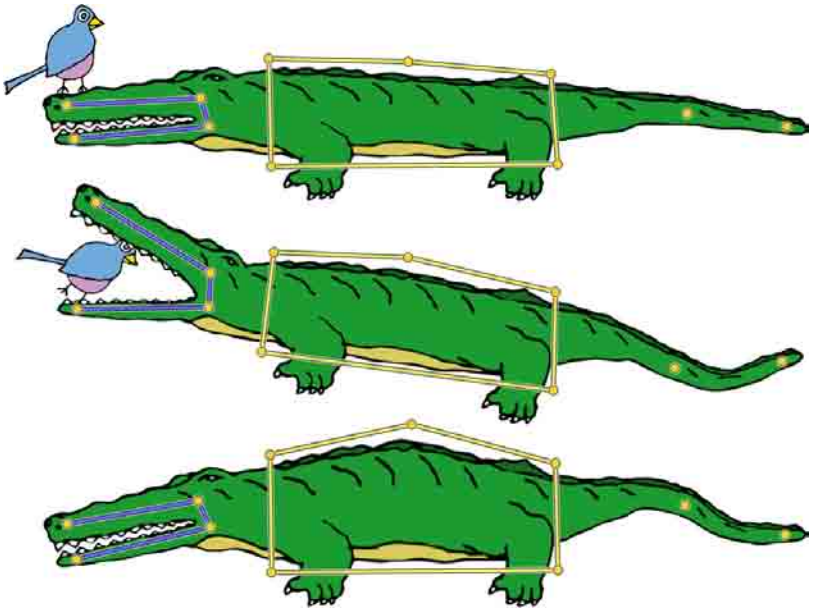
Diss. ETH N° 21189

Algorithms and Interfaces for Real-Time Deformation of 2D and 3D Shapes



Alec Jacobson

PhD Thesis
May 2013

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

igl
INTERACTIVE GEOMETRY LAB

Diss. ETH N^{o} 21189

ALGORITHMS AND INTERFACES FOR REAL-TIME DEFORMATION OF 2D AND 3D SHAPES

DISSERTATION

Submitted to

ETH ZURICH

for the degree of

DOCTOR OF SCIENCES

presented by

ALEC JACOBSON

M.A., New York University

November 4, 1987

citizen of
The United States of America

accepted on the recommendation of

Prof. Dr. Olga Sorkine-Hornung
Dr. Jovan Popović
Prof. Dr. Mario Botsch

2013

Abstract

This thesis investigates computer algorithms and user interfaces which assist in the process of deforming raster images, vector graphics, geometric models and animated characters in real time. Many recent works have focused on deformation quality, but often at the sacrifice of interactive performance. A goal of this thesis is to approach such high quality but at a fraction of the cost. This is achieved by leveraging the geometric information implicitly contained in the input shape and the semantic information derived from user constraints.

Existing methods also often require or assume a particular interface between their algorithm and the user. Another goal of this thesis is to design user interfaces that are not only ancillary to real-time deformation applications, but also endowing to the user, freeing maximal creativity and expressiveness.

This thesis first deals with discretizing continuous Laplacian-based energies and equivalent partial differential equations. We approximate solutions to higher-order polyharmonic equations with piecewise-linear triangle meshes in a way that supports a variety of boundary conditions. This mathematical foundation permeates the subsequent chapters. We aim this energy-minimization framework at skinning weight computation for deforming shapes in real-time using linear blend skinning (LBS). We add additional constraints that explicitly enforce boundedness and later, monotonicity. We show that these properties and others are mandatory for intuitive response. Through the boundary conditions of our energy optimization and tetrahedral volume meshes we can support all popular types of user control structures in 2D and 3D. We then consider the skeleton control structure specifically, and show that with small changes to LBS we can expand the space of deformations allowing individual bones to stretch and twist without artifacts. We also allow the user to specify only a subset of the degrees of freedom of LBS, automatically choosing the rest by optimizing nonlinear, elasticity energies within the LBS subspace. We carefully manage the complexity of this optimization so that real-time rates

are undisturbed. In fact, we achieve unprecedented rates for nonlinear deformation. This optimization invites new control structures, too: shape-aware inverse kinematics and disconnected skeletons. All our algorithms in 3D work best on volume representations of solid shapes. To ensure their practical relevancy, we design a method to segment inside from outside given a shape represented by a triangle surface mesh with artifacts such as open boundaries, non-manifold edges, multiple connected components and self-intersections. This brings a new level of robustness to the field of volumetric tetrahedral meshing.

The resulting quiver of algorithms and interfaces will be useful in a wide range of applications including interactive 3D modeling, 2D cartoon keyframing, detailed image editing, and animations for video games and crowd simulation.

Zusammenfassung

Diese Dissertation erforscht Computer-Algorithmen und Benutzerschnittstellen, die mit dem Prozess der Deformation der Vektorgrafiken, der geometrischen Modelle, und der animierten Figuren helfen. Viele vorgeschlagene Methoden konzentrieren sich auf Deformation-Qualität ohne Rücksicht auf interaktive Leistung. Ein Ziel dieser Dissertation ist hohe Qualität bei Deformationen in vernünftigen Rechenzeiten zu bekommen. Wir werden unser Ziel mit der Kombination von der geometrischen Information der Eingabe-Figuren und der semantischen Information der vom Benutzer eingegebenen Bedingungen erreichen.

Existierende Methoden brauchen eine spezifische Benutzerschnittstelle zwischen den Algorithmen und dem Benutzer. Ein anderes Ziel dieser Dissertation ist Benutzerschnittstellen zu erstellen, die nicht nur hilfreich für Echtzeit-Deformationen sind, sondern auch maximale Kreativität und Ausdrucksfähigkeit inspirieren.

Diese Dissertation beginnt mit der Diskretisierung von stetigen Laplacian-basierten Energien und der äquivalenten partiellen Differentialgleichungen. Wir nähern Lösungen für polyharmonische Gleichungen höherer Ordnungen mit stückweise-linearen Funktionen auf Dreiecksnetze so an, dass viele Randbedingungen unterstützt werden können. Diese mathematische Grundlage dringt durch die nächsten Kapiteln durch. Wir verwenden diese Energieoptimierung für die Berechnung von Linear Blend Skinning (LBS) Gewichten. Wir fügen andere Nebenbedingungen hinzu, die die Variablen begrenzen und später Monotonie erzwingen. Wir zeigen, dass diese und andere Eigenschaften obligatorisch für intuitives Kontrollieren sind. Mit unseren Randbedingungen zur Energieoptimierung und volumetrischen Tetraeder-Netzen unterstützen wir alle populären Benutzerschnittstellen in 2D und 3D.

Als nächstes betrachten wir die besondere Benutzerschnittstelle für Knochengerüste (*skeleton*). Wir zeigen, dass der Raum der möglichen Deformationen durch kleine Änderungen zu LBS erweitert werden kann. Einzelne Knochen können ohne Artefakte verdreht und gedehnt werden.

