use an *intermediate* representation involving *continuous* basis functions that approximate the bilinear basis. Specifically, we introduce:

- A novel function space spanned by basis functions that reproduce the bilinear reconstruction kernel in the interior of a chart and are continuous across chart boundaries.
- A basis for cotangent vector fields to represent the target texture differential.
- Metric-aware Hodge stars for constructing the mass and stiffness matrices in the discretization of Equation (5.1) over texels.

In effect, we form a linear system over the texel values of an ordinary texture atlas, but using system matrix coefficients derived from an approximating continuous function space.

To efficiently solve this system, we present a novel multigrid algorithm that exploits grid regularity within chart interiors while correctly handling irregularity across chart boundaries.

Our work does not address seamless texturing. Because the output representation, like the input, is a general texture atlas evaluated using bilinear hardware rasterization, continuity can only be attained by blurring the signal along chart boundaries. However, we find that formulating signal processing operations using an intermediate continuous representation yields results in which chart seams are usually imperceptible. Our strategy is more effective than introducing inter-chart continuity constraints (Section 5.5.1).

We demonstrate the effectiveness of our approach in applications including signal smoothing and sharpening, texture stitching, geodesic distance computation, and line integral convolution.

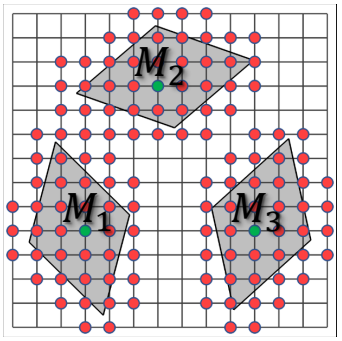
5.4 Preliminaries

The input to our algorithm is an atlas parameterization of a 2-manifold immersed in 3D. It consists of a triangle mesh (V, T) residing in the unit-square, an equivalence relation $\sim$ on V indicating if two boundary vertices correspond to the same point on the manifold, and a map $\pi : V \to \mathbb{R}^3$ giving the immersion.

5.4.1 Texture atlas

The mesh atlas induces a partition of triangles into connected components, each defining a chart domain $M_i \subset [0,1] \times [0,1]$ formed by the union of its triangles. We let $M = \bigcup M_i$ denote the parameterization domain. We extend the map $\pi : V \to \mathbb{R}^3$ to the map $\pi : M \to \mathbb{R}^3$ by linear interpolation within triangles. We extend the equivalence relation $\sim$ to M by linear interpolation along boundary edges, setting $p \sim q$ if there exists boundary edges (v_1, v_2) and (w_1, w_2) and interpolation weight $\alpha \in [0,1]$ such that $v_1 \sim w_1$, $v_2 \sim w_2$, $p = (1-\alpha)v_1 + \alpha v_2$, and $q = (1-\alpha)w_1 + \alpha w_2$.

We say that points $p, q \in M$ are *on opposite sides of a seam* if $p \sim q$ and that a function $\phi : M \to \mathbb{R}$ is *seam-continuous* if it is continuous on M and has the same values on opposite sides of a seam.



Given a $W \times H$ texture image, we partition the unit square into $W \times H$ cells and compute the dual graph (shown in black in the inset). As our goal is to define a function space which mimics the bilinear functions, we define the *footprint* of a node to be the four incident quads (the support of the bilinear kernel centered at the node). We define a *texel* to be any node whose footprint overlaps M and denote the set of texels by $\mathcal{T}$. We assume that the footprint intersects exactly one M_i and say a texel is *interior* if its footprint is contained within a chart (green nodes) and *boundary* otherwise (red nodes).[2]

5.4.2 Metric

As described in **??**, to integrate functions over the triangulation, we require a Riemannian metric g on M. In the context of gradient domain processing, the metric needs only be integrable. Therefore we restrict ourselves to the set of piecewise-constant metrics. That is, given the canonical coordinate frame on the unit square containing M, and given a triangle $t \in T$, we consider metrics for which the matrix expression of g is the same for all $p \in t$.

From the parameterization $\pi : M \to \mathbb{R}^3$, we construct an *immersion metric* μ that captures the intrinsic geometry of the surface. This metric is defined through the inner product:

$$\langle X, Y \rangle_\mu := (d\pi X)^\top (d\pi Y) = X^\top (d\pi^\top d\pi) Y, \qquad \forall X, Y \in TM\,.$$

[2]Charts can always be translated by different integer offsets to ensure that the footprint of a texel intersects exactly one chart.

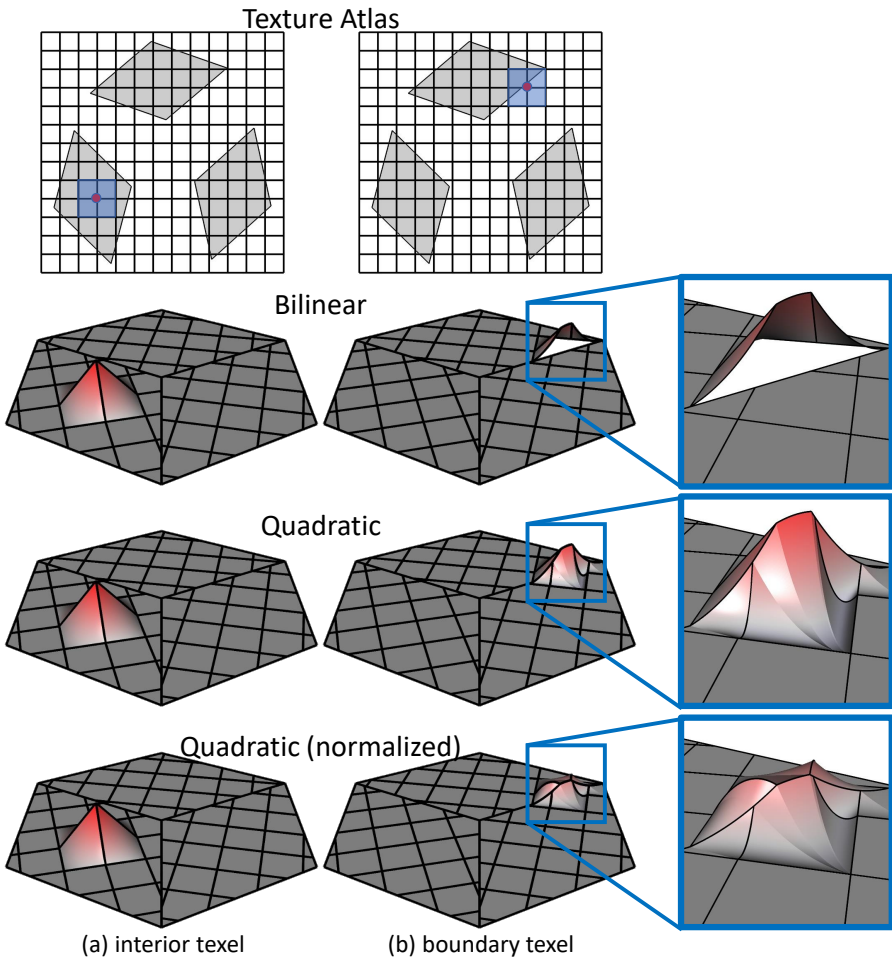


FIGURE 5.2: Visualization of interior and boundary basis functions of the Bilinear, Quadratic, and Normalized Quadratic function spaces.

For most of our applications we assume $g = \mu$. In Section 5.9.4 we construct a new metric to stretch and shrink distances according to a vector field.

5.5 Functional basis

Our goal is to associate a basis function to each texel $t \in \mathcal{T}$ so that a set of discrete texture values can be interpreted as a function that can be evaluated anywhere on M.

Perhaps the simplest approach is to associate texel $t \in \mathcal{T}$ with the bivariate, first-order B-spline B_t centered at t. This conforms to the bilinear rasterization performed by graphics hardware. While such functions are well-behaved for interior texels, they are not seam-continuous for boundary texels, dropping to zero on the opposite side of the seam (Figure 5.2, second row).

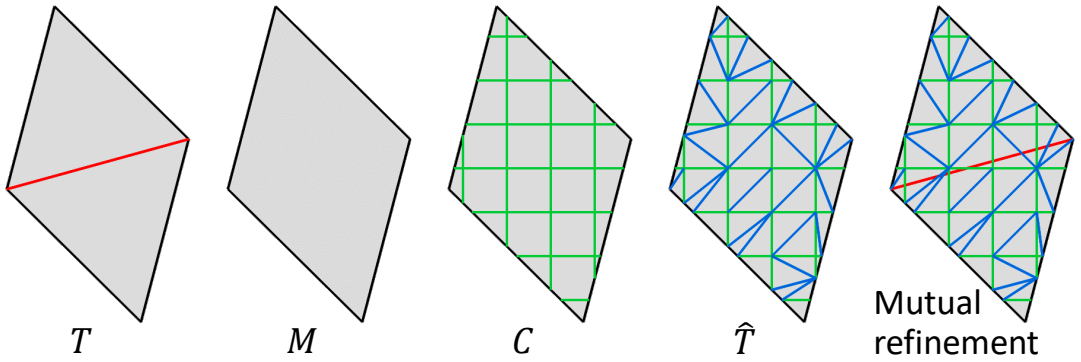


FIGURE 5.3: Notation for the introduced triangulations and polygonizations of the texture domain.

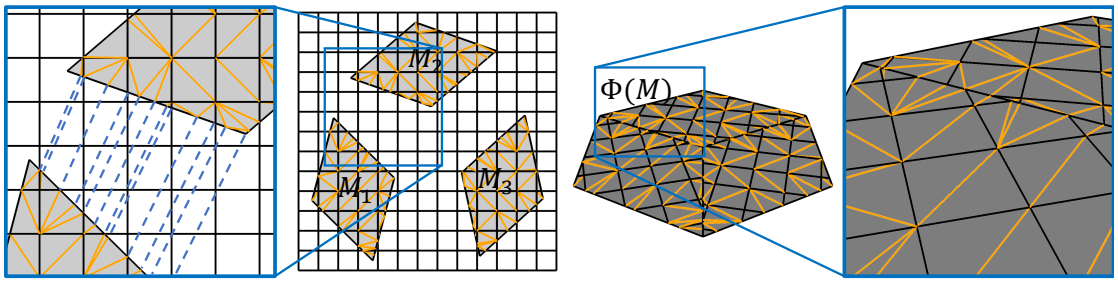


FIGURE 5.4: Boundary cells are tessellated by transferring intersections from opposite sides of the seam (left). The result is a triangulation of the surface free of T-junctions (right).

Instead, we define a basis $\mathscr{B} = \{\phi_t\}_{t \in \mathcal{T}}$ consisting of seam-continuous functions that *approximate* the bilinear kernels $\{B_t\}$. Since the bilinear kernels are piecewise-quadratic polynomials, we define the basis $\mathscr{B}$ to be piecewise-quadratic as well.

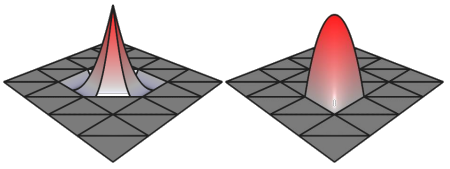
We proceed in three steps: (1) computing a new triangulation $\widehat{T}$ of the texture domain; (2) using $\widehat{T}$ to define a seam-continuous basis of piecewise-quadratic functions $\{\widetilde{Q}_{\tilde{n}}\}$ on M; (3) defining bilinear-like seam-continuous basis $\mathscr{B}$ as linear combinations of $\{\widetilde{Q}_{\tilde{n}}\}$.

(1) Triangulating the texture domain

We decompose the atlas domain M into a set of polygonal cells C by tessellating M using the texel lattice (Figure 5.3). For each vertex introduced along a seam, we insert a corresponding vertex on the opposite side of the seam (shown as dashed lines in Figure 5.4). Then, we compute a constrained Delaunay triangulation $\widehat{T}$ of these polygons.

(2) Defining a quadratic seam-continuous function basis

We associate a quadratic Lagrange basis function to each vertex and each edge in the triangulation $\widehat{T}$



[72]. These functions form a partition of unity, reproduce continuous piecewise quadratic polynomials, and are interpolatory, i.e. a function centered at a node evaluates to 1 at that node and to 0 at all other nodes. (The inset shows elements centered on a vertex and edge of a triangle mesh.) We denote the set of nodes (vertices and edges) by $\mathcal{N}$ and the basis as $\{Q_n\}_{n\in\mathcal{N}}$.

To obtain a seam-continuous function-space, we merge the $\{Q_n\}$ across seams into a single function. Specifically, let $\widetilde{\mathcal{N}} = \mathcal{N}/\sim$ be the set of equivalence classes in $\mathcal{N}$ modulo seam-equivalence. (We implicitly treat a node $n \in \mathcal{N}$ as a point on M, using the vertex position if n is a vertex and the midpoint if n is an edge.) We associate a seam-continuous function $\widetilde{Q}_{\tilde{n}}$ to each equivalence class $\tilde{n} \in \widetilde{\mathcal{N}}$ by summing the quadratic Lagrange elements associated to nodes in the equivalence class:

$$\widetilde{Q}_{\tilde{n}} = \sum_{n\in\tilde{n}} Q_n \,.$$

These functions also form a partition of unity, reproduce seam-continuous piecewise quadratic polynomials, and are interpolatory.

(3) Defining a bilinear-like seam-continuous basis of texel functions

Given a texel $t \in \mathcal{T}$, we define the function $\overline{\phi}_t : M \to \mathbb{R}$ to be the linear combination of $\{\widetilde{Q}_{\tilde{n}}\}$, with coefficients given by evaluating the bilinear function B_t at the node positions:

$$\overline{\phi}_t(p) \equiv \sum_{\tilde{n}\in\widetilde{\mathcal{N}}} \left(\sum_{n\in\tilde{n}} B_t(n) \right) \cdot \widetilde{Q}_{\tilde{n}}(p) \,.$$

By construction, the $\{\overline{\phi}_t\}$ are seam-continuous since they are the linear combinations of seam-continuous functions. Furthermore, due to the interpolatory property of the Lagrange elements, the function $\overline{\phi}_t$ reproduces the bilinear function B_t whenever t is an interior texel (Figure 5.2a). Generally, the functions $\overline{\phi}_t$ and B_t agree on the intersection of M with the footprint of t.[3] (Compare Figure 5.2b, second and third rows.)

The limitation of using the functions $\{\overline{\phi}_t\}$ is that they do not form a partition of unity. To address this, we normalize the coefficients by the number of seams on which the node is located:

$$\phi_t(p) \equiv \sum_{\tilde{n}\in\widetilde{\mathcal{N}}} \left(\frac{1}{|\tilde{n}|} \sum_{n\in\tilde{n}} B_t(n) \right) \cdot \widetilde{Q}_{\tilde{n}}(p) \,,$$

[3] A rare exception is if the footprint of a texel contains nodes that are on opposite sides of a seam, e.g. at the poles of the sinusoidal projection.

where $|\tilde{n}|$ is the cardinality of the equivalence class $\tilde{n}$.

This still associates a seam-continuous, piecewise quadratic function to each texel and reproduces the bilinear functions at interior texels. However, for a boundary texel t, the functions ϕ_t and B_t no longer agree on the intersection of M with the footprint of t. (See Figure 5.2b, bottom row.)

For a thorough derivations of these results please refer to the appendix of [73].

5.5.1 Comparison with soft continuity constraints

We compare our construction of a continuous function space $\mathscr{B}$ to the approach of Liu et al. [69] which enforces continuity on the traditional bilinear basis by introducing a soft constraint E_C. For a general function ϕ, the energy $\mathrm{E}_C(\phi; g)$ measures the integrated squared difference between the values of ϕ on opposite sides of a seam. We can include this continuity energy into Equation (5.1) as an additional term:

$$\mathrm{E}(\phi; g, \alpha, \psi, X) + \lambda \mathrm{E}_C(\phi; g)\,,$$

where λ modulates the importance of continuity across the seam.

Figure 5.5 shows examples of signal diffusion using two different diffusion scales (Section 5.9.1), comparing the results obtained using the bilinear basis with soft constraints to the results obtained using our continuous basis. Renderings are obtained using the texture mapping hardware, with basis coefficients used as texel values. For large-time-scale diffusion (top), a low continuity weight results in insufficient cohesion between charts, and colors do not diffuse across chart boundaries. For short-time-scale diffusion (bottom), a high continuity weight encourages the function to be constant along the seam, resulting in perceptible color "smearing". Our continuous basis provides correct results for both scenarios and does not require any parameter tuning.

5.6 Vector fields

We use the Whitney basis to represent vector fields. As described in Section 2.3.1.4, each basis element is associated to an unordered pair of adjacent texels and is defined as the symmetric difference of the product of the scalar function at one texel times the differential of the scalar function at the other. Because the function basis $\mathscr{B}$ forms a partition of unity, we obtain a discretization of the exterior derivative, given in terms of finite differences [74].

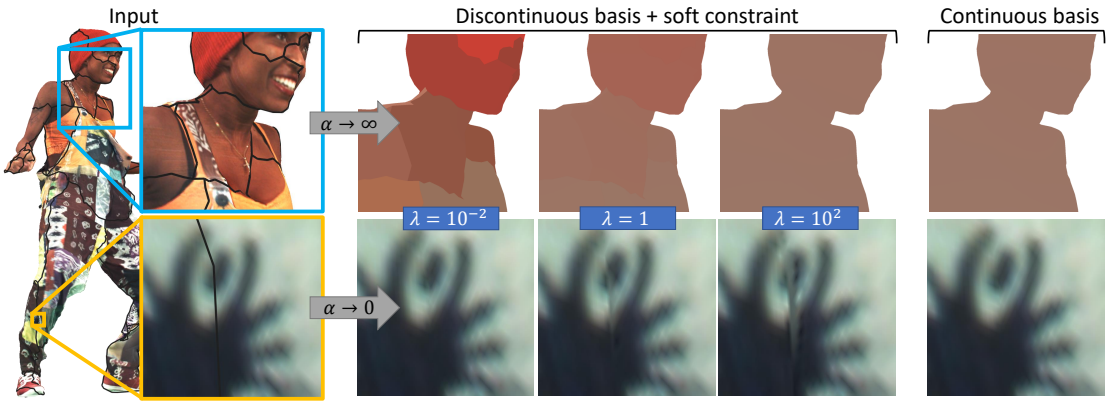


FIGURE 5.5: Comparison of diffusion using the standard bilinear basis with soft constraints and our continuous basis.

Whitney basis Let "$\prec$" be some precedence operator on texels, and let $\mathscr{E}$ denote the set of adjacent texels, i.e. pairs of texels whose basis functions have overlapping support:

$$\mathscr{E} \equiv \left\{ (s,t) \in \mathscr{T} \times \mathscr{T} \,\middle|\, s \prec t \text{ and } \operatorname{supp}(\phi_s) \cap \operatorname{supp}(\phi_t) \neq \emptyset \right\}.$$

Given the scalar function basis $\mathscr{B} = \{\phi_t\}$, the Whitney basis, $\mathscr{B}_1^W = \{X_a\}$, is defined as:

$$X_a = \phi_s \cdot \nabla_g \phi_t - \phi_t \cdot \nabla_g \phi_s, \qquad \forall a = (s,t) \in \mathscr{E}.$$

Discrete exterior derivative We denote by $\mathbf{d} \in \mathbb{R}^{|\mathscr{T}| \times |\mathscr{E}|}$ the matrix giving the signed incidence of texels for adjacent texel pairs:

$$\mathbf{d}_{r(s,t)} = \begin{cases} -1 & \text{if } r = s \\ 1 & \text{if } r = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall r \in \mathscr{T} \text{ and } (s,t) \in \mathscr{E}.$$

We recall that since $\mathscr{B}$ forms a partition of unity, the matrix $\mathbf{d}$ gives the discretization of the exterior derivative in the bases $\mathscr{B}$ and $\mathscr{B}_1^W$. That is, for a given scalar basis function ϕ_t we have:

$$\begin{aligned} \nabla_g \phi_t &= \left(\sum_{s \in \mathscr{T}} \phi_s \right) \nabla_g \phi_t - \phi_t \nabla_g \left(\sum_{s \in \mathscr{T}} \phi_s \right) \\ &= \sum_{s \in \mathscr{T}} (\phi_s \nabla_g \phi_t - \phi_t \nabla_g \phi_s) \\ &= \sum_{a \in \mathscr{E}} \mathbf{d}_{ta} X_a \,. \end{aligned}$$

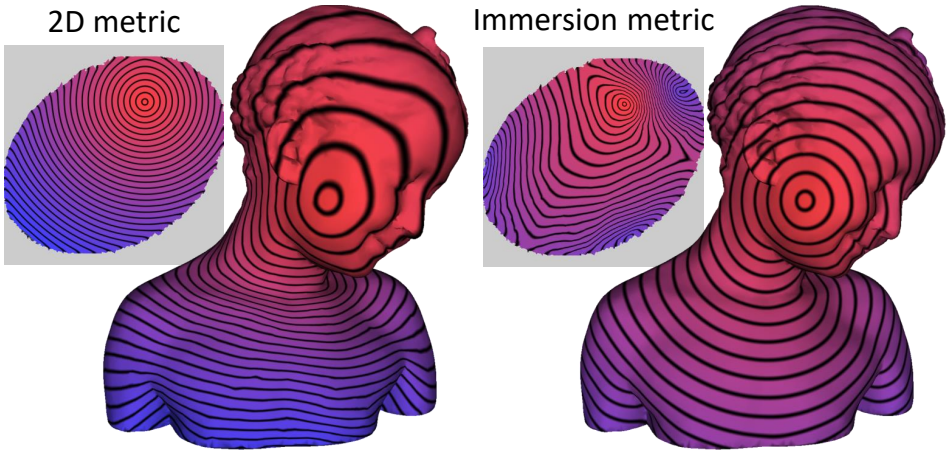


FIGURE 5.6: Comparison of geodesic distance computation using the 2D Euclidean metric and the immersion metric.

5.7 Linear operators

Given a Riemannian metric g we define mass matrices for signals $\mathbf{M} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$ and vector fields $\mathbf{M}_1 \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$ by:

$$(\mathbf{M})_{s,t} = \int_M \phi_s \phi_t \sqrt{|g|}\, dA\,,$$
$$(\mathbf{M}_1)_{a,b} = \int_M \langle X_a, X_b \rangle_g \sqrt{|g|}\, dA\,.$$

We compute the integrals by using the canonical coordinate frame for $[0,1] \times [0,1] \supset M$ and combining the two triangulations described earlier: the parameterized surface triangulation T and the triangulation $\widehat{T}$ obtained by tessellating M using the texel lattice.

By assumption, the matrix expression for g is constant on each triangle in T. By construction, the basis $\mathcal{B}$ and $\mathcal{B}_1^W$ are polynomial on each triangle in $\widehat{T}$. Thus, computing a mutual refinement $T \oplus \widehat{T}$ of the two triangulations (Figure 5.3) and summing the integrals over the faces of $T \oplus \widehat{T}$, the computation of the matrices reduces to integrating polynomials over 2D polygons.

We compute the integrals over each face in the refinement by triangulating the face and using 11-point quadrature [75], which is exact for polynomials up to degree six. (Since ϕ_t is a piece-wise quadratic polynomial and X_a is piece-wise cubic, computing the signal mass matrix $\mathbf{M}$ requires integrating fourth-order polynomials and computing the vector mass matrix $\mathbf{M}_1$ requires integrating sixth-order polynomials.)

5.7.1 Defining the linear system

In our applications, we are interested in computing functions minimizing the quadratic energy $\mathrm{E}(\phi; g, \alpha, \psi, X)$ from Equation (5.1). Using the Euler-Lagrange formulation and discretizing with respect to the function basis, the coefficients of the minimizer $\boldsymbol{\phi} \in \mathbb{R}^{|\mathcal{T}|}$ are given as the solution to the linear system

$$(\mathbf{M} + \alpha\mathbf{S})\,\boldsymbol{\phi} = \mathrm{mass}(\psi) + \alpha\mathrm{div}(X)\,.$$

Here $\mathbf{S}$ is the stiffness matrices, given by[4] $\mathbf{S}_0 \equiv \mathbf{d}^\top\mathbf{M}_1\mathbf{d}$, and $\mathrm{mass}(\psi) \in \mathbb{R}^{|\mathcal{T}|}$ and $\mathrm{div}(X) \in \mathbb{R}^{|\mathcal{T}|}$ are obtained by integrating against the basis functions:

$$\begin{aligned}\mathrm{mass}(\psi)_t &\equiv \int_M \phi_t\psi\sqrt{|g|}dA\,,\\ \mathrm{div}(X)_t &\equiv \int_M \langle\nabla_g\phi_t, X\rangle_g\sqrt{|g|}\;dA\,.\end{aligned}$$

When ψ or X can be expressed as a linear combination of basis functions (e.g. in smoothing and sharpening, stitching, and line integral convolution applications), the constraints simplify:

$$\psi = \sum_{t\in\mathcal{T}} \boldsymbol{\psi}_t\phi_t \quad\Rightarrow\quad \mathrm{mass}(\psi) = \mathbf{M}\boldsymbol{\psi}\,, \tag{5.12}$$

$$X = \nabla_g\left(\sum_{t\in\mathcal{T}} \boldsymbol{\psi}_t\phi_t\right) \quad\Rightarrow\quad \mathrm{div}(X) = \mathbf{S}\boldsymbol{\psi}\,, \tag{5.13}$$

$$X = \sum_{a\in\mathcal{E}} \boldsymbol{z}_a X_a \quad\Rightarrow\quad \mathrm{div}(X) = \mathbf{d}^\top\mathbf{M}_1\boldsymbol{z}\,. \tag{5.14}$$

When ψ or X cannot be expressed as a linear combination of basis functions (e.g. in computing single-source geodesic distances), we approximate the integrals using quadrature.

5.8 Multigrid solver

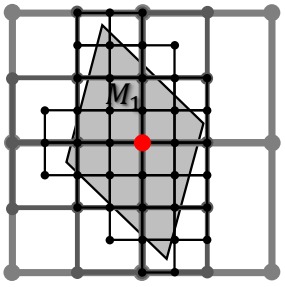
The applications we consider are formulated as solutions to sparse symmetric positive-definite linear systems. On domains with irregular connectivity like triangle meshes, these type of systems are commonly solved either through direct methods, like sparse Cholesky factorization, or through iterative methods, like conjugate gradients. Both

[4] In practice $\mathbf{S}$ is computed directly by integrating the dot products of the differentials of the scalar functions $\{d\phi_t\}$. This is more efficient because $|\mathcal{E}| \approx 4|\mathcal{T}|$ and more stable because the integrands are only second-order polynomials.

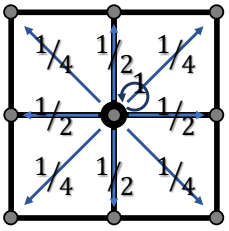
approaches have limitations within an interactive system: Cholesky factorization requires expensive precomputation and the back-substitution is hard to parallelize, while iterative methods like conjugate gradients converge too slowly.

To support interactivity, we implement a multigrid solver that exploits the regularity of the texture domain. The challenge in doing so is handling the irregularity that arises at the seams. We resolve this by using domain-decomposition [76], partitioning the degrees of freedom into *interior*, where we leverage regularity, and *boundary*, where the system is small enough to be handled by a direct solver. We start by describing the implementation of the multigrid solver and then discuss performance.

5.8.1 Hierarchy construction



Our input is a texture grid where charts are separated sufficiently so that the footprint of each texel intersects a single chart. The set of texels in the input grid defines the *finest resolution* of our hierarchy. We construct the coarser levels by generating a multiresolution grid for each chart independently, as shown in the inset. We select a texel in the finest resolution (level 0) as the origin (shown in red in the inset), and define the texels $\mathcal{T}^l$ at the l-th hierarchy level as the subset of finest-level grid nodes with indices $(2^l m, 2^l k)$ whose $[-2^l, 2^l] \times [-2^l, 2^l]$ footprints intersect the chart. Extending the definitions from Section 5.4.1, we classify texels at coarser levels of the hierarchy as *interior* or *boundary* by checking whether their footprint is entirely contained within a chart.



Each texel of the hierarchy indexes a basis function. Texels at the finest resolution are associated with the continuous basis $\{\phi_t\}$ introduced in Section 5.5. We implicitly construct the coarse function spaces using the Galerkin approach, defining a prolongation matrix $\mathbf{P}^l$ that expresses basis functions at coarser level $l+1$ as linear combinations of (at most) 9 basis functions at level l. The coefficients are given by the bilinear up-sampling stencil, (see inset). The restriction matrix is defined as $\mathbf{R}^l \equiv (\mathbf{P}^l)^\top$. Then, given a matrix $\mathbf{A}$ defined at the finest resolution, we recursively construct the restriction of this matrix to the coarser levels of the hierarchy, setting $\mathbf{A}^0 = \mathbf{A}$ and $\mathbf{A}^{l+1} = \mathbf{R}^l \mathbf{A}^l \mathbf{P}^l$.

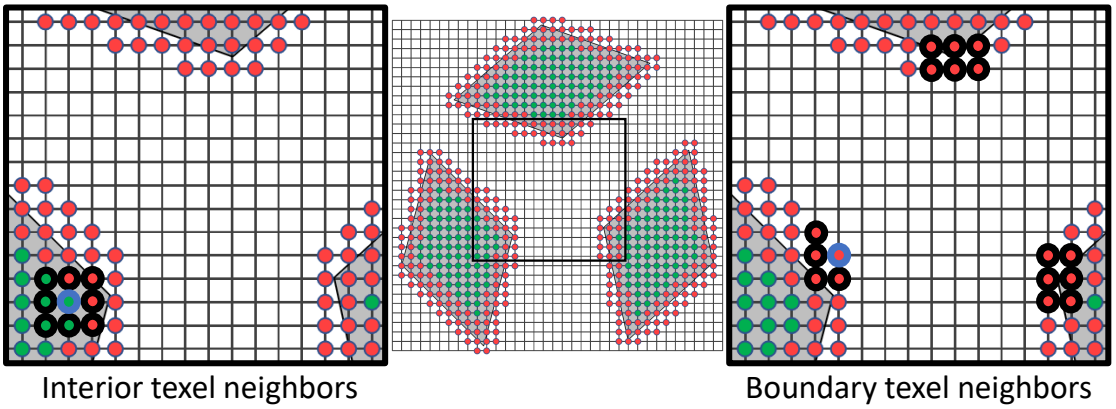


FIGURE 5.7: Visualization of the neighbors of an interior texel and a boundary texel.

5.8.2 Solution update

To solve the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ (with known constraints $\boldsymbol{b}$ and unknown coefficients $\boldsymbol{x}$), we update the estimated solution by performing a V-cycle [77]. Starting at the finest resolution, we recursively relax the solution and restrict the residual to the next coarser level. At the coarsest resolution, we solve the small system using a direct solver. Then, we recursively add the prolonged correction to the estimated solution at the next finer level, and apply further relaxation.

V-cycle$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x})$

R.1	`for` $l = 0, \ldots, L-1$	*restriction phase*
R.2	$\quad \boldsymbol{r}^l_{\mathbf{i}} \leftarrow \boldsymbol{b}^l_{\mathbf{i}} - \mathbf{A}^l_{\mathbf{ib}} \boldsymbol{x}^l_{\mathbf{b}}$	*boundary-relative residual*
R.3	$\quad \boldsymbol{x}^l_{\mathbf{i}} \leftarrow$ **GaussSeidelRelax**$(\ \mathbf{A}^l_{\mathbf{ii}}\ ,\ \boldsymbol{r}^l_{\mathbf{i}}\ ,\ \boldsymbol{x}^l_{\mathbf{i}}\ ,\ n\ )$	
R.4	$\quad \boldsymbol{r}^l_{\mathbf{b}} \leftarrow \boldsymbol{b}^l_{\mathbf{b}} - \mathbf{A}^l_{\mathbf{bi}} \boldsymbol{x}^l_{\mathbf{i}}$	*interior-relative residual*
R.5	$\quad \boldsymbol{x}^l_{\mathbf{b}} \leftarrow$ **Solve** $(\ \mathbf{A}^l_{\mathbf{bb}}\ ,\ \boldsymbol{r}^l_{\mathbf{b}}\ )$	
R.6	$\quad \boldsymbol{b}^{l+1} \leftarrow \mathbf{R}^l \left(\boldsymbol{b}^l - \mathbf{A}^l \boldsymbol{x}^l \right)$	
C.1	$\boldsymbol{x}^L \leftarrow$ **Solve**$(\ \mathbf{A}^L\ ,\ \boldsymbol{b}^L\ )$	*coarse level solution*
P.1	`for` $l = L-1, \ldots, 0$	*prolongation phase*
P.2	$\quad \boldsymbol{x}^l \leftarrow \boldsymbol{x}^l + \mathbf{P}^l \boldsymbol{x}^{l+1}$	
P.3	$\quad \boldsymbol{r}^l_{\mathbf{b}} \leftarrow \boldsymbol{b}^l_{\mathbf{b}} - \mathbf{A}^l_{\mathbf{bi}} \boldsymbol{x}^l_{\mathbf{i}}$	*interior-relative residual*
P.4	$\quad \boldsymbol{x}^l_{\mathbf{b}} \leftarrow$ **Solve**$(\ \mathbf{A}^l_{\mathbf{bb}}\ ,\ \boldsymbol{r}^l_{\mathbf{b}}\ )$	
P.5	$\quad \boldsymbol{r}^l_{\mathbf{i}} \leftarrow \boldsymbol{b}^l_{\mathbf{i}} - \mathbf{A}^l_{\mathbf{ib}} \boldsymbol{x}^l_{\mathbf{b}}$	*boundary-relative residual*
P.6	$\quad \boldsymbol{x}^l_{\mathbf{i}} \leftarrow$ **GaussSeidelRelax**$(\ \mathbf{A}^l_{\mathbf{ii}}\ ,\ \boldsymbol{r}^l_{\mathbf{i}}\ ,\ \boldsymbol{x}^l_{\mathbf{i}}\ ,\ n\ )$	

To perform the V-cycle efficiently, we rearrange variables in blocks of interior (**i**) and boundary (**b**) texels, and rewrite the linear system $\mathbf{A}^l \boldsymbol{x}^l = \boldsymbol{b}^l$ at each level as

$$\begin{pmatrix} \mathbf{A}^l_{\mathbf{ii}} & \mathbf{A}^l_{\mathbf{ib}} \\ \mathbf{A}^l_{\mathbf{bi}} & \mathbf{A}^l_{\mathbf{bb}} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^l_{\mathbf{i}} \\ \boldsymbol{x}^l_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}^l_{\mathbf{i}} \\ \boldsymbol{b}^l_{\mathbf{b}} \end{pmatrix} .$$

We update the solution at interior texels by locking the boundary coefficients, adjusting the constraints to account for the solution met at the boundary, and performing

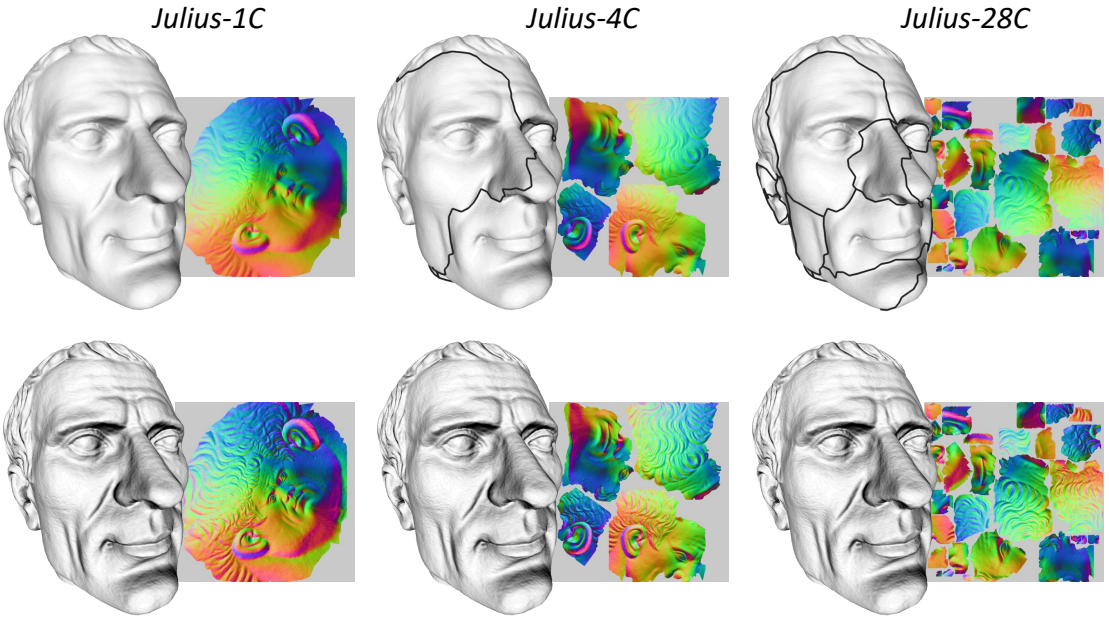


FIGURE 5.8: Comparison of normal-map sharpening using atlases with 1, 4, and 28 charts after one V-cycle.

multiple passes of Gauss-Seidel relaxation over the interior coefficients. Leveraging the grid-regularity of texel adjacency (Figure 5.7, left), relaxation of interior texels can be done efficiently using multi-coloring (parallelization) and temporal-blocking (memory coherence) [78].