Wir erlauben auch, dass die Benutzerin nur eine Untermenge der Freiheitsgrade der LBS angibt, und die Übrigen werden automatisch durch eine nichtlineare, elastische Energieoptimierung in dem Unterraum der LBS erzeugt. Wir gehen mit der Komplexität dieser Optimierung vorsichtig um, damit wir immer noch in Echtzeit arbeiten können. Auf diese Weise realisieren wir beispiellose Frameraten für Echtzeit Deformationen. Dieses Optimierungsverfahren erlaubt neue Benutzerschnittstellen wie Formen-bewusste inverse Kinematik und disjunkte Knochengerüste. Unsere Algorithmen in 3D funktionieren am besten mit volumetrisch festen Formdarstellungen. Für die praktische Relevanz dieser Algorithmen schlagen wir auch eine Methode vor, die das Innere und das Äussere eines Dreieckenetzes mit Artefakten wie Löcher, vielen zusammenhängenden Räumen und nicht-Mannigfaltigen Kanten segmentieren und volumetrisch feste Forme erzeugen kann. Diese Methode bringt ein neues Maß der Robustheit zur Schaffung von volumetrischen Tetraeder-Netzen.

Die resultierende Sammlung von Algorithmen und Benutzerschnittstellen ist nützlich für viele Anwendungen wie interaktive 3D Modellierung, 2D Schlüsselbildanimation, detaillierte Bildbearbeitung und Animationen in Videospielen und Gruppensimulationen.

Acknowledgments

This thesis is dedicated to Annie Ytterberg. She is my ambition, my muse, my idol, and my best friend.

As a child, my mother gave me math homework over the summer breaks and extra homework during the school year. She always said I would thank her later. I said I never would. Mom, thank you. I admire my father as a model scientist and researcher. If you can only be smarter by surrounding yourself by people smarter than you, then I am triply lucky to have my brother, Seth, and my sisters, Tess & Elli.

Thank you, Kendalle Jacobson, for sending me cards all my life, even while I live overseas. My grandfather, via the ancient philosophy of cynicism, has made my research beliefs more concrete.

Denis Zorin encouraged me to undertake this PhD and introduced me to my advisor. He has helped me every step of the way. His openness to new ideas and patience with lesser minds are inspirational.

My internship with Jovan Popović in David Salesin's Creative Technologies Lab at Adobe was a turning point in my research. I am thankful to both for giving me this opportunity. Jovan has been not just a collaborator and coauthor, but also a role-model, mentor, and advisor.

Thank you, Jovan and Mario Botsch, for refereeing my examination and reviewing my thesis.

I would have published nothing without my coauthors. I have enjoyed collaborating with all of them: Ladislav Kavan, Ilya Baran, Alex Sorkine-Hornung, Kaan Yücer, Tino Weinkauf, David Günther, Jan Reininghaus.

I am fortunate to have worked on my PhD in two labs. My years at the Media Research Lab in the Courant Institute at NYU were formative to my research in many ways. Conversations

with Ken Perlin and Murphy Stein kept my mind open and engaged. My officemate, Ofir Weber, taught me immeasurably. I am also thankful to the others at MRL: Yotam Gingold, James Zhou, David Harmon, Rob Fergus, Denis Kovacs, Ashish Myles, Yang Song.

When I arrived at ETH Zurich, the Interactive Geometry Lab doubled in size. For those first few months Markus Gross's Computer Graphics Lab and the Disney Research Zurich group were my new families. I am thankful for the welcoming environment cultivated by Markus and his PhD students, postdocs and researchers: Thomas Oskam, Masi Germann, Tanja Käser, Claudia Kuster, Jean-Charles Bazin, Tiberiu Popa, Sebi Martin, Peter Kaufman, Barbara Solenthaler, Tobias Pfaff. I am also thankful to the intellectual coffee breaks with newer members: Alex Chapiro, Changil Kim, Pascal Bérard, Fabian Hahn, Mélina Skouras. Of course, IGL is growing now, and I have learned and benefited so much from its members: Olga Diamanti, Daniele Panozzo, Kenshi Takayama, Emily Whiting, Leo Koller-Sacht.

Many administrators have made my PhD easier. I am thankful to Marianna Berger, Sarah Disch, Hong Tam and Denise Spicher.

Dr. Željko Bajzer of the Mayo Clinic gave me my first introduction to scientific and mathematical research.

My friends Brandon Reiss and Dan Leventhal politely listened to my harebrained ideas.

Our papers would be uglier without the help of Maurizio Nitti and Felix Hornung.

I am indebted to my funders: Intel, SNF, Adobe Systems Inc. and the MacCracken Fellowship.

The enormous public database of old photographs provided by US Library of Congress inspires my research.

I have been fortunate to have the opportunity to discuss my research with many great minds in computer graphics as they visited our lab. Conversations with Craig Gotsman, Kai Horman, Tamy Boubekeur, Jernej Barbić, Leo Guibas, Pierre Alliez, and Peter Schröder have been invaluable.

Giving presentations to other groups has been a delight, and I am very thankful to Baoquan Chen, Konrad Polthier, Remo Ziegler and Enrico Puppo for inviting me into their labs.

I am grateful to fellow researchers who shared their code and kindly responded to requests and questions throughout my thesis, notably Hang Si, Marco Attene, Scott Schaefer, Pascal Frey, Philippe Decaudin and Eftychios Sifakis.

I would like to thank all the peer reviewers who anonymously improved my work over the years. Countless insights are due to perspectives gained while reading our reviewers. Though we often accommodated required changes begrudgingly at the time, I am confident our publications and this thesis are better as a result.

Of course, this thesis would not exist without my advisor Olga Sorkine-Hornung. She has taught me so many things. Her enthusiasm for new research and scientific rigor are reassuring and inspiring. I am forever thankful to be her student.

Contents

1

Introduction

> *We do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism [Phong 1975].*

Thirty-eight years later, textures and shadows have long been incorporated into rendering pipelines. However, the continued ubiquity of the Phong shading and Phong lighting model is evidence of their success as computer graphics algorithms. Phong's algorithms incorporate principles relevant to the entirety of computer graphics research. Our algorithms should:

1. approximate human perception of reality rather than reality directly,
2. prefer models with a simple mathematical result,
3. execute efficiently on a computer, and
4. be easy for humans to understand and control.

As we turn to the topic of this thesis, shape deformation, we will remember these as tenants to which our solutions should also adhere. With these in mind we remain faithful both to the audience of computer graphics (film-goers, video-gamers) and the authors of computer graphics (artists, designers).

1.1 Shape deformation

The research field of geometry processing studies the entire life of a shape. A shape is born by geometry acquisition (e.g. scanning a physical object) or by creation with a modeling tool (e.g.

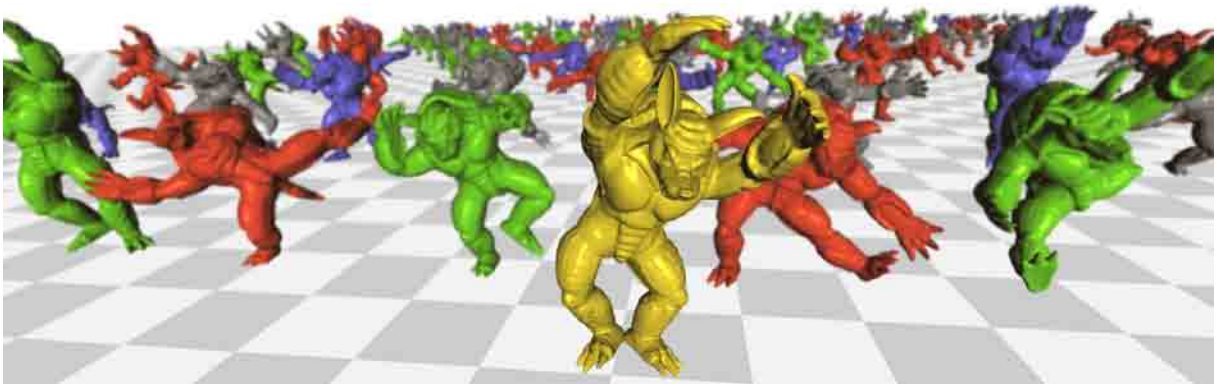


Figure 1.1: *Our algorithms form a framework that can deform 100* Armadillos*, each 80,000 triangles at 30 frames per sec. The deformations minimize volumetric, nonlinear, elasticity energies defined over a tetrahedral mesh, but reduced to the LBS subspace using our automatically computed bounded biharmonic weights.*

the advanced AUTODESK MAYA software). The life of a shape ends with consumption. It is rendered on screen, e.g. in film or video game, or it is fabricated into a physical object. Throughout the other stages of this geometry processing pipeline, a shape is analyzed and manipulated.

One of those manipulations stages is articulated shape deformation, where a *user* is directing changes to a shape's geometry to achieve a specific goal. The direction may be either explicit, for example, fixing regions of the shape to certain positions; or implicit, simulating the elastic behavior of a gummy bear. The goal may even be unknown or ill-posed when the process is begun, for example, modeling novel poses of a dancing hippopotamus. Such articulated shape deformation is fundamental to computer graphics applications such as 3D character animation and interactive shape modeling. If we treat images as planar geometry (often just rectangles) with color attributes attached to points in 2D space (pixels), then shape deformation is also fundamental to image manipulation, a subfield of the related field of image processing.

Tasks like interactive modeling and graphic design require immediate visual feedback to facilitate user exploration of *shape space*: the typically unbounded space of all possible configurations of a given shape. Real-time performance becomes especially important when dealing with high-resolution data, such as megapixel images or scanned 3D geometry with millions of vertices. Applications like crowd simulation and video games demand that deformations be computed repeatedly for many objects on screen simultaneously. Previous work has shown that it is difficult to achieve both high deformation quality and high framerates. In this thesis, we will develop deformation algorithms that approach the quality of more computationally expensive methods but remain inherently simple and extremely fast.

Broadly, interactive shape deformation is also the task of *assisting* the user to reconfigure a shape. In the case of surface mesh deformations, the user could manually reposition each mesh vertex, but this is unnecessarily tedious. The space of all possible positions for every vertex of a mesh is much larger than the space of coherent reconfigurations. Rather, we would like the user to only provide a few, high-level constraints like "move the feet here", "turn the head 90 degrees" or "stretch the ears to there." The rest of the shape should immediately deform in an intuitive manner. We develop algorithms that efficiently navigate in the space of coherent deformations which meet the user's constraints.

Exactly defining what is an "intuitive", "coherent", or "high quality" deformation in a mathematical way is, in general, impossible. Rewording these demands in terms of an energy minimization, we could say that ideally our deformation should *minimize user surprise* [Gingold 2008]. We approximate this ill-posed and highly subjective energy with objective, mathematical quantities which directly measure properties like surface curvature or local rigidity. We prioritize quantities derived from continuous differential geometry, which when carefully discretized lead to practical assurances with respect to discretization.

Variational techniques provide a straightforward setup for modeling the deformation problem, but complicated objective terms and nonlinear constraints quickly lead to slow optimizations, far from real-time. Rather than minimize deformation energies directly on the shape, we build our foundation around Linear Blend Skinning (LBS) and point our high-powered optimizations at the skinning weight functions and transformations. LBS is a time-tested standard for real-time deformation, dating to at least [Magnenat-Thalmann et al. 1988], but probably used frequently before [Badler and Smoliar 1979, Magnenat-Thalmann and Thalmann 1987]. Despite its known artifacts, LBS remains a popular mechanism for shape deformation because it is simple to implement and fast at runtime. New shape positions are computed as a linear combination of a small number of affine transformations:

$$\mathbf{x}'_j = \sum_{i=0}^{m} w_i(\mathbf{x}_j)\, \mathbf{T}_i \begin{pmatrix} \mathbf{x}_j \\ 1 \end{pmatrix}. \tag{1.1}$$

Once the weight functions w_i and transformations matrices $\mathbf{T}_i$ are defined, the deformation of a point on a shape from its rest position $\mathbf{x}_j$ to a new position $\mathbf{x}'_j$ is simple and embarrassingly parallel. LBS is easily integrated into the larger computer graphics pipeline using GPU vertex shaders, where vertices are deformed immediately before rendering at only a small overhead. However, a large amount of time and effort is spent by specialized *riggers* to paint the weight functions w_i manually. A core contribution of this thesis is to define w_i functions automatically based on an input shape and control structure (e.g. internal skeleton).

We first investigate discretizations of continuous quadratic energies resulting in high quality deformations of surfaces represented as triangle meshes (Chapter 3). We focus on quadratic energies corresponding to polyharmonic equations: $\Delta^k u = 0$. Constraints may then be added to similar optimizations in order to design LBS weight functions automatically, achieving a number of important qualities necessary for high quality *and intuitive* deformation (Chapters 4 & 5). By utilizing these optimized weights and making small changes to the LBS formula, we may then greatly expand the space of deformations achievable in real time (Chapter 6). We also employ further energy optimization to assist the user in defining the LBS transformation matrices based on just a small set of constraints (Chapter 7). Our optimizations never interfere with the runtime performance (see Figure 1.1). We do this by maximizing precomputation and by exploiting LBS as a limited, but potentially high quality subspace of all possible deformations.

We focus on deforming shapes described by popular parametric representations: surfaces are triangle meshes and solids are tetrahedral meshes. Often only the triangle mesh of the surface of a solid shape is given. To ensure practical relevance of our deformation methods for solids, we design a method to robustly determine inside from outside given a triangle mesh with imperfections like holes, self-intersections and non-manifold components (Chapter 8).

How the user interacts with a particular deformation method is arguably just as important as

the underlying mathematics. Methods that restrict a user to a particular control structure (e.g. an internal skeleton or exterior cage) place a burden on the user that permeates the entire process (Chapter 4). To this end, this thesis investigates the appropriateness of proposed control structures for given tasks. Hypothesizing that all existing structures have advantages and are thus useful options to the user, we design algorithms which treat the choice of control structure as a first-class input parameter. We also reduce the amount of necessary control, allowing the user to make high level directions with minimal effort (Chapter 7). In this way, the original control structure and associated LBS weights not only define a more manageable optimization search space, but also encode semantic information, otherwise not available when crafting a deformation energy.