As boundary texels have irregular adjacency patterns (Figure 5.7, right), Gauss-Seidel relaxation is less efficient. However, because the number of boundary texels is small, these can be updated using a direct solver at interactive rates. This time we lock interior coefficients, adjust the constraints to account for the solution met in the interior, and perform a direct solve for the boundary coefficients.

Our **V-cycle** algorithm performs the interior relaxation before the boundary solution in the restriction phase, and after in the prolongation phase. ($\mathbf{Solve}(\mathbf{A}, \boldsymbol{b})$ computes the solution to the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ using a direct solver and $\mathbf{GSRelax}(\mathbf{A}, \boldsymbol{b}, \boldsymbol{x}, n)$ performs n Gauss-Seidel relaxations with $\boldsymbol{x}$ as the initial guess.)

5.8.3 Performance

We analyze the performance of our multigrid solver by sharpening normal-maps over three different chartifications of the Julius model, shown in Figure 5.8. Sharpening is done by solving the gradient-domain problem in Equation (5.1), setting $g = \mu$ (immersion metric), $\alpha = 10^{-4}$, ψ equal to the input normal map, and $X = 3\nabla_g \psi$, and using a multigrid system with $L = 4$ hierarchy levels and $n = 3$ Gauss-Seidel relaxations per level. The solutions for the boundary texels and for the full system at the coarsest

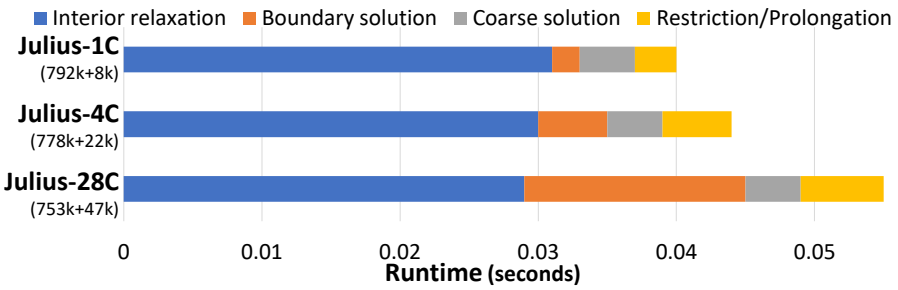


FIGURE 5.9: Breakdown of V-cycle computations times.

level are obtained using CHOLMOD [79]. These tests are performed on a quad-core i7-6700HQ processor.

Runtime Figure 5.9 shows the runtime decomposition for a single V-cycle using double precision. We plot the aggregate times for interior relaxation, boundary solution, solution at the coarsest level, and restriction and prolongation. Memory coherence and parallelism make the average cost of relaxing an interior texel significantly lower than solving for a boundary texel. Thus, a V-cycle becomes less efficient as the atlas becomes more fragmented. The cost of solving at the coarse level and the cost of applying restriction and prolongation is a small fraction of the overall runtime. Evaluating using texture maps with 0.2M, 0.8M, 3.2M, and 12.8M texels, we found that performance scales almost linearly with the number of texels, with improved parallelism at higher resolutions due to the increased per-thread workload.

Comparison to direct solvers Table 5.1 compares the performance of our multigrid system with two direct solvers: CHOLMOD [79] and PARDISO [80, 81]. All solvers are run in double precision. For each one, we report three timings:

- **Initialization**: For direct solvers, this is the symbolic factorization of the fine system $\mathbf{A}^0$. For multigrid, this is the symbolic factorization of the boundary and coarse systems $\{\mathbf{A}^l_{\mathbf{bb}}\}$ and $\mathbf{A}^L$.
- **Update**: For direct solvers, this is the numerical factorization of $\mathbf{A}^0$. For multigrid, this is the numerical factorization of $\{\mathbf{A}^l_{\mathbf{bb}}\}$ and $\mathbf{A}^L$ as well as the computation of the intermediate linear systems $\{\mathbf{A}^{l+1} = \mathbf{R}^l\mathbf{A}^l\mathbf{P}^l\}$.
- **Solution**: For direct solvers, this is back-substitution updating the three coordinates separately. For multigrid, this is a single (parallelized) V-cycle pass updating the coordinates together.

Model	CHOLMOD	PARDISO	Our multigrid
Julius-1C	3.8 : 1.2 : 0.2	2.8 : 0.8 : 0.2	0.4 : 0.1 : 0.04
Julius-4C	4.0 : 1.4 : 0.2	2.9 : 0.8 : 0.3	0.5 : 0.2 : 0.04
Julius-28C	4.0 : 1.3 : 0.2	3.1 : 0.9 : 0.3	0.6 : 0.4 : 0.06

TABLE 5.1: Comparison of the time for initialization, update, and solution (in seconds) of direct solvers to our multigrid method.

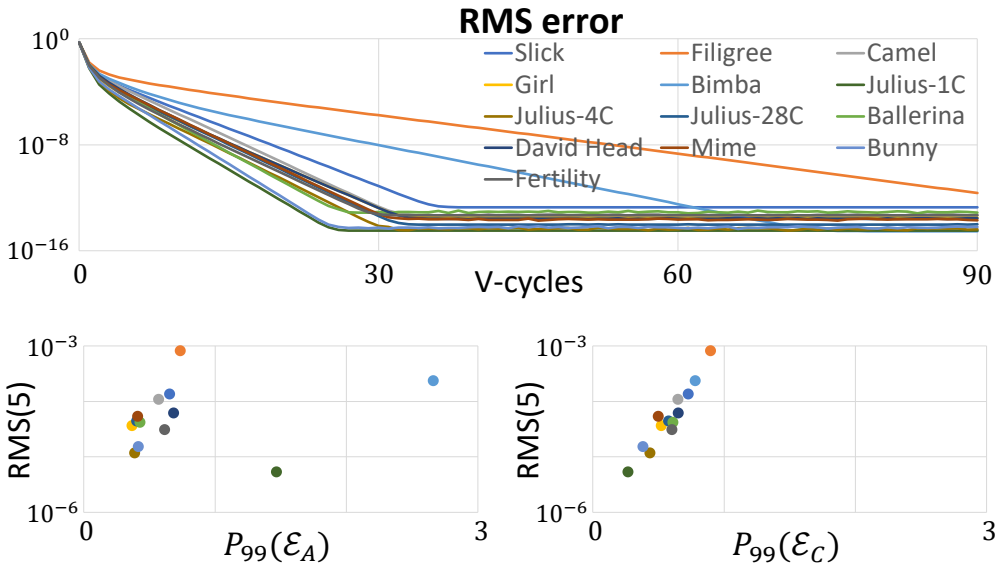


FIGURE 5.10: RMS error as a function of the number of V-cycles (top). RMS after five V-cycles as a function of authalic and conformal energies (bottom).

As Table 5.1 shows, direct solvers incur heavy initialization and update costs due to the factorization (symbolic and numerical, respectively) of large system matrices. In contrast, our approach only requires factorization of small matrices: the ones associated to the boundary nodes and the one at the coarsest resolution. Our multigrid approach also updates the solution at interactive rates, five times faster than a direct solver. In practice, we have found that it takes between two and four V-cycles to obtain a solution that is indistinguishable from a direct solver's solution.

5.8.4 Convergence

We assess the convergence of our solver by analyzing how RMS error decreases with the number of V-cycles. Figure 5.10 (top) shows plots of the RMS error for the models shown in the paper, using the same linear system ($g = \mu$, $\alpha = 10^{-4}$, $X = 0$, and ψ set to random texture), at the same resolution (texture images are rescaled to have 800K texels), with ground-truth obtained using a direct solver.

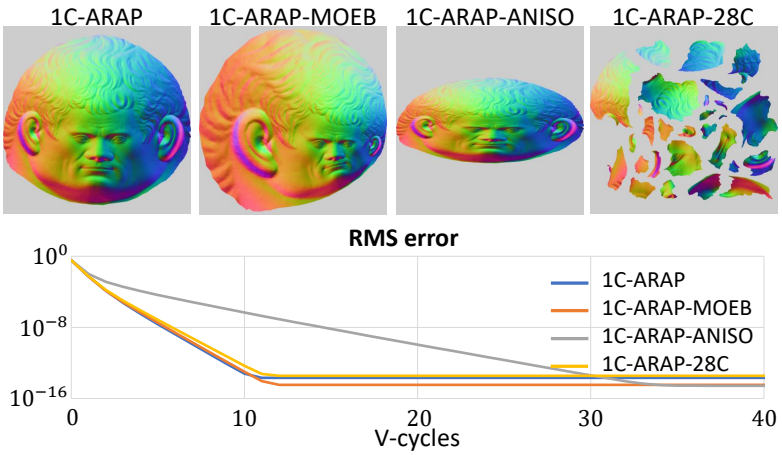


FIGURE 5.11: Convergence plots of four different atlases for the Julius head evidence negative impact of anisotropic distortion and invariance to conformal distortion.

For all models the RMS error decays exponentially up to machine precision. To better understand the different convergence rates, we analyze the effects of parametric distortion on the solver.

Distortion To measure distortion, we scale each 3D model so that its surface area equals the area of the triangulation in the parametric domain and then consider the singular values of the affine transformations mapping 3D triangles into 2D. As in the work of Smith and Schaefer [82], we use a symmetric Dirichlet energy that equally penalizes singular values and their reciprocals. Unlike the earlier work, we define this energy in log-space:

$$\mathcal{E}_D(\sigma_1, \sigma_2) = \log^2(\sigma_1) + \log^2(\sigma_2).$$

An advantage of this formulation is that we can express the energy as the sum $\mathcal{E}_D = \mathcal{E}_A + \mathcal{E}_C$ of authalic and conformal energies:

$$\mathcal{E}_A(\sigma_1, \sigma_2) = \frac{1}{2}\Big(\log(\sigma_1) + \log(\sigma_2)\Big)^2 = \frac{1}{2}\log^2(\sigma_1 \cdot \sigma_2)$$
$$\mathcal{E}_C(\sigma_1, \sigma_2) = \frac{1}{2}\Big(\log(\sigma_1) - \log(\sigma_2)\Big)^2 = \frac{1}{2}\log^2(\sigma_1/\sigma_2).$$

To better understand how distortion affects convergence rates, we plot the RMS error after five V-cycles against the 99-th percentile distortion in Figure 5.10 (bottom). Surprisingly, convergence is weakly correlated with area (authalic) distortion. Rather, it is the deviation from conformality, as reflected by larger values of $\mathcal{E}_C$, that correlates strongly with slower convergence.

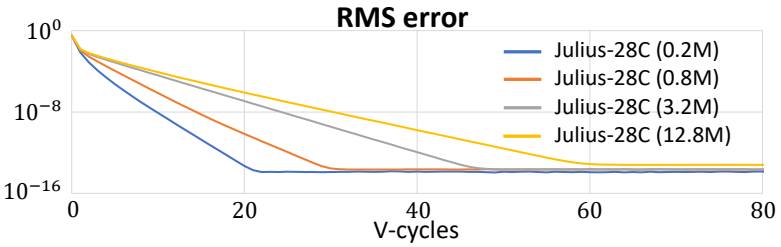


FIGURE 5.12: Convergence of our solver for a single texture atlas using texture images with 0.2M, 0.8M, 3.2M, and 12.8M texels.

We corroborate this empirical observation by computing an as-rigid-as-possible [83] parameterization of the Julius head (1C-ARAP). Then, we obtain new charts by applying a Möbius transformation (1C-ARAP-MOEB), applying an anisotropic scale (1C-ARAP-ANISO), and partitioning into 28 charts (1C-ARAP-28C). Note that 1C-ARAP, 1C-ARAP-MOEB, and 1C-ARAP-28C have the same conformal distortions while 1C-ARAP, 1C-ARAP-ANISO, and 1C-ARAP-28C have the same authalic distortions.

Figure 5.11 shows the convergence plots for the four different atlases. As the figure shows, neither the application of a Möbius transformation nor the introduction of new seams significantly affects the convergence rate of the solver. In contrast the introduction of anisotropy significantly degrades the solver's performance.

Note that though it does not necessarily improve convergence, reducing area distortion is still important for ensuring that the discretization samples the function space uniformly.

Resolution We also analyze the performance of our multigrid solver as a function of resolution. Fixing the parameterization, we up-sample the texture map and consider the convergence of the multigrid solver at different resolutions.

Figure 5.12 shows representative results for four different resolutions of the Julius-28C atlas. As the figure shows, though the RMS error decays exponentially, the convergence rate slows as resolution is increased. We do not have a satisfying explanation for this behavior and intend to continue studying this in the future.

Single precision solver Using single precision, we obtain a roughly 2× speedup for the interior relaxation and for the restriction and prolongation stages, though numerical precision limits the achievable accuracy. The error reduction is similar to that of double precision for the first 5-8 iterations, at which point the single precision solver plateaus to an RMS error of roughly 10^{-5}.

Triangle quality Though convergence efficiency depends on the parametric distortion, it is less dependent on the quality of the triangulation. For example, if there is no distortion, the discretization of the linear system depends only on the parameterization of the chart boundaries and not on the shapes of the triangles.

5.8.5 Implementation of interior relaxation

The objective of our multigrid method is to provide a fast update of the system solution by taking advantage of grid regularity. This regularity facilitates coherent memory access and concurrent processing.

In Figure 5.13 we present a coarse-to-fine view of the elements that form the data decomposition in our multigrid system. In the top row we show elements we have already introduced: the system hierarchy, an atlas at a fixed resolution, and a chart in this atlas.

Charts are decomposed into 2D arrays of overlapping *blocks*. In the bottom row of Figure 5.13 we show one of these blocks. Each block is divided into a sequence of *rows*, and each row is split into a collection of *segments*. Each segment is a maximal set of consecutive interior texels.

Within each hierarchical atlas we index texels one chart a time. For each chart the texels are indexed from top to bottom and left to right. In this way, texels within a row are adjacent in memory.

Our relaxation method updates the interior texels of an atlas by iterating across charts, blocks, rows and segments. Applying Gauss-Seidel relaxation to a segment is straightforward. In the hierarchy construction stage, we compute pointers to the first and last texel of each segment, and pointers to the texels immediately above and below the first texel. In the solution update stage, we traverse each segment by simultaneously advancing the pointers to the first texel and to the texels above and below. We terminate when the central pointer reaches the last texel of the segment[5].

The reason why we decompose charts into blocks is twofold. The first is to limit the row length so we can fit multiple rows in cache. The second is to allow concurrent relaxation of a chart.

Cache coherence It is desirable that after traversing a row, the next row remain in cache so it does not need to be reloaded for its own relaxation. By fitting multiple rows in

[5]We also store the system coefficients and constraint values in memory adjacent locations, so they can be easily traversed by increasing a pointer.

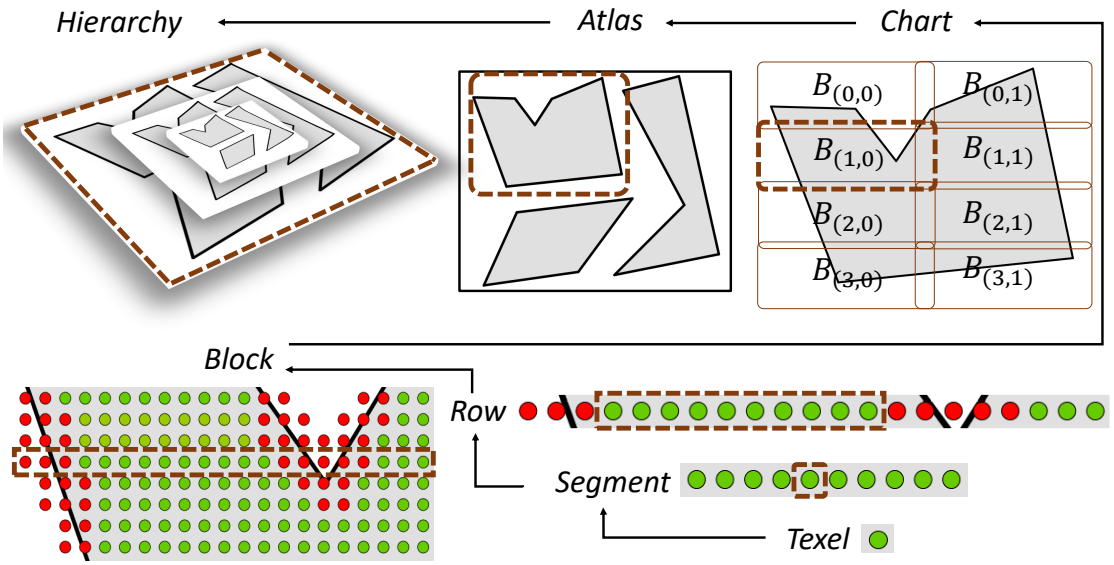


FIGURE 5.13: Components of our multigrid data decomposition.

cache we can further exploit memory coherence through temporal blocking [84, 85] : after relaxing a row (r_i) we relax a small preceding band of in-memory rows (r_{i-1},r_{i-2},...,r_{i-k}) before proceeding to the next row (r_{i+1}). In this way multiple Gauss-Seidel relaxation passes are executed within a single data load.

Concurrent relaxation We want to exploit parallelism by relaxing independent blocks with different threads. To do this we group blocks into lines, and classify lines as even or odd. For instance, in Figure 5.13, $B_{(0,0)}, B_{(0,1)}$ is the first even line and $B_{(1,0)}, B_{(1,1)}$ is the first odd line. First we process all the even lines in parallel, assigning a single thread to each line. Then we repeat the process for the odd lines. This task decomposition avoids race conditions and reduces synchronization.

5.9 Applications

We demonstrate the versatility of our approach by considering a number of applications of gradient-domain processing. For each of these, the solution is obtained by solving

$$\operatorname*{argmin}_{\phi} \mathrm{E}(\phi; g, \alpha, \psi, X)\,,$$

with g the metric, α the screening weighting, ψ the target scalar field, and X the target vector field.

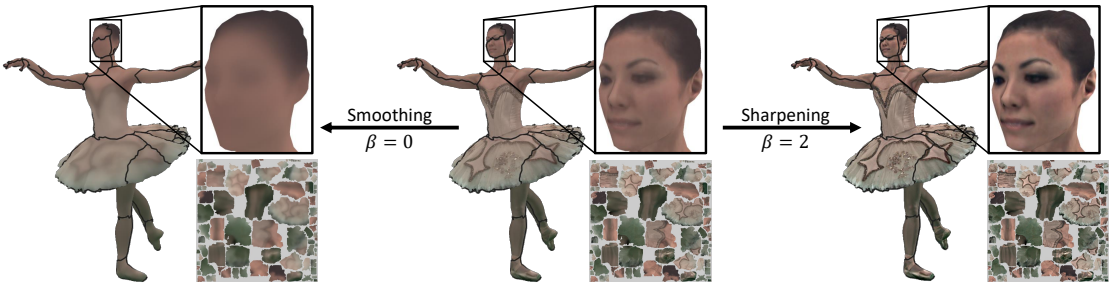


FIGURE 5.14: We smooth (left) and sharpen (right) a texture by solving linear systems that dampen and amplify the local color variation.