1.2 Organization

This thesis is organized around seven core chapters: one background chapter and six chapters associated with interim publications of this thesis's contributions (for a complete list of publications see Section 9.2). The content of these six chapters largely mirrors the respective publications, but more details, derivations and results are supplied for each. Errata are corrected and connections are drawn between works which would have been otherwise anachronistic. The one-column format of this thesis better facilitates large figures and mathematical derivations than the typical two-column format of graphics journal publications (e.g. Transactions on Graphics, Computer Graphics Forum). Rather than inconspicuously slip significant, previously unpublished material in the middles of these six chapters, such larger discussions will instead appear as supplemental appendices following the main text in each chapter.

2

Mathematical foundation

The chapters in this thesis will follow a similar story. First the goal will be stated in English. This goal will be approximated by concretely defining a mathematical energy that attempts to measure deviation from this goal. The energy is defined in terms of the input geometry. An attempt will be made to define this energy in a way that is derived from continuous quantities. Once discretized, the result is a system of differential equations.

This chapter contains the background information about discretization and energy minimization necessary for understanding and reproducing the technical contributions in following chapters. While it is assumed that linear algebra, trigonometry and calculus are known to the reader, we will make an effort to provide simple in-place derivations in this chapter. These are useful not only as a reference, but also to provide insights and heighten our intuition.

As a warm up, we first consider a very common problem in computer graphics and mathematics at large: the Dirichlet problem. Variants of this problem appear in each of the six core chapters of this thesis.

2.1 Dirichlet energy and the Laplace equation

The Dirichlet energy $E_D(u)$ is a quadratic functional that measures how much a differentiable function u changes over a domain Ω:

$$E_D(u) = \int_\Omega \|\nabla u(\mathbf{x})\|^2 d\mathbf{x}. \tag{2.1}$$

Colloquially we can say that minimizers of the Dirichlet energy are *as-constant-as-possible*.

The corresponding Euler-Lagrange equation is the Laplace equation, a second-order partial differential equation (PDE):

$$\Delta u = 0. \tag{2.2}$$

Solutions of the Laplace equation are called harmonic functions and enjoy a host of special properties. Besides minimizing the Dirichlet energy in Equation (2.1), harmonic functions:

1. are uniquely defined by their value on the boundary of the domain $\partial\Omega$,
2. obey the maximum principle,
3. and obey the mean-value property.

These functions and their higher-order cousins (biharmonic, triharmonic, etc.) will prove very useful in shape deformation algorithms as they help approximate surface fairness and intuitive deformation behavior. They are smooth and may be efficiently computed numerically for a given set of boundary values (realizations of user constraints).

When solving the Dirichlet boundary value problem, we seek solutions to the optimization problem:

$$\arg\min_u \int_\Omega \|\nabla u\|^2 dA \tag{2.3}$$

$$\text{subject to } \; u|_{\partial\Omega} = u_0. \tag{2.4}$$

or equivalently

$$\text{solve } \; \Delta u|_\Omega = 0 \tag{2.5}$$

$$\text{with } \; u|_{\partial\Omega} = u_0. \tag{2.6}$$

For the sake of generality we will consider the Poisson equation with a nonzero right-hand side:

$$\text{solve } \; \Delta u|_\Omega = f \tag{2.7}$$

$$\text{with } \; u|_{\partial\Omega} = u_0. \tag{2.8}$$

We can write Equation (2.7) in its equivalent *weak form*:

$$\int_\Omega \Delta u \, v \, dA = \int_\Omega f v \, dA, \; \forall v \text{ s.t. } \; v|_{\partial\Omega} = 0, \tag{2.9}$$

where v is an arbitrary test function. We can then apply integration by parts and rewrite the left-hand side:

$$\int_\Omega \Delta u \, v \, dA = \cancel{\oint_{\partial\Omega} \frac{\partial u}{\partial n} v \, ds} - \int_\Omega \nabla u \, \nabla v \, dA, \tag{2.10}$$

where the boundary term may be ignored if we choose our test functions v such that $v|_{\partial\Omega} = 0$. Note that now our equation only involves first derivatives. This means we can use piecewise linear functions as their first derivatives are defined *almost everywhere*: everywhere except a Lebesgue measure zero subset.

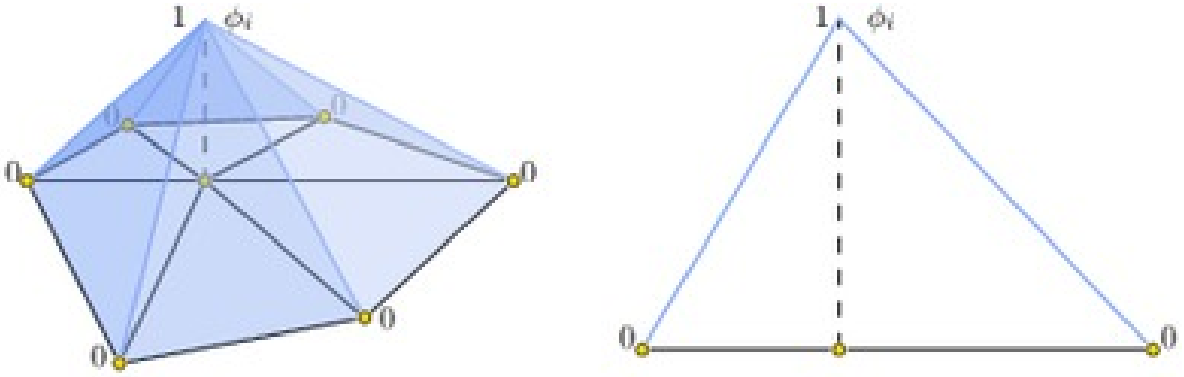


Figure 2.1: *Left: a piecewise-linear hat function visualized as a height field above a 2D triangle mesh. Right: a piecewise-linear hat function in 1D.*

We can discretize Equation (2.10) with the finite element method (FEM) by defining *hat functions* ϕ_i at each node or vertex of a simplicial mesh. For now, assuming $\Omega \subset \mathbb{R}^2$ we use hat functions defined over a triangle mesh (see Figure 2.1). Then we approximate the unknown function and test functions as:

$$u = \sum_{i \in I} u_i \, \phi_i, \tag{2.11}$$

$$v = \sum_{j \in I} v_j \, \phi_j, \;\; v_j|_{\partial\Omega} = 0, \tag{2.12}$$

where I is the set of all mesh vertices.

If Equation (2.9) holds for $v = \phi_j$ it also holds for any piecewise-linear v. So, it is sufficient to use all v such that $v = \phi_j$ for some j. Considering each of these gives a system of n equations with the jth equation being:

$$-\int_\Omega \nabla \left(\sum_{i \in I} u_i \, \phi_i \right) \nabla \phi_j dA = \int_\Omega \left(\sum_{i \in I} f_i \phi_i \right) \phi_j dA, \tag{2.13}$$

and moving the integrals, gradients and products inside the summations gives:

$$\sum_{i \in I} u_i \int_\Omega -\nabla \phi_i \, \nabla \phi_j dA = \sum_{i \in I} f_i \int_\Omega \phi_i \, \phi_j dA. \tag{2.14}$$

Again, this is the jth equation in a system of equations, each corresponding to an interior node $j \in I_{\Omega \setminus \partial\Omega}$, where $I = I_{\Omega \setminus \partial\Omega} \cup I_{\partial\Omega}$, with cardinalities $|I_{\Omega \setminus \partial\Omega}| = n$ and $|I_{\partial\Omega}| = n_b$.

Because the values of u are known on the boundary we move these to the right-hand side:

$$\sum_{i \in I_{\Omega \setminus \partial\Omega}} u_i \int_\Omega -\nabla \phi_i \, \nabla \phi_j dA = \sum_{i \in I} f_i \int_\Omega \phi_i \, \phi_j dA + \sum_{i \in I_{\partial\Omega}} u_i^0 \int_\Omega \nabla \phi_i \, \nabla \phi_j dA. \tag{2.15}$$

By declaring that:

$$\mathbf{L}_{ij} = \int_\Omega -\nabla\phi_i \, \nabla\phi_j dA \tag{2.16}$$

$$\mathbf{b}_j = \sum_{i\in I} f_i \int_\Omega \phi_i \, \phi_j dA + \sum_{i\in I_{\partial\Omega}} u_i^0 \int_\Omega \nabla\phi_i \, \nabla\phi_j dA, \tag{2.17}$$

we arrive at an $n \times n$ linear system in matrix form:

$$\mathbf{Lu} = \mathbf{b}, \tag{2.18}$$

where the system matrix $\mathbf{L}$ is called the *stiffness matrix* of this elliptic PDE.

2.1.1 Neumann boundary conditions

Up to computing the entries of $\mathbf{L}$ and $\mathbf{b}$ in Equations (2.16) & (2.17), we have sufficiently described the finite element method for solving a Poisson equation with Dirichlet boundary conditions (i.e. fixed boundary values). Now we will see that by altering the choice of test functions v we may enable Neumann boundary conditions (i.e. fixed boundary derivatives). The Poisson problem in Equation (2.7) rewritten with Neumann boundary conditions is:

$$\text{solve } \Delta u|_\Omega = f \tag{2.19}$$

$$\text{with } \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = g, \tag{2.20}$$

where n is the outward pointing normal vector along the boundary. Derivative control like this will become an especially powerful tool when defining bi-, triharmonic functions where derivatives are defined in combination with Dirichlet boundary conditions: the combination called Cauchy boundary conditions (see Chapter 3).

Recall that for Dirichlet boundary conditions we required that the test functions v vanished on the boundary. That is, in Equation (2.10) we had

$$\int_\Omega \Delta u \, v \, dA = \oint_{\partial\Omega} \underbrace{\frac{\partial u}{\partial n}}\, v \, ds - \int_\Omega \nabla u \, \nabla v \, dA. \tag{2.21}$$

$$\text{might not be zero, so we needed } v|_{\partial\Omega} = 0 \tag{2.22}$$

However, if the unknown function satisfies Neumann conditions then we may use test functions with nonzero boundary values:

$$\int_\Omega \Delta u \, v \, dA = \oint_{\partial\Omega} \underbrace{\frac{\partial u}{\partial n}}\, v \, ds - \int_\Omega \nabla u \, \nabla v \, dA. \tag{2.23}$$

$$\text{because } \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = g \text{ this is known, move to right-hand side} \tag{2.24}$$

So, the jth equation in our system becomes:

$$\sum_{i \in I} u_i \int_\Omega -\nabla \phi_i \, \nabla \phi_j dA = \sum_{i \in I} f_i \int_\Omega \phi_i \, \phi_j dA + \sum_{i \in I_{\partial\Omega}} g_i \oint_{\partial\Omega} \phi_i \, \phi_j ds. \tag{2.25}$$

Because of the choice of test functions, we notice that we have nontrivial equations corresponding to $j \in I_{\partial\Omega}$. Thus, whereas for Dirichlet boundary conditions in Equation (2.15) we had n equations, for Neumann boundary conditions we have $n + n_b$ equations. This, of course, makes sense because we need to solve for u not just for the n interior nodes but also for the n_b boundary nodes.

2.1.2 Cotangent weights

By defining ϕ_i as piecewise-linear hat functions, the values in the system matrix $\mathbf{L}_{ij}$ are uniquely determined by the geometry of the underlying mesh. These values are famously known as *cotangent weights*. "Cotangent" because, as we will shortly see, of their trigonometric formulae and "weights" because as a matrix $\mathbf{L}$ they define a weighted graph Laplacian for the given mesh. Graph Laplacians are employed often in geometry processing, and often in discrete contexts ostensibly disconnected from FEM (e.g. [Floater 2003, Sorkine et al. 2004, Zhou et al. 2005]). The choice or manipulation of Laplacian weights and subsequent use as a discrete Laplace operator has been a point of controversy in geometry processing research [Grinspun et al. 2006, Wardetzky et al. 2007b].

Though the cotangent formulae resulting from:

$$\mathbf{L}_{ij} = \int_\Omega -\nabla \phi_i \, \nabla \phi_j dA, \tag{2.26}$$

in Equation (2.16) are long known [MacNeal et al. 1949], it is worth clearly explaining its derivation here as it provides geometric insights.

We first notice that $\nabla \phi_i$ are constant on each triangle, and only nonzero on triangles incident on node i. For such a triangle, T_α, this $\nabla \phi_i$ points perpendicularly from the opposite edge e_i with inverse magnitude equal to the height h of the triangle treating that opposite edge as base:

$$\|\nabla \phi_i\| = \frac{1}{h} = \frac{\|\mathbf{e}_i\|}{2A}, \tag{2.27}$$

where $\mathbf{e}_i$ is the edge e_i as a vector and A is the area of the triangle (see Figure 2.2).