We use single precision and, with the exception of the last application, results are obtained using our multigrid solver, with $L = 4$ hierarchy levels, $n = 3$ Gauss-Seidel iterations per level, and using CHOLMOD to solve for the boundary nodes and coarsest resolution system. Immersions are scaled so the surface has unit area (because the effects of α and g are scale-dependent). All parameterizations, with the exception of those shown in Figure 5.11, are obtained using UVAtlas [86]. Please see the appendix of [73] for performance statistics.

Source code for our gradient-domain applications can be found at `https://github.com/fabianprada/GradientDomainTextureProcessing`.

5.9.1 Isotropic filtering

A signal ψ is smoothed and sharpened by solving for a new signal with scaled gradient. Following the approach of Bhat et al. [18], we compute a filtered signal ϕ as the minimizer:

$$\underset{\phi}{\operatorname{argmin}}\, \mathrm{E}(\phi; \mu, 10^{-4}, \psi, \beta\nabla_{\mu}\psi)\,,$$

with β the differential scaling term (and μ the immersion metric). Setting $\boldsymbol{\psi}$ to the coefficients of the input signal and using Equations (5.12) and (5.13), the coefficients $\boldsymbol{\phi}$ of the minimizer are given by

$$\boldsymbol{\phi} = \left(\mathbf{M} + 10^{-4}\mathbf{S}\right)^{-1}\left(\mathbf{M} + 10^{-4}\beta\mathbf{S}\right)\boldsymbol{\psi}\,.$$

When $\beta < 1$, the differential of the input signal is dampened and the signal is smoothed. When $\beta > 1$, the differential is amplified, and the signal is sharpened. Figure 5.8 shows results of sharpening a normal map and Figure 5.14 shows results of smoothing and sharpening a color texture.

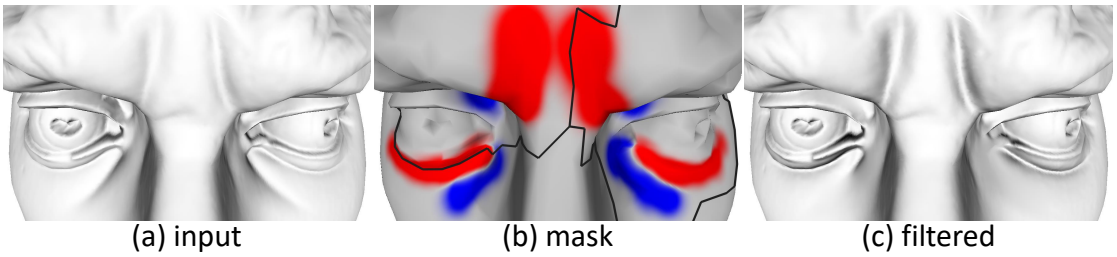


FIGURE 5.15: We design a user interface for local filtering. In the middle we visualize the differential modulation mask used for this example, showing attenuation (smoothing) in blue and amplification (sharpening) in red.

Local filtering Selective removal or enhancement of signal detail is obtained by allowing β to vary spatially. Figure 5.15 shows an example of local filtering where an input texture (a) is filtered to produce both sharpening and smoothing effects (c). The spatially varying modulation mask (b) prescribes that the furrow should be amplified (red) while the bags under the eyes should be removed (blue). We represent β as a piecewise constant function, with a value associated to each cell $c \in C$. The minimizer is given by

$$\boldsymbol{\phi} = \left(\mathbf{M} + 10^{-4}\mathbf{S}\right)^{-1} \left(\mathbf{M} + 10^{-4} \sum_{c \in C} \beta_c \mathbf{S}_c\right) \boldsymbol{\psi}\,,$$

where $\mathbf{S}_c$ is the stiffness matrix with integration restricted to c,

$$(\mathbf{S}_c)_{s,t} = \int_c \langle \nabla_g \phi_s, \nabla_g \phi_t \rangle_g \sqrt{|g|} dA\,,$$

and β_c is the differential modulation factor at c.

We designed an interactive system for texture filtering using a spray-can interface to prescribe local modulation weights β. We precompute the matrices $\mathbf{S}_c$. Then, at runtime, the user-specified modulation weights are transformed into linear constraints and our multigrid solver generates the new texture values at interactive rates, approximately 18 frames per second on the ballerina model (740k texels).

5.9.2 Texture stitching

Previous works in image and geometry processing merge multiple signals by formulating stitching as a gradient-domain problem [55–57]. These approaches use the input signals to compute differences between pairs of adjacent elements and solve for a global signal that matches the differences in a least squares sense. Here, we describe how to use our framework to stitch together textures obtained by imaging a static object from multiple viewpoints.

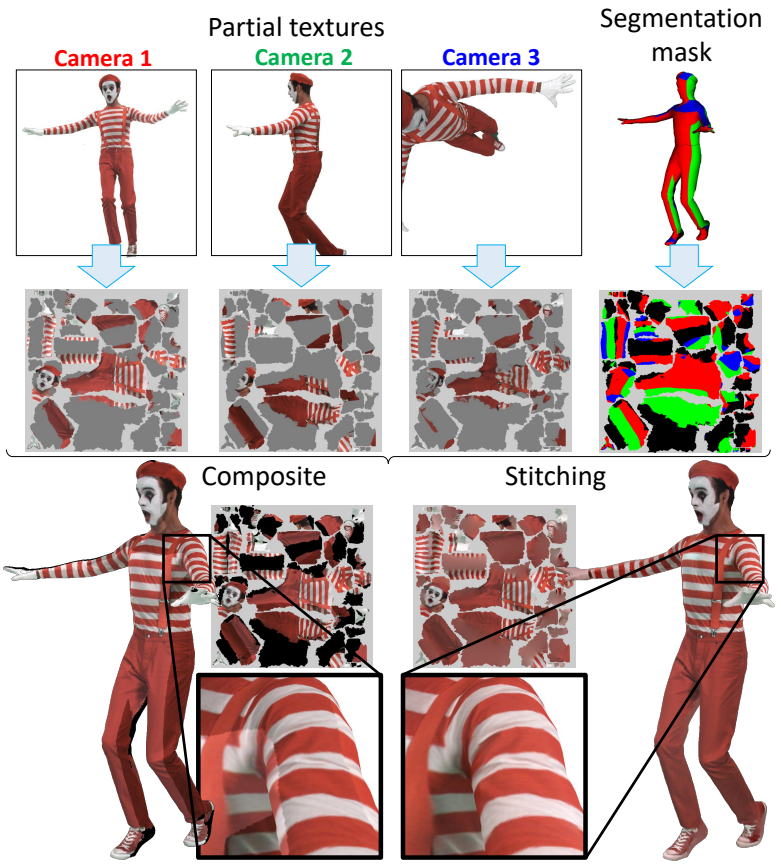


FIGURE 5.16: Gradient-domain stitching generates a texture that does not exhibit discontinuities due to the lighting variance of the partial textures.

Our input is a texture-atlased surface, together with a collection of *partial textures* $\{\psi_k\}$ and a *segmentation mask* $\boldsymbol{\zeta}$. Figure 5.16 shows an example with three partial textures. The partial textures sample the color of visible texels from each camera's viewpoint, and the segmentation mask specifies the camera providing the best view. (The quality of a view is determined by visibility as well as the alignment of the surface normal to the camera's view direction.)

A naive solution is to create a composite texture ψ by using the camera with the best view to assign a texel's color. As shown in the middle left of Figure 5.16, this reveals abrupt illumination changes at the transitions between regions covered by different cameras.

These discontinuities are removed by solving for a texture that preserves the differential within the partial textures and is smooth across the boundaries. To achieve this, we set the target texel difference to zero for texel pairs residing on different partial textures:

$$\boldsymbol{z}_d \equiv \begin{cases} \boldsymbol{\psi}_t - \boldsymbol{\psi}_s & \text{if } \boldsymbol{\zeta}_s = \boldsymbol{\zeta}_t \\ 0 & \text{otherwise} \end{cases} \qquad \forall \boldsymbol{d} = (s, t) \in \mathscr{E}.$$

In regions not seen by any camera, differences are also set to zero to encourage a smooth fill-in. We construct the associated vector field $X = \sum_d \boldsymbol{z}_d X_d$, and solve for the signal with matching differential:

$$\operatorname*{argmin}_{\phi} \mathrm{E}(\phi; \mu, 10^{-2}, \psi, X)\,.$$

Setting $\boldsymbol{\psi}$ to the coefficients of the composite and using Equations (5.12) and (5.14), the coefficients $\boldsymbol{\phi}$ of the solution are given by

$$\boldsymbol{\phi} = \left(\mathbf{M} + 10^{-2}\mathbf{S}\right)^{-1} \left(\mathbf{M}\boldsymbol{\psi} + 10^{-2}\mathbf{d}^{\top}\mathbf{M}_1\boldsymbol{z}\right)\,.$$

Results of gradient-domain stitching are shown in Figures 5.1 (bottom left) and 5.16 (middle right), where lighting differences between the cameras are removed, while details in the interior are preserved.

5.9.3 Single-source geodesic distances

We demonstrate the robustness of our approach by computing single-source geodesic distances using the Geodesics-in-Heat method [87]. The approach computes distances to a source point $p \in M$ by solving two successive systems. The first solves for a short-time-scale diffusion of an impulse δ_p at the surface point:

$$\operatorname*{argmin}_{\psi} \mathrm{E}(\psi; \mu, 10^{-3}, \delta_p, 0)\,. \tag{5.15}$$

The second solves for the function whose differential best matches the (negated) normalized differential of the diffused impulse:

$$\operatorname*{argmin}_{\phi} \mathrm{E}\left(\phi; \mu, \infty, \bullet, -\frac{\nabla_\mu \psi}{|\nabla_\mu \psi|}\right)\,. \tag{5.16}$$

(Setting $\alpha \to \infty$, the target scalar field has no effect.)

In the texture domain, we associate the impulse with a texel $t \in \mathcal{T}$, defining $\boldsymbol{\delta}^{\boldsymbol{t}}$ to be the vector whose coefficients is one at the t-th texel and zero for all others. Using Equation (5.12) we obtain the coefficients of the smoothed impulse by solving $(\mathbf{M} + 10^{-3}\mathbf{S})\boldsymbol{\psi} = \mathbf{M}\boldsymbol{\delta}^{\boldsymbol{t}}$.

The coefficients $\boldsymbol{\phi} \in \mathbb{R}^{|\mathcal{T}|}$ of the geodesic function are obtained by solving the system $\mathbf{S}\boldsymbol{\phi} = -\mathrm{div}(\nabla_\mu \psi / |\nabla_\mu \psi|)$. Leveraging the smoothness of ψ, we use one-point quadrature

to approximate the values of $\mathrm{div}(\nabla_\mu \psi / |\nabla_\mu \psi|)$. Assuming a connected surface, the solution is unique up to a constant factor and we offset the solution so that the distance is 0 at the source: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \boldsymbol{\phi}_t$.

Figure 5.17 (top) shows the distance function for a source point selected on the cheek of the bunny. Note that after three V-cycles (second row), the result is indistinguishable from the result obtained with a direct solver (third row).

This application is unusual in that the constraint to the second linear system depends on the solution to the first, and hence evolves with the V-cycle iterations. Nonetheless, Figure 5.17 (bottom) shows that the RMS error for both $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ decays exponentially.

We also validated the efficiency of our multigrid solver in an interactive application in which a user picks a source texel and the application displays the estimated geodesic distances after each multigrid pass. When a source texel is selected the smoothed impulse and distance functions are initialized to zero. Then, at each frame, one V-cycle is performed for the impulse diffusion system and a second is performed for the distance estimation (using the solution from the first V-cycle to define the constraints for the second). We achieve an interactive rate of 17 frames per second on the bunny model (670k texels). Please refer to the accompanying video for a demonstration.

5.9.4 Line integral convolution

Last, we consider the application of line integral convolution [88] to surface vector field visualization. Teitzel et al. [89] achieve this by tracing streamlines over the triangulation and averaging a random signal over these paths, obtaining a signal defined over the mesh. Palacios and Zhang [90] interactively project the field onto the view-plane, obtaining a signal in screen-space. Diewald et al. [91] formulates vector field visualization on images and surfaces using anisotropic diffusion [15]. We introduce line integral convolution in the texture domain by using gradient-domain processing with an anisotropic metric.

Given a vector field Y, we first define a metric g_Y that stretches distances along the direction perpendicular to Y in proportion to the magnitude of Y. Then, we diffuse a random texture ψ along the stream-lines by solving

$$\operatorname*{argmin}_{\phi} \mathrm{E}(\phi; g_Y, 1, \psi, 0) \,.$$

The anisotropic diffusion on the random texture ψ produces a signal ϕ where texels along the same integral line have similar color, but the contrast across different integral lines is low. We run gradient domain sharpening to increase contrast across integral

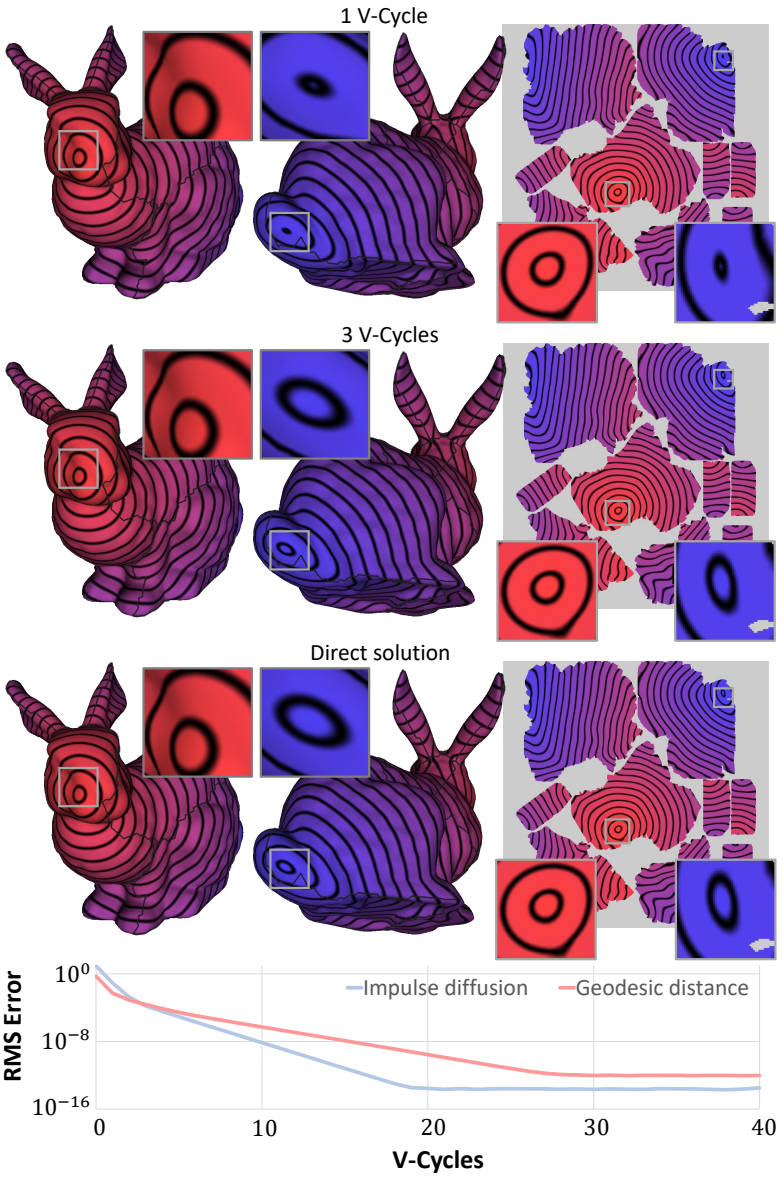


FIGURE 5.17: In a single V-cycle our multigrid solver produces a result very similar to the exact solution of a direct solver (RMS= $5.2 \cdot 10^{-3}$). After three V-cycles, the result is almost indistinguishable (RMS= $2.7 \cdot 10^{-4}$).

lines. The signal $\tilde{\phi}$ that we use to visualize vector fields is the solution to

$$\operatorname*{argmin}_{\tilde{\phi}} \mathrm{E}(\tilde{\phi}; \mu, 10^{-4}, \phi, 100\nabla_{\mu}\phi)$$

Given a normalized vector field Y, which is constant per triangle in the canonical coordinate frame of M, we define the anisotropic metric by setting

$$g_Y(X_1, X_2) \equiv 10^4 \langle X_1, J_\mu Y \rangle_\mu \cdot \langle X_2, J_\mu Y \rangle_\mu + \langle X_1, X_2 \rangle_\mu \,,$$

for all vector fields $X_1, X_2 \in TM$. Here J_μ is the direction field perpendicular to Y (relative to μ).

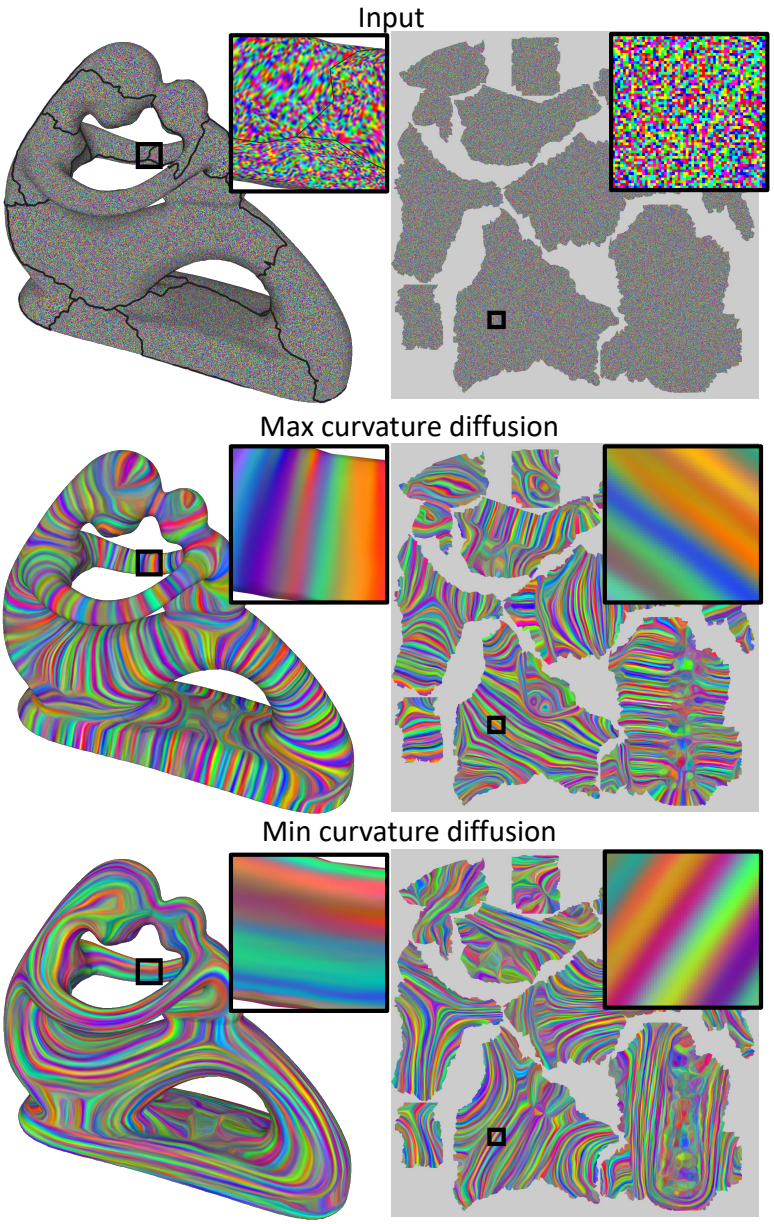


FIGURE 5.18: We perform line integral convolution by modifying the surface metric to diffuse a random signal along the directions of principal curvature.

Figure 5.1 (bottom right) and Figure 5.18 show visualizations of surface curvature on the *Camel* and *Fertility* models. For these we define Y by scaling principal curvature directions by the absolute difference in principal curvature values. Figure 4.3 shows visualization of the harmonic vector fields in the Fertility model.

This application highlights the robustness of our finite-elements discretization, which provides high-quality vector-field visualizations despite significant distortion in the metric.

Chapter 6

Evolving Meshes

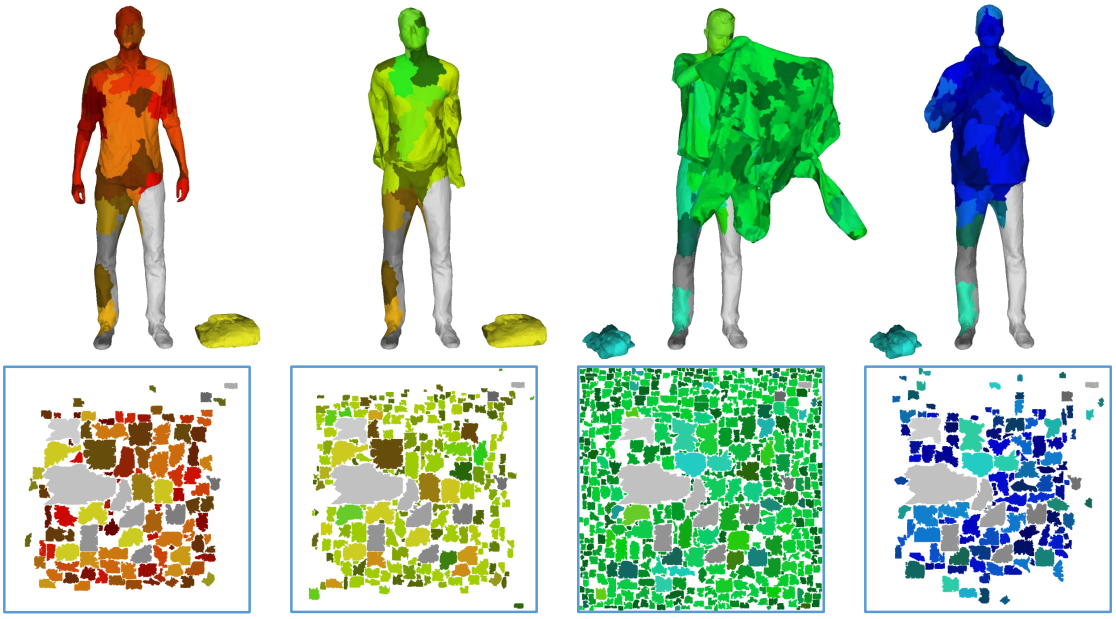


FIGURE 6.1: Spatiotemporal atlas parameterization of the *Clothing* sequence.

In the previous chapters we introduce techniques for processing signals on static triangle meshes. In this chapter, we present the first steps in generalizing these techniques to evolving meshes, i.e., meshes whose embedding and connectivity change over time.

We start by introducing a spatiotemporal texture atlas parameterization for evolving meshes. Our method is robust and flexible, performing local remeshing operations in case of tracking failure, surface stretching and topological changes. The proposed parameterization aims to maximize coherence and compactness, improving texture compression over state-of-the-art methods.

We close this chapter with preliminary results for gradient domain processing on an evolving texture atlas. A more comprehensive exploration of temporally coherent signal processing in the presence of partial correspondences is left for future research.

6.1 Introduction

The goal of surface tracking is to represent a deforming surface using a template mesh whose vertex positions are updated as the surface moves [52, 92]. The final result is a sequence of meshes with identical triangulation that establish a perfect correspondence between surface points at any two frames.

In computer graphics tracking is particularly important since it provides a compact representation of realistic motion. Additionally, it is a prerequisite for tasks like motion classification [93], and for applications like motion synthesis [94] and transfer [95].

However, there are multiple challenges that need to be addressed when tracking a surface. In particular we highlight:

1. Establishing correct correspondences between the template and target meshes.
2. Deforming the template mesh to capture the fine detail in the target.
3. Avoiding degradation of the template mesh triangulation over prolonged tracking.

These kinds of challenges make tracking computationally expensive. Techniques that provide successful tracking of long temporal sequences require user intervention [96] or multiple passes over the data [97].

Having a robust tracking algorithm is not necessarily possible: when the target surface has different topology or presents drastic geometric changes (e.g., occluded regions become visible), it might not be representable by the template mesh. This is the case for the *Clothing* capture in the top row of Figure 6.1, where the character takes his shirt off, drop it to his right, and then puts on the jacket that is to his left. Capturing all these changes with a single template is extremely challenging.

The work of Collet et al. [98] solves the limitations of single-template tracking by representing temporal sequences using multiple keyframes. The authors choose an optimal keyframe mesh based on topological criteria (largest number of connected components, smallest genus) and geometric criteria (largest area) and use it to track a window of frames around it. Whenever the deformed keyframe does not provide a good representation of the target mesh the process restarts: a new keyframe is selected and locally

propagated. However, doing a complete re-triangulation increases the storage requirements of the sequence, and limits global editing operations.

Evolving meshes addresses the limitations of both the single template and the keyframe approaches, while preserving fidelity to the deforming surface and providing a compact representation. The concept of evolving mesh was initially introduced by Wojtan et al. [99] in the context of fluid simulation. In that work, the authors deform the surface of a fluid by advancing a differential equation, and applying local remeshing operations to compensate for topological changes. The mesh is preserved almost everywhere, except at the critical regions where new triangles are added or removed to capture the changes.

Bojsen-Hansen et al. [100] extended the use of evolving meshes to surface tracking. In that work remeshing operations are not only used to compensate for topological changes but also to support possible tracking failures. Furthermore, [100] establish surface correspondences across remeshing operations to propagate attributes like surface albedo over mesh vertices during animation.

Our construction of evolving meshes differ from Bojsen-Hansen et al. [100] approach in the definition of remeshing events. In [100], the authors compare the intersections of the deformed template and the target mesh against a volumetric grid, and run remeshing operations to resolve topological inconsistencies. This can introduce remeshing when both the deformed template and target look similar but are slightly misaligned with respect to the reference grid. In order to get a more spatiotemporal coherent representation, we only trigger remeshing events when the deformed template is perceptually a poor representation of the target. We define our remeshing policy based on visibility (ambient occlusion), detail (curvature), and correspondences (closest point distance).

In Figure 6.2 we compare tracking the motion of finger with and without remeshing enabled. The changing topology (fingers touching and separating) and the fast motion makes this sequence particularly challenging. When remeshing is disabled (middle row), we observe how tracking failure propagates to subsequent frames. Instead, when remeshing is enabled, tracking failure is resolved, and the updated template provides a good approximation to the input sequence. In Section 6.3 we provide a brief description of the integration between tracking and remeshing.

Robust tracking and sparse remeshing are prerequisites for our major contribution: the construction of a spatiotemporal coherent parameterization for evolving meshes (Figure 6.1). Our construction provides an intuitive extension of standard concepts like charts and atlas from 2D to 3D. Furthermore, our method parallels the traditional procedure for texture atlas parameterization, decomposing the process into three major steps:

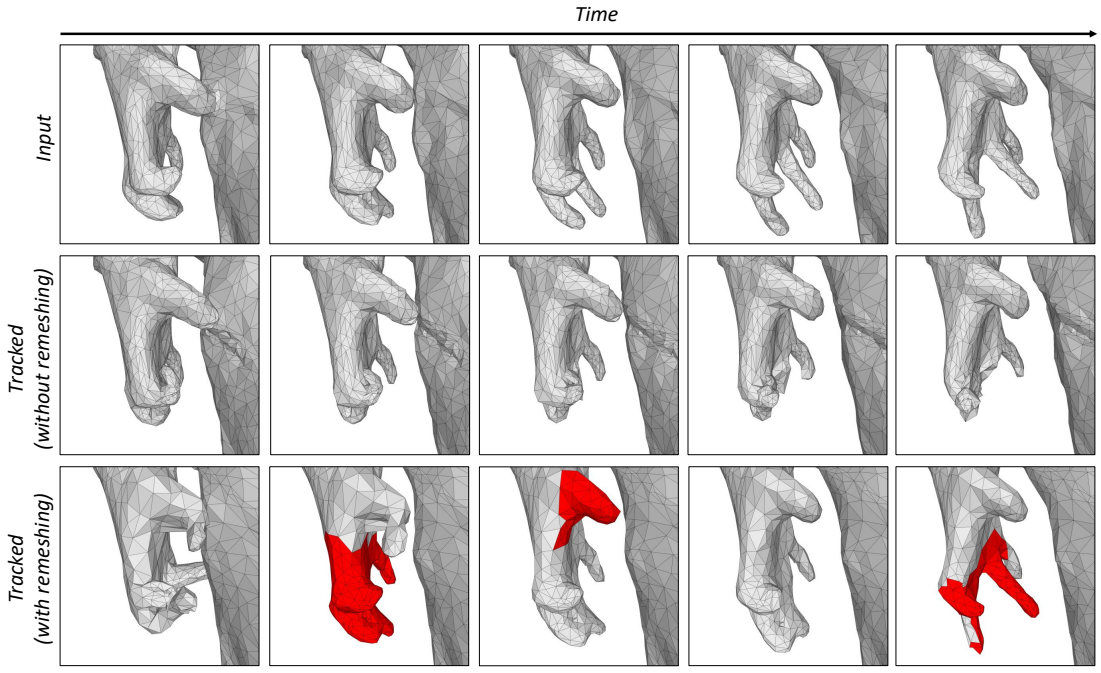


FIGURE 6.2: Comparison of finger motion tracking with remeshing enabled and disabled.

charting, unwrapping, and packing [3, 61]. In Section 6.4, we describe how these steps are adapted to the context of evolving meshes.

Finally, we are interested in exploring signal processing over evolving meshes. In particular we are interested in exploiting partial correspondences to generate globally coherent filtering results. In Section 6.5 we extend the gradient domain formulation of Chapter 5 to show an applications for lighting removal.

6.2 Overview

The input to our algorithm is a temporal sequence of unstructured meshes $\{F_i = (T_i, V_i)\}$, where each frame F_i is represented by a triangulation T_i and a set of 3D vertex positions V_i.

Our goal is to create a *time-evolving mesh* $\{F'_i = (T'_i, V'_i, U'_i)\}$, with texture coordinates U'_i, that looks similar to the input sequence but has temporally coherent in connectivity, geometry, and parameterization.

The creation of the time-evolving mesh $\{F'_t\}$ proceeds in 3 steps:

Tracking: Given the new geometry (T'_i, V'_i) at time i, we deform its vertex positions, to obtain the geometry $(T'_i, \widetilde{V}_{i+1})$ that fits the input geometry (T_{i+1}, V_{i+1}) from the next frame.

Remeshing: We identify regions in the deformed mesh $(T'_i, \widetilde{V}_{i+1})$ that fail to match the input and replace these with input geometry, obtaining the remeshed geometry (T'_{i+1}, V'_{i+1}) for the next frame.

Parameterization: We leverage the temporal coherence of the triangles T'_i in the output geometry to define coherent texture atlases U'_i for better compressibility.

6.3 Evolving mesh construction

6.3.1 Tracking

Figure 6.3 provides an example of the construction of an evolving mesh by sequentially deforming a template and applying local remeshing operations. In the top row we show the input to our algorithm: a temporal sequence of meshes with different connectivity. We start the tracking process (depicted in the second row) by initializing the template as a copy of the first mesh in the sequence. Then, we deform the template to match the second mesh in the sequence. If the deformed mesh provides a satisfactory match to the target geometry, we continue the tracking process, computing a new deformation that makes the template match the next target configuration.

When the deformed template does not provide a good match to the target, as highlighted by the circled regions in Figure 6.3, we perform a local remeshing procedure that compensates for the differences. Then, we continue the tracking process using the remeshed surface as the new template.

Our mesh deformation algorithm is an extension of the algorithm introduced by Sumner et al. [52]. In that work the authors identify closest point correspondences between the template and the target, and optimize for a deformation that fits the correspondences while preserving the local rigidity of the template. We extend this algorithm to do robust pruning of correspondences using geodesic descriptors, and identify tangential correspondences (instead of closest point) using mesh optical flow as described in Section 4.8.2. For further details of the tracking process, please refer to [101].

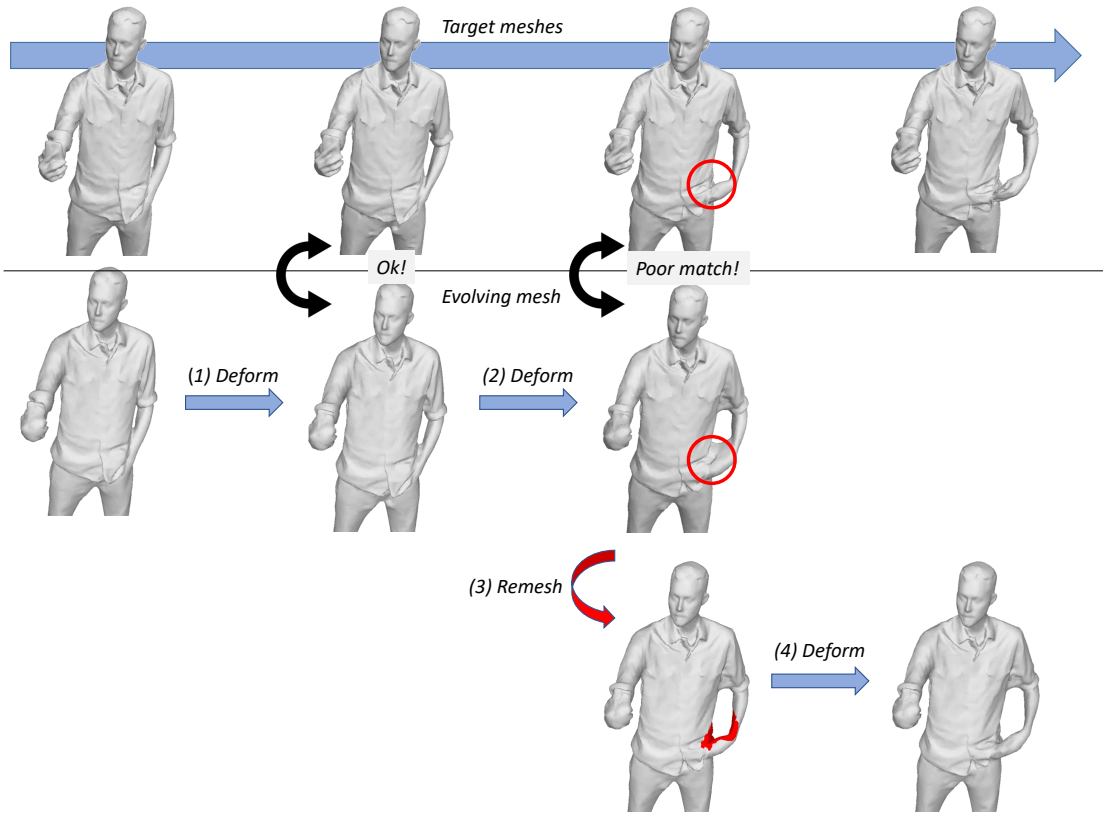


FIGURE 6.3: Integration of localized remeshing on the tracking pipeline.

6.3.2 Remeshing

Localized remeshing is the key feature to maximize the persistence of the template triangulation while providing a faithful reconstruction of the target sequence. In Figure 6.4 we illustrate the major steps of the remeshing process. Our construction is similar to [99, 100] in its use of a volumetric grid to enforce consistent merging. However, the way we identify mismatched regions and stitch surfaces is different. Our algorithm can be summarized as follows:

1. **Region selection**: We identify as *seeds* all the vertices in the template and target mesh that have no good correspondence with any point in the other mesh. In our implementation, the correspondence score at a vertex is given by its distance to the other mesh, and is modulated by its ambient occlusion and curvature. Vertices that have low visibility are less likely to be marked as seeds since they do not affect the perceptible quality of the reconstruction, and might belong to regions that are temporally occluded (e.g., surfaces that are in contact with each other). Vertices with high curvature are more likely to be marked as seeds since they convey meaningful detail (e.g., finger position, facial features, etc). Finally, we mark as seeds the vertices of triangles on the template mesh that have excessive distortion

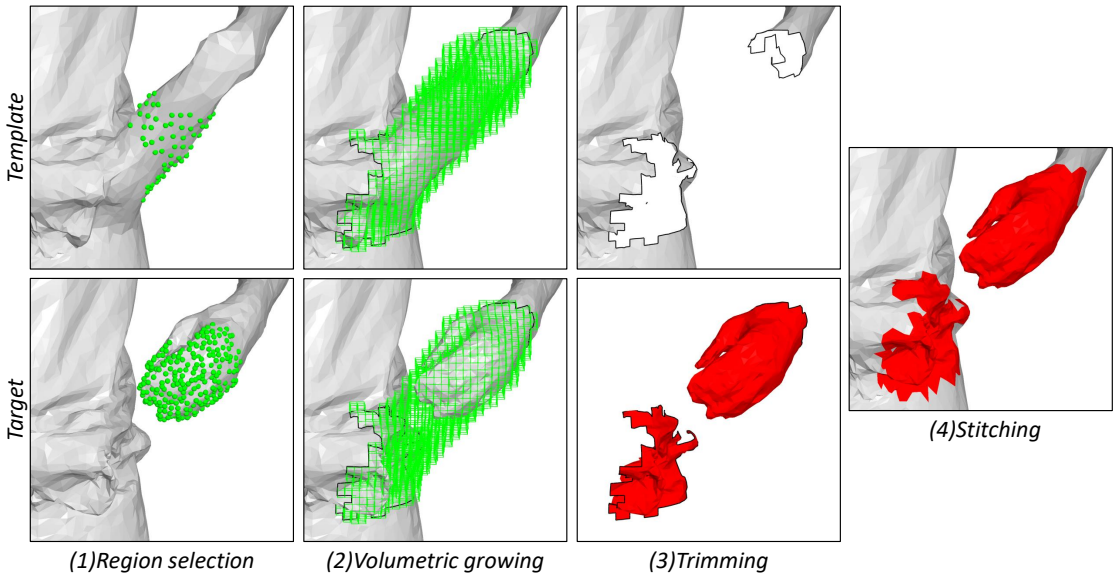


FIGURE 6.4: Major steps of the remeshing algorithm.

with respect to the initial configuration. Ensuring that the shape of a triangle is preserved during the tracking process is a prerequisite for the construction of the spatiotemporal atlas parameterization in Section 6.4.