Now, consider two neighboring nodes i and j, connected by some edge $\mathbf{e}_{ij}$. Then $\nabla \phi_i$ points toward node i perpendicular to $\mathbf{e}_i$ and likewise $\nabla \phi_j$ points toward node j perpendicular to $\mathbf{e}_j$. Call the angle formed between these two vectors θ. So we may write:

$$\nabla \phi_i \cdot \nabla \phi_j = \|\nabla \phi_i\| \|\nabla \phi_j\| \cos \theta = \frac{\|\mathbf{e}_j\|}{2A} \frac{\|\mathbf{e}_i\|}{2A} \cos \theta. \tag{2.28}$$

Now notice that the angle between $\mathbf{e}_i$ and $\mathbf{e}_j$, call it α_{ij}, is $\pi - \theta$, but more importantly that:

$$\cos \theta = -\cos (\pi - \theta) = -\cos \alpha_{ij}. \tag{2.29}$$

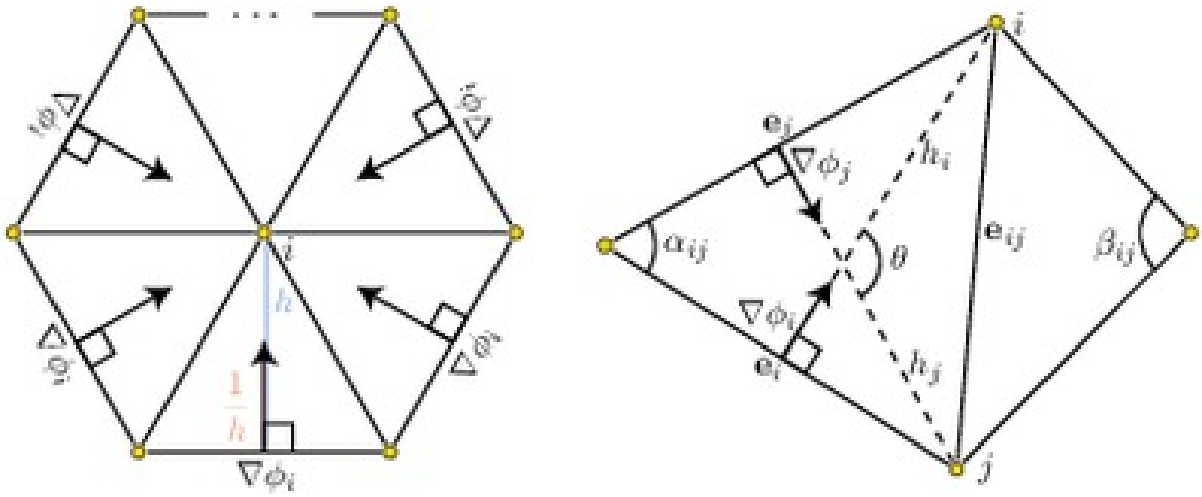


Figure 2.2: *Left: the gradient $\nabla\phi_i$ of a hat function ϕ_i is piecewise-constant and points perpendicular to opposite edges. Right: hat function gradients $\nabla\phi_i$ and $\nabla\phi_j$ of neighboring nodes meet at angle $\theta = \pi - \alpha_{ij}$.*

So, we can rewrite equation (2.28) into:

$$-\frac{\|\mathbf{e}_j\|}{2A}\frac{\|\mathbf{e}_i\|}{2A}\cos\alpha_{ij}. \tag{2.30}$$

Now, apply the definition of sine for right triangles:

$$\sin\alpha_{ij} = \frac{h_j}{\|\mathbf{e}_i\|} = \frac{h_i}{\|\mathbf{e}_j\|}, \tag{2.31}$$

where h_i is the height of the triangle treating $\mathbf{e}_i$ as base, and likewise for h_j. Rewrite (2.30), replacing one of the edge norms, e.g. $\|\mathbf{e}_i\|$:

$$-\frac{\|\mathbf{e}_j\|}{2A}\frac{\frac{h_j}{\sin\alpha_{ij}}}{2A}\cos\alpha_{ij}. \tag{2.32}$$

Combine the cosine and sine terms:

$$-\frac{\|\mathbf{e}_j\|}{2A}\frac{h_j}{2A}\cot\alpha_{ij}. \tag{2.33}$$

Finally, since $\|\mathbf{e}_j\|h_j = 2A$, our constant dot product of these gradients in our triangle is:

$$\nabla\phi_i \cdot \nabla\phi_j = -\frac{\cot\alpha_{ij}}{2A}. \tag{2.34}$$

Similarly, inside the other triangle T_β incident on nodes i and j with angle β_{ij} we have a constant dot product:

$$\nabla\phi_i \cdot \nabla\phi_j = -\frac{\cot\beta_{ij}}{2B}, \tag{2.35}$$

where B is the area T_β.

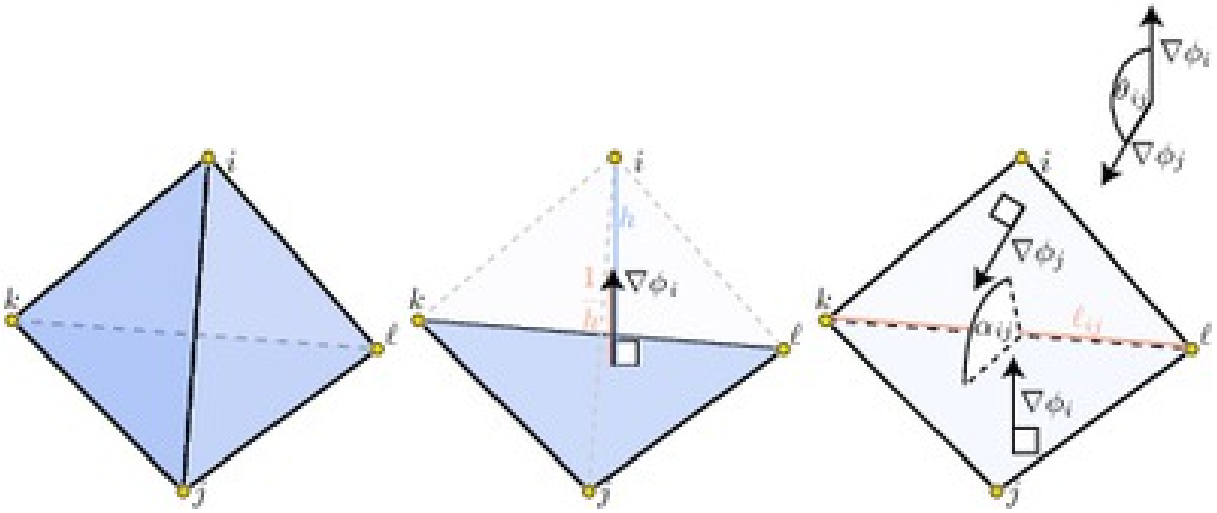


Figure 2.3: *Inside a tetrahedron (left), a hat function gradient $\nabla\phi_i$ is constant and points orthogonal to the face opposite node i (middle). The dihedral angle α_{ij} between faces (right) is related to the angle between overlapping hat function gradients $\nabla\phi_i$ and $\nabla\phi_j$: $\theta_{ij} = \pi - \alpha_{ij}$ (corner).*

Recall that ϕ_i and ϕ_j are only both nonzero inside these two triangles, T_α and T_β. So, since these constants are inside an integral over area we may write:

$$\int_\Omega \nabla\phi_i \cdot \nabla\phi_j = A\nabla\phi_i \cdot \nabla\phi_j\Big|_{\mathrm{T}_\alpha} + B\nabla\phi_i \cdot \nabla\phi_j\Big|_{\mathrm{T}_\beta} = -\frac{1}{2}\left(\cot\alpha_{ij} + \cot\beta_{ij}\right). \tag{2.36}$$

which is also the $\mathbf{L}_{ij}$ term in the Laplacian matrix $\mathbf{L}$ in Equation (2.16).

Tetrahedral volume meshes

We have derived the cotangent Laplacian for triangle meshes in $\mathbb{R}^2$. The result of this derivation may be extended—without modification—to surfaces (2-manifolds in $\mathbb{R}^3$) [Pinkall and Polthier 1993]. Often we will want to work with not just surfaces, but also volumes in $\mathbb{R}^3$. These may be discretized with a tetrahedral mesh. We may similarly define the continuous Dirichlet energy in Equation (2.3), but now $\Omega \subset \mathbb{R}^3$, and again we may *geometrically* derive the contents of the Laplacian matrix for an FEM discretization, where now elements are the tetrahedra of our volume mesh. Recall, the integral in question is:

$$\mathbf{L}_{ij} = \int_\Omega -\nabla\phi_i \, \nabla\phi_j dV, \tag{2.37}$$

where ϕ_i are again hat functions, but now defined to be linear functions in each tetrahedron incident on node i. These hat functions are again locally compact so we are only concerned with neighboring nodes i and j. Just as in $\mathbb{R}^2$, because ϕ_i is linear in each tetrahedron, $\nabla\phi_i$ is constant in each tetrahedron and only nonzero for neighboring tetrahedra. For such a tetrahedron, T, this $\nabla\phi_i$ points perpendicularly from the opposite *face* with magnitude inverse of the height of the tetrahedron treating that opposite face as base (see Figure 2.3):

$$\|\nabla\phi_i\| = \frac{1}{h} = \frac{A_i}{3V}, \tag{2.38}$$

where A_i is the area of the opposite face (triangle) from node i, h is the height of T treating this face as base, and V is the volume of the tetrahedron.

If we restrict our integral to T and consider the hat function of another incident node ϕ_j (whose $\nabla\phi_j$ is also constant in T) then we have:

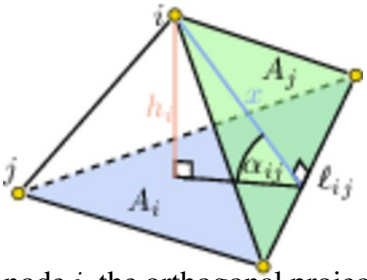
$$\int_T -\nabla\phi_i \nabla\phi_j dV = -V\|\nabla\phi_i\|\|\nabla\phi_j\|\cos\theta_{ij} \tag{2.39}$$

$$= V\|\nabla\phi_i\|\|\nabla\phi_j\|\cos\alpha_{ij} \tag{2.40}$$

$$= \frac{A_i A_j}{9V}\cos\alpha_{ij}, \tag{2.41}$$

where θ_{ij} is the angle between $\nabla\phi_i$ and $\nabla\phi_j$. Let $\alpha_{ij} = \pi - \theta_{ij}$ and notice that α_{ij} is also the dihedral angle between the faces opposite nodes i and j.

A law of sines for tetrahedra



Deriving a law of sines for tetrahedra is straightforward and relies on simple geometric rules.[1] Consider node i of a tetrahedron, we start with the typical formula for volume:

$$V = \frac{1}{3}A_i h_i. \tag{2.42}$$

Now, consider another node j. Make a right triangle perpendicular to the face opposite node i. Let the triangle's corners be: node i, the orthogonal projection of node i onto the opposite face, and the orthogonal projection of node i on to the edge shared by the faces opposite nodes i and j. The angle in this triangle opposite node i is by definition the dihedral angle between those faces α_{ij}. By the trigonometric definition of sine we have:

$$\sin\alpha_{ij} = \frac{h_i}{x}\,,\; h_i = x\sin\alpha_{ij}, \tag{2.43}$$

where x is the hypotenuse of this right triangle. But more importantly, x is also the height of the face opposite node j, which we know is related to the area via:

$$A_j = \frac{x\ell_{ij}}{2}\,,\; x = \frac{2A_j}{\ell_{ij}}. \tag{2.44}$$

Substituting Equation (2.43) and Equation (2.44) into Equation (2.42) reveals:

$$V = \frac{1}{3}A_i x\sin\alpha_{ij} \tag{2.45}$$

$$= \frac{1}{3}A_i\frac{2A_j}{\ell_{ij}}\sin\alpha_{ij} \tag{2.46}$$

$$= \frac{2}{3\ell_{ij}}A_i A_j\sin\alpha_{ij}. \tag{2.47}$$

[1] Thanks to Leo Koller-Sacht for assisting in this derivation.

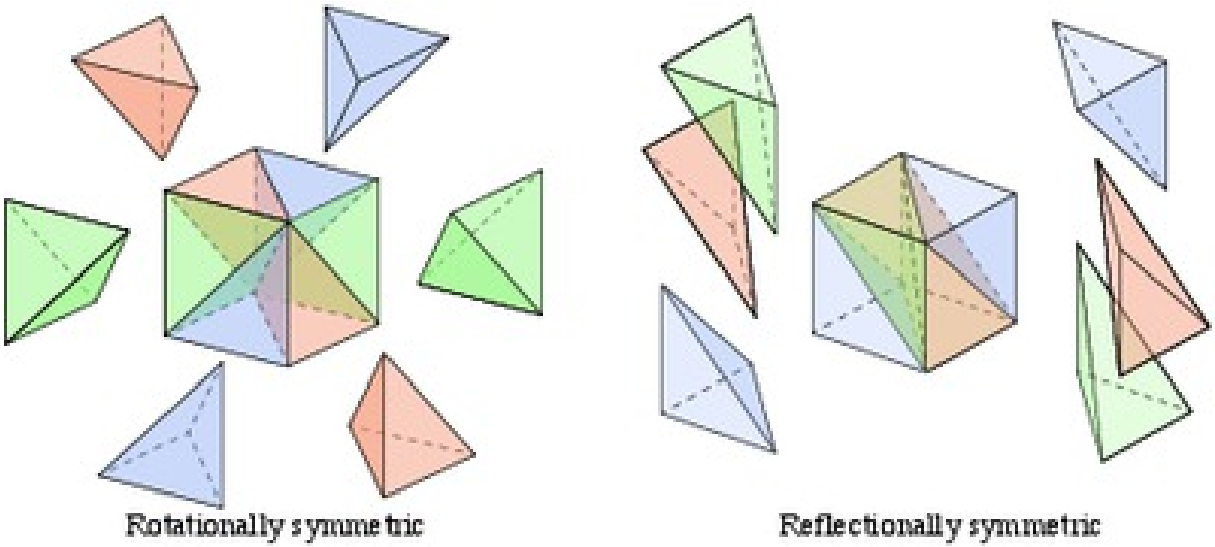


Figure 2.4: *Delaunay tetrahedralizations of a regular grid cell in $\mathbb{R}^3$ may vary drastically. Compare the six rotationally symmetric tetrahedra (left) to the six reflectionally symmetric tetrahedra (right).*

This result was equivalently derived by [Lee 1997] as an application of the law of cosines for tetrahedra.

Substituting Equation (2.47) into Equation (2.41) produces:

$$\int_T -\nabla\phi_i \nabla\phi_j dV = \frac{A_i A_j}{3\frac{2}{\ell_{ij}} A_i A_j \sin\alpha_{ij}} \cos\alpha_{ij} = \frac{\ell_{ij}}{6}\cot\alpha_{ij}. \tag{2.48}$$

A pair of nodes i and j will in general have many shared tetrahedra in $\mathbb{R}^3$, so we write the entire integral as:

$$\mathbf{L}_{ij} = \int_\Omega -\nabla\phi_i \,\nabla\phi_j dV = \sum_{T\in N(i)\cap N(j)} \frac{\ell_{ij}^T}{6}\cot\alpha_{ij}^T. \tag{2.49}$$

where $N(i)$ are the edge neighbors of node i.

Meyer et al. provide a similar derivation [2003], writing that "we have the same formula as [our Equation (2.36)], this time with cotangents of the dihedral angles opposite to the edge". Taken at face value, this is statement is only *almost* true. A naïve reader might blindly replace triangle angles with dihedral angles in Equation (2.36), but this results in an incorrect operator. Indeed, the ℓ_{ij}^T length factor is important to ensure correct behavior of the operator in $\mathbb{R}^3$. A full derivation is given in [Barth 1992], and for general n-manifolds in [Xu and Zikatanov 1999]. This cotangent formula was successfully employed by [Liao et al. 2009, Chao et al. 2010, Li et al. 2012]. The geometric derivation here agrees with the results of discrete calculus [Tong et al. 2003] and "by the book" FEM discretization [Sharf et al. 2007].

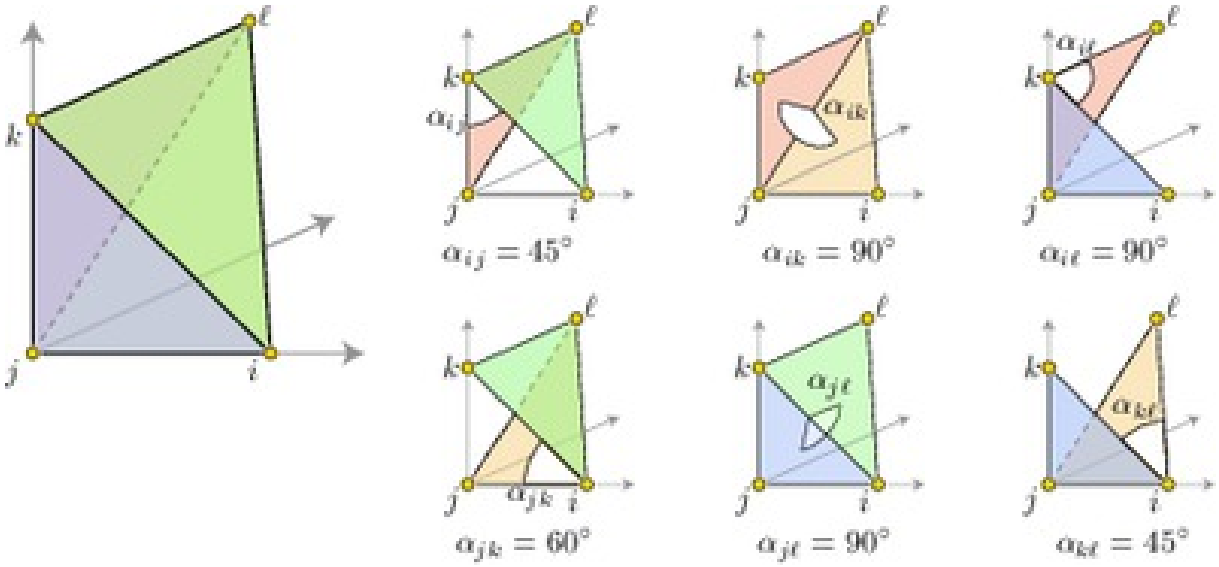


Figure 2.5: *Six tetrahedra rotational symmetric about $e_{i\ell}$ fit in one regular grid cell. With loss of generality assume that $\ell_{ij} = 1$ then $\ell_{jk} \cot \alpha_{jk} = \sqrt{3} \cot 60° = 1 = 1 \cot 45° = \ell_{ij} \cot \alpha_{ij}$ as expected to match the FD stencil.*

Relationship to finite differences

If we consider a regular rectangular grid in $\mathbb{R}^2$, then our FEM stiffness matrix matches exactly one derived using the finite-difference method (FD). This is evident by examining the cotangent formula: diagonal edges are *nullified* because $\cot 90° = 0$. This is true regardless of connectivity, so long as it is Delaunay—each regular grid cell has four points on a circle so Delaunay triangulations are not unique.

With regular grids in $\mathbb{R}^3$ we must be a bit more careful, as even Delaunay connectivities can result in a skewed stiffness matrix using FEM. Each grid cube may be split into five or six tetrahedra, assuming no new vertices are introduced. All eight vertices are on the same circumsphere, so there are different ways to cut up the cube while still Delaunay. For example, we may pack six rotationally symmetric tetrahedra *around* the axis formed between two opposite vertices (see Figure 2.4 left). This results in a balanced FEM stiffness matrix which matches FD (see Figure 2.5). Alternatively, we could divide the cube in half with a plane cutting $45°$ across two opposite faces. On either side we place one trirectangular tetrahedron and two oblong tetrahedra, totaling six (see Figure 2.4 right). This arrangement is Delaunay, but the resulting FEM stiffness matrix does not match finite-differencing. There are even negative off-diagonal coefficients, meaning that solutions to the Laplace equation will not necessarily obey the maximum principle. Related criteria and yet another connectivity (using five tetrahedra per cube) are discussed and illustrated in [Fleishmann et al. 1999].

2.1.3 Mass matrix

We previously derived the assembly of the stiffness matrix *geometrically*. Now we derive the matrix assembly of another important matrix: the mass matrix. Here we wish to determine the

entries:

$$\mathbf{M}_{ij} = \int_{\Omega} \phi_i \, \phi_j dA \tag{2.50}$$

Recall again, our hat functions ϕ_i are locally supported on our triangle mesh, so $\mathbf{M}_{ij}$ is only nonzero for neighboring nodes i and j (and along the diagonal when $i = j$).

Quadrature rules

A large theory surrounds the method of numerical integration by a carefully weighted average of function evaluations at carefully chosen *quadrature* points:

$$\int_0^1 f(x)dx \approx \sum_{i=1}^{m} w_i f(x_i), \tag{2.51}$$

where we approximate the finite integral of the function $f(x)$ as a sum of evaluations of f at some chosen points x_i weighted by scalars w_i. Importantly, this approximation becomes exact when $f(x)$ is a polynomial of degree n and we use an appropriate choice of weights and points, for example if we employ *Gaussian quadrature rules*. In the case of Equation (2.50), we again can divide the integral into the contributions of each triangle simultaneously incident on node i and node j:

$$\mathbf{M}_{ij} = \sum_{T \in N(i) \cap N(j)} \int_T \phi_i \, \phi_j dA, \tag{2.52}$$

where T are those triangles incident on nodes i and j (typically there are two). Now we consider a Gaussian quadrature rule applied