2. **Volumetric growing**: We define a common voxelization to identify consistent boundaries for surface insertion and deletion operations. We mark the voxels containing seeds of either the template or target and propagate the marked voxels until the intersections of both the template and target meshes with the grown volume boundary are consistent.

3. **Trimming and stitching**: We clip the meshes against the grown volume, removing from the template the triangles interior to the volume, and inserting the triangles from the target mesh. Since the boundary curves in the template and the target are topologically consistent, we do the stitching by first identifying corresponding vertices, and then merging them together into their average position. Finally, we run a pass of mesh simplification and Laplacian smoothing [58] to improve the quality of the triangulation around the stitching curve.

For further details on the tracking process, please refer to [101]. The implementation of our remeshing algorithm can be found at `https://github.com/fabianprada/MeshStitching`.

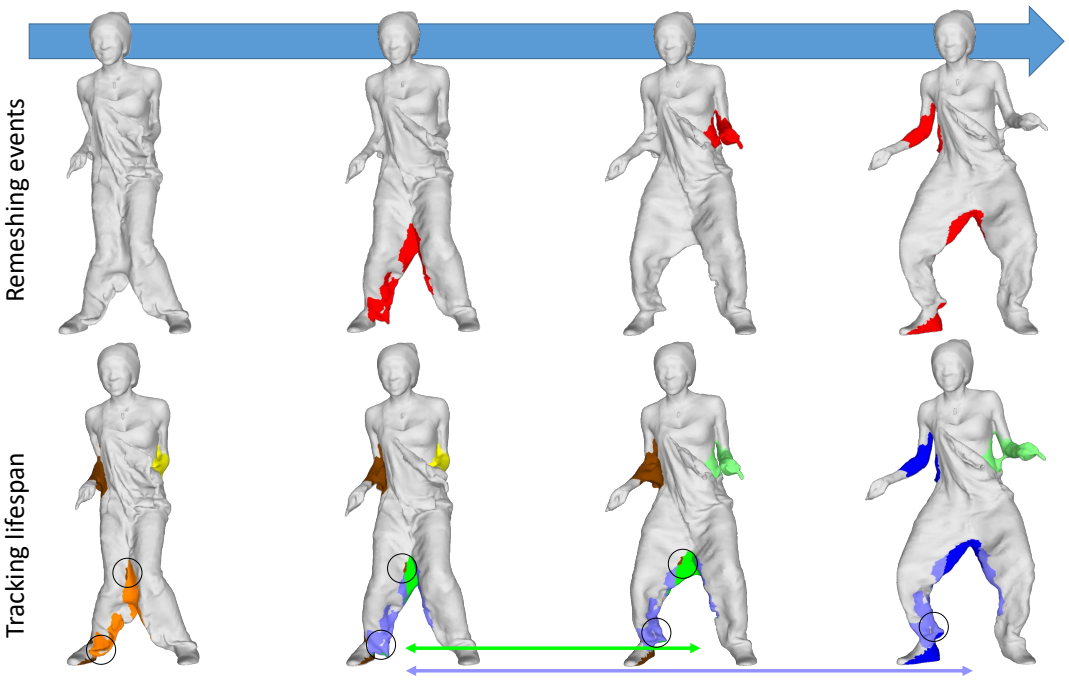


FIGURE 6.5: Remeshing operations induce a partition of triangles on regions with identical lifespan.

6.4 Evolving mesh parameterization

Once the tracking process is completed we construct a texture atlas parameterization for the evolving mesh. Shown in the bottom row of Figure 6.1 is the atlas parameterization of the *Clothing* sequence, which is just a standard atlas parameterization for each frame. As in the standard case, our atlas construction requires decomposing the evolving mesh into a collection of charts, unwrapping, and packing them into the texture domain.

Since our tracking process aims to preserve most of the triangulation from one frame to the next, we would like to make our atlas parameterization as temporally coherent as possible. For instance, if a triangle appears in multiple frames of the evolving mesh, it is desirable to map it to the same position in the texture domain: this makes the atlas parameterization more compressible.

In the top row of Figure 6.5 we show the output of our tracking method on a short sequence of the *Girl* capture, highlighting in red the regions modified by localized remeshing. Once the tracking stage terminates we can associate a lifespan to each triangle in the template, i.e, identify the first and last frame where the triangle appears in the template. In the second row of Figure 6.5 we color each triangle in the template according to its tracking-lifespan. For instance, we color in green the triangles in the pants of the girl that are added in the second frame and removed in the fourth frame, and we color in light blue, the triangles that are added in the second frame and persist through the

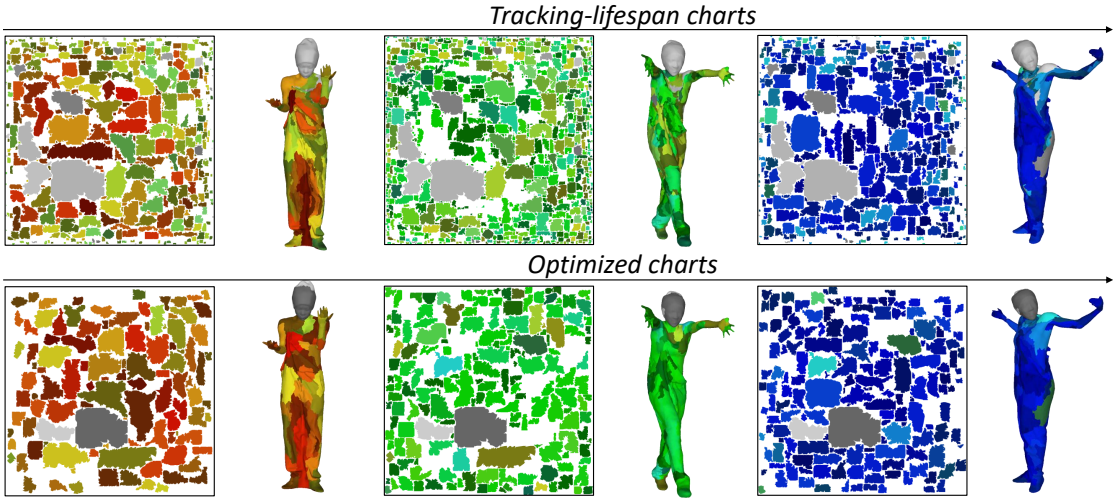


FIGURE 6.6: Atlas generated from tracking-lifespan charts exhibit excessive fragmentation. Our optimization sacrifice a bit of temporal coherence for improved spatial coherence.

fourth frame. By construction, the tracking-lifespans induce a partition of the triangles in the evolving mesh.

Naively we could just use the tracking-lifespan clusters as the charts of our parameterization. However, as highlighted in the circles in Figure 6.5, the tracking process can produce excessive fragmentation (i.e., too many small patches) which is undesirable for texture mapping. In Figure 6.6 we compare the atlas parameterization for three frames of the *Girl* capture using the charts given by the lifespan clustering (top) and our optimized parameterization (bottom). Our result look as temporally coherent as the one from tracking-lifespan charts, but with significantly less fragmentation.

In determining the partition into charts, we seek to maximize both temporal and spatial coherence. To measure *temporal coherence* we introduce the concept of parametric-lifespan. A parametric-lifespan of a triangle is an interval of frames on which the triangle use the same texture coordinates. A triangle can have a parametric-lifespan equal to its tracking-lifespan, or it can have multiple parametric-lifespan if it changes its texture coordinates. Our measure of temporal coherence is the average parametric-lifespan. We favor longer parametric-lifespans to reduce the number of stored texture coordinates.

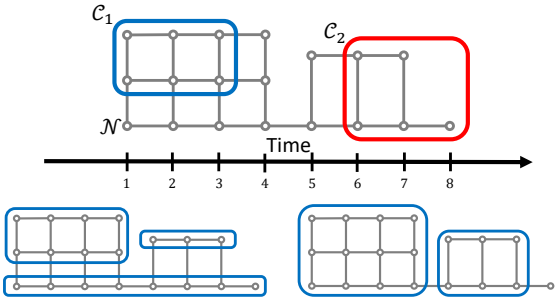
We measure *spatial coherence* as the average number of charts per frame. We favor fewer charts to reduce unused gutter space needed to separate charts and to minimize the presence of texture discontinuity curves on the evolving mesh.

In this section we describe our approach for simultaneously optimizing both objectives, parameterizing the resulting charts, and assembling them into a 3D texture map.

6.4.1 Spatiotemporal atlas notation

Let T'_i be the triangles in the remeshed output of frame i. Given triangles $t_i \in T'_i$ and $t_{i+1} \in T'_{i+1}$, we write $t_i \mapsto t_{i+1}$ when triangle t_i is deformed into triangle t_{i+1} by the tracking.

- A *chain* is a subset of nodes $\{t_s, \ldots, t_e\}$ with $t_i \mapsto t_{i+1}$.
- The *lifespan* of a chain $\{t_s, \ldots, t_e\}$ is the interval $[s, e]$.
- The *neighbor graph* on $\bigcup T'_i$, denoted $\mathcal{N}$, is the graph with an edge between two triangles/nodes if either they are adjacent in the same frame, or one tracks to the other (in adjacent frames).
- A *chart* is a connected subgraph $\mathscr{C} \subset \mathcal{N}$ whose nodes form chains with identical lifespan, denoted $I_{\mathscr{C}} = [s_{\mathscr{C}}, e_{\mathscr{C}}]$.
- Two charts are *neighbors* if they are connected by an edge in $\mathcal{N}$.
- The *cross-section* of a chart at time i is $\mathscr{C}(i) = \mathscr{C} \cap T'_i$.
- An *atlas* is a set of charts that partitions the neighbor graph $\mathcal{N}$.



The top of the inset figure shows a visualization of a neighbor graph with subgraphs $\mathscr{C}_1$ and $\mathscr{C}_2$ highlighted. Though both are composed of two chains, only $\mathscr{C}_1$ is a chart. In the bottom we show two different atlases on $\mathcal{N}$. The one on the left maximizes temporal coherence and, among all such maxima, also maximizes spatial coherence. The one on the right maximizes spatial coherence and, among all such maxima, also maximizes temporal coherence.

6.4.2 Charting

To identify a good atlas, we define an energy on the space of atlases, design editing operators to transform one atlas into another, and greedily choose edits that reduce the energy.

Geometric interpretation We think of a chart $\mathscr{C}$ as a (right) prism through the neighbor graph, with time as its axis, and the cross-section at time i given by $\mathscr{C}(i)$. This allows us to define the *cap area* as twice the average of cross-sectional areas and the *side area* as the sum of the perimeter lengths of the cross-sections:

$$\|\mathscr{C}\|_C = \frac{2}{|I_{\mathscr{C}}|} \cdot \sum_{s \in I_{\mathscr{C}}} |\mathscr{C}(s)| \qquad \text{and} \qquad \|\mathscr{C}\|_S = \sum_{s \in I_{\mathscr{C}}} |\partial \mathscr{C}(s)| \,,$$

where $\partial \mathscr{C}(s)$ is the boundary of the cross section $\mathscr{C}(s)$.

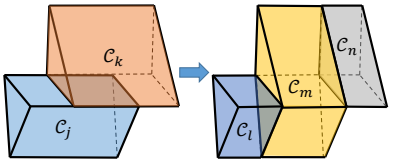
Optimization energy We define the energy of an atlas $\mathscr{A}$ to be the sum of temporal and spatial coherence terms, summed over the atlas charts. We note that a chart ceases to be temporally coherent at its temporal boundaries (its two caps). Similarly, it ceases to be spatially coherent along its spatial boundaries (its sides) and becomes less coherent as it narrows. Taking these in conjunction, we set:

$$E(\mathscr{A}) = \sum_{\mathscr{C} \in \mathscr{A}} E(\mathscr{C}) \quad \text{with} \quad E(\mathscr{C}) = \|\mathscr{C}\|_C + \alpha \cdot \frac{\|\mathscr{C}\|_S}{\|\mathscr{C}\|_C}$$

(We fix α to 10 times the average radius of a triangle.)

Atlas editing operators To support exploration of the space of atlases, we define an atlas edit operator as the composition of several primitive transformations. The *spatial merge* transformation takes two neighboring charts with identical lifespans and joins them into a single chart. It keeps the temporal coherence energy fixed and improves spatial coherence by removing boundaries and increasing the cross-sectional area. The *temporal split* transformation takes a chart and divides it along a cross-sectional slice. It worsens temporal coherence, but lets us trim the lifespans of adjacent charts so they can be merged.

Then, given spatially neighboring charts $\mathscr{C}_j$ and $\mathscr{C}_k$, we construct an edit operator composed of at most two temporal splits (to align the lifespans of the charts) followed by a spatial merge of the charts with identical lifespans. This edit creates at most three charts $\mathscr{C}_l$, $\mathscr{C}_m$, and $\mathscr{C}_n$ and its change in energy is



$$\Delta E = \Big(E(\mathscr{C}_l) + E(\mathscr{C}_m) + E(\mathscr{C}_n)\Big) - \Big(E(\mathscr{C}_j) + E(\mathscr{C}_k)\Big)$$

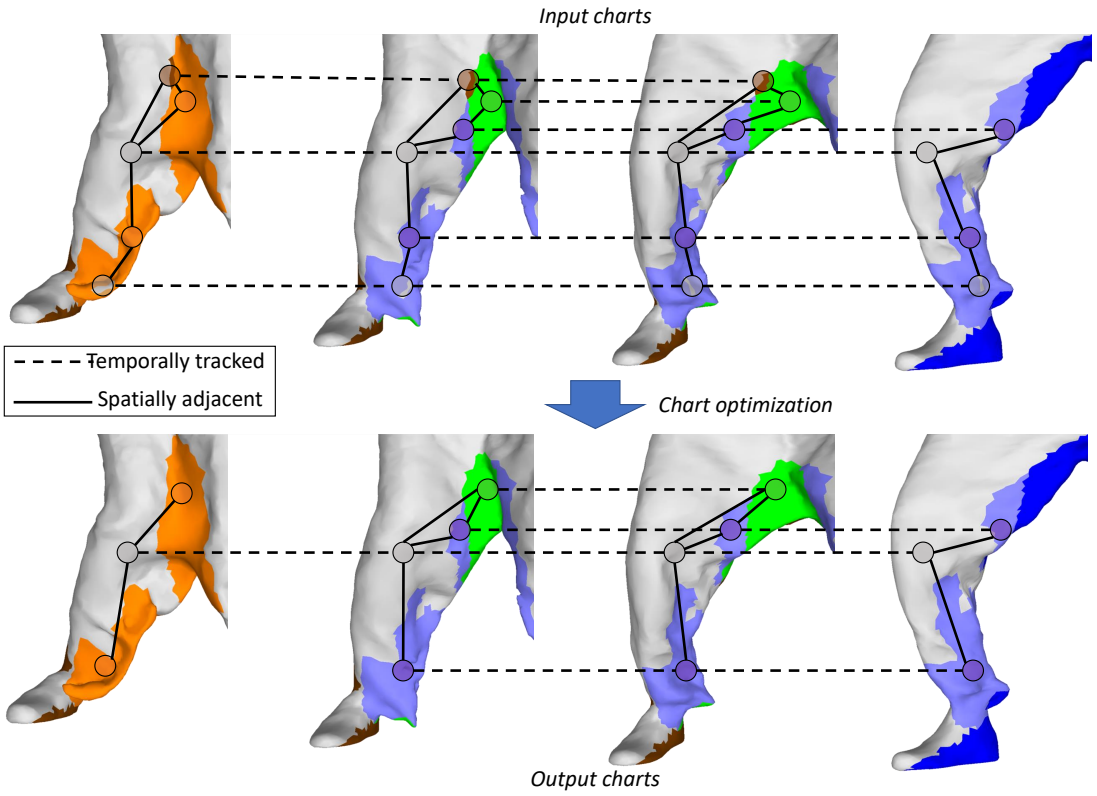


FIGURE 6.7: A sequence of merging and split operations on the neighbour graph produce a more spatially coherent parameterization.

Greedy energy descent We define an initial atlas by computing maximal chains in the neighbor graph and clustering neighboring chains with identical tracking-lifespan. The top row of Figure 6.7 show this initialization for a section of the evolving mesh depicted in Figure 6.5. This atlas minimizes the temporal coherence energy and, of all such minimizers, it is the one that minimizes the spatial coherence energy.

We optimize the atlas by maintaining a heap of candidate edits, sorted by their associated change in energy. We pop a candidate edit off the heap, perform the edit if the associated charts have not been modified already, create new candidate edits between the new chart and its neighbors, and insert these candidates into the heap if they have negative energy change ΔE.

In the second row of Figure 6.7 we show the neighbour graph after a sequence of splitting and merge operations. The small fragments have been absorbed by the large patches improving the overall quality of the chartification.

6.4.3 Unwrapping

Given a chart decomposition, we unwrap and map each chart into texture space. For a chart $\mathscr{C}$ we take geometry from the first cross-section, $\mathscr{C}(s_{\mathscr{C}})$, and use UVAtlas [86]

to obtain the parameterization of the 2D cross-section. This parameterization is then extruded across the lifespan of the chart, as shown in the top of Figure 6.8.

We note that even though each chart is connected, we are not guaranteed that the unwrapping will occupy a contiguous region in texture space because the parameterization may divide a chart into multiple components to limit geometric distortion. These components form a refined chartification, which we call the *parameterized atlas*.

6.4.4 Packing

Given the unwrapped charts, we pack them into a 3D texture atlas. Our design strategy aims to support a streamable representation whereby a lightweight client need only store and access a single frame of texture video per rendered frame. Also, we constrain the 3D texture charts to be *right* prisms, so vertices have constant texture coordinates over their lifespans.

To assign texture coordinates to the unwrapped charts, we sort charts by cap area and incrementally place them in the texture volume. (We also tried sorting by chart volume and lifespan, but found that cap area gives the most efficient packing.)

Using the fact that the unwrapped charts are extrusions of 2D cross-sections, we can reduce the placement to a 2D problem. Specifically, given the chart $\mathscr{C}$, we consider all previously placed charts which overlap the lifespan of $\mathscr{C}$, flatten them onto the 2D plane and search for locations in 2D that are empty of previously placed charts and can fit the cross-section of $\mathscr{C}$.

We find such locations by using the approach of [61] which defines *horizons* for both the current texture map and the chart to be inserted. The horizon is an envelope defined around a line, with one horizon contained in the free space of the texture domain and the other containing the chart. Shifting one horizon along the other provides an efficient way to identify locations in texture space which can accommodate the new chart.

While a direct implementation successfully places the charts, it is not efficient. This is because the original method uses a single texture horizon at the top of the 2D domain, and places the new charts as close to the bottom as possible. As the packing is performed from bottom to top, the horizon provides a good representation of the remaining free space. In our context, we perform the packing in 2D texture space, but then extrude the result into the 3D texture volume, with different charts extruded by different lifespans. Thus, even if charts appear to be tightly packed at the bottom of one 2D flattening, the packing may not be tight when flattened onto a different lifespan.

Instead, we use multiple horizon lines for the 2D texture space, testing each one to identify candidate locations. Once the locations are identified, we place the chart at the location closest to the projected center of *all* the charts already placed into the atlas. (We use 16 evenly spaced horizontal and vertical lines in our experiments and have found diminishing returns when using more.)

In the bottom of Figure 6.8 we show the output of the packing process. The extruded prisms, each representing a different chart, are placed in the 3D atlas, trying to maximize occupancy and avoiding overlaps. The texture atlas at each frame (shown to the left of each model) corresponds to a slice of the 3D atlas.

6.4.5 Parameterization results

In Figure 6.1 we show the spatiotemporal texture atlas parameterization for a collection of captures. For each capture we show the texture atlas at the first frame (left column), middle frame (center column) and last frame (right column) of the sequence. We color-code the charts according to its lifespan. Charts with large lifespan have low saturation, and in particular, charts in gray have full lifespan. The hue of the chart encodes the midpoint of its lifespan: charts that appear at the beginning of the sequence are colored in red, charts that appear at the middle are colored in green and charts that appear at the end are colored in blue.

6.5 Signal processing

6.5.1 Texture videos

The atlas parameterization of the evolving mesh allows us to store high resolution signals in the form of images that are sampled on the surface using standard texture mapping techniques. In the top row of Figure 6.10 we show the parameterization of three consecutive frames of the Ballerina sequence, and in the second row we visualize the rendered meshes using the camera re-projected textures.

We call the sequence of textures $\{I_i\}$ the *texture video*. On our experiments, the texture videos generated by our spatiotemporal parameterization had a gain of 12% in MP4 compression over the texture videos generated from the keyframe approach of Collet et al. [98]. For a more detailed discussion of texture and geometry compression please refer to [101].

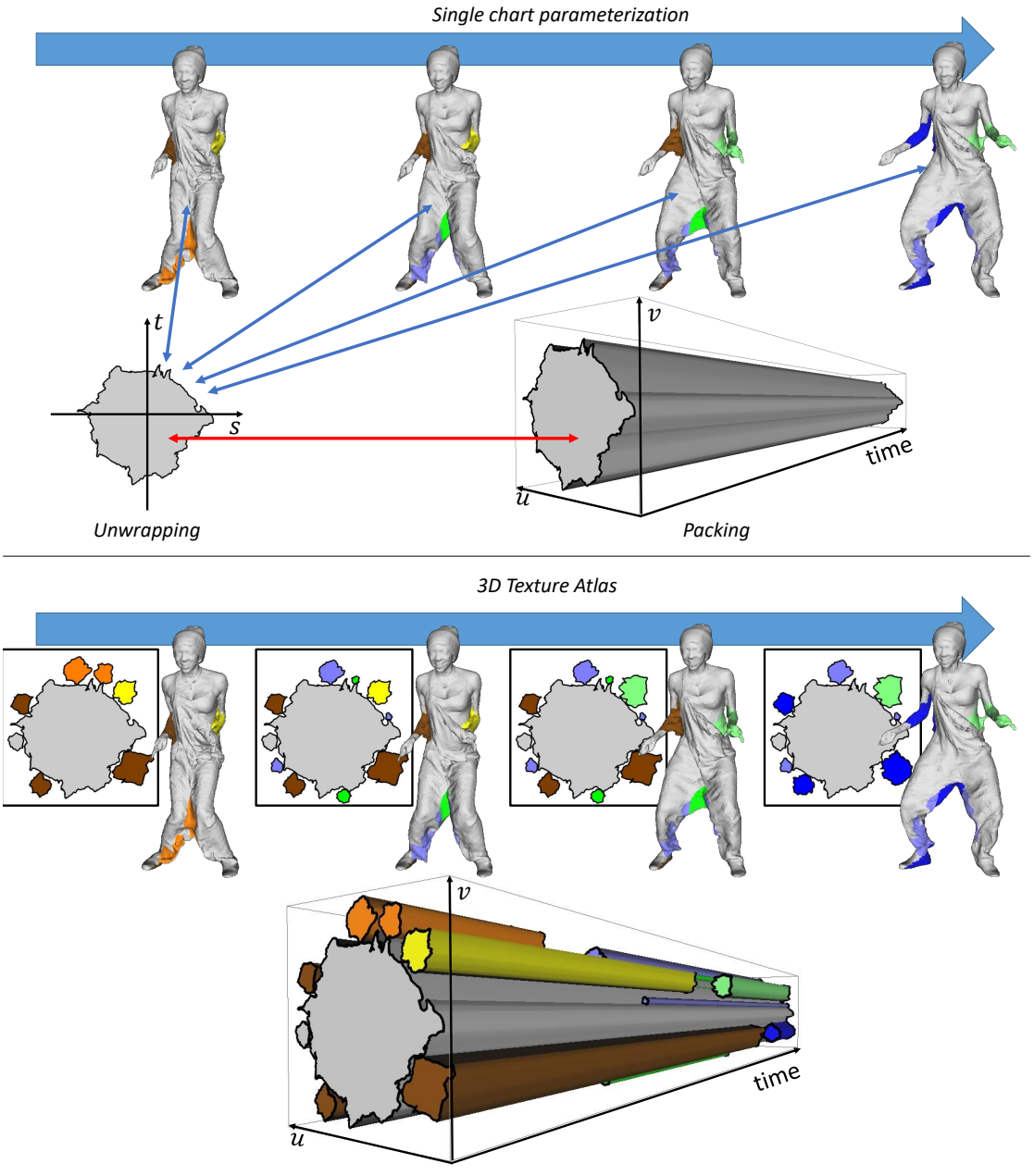


FIGURE 6.8: Each chart of the evolving mesh is unwrapped, extruded along its temporal axis, and packed into the 3D texture atlas.

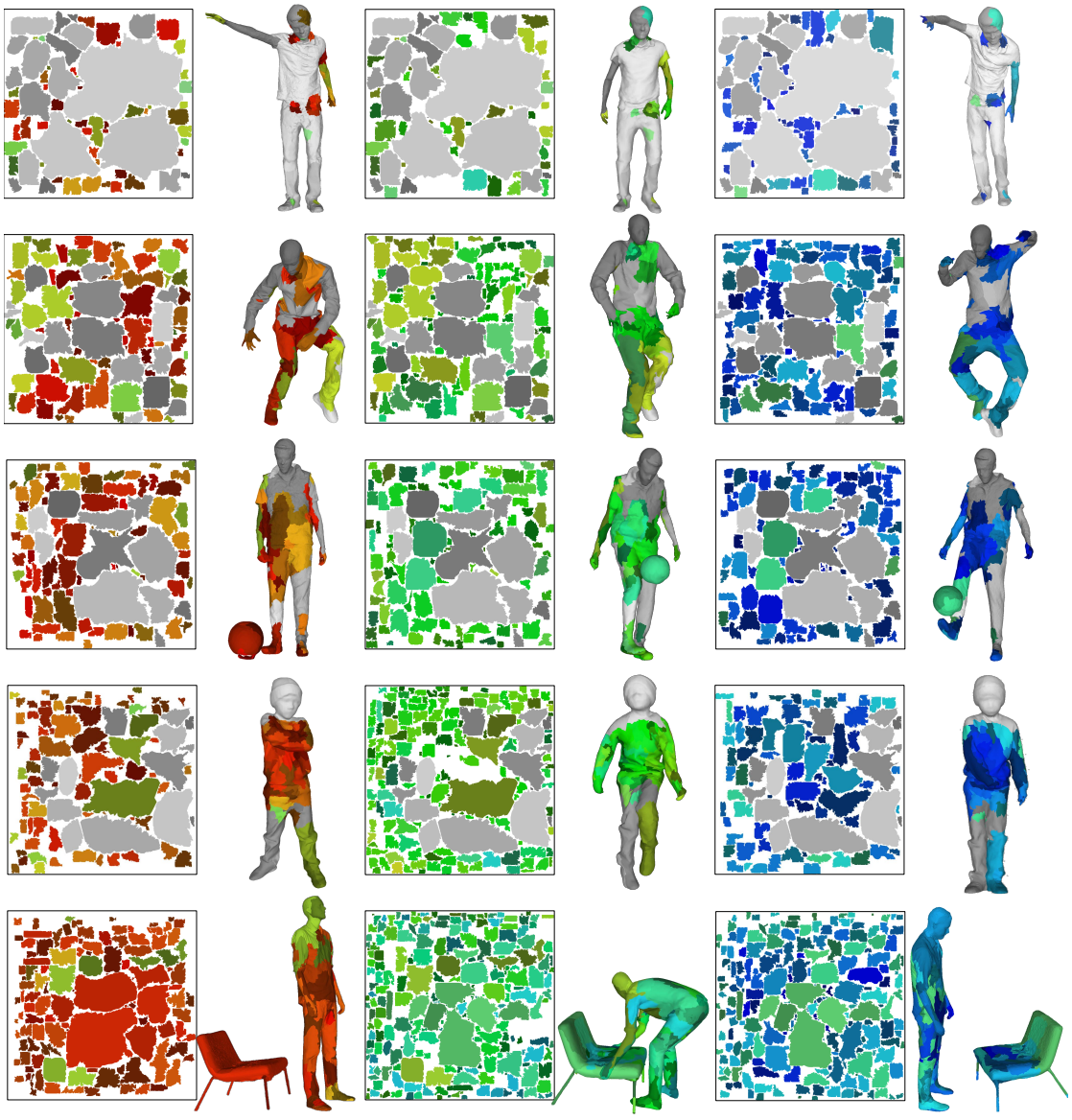


FIGURE 6.9: Spatiotemporal atlas parameterization of the *Macarena, Slick, Soccer, Breakers*, and *Chair* sequences.

It is important to notice that even though a chart can be replicated across multiple frames, we allow the signal associated to the chart to change frame to frame. While this might be unnecessary for some applications, in the case of the example in Figure 6.10[1] this allow us to capture details like the motion of the wrinkles in the clothes and changing facial expressions.

However, we can see that the sampled textures also capture strong lighting variations due to the motion of the character. We highlight a patch of the texture atlas, covering

[1]The textures on the second row of Figure 6.10 were constructed by assigning to each texel the color from the camera with the best visibility and stitching (see Section 5.9.2).

part of the Ballerina dress, where these lighting variations are visible. In the next section we describe how to attenuate these artifacts using gradient domain techniques.

6.5.2 Lighting removal

Our hypothesis is that color variation due to lighting effects occurs smoothly on the surface but abruptly over time, and consequently it is captured in the low frequencies of the signal. In order to remove the lighting artifacts, we compute a new estimate of the low frequencies components by averaging the signal across time. The result of this temporal smoothing, $\{\bar{I}_i\}$, is shown in the third row of Figure 6.10. In our implementation, we set the smoothed values at each frame to be the weighted average (using cubic b-spline weights) of a 10-radius window around the frame. When the lifespan of a chart does not cover the entire 10 radius window, we just take the weighted average of the covered subset of frames.

Temporal smoothing is effective in removing the abrupt lighting variations but also introduces significant blurring. At each frame, we want a signal that captures the low frequencies of the temporally smoothed signal, $\bar{I}_i$, and the high frequencies of the input, I_i. This can be formulated as the solution to the following screened-Poisson system:

$$\min_{I_i^*} ||\bar{I}_i - I_i^*||^2 + \alpha||\nabla I_i - \nabla I_i^*||^2 \tag{6.1}$$

The fourth row of Figure 6.10 shows the solution of this system for $\alpha = 5 \times 10^{-6}$. Lighting artifacts are removed and texture details preserved.

6.5.3 Temporal screened-Poisson

In the previous application we decomposed the filtering process in two steps: temporal smoothing and per-frame reconstruction. The formulation allows each frame to be solved independently and keep the size of the linear system on the order of the texture size.

An alternative formulation could enforce the temporal smoothness as part of the optimization energy. This is an instance of the temporal screened-Poisson energy:

$$E(\phi; \psi, X, \nu, \alpha, \gamma, g) = \int \|\phi - \psi\|_g^2 + \alpha\|\nabla\phi - X\|_g^2 + \gamma||\frac{\partial\phi}{\partial t} - \nu||_g^2 d\mu \tag{6.2}$$

Exploring applications of this formulation and efficient solutions through hierarchical methods is a future research direction.

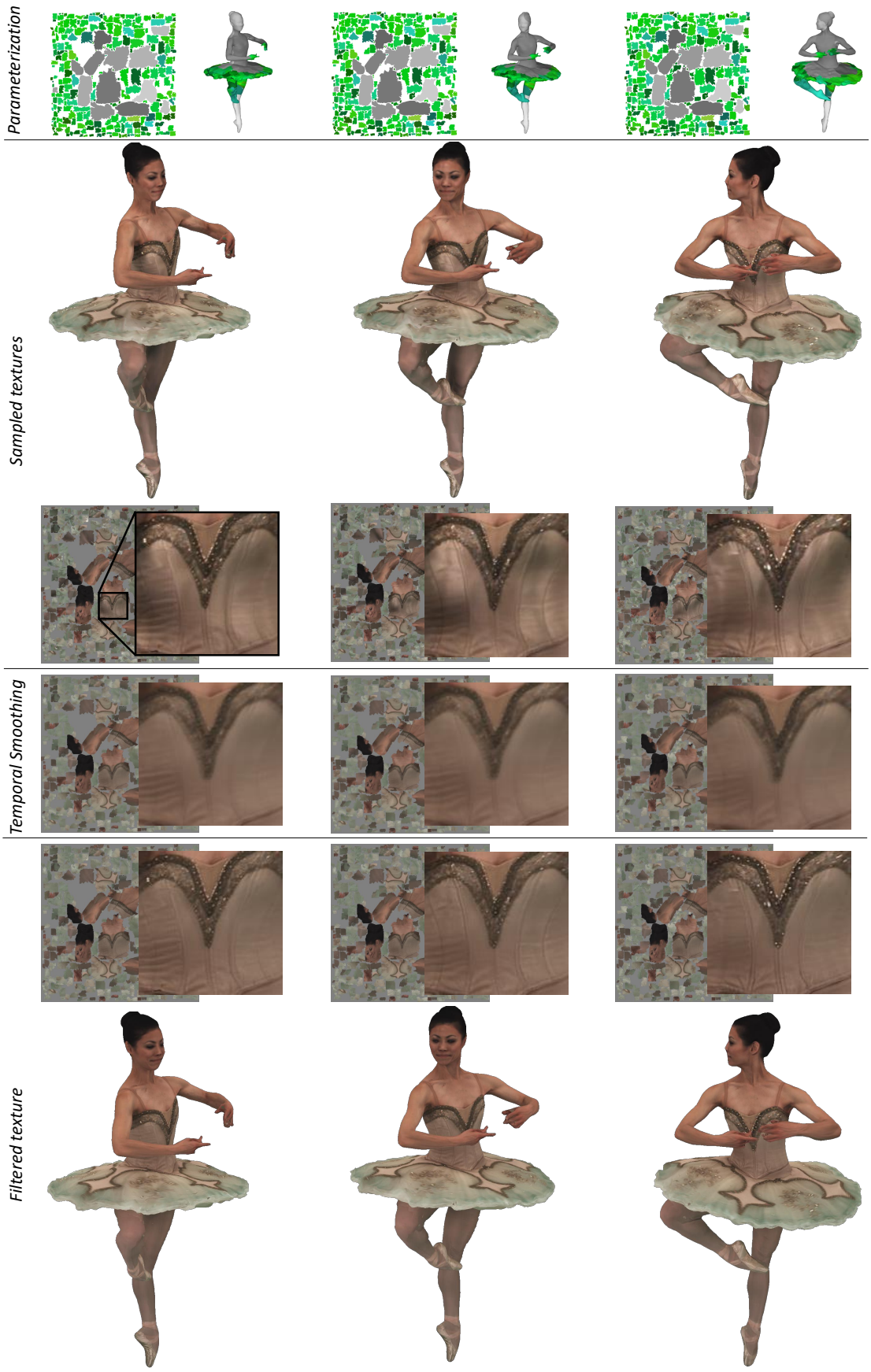


FIGURE 6.10: Evolving mesh filtering

Chapter 7

Conclusions

In this thesis we extended image processing techniques like Shock Filters, Optical Flow and Gradient-Domain processing to signals defined on static and dynamic surfaces. Our methods are robust to irregular connectivity, non-uniform sampling, and parametric distortion. We describe the main properties of our methods and the specific components that sustain them:

Metric awareness We parameterize 3D surfaces as 2D Riemannian manifolds with piecewise constant metrics. We use the Finite Elements Method (FEM) and Discrete Exterior Calculus (DEC) to construct linear operators that capture the geometry of function and vector spaces. FEM allows efficient construction of mass and stiffness matrices, sufficient to implement most of the applications described in our work. DEC provides a framework to operate on discrete differential forms, establishing a separation between combinatorial operators (e.g. exterior derivatives) and metric operators (e.g. Hodge stars). We use DEC to define smoothing operators on vector fields (the Hodge Laplacian) that preserve harmonics.

Quality preservation Signal degradation is a consequence of excessive resampling. This is particular common of explicit Eulerian integration methods. In this thesis, we preserved quality by using Lagrangian Integration (for Shock Filters and Optical Flow) and implicit Euler methods (for Gradient-Domain processing).

Efficient solution Fast signal processing on triangle meshes is specially hindered by irregular vertex connectivity. We leverage on texture atlas parameterization and the local regularity of the texture domain to accelerate signal processing on surfaces. We use domain decomposition to exploit memory coherence and parallelism in the interior

of charts, and a direct solver to handle arbitrary chart boundaries. We incorporate the domain decomposition in a multigrid method to reduce computation and accelerate convergence.

Temporal coherence Processing of temporal signals on surfaces requires information to flow across multiple frames to ensure temporal coherence (e.g. avoid flickering). Previous approaches solve for full surface correspondence on the temporal sequence as a prerequisite for signal processing. We argue that full correspondences are not always required, and (dense) partial correspondences can be sufficient. We construct a spatiotemporal atlas parameterization of an evolving mesh to capture (dense) partial correspondences, and show its usage for temporally coherent signal processing.

7.1 Open Problems

During the development of the methods introduced in this thesis we encountered very interesting problem that are worth to explore in future research. We highlight the most relevant problem within each chapter:

Chapter 3: Efficient signal advection on surfaces We represent intra-surface maps (i.e., within points in a fixed surface) as the flow induced by a vector field. Efficient integration of these vector field is an interesting computational problem. In particular, it is important to handle the skew generated by processing paths independently, and improve memory coherence on the triangle traversal.

Chapter 4: Hierarchical optical flow in meshes Our current formulation of optical flow on meshes uses a linear scale approach to handle large displacement. However, this requires to recompute the alignment vector field at the finest resolution, a task that is computationally expensive. Ideally, a multiresolution representation (e.g., via mesh simplification) should accelerate the process by solving first at coarser resolutions. Defining such multiresolution methods for vector fields is an open problem that is also relevant in other applications like vector field design and fluid simulation.

Chapter 5: Vector field processing in the texture atlas We would like to extend our seamless processing of signals in the texture atlas, to support seamless processing of vector fields in this same domain. In our current formulation we use the Whitney basis to represent vector fields. This has two major limitations: some elements of the basis

does not correspond to elements of the grid (node, edges, or cells) and it defines an overcomplete set (its size is four times the resolution of the grid rather than twice). Besides overcoming these limitations, a satisfactory vector field representation must enable the construction of robust curl and divergence operators, and must be easy to integrate. Finding appropriate representations of vector fields is a prerequisite for extending Shock Filters and Optical Flow to the texture atlas domain.

Chapter 6: Online atlas parameterization Our current approach to atlas parameterization of evolving meshes decouples tracking (online) from parameterization (offline). Having full knowledge of the tracking process allow us to optimize the charting and packing tasks. However, a more realistic (but harder) scenario is to update the atlas parameterization on the fly. We think that a compact atlas parameterization can still be constructed by customizing charting and packing, and developing statistical methods that anticipate remeshing.

Bibliography

[1] S. Osher and L. Rudin. Feature-oriented image enhancement using shock filters. *SIAM Journal of Numerical Analysis*, 27:919–940, 1990.

[2] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artif. Intell.*, 17(1-3):185–203, August 1981.

[3] Pedro Sander, John Snyder, Steven Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proc. ACM SIGGRAPH*, pages 409–416, 2001.

[4] M. Chuang, L. Luo, B. Brown, S. Rusinkiewicz, and M. Kazhdan. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Computer Graphics Forum (Symposium on Geometry Processing)*, pages 1475–1484, 2009.

[5] William Munger Boothby. *An introduction to differentiable manifolds and Riemannian geometry; 2nd ed.* Pure Appl. Math. Academic Press, Orlando, FL, 1986.

[6] M.P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice-Hall, 1976.

[7] Herbert Edelsbrunner. *A Short Course in Computational Geometry and Topology.* Springer Publishing Company, Incorporated, 2014.

[8] Mathieu Desbrun Peter Schrder Keenan Crane, Fernando de Goes. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

[9] Konrad Polthier and Eike Preu. Variational approach to vector field decomposition. In *Proc. Eurographics Workshop ON Scientific Visualization*, pages 147–156. Springer Verlag, 2000.

[10] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Trans. Graph.*, 26(3), July 2007.

[11] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *SIGGRAPH Asia 2015 Courses*, SA '15, pages 17:1–17:48, New York, NY, USA, 2015. ACM.

[12] Anil Nirmal Hirani. *Discrete Exterior Calculus*. PhD thesis, Pasadena, CA, USA, 2003. AAI3086864.

[13] L. Alvarez and L. Mazorra. Signal and image restoration using shock filters and anisotropic diffusion. *SIAM Journal of Numerical Analysis*, 31:590–605, 1994.

[14] G. Gilboa, N. Sochen, and Y. Zeevi. Regularized shock filters and complex diffusion. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *Computer Vision ECCV 2002*, volume 2350 of *Lecture Notes in Computer Science*, pages 399–413. Springer Berlin Heidelberg, 2002.

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

[16] Luis Alvarez, Pierre-Louis Lions, and Jean-Michel Morel. Image selective smoothing and edge detection by nonlinear diffusion. ii. *SIAM J. Numer. Anal.*, 29(3): 845–866, 1992.

[17] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceeddings of the International Conference on Computer Vision*, pages 839–846, 1998.

[18] P. Bhat, B. Curless, M. Cohen, and L. Zitnick. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *Proceedings of the 10th European Conference on Computer Vision*, pages 114–128, 2008.

[19] Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. Image smoothing via L_0 gradient minimization. *ACM Trans. Graph.*, 2011.

[20] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. Anisotropic geometric diffusion in surface processing. *Visualization Conference, IEEE*, 0:70, 2000.

[21] Chandrafit L. Bajaj, Guo Liang Xu, Rafit L. Bajaj, and Guoliang Xu T. Anisotropic diffusion of subdivision surfaces and functions on surfaces. *ACM Transactions on Graphics*, 22:4–32, 2002.

[22] Tolga Tasdizen, Ross Whitaker, Paul Burchard, and Stanley Osher. Geometric surface smoothing via anisotropic diffusion of normals. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 125–132, 2002.

[23] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Transactionson Graphics*, 22:950–953, 2003.

[24] B. Vallet and B. Lévy. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum (Proceedings Eurographics)*, 27:251–160, 2008.

[25] Raanan Fattal. Upsampling via imposed edges statistics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):to appear, 2007.

[26] Jian Sun, Zongben Xu, and Heung-Yeung Shum. Image super-resolution using gradient profile prior. *CVPR*, 2008.

[27] Qi Shan, Zhaorong Li, Jiaya Jia, and Chi-Keung Tang. Fast image/video upsampling. *ACM Trans. Graph.*, 2008.

[28] Gilad Freedman and Raanan Fattal. Image and video upscaling from local self-examples. *ACM Trans. Graph.*, 2010.

[29] J. Stam. Stable fluids. In *ACM SIGGRAPH Conference Proceedings*, pages 121–128, 1999.

[30] J. Stam. Flows on surfaces of arbitrary topology. *ACM Transactions on Graphics (SIGGRAPH '03)*, 22:724–731, 2003.

[31] Alexandre J. Chorin and J.E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer, 1993.

[32] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-S. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics*, 23:644–651, 2004.

[33] Justin Solomon, Keenan Crane, Adrian Butscher, and Chris Wojtan. A general framework for bilateral and mean shift filtering. *CoRR*, abs/1405.4734, 2014.

[34] Luis Alvarez, Joachim Weickert, and Javier Sánchez. Reliable estimation of dense optical flow fields with large displacements. *International Journal of Computer Vision*, 2000.

[35] Zhuoyuan Chen, Hailin Jin, Zhe Lin, Scott Cohen, and Ying Wu. Large displacement optical flow from nearest neighbor fields. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, 2013.

[36] E. C. Hildreth and C. Koch. The analysis of visual motion: From computational theory to neuronal mechanisms. *Annual Review of Neuroscience*, 10(1):477–533, 1987. doi: 10.1146/annurev.ne.10.030187.002401. PMID: 3551763.

[37] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision*, 92(1), March 2011.

[38] Jing Liao, Rodolfo S. Lima, Diego Nehab, Hugues Hoppe, Pedro V. Sander, and Jinhui Yu. Automating image morphing using structural similarity on a halfway domain. *ACM Trans. Graph.*

[39] Dhruv Mahajan, Fu-Chung Huang, Wojciech Matusik, Ravi Ramamoorthi, and Peter Belhumeur. Moving gradients: A path-based method for plausible image interpolation. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, 2009.

[40] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, 1981.

[41] F. Glazer. Multilevel relaxation in low-level computer vision. *Multi-Resolution Image Processing and Analysis*, pages 312–330, 1984.

[42] Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, 43(2):150 – 177, 1988.

[43] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

[44] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, 1995.

[45] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, 1996.

[46] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, 1997.

[47] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, 1999.

[48] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), July 2013.

[49] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 105–114, New York, NY, USA, 1998. ACM.

[50] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[51] Patrick Mullen, Pooran Memari, Fernando de Goes, and Mathieu Desbrun. Hot: Hodge-optimized triangulations. *ACM Trans. Graph.*, 30(4):103:1–103:12, July 2011.

[52] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Trans. Graph.*, 26(3), July 2007.

[53] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. *ACM Trans. Graphics*, 21:249–256, 2002.

[54] Tim Weyrich, Jia Deng, Connelly Barnes, Szymon Rusinkiewicz, and Adam Finkelstein. Digital bas-relief from 3D scenes. *ACM Trans. Graphics*, 26:32:1–32:7, 2007.

[55] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graphics*, 22:313–318, 2003.

[56] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *European Conf. Computer Vision*, pages 377–389, 2003.

[57] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graphics*, 23:294–302, 2004.

[58] Gabriel Taubin. A signal processing approach to fair surface design. In *ACM SIGGRAPH Conf. Proc.*, pages 351–358, 1995.

[59] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *ACM SIGGRAPH Conf. Proc.*, pages 317–324, 1999.

[60] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hand-Peter Seidel. Laplacian surface editing. In *Symposium on Geometry Processing*, SGP '04, pages 175–184, 2004.

[61] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proc. ACM SIGGRAPH*, Jul 2002.

[62] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic Parameterizations of Surface Meshes. *Computer Graphics Forum*, 2002.

[63] Francisco González and Gustavo Patow. Continuity mapping for multi-chart textures. *ACM Trans. Graphics*, 28:109:1–109:8, 2009.

[64] Marco Tarini. Volume-encoded uv-maps. *ACM Trans. Graph.*, 35(4):107:1–107:13, July 2016.

[65] Nathan Carr and John Hart. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graphics*, 21:106–131, 2002.

[66] Nathan Carr and John C. Hart. Painting detail. *ACM Trans. Graphics*, 23:845–852, 2004.

[67] Cem Yuksel. Mesh color textures. In *High Performance Graphics*, pages 17:1–17:11, 2017.

[68] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. *ACM Trans. Graphics*, 25:541–548, 2006.

[69] Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.*, 36:216:1–216:15, 2017.

[70] Chandrajit Bajaj and Guoliang Xu. Anisotropic diffusion of surfaces and functions on surfaces. *ACM Trans. Graphics*, 22:4–32, 2003.

[71] Ming Chuang, Szymon Rusinkiewicz, and Michael Kazhdan. Gradient-domain processing of meshes. *J. Computer Graphics Techniques*, 5:44–55, 2016.

[72] Paul Heckbert. Introduction to finite element methods. In *ACM SIGGRAPH 1993 Courses*, 1993.

[73] Fabián Prada, Misha Kazhdan, Ming Chuang, and Hugues Hoppe. Gradient-domain processing within a texture atlas. *ACM Trans. Graph.*, 37(4), August 2018.

[74] Alain Bossavit. Whitney forms: a class of finite elements for three-dimensional computations in electromagnetism. *IEE Proc. A (Physical Science, Measurement and Instrumentation, Management and Education, Reviews)*, 135:493–500, 1988.

[75] David Day and Mark Taylor. A new 11 point degree 6 cubature formula for the triangle. *Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting*, 7:1022501–1022502, 2007.

[76] Barry Smith, Petter Bjørstad, and William Gropp. *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations.* Cambridge Univ. Press, 1996.

[77] William Briggs, Van Emden Henson, and Steve McCormick. *A multigrid tutorial (2nd ed.).* Society for Industrial and Applied Mathematics, 2000.

[78] Christian Weiss, Wolfgang Karl, Markus Kowarschik, and Ulrich Rüde. Memory characteristics of iterative methods. In *1999 ACM/IEEE Conf. Supercomputing*, 1999.

[79] Yanqing Chen, Timothy Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35:22:1–22:14, 2008.

[80] Cosmin Petra, Olaf Schenk, and Mihai Anitescu. Real-time stochastic optimization of complex energy systems on high-performance computers. *Computing in Science & Engineering*, 16(5):32–42, 2014.

[81] Cosmin G. Petra, Olaf Schenk, Miles Lubin, and Klaus Gäertner. An augmented incomplete factorization approach for computing the schur complement in stochastic optimization. *SIAM J. on Scientific Computing*, 36:C139–C162, 2014.

[82] Jason Smith and Scott Schaefer. Bijective parameterization with free boundaries. *ACM Trans. Graph.*, 34:70:1–70:9, 2015.

[83] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven Gortler. A local/global approach to mesh parameterization. In *Symposium on Geometry Processing*, pages 1495–1504, 2008.

[84] C.J. Pfeifer. Data flow and storage allocation for the PDQ-5 program on the Philco-2000. *Communications of the ACM*, 6:365–366, 1963.

[85] C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss. Cache optimization for structured and unstructured grid multigrid. *Electronic Trans. on Numerical Analysis*, 10:21–40, 2000.

[86] Microsoft. Uvatlas: isochart texture atlasing . https://github.com/Microsoft/UVAtlas, 2018.

[87] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graphics*, 32:152:1–152:11, 2013.

[88] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *20th Annual Conf. Computer Graphics and Interactive Techniques*, pages 263–270, 1993.

[89] Christian Teitzel, Roberto Grosso, and Thomas Ertl. Line integral convolution on triangulated surfaces. In *Conf. World Society for Computer Graphics 1997*, pages 572–581, 1997.

[90] Jonathan Palacios and Eugene Zhang. Interactive visualization of rotational symmetry fields on surfaces. *IEEE Trans. Visualization and Computer Graphics*, 17: 947–955, 2011.

[91] Udo Diewald, Tobias Preuber, and Martin Rumpf. Anisotropic diffusion in vector field visualization on euclidean domains and surfaces. *IEEE Transactions on Visualization and Computer Graphics*, pages 139–149, 2000.

[92] Hao Li, Robert W. Sumner, and Mark Pauly. Global correspondence optimization for non-rigid registration of depth scans. In *Symposium on Geometry Processing*, 2008.

[93] Meinard Müller and Tido Röder. Motion templates for automatic classification and retrieval of motion capture data. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 137–146. Eurographics Association, 2006.

[94] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, July 2002.

[95] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, August 2004.

[96] Chris Budd, Peng Huang, Martin Klaudiny, and Adrian Hilton. Global non-rigid alignment of surface sequences. *Int. J. Comput. Vision*, 102, 2013.

[97] Hao Li, Linjie Luo, Daniel Vlasic, Pieter Peers, Jovan Popović, Mark Pauly, and Szymon Rusinkiewicz. Temporally coherent completion of dynamic shapes. *ACM Trans. Graph.*, 31(1), February 2012.

[98] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4):69:1–69:13, July 2015.

[99] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3), July 2009.

[100] Morten Bojsen-Hansen, Hao Li, and Chris Wojtan. Tracking surfaces with evolving topology. *ACM Trans. Graph.*, 31(4), July 2012.

[101] Fabián Prada, Misha Kazhdan, Ming Chuang, Alvaro Collet, and Hugues Hoppe. Spatiotemporal atlas parameterization for evolving meshes. *ACM Trans. Graph.*, 36(4):58:1–58:12, July 2017.

Fabián Andrés Prada Niño

Bucaramanga, Colombia. 14 November 1989

fabianprada@gmail.com

http://www.cs.jhu.edu/∼fpradan1

Education

- **Johns Hopkins University** — Baltimore, MD
  PhD in Computer Science — *2013-2018*
- **Instituto Nacional de Matmetica Pura e Aplicada** — Rio de Janeiro, Brazil
  Master of Science in Mathematics, Computer Graphics option — *2011-2013*
- **Universidad de Los Andes** — Bogotá, Colombia
  Bachelor of Science in Mathematics, Summa Cum Laude — *2007-2011*

Professional Experience

- **Johns Hopkins University** — Baltimore, MD
  Research and Teaching Assistant — *2013 - Present*
  - Instructor of *Introduction to Geometry Processing.*
  - Teaching assistant of *Computer Graphics* and *Computational Geometry.*
- **Microsoft Research** — Redmond, WA
  Research Intern — *Summer 2015 and 2016*
  - Similarity detection and transition synthesis for stochastic playback of 4D videos.
  - Tracking and parameterization of scanned human performances for 4D video compression.
- **Colombian Mathematical Olympiad** — Bogotá, Colombia
  Organizing Committee Assistant — *2009 - 2011*
  - Tutor of the Colombian delegation at 13th Centroamerican Mathematical Olympiad. Mexico, 2011.
  - Problem coordinator at 54th International Mathematical Olympiad. Colombia, 2013.
- **Universidad de Los Andes** — Bogotá, Colombia
  Teaching Assistant — *2010*
  - Teaching assistant of *Integral Calculus* and *Linear Algebra.*

Publications

- F. Prada, M. Kazhdan, M. Chuang, and H. Hoppe. *Gradient-Domain Processing within a Texture Atlas.* SIGGRAPH 2018.
- F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe. *Spatiotemporal Atlas Parameterization for Evolving Meshes.* SIGGRAPH 2017.
- F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe. *Motion Graphs for Unstructured Textured Meshes.* SIGGRAPH 2016.
- F. Prada, and M. Kazhdan. *Unconditionally Stable Shock Filters for Image and Geometry Processing.* SGP 2015.
- F. Prada, L. Cruz, and L. Velho. *Improving Object Extraction With Depth-Based Methods.* CLEI 2013.

Awards

- **Ramón de Zubiría**: Highest GPA in the School of Science. Universidad de Los Andes, 2009.
- **ICFES**: Highest score in college admission test. Ministry of Education of Colombia, 2006.
- **Honorable Mention**: Ranked 254th among 498 high school students at the 47th International Mathematical Olympiad. Ljubljana, Slovenia 2006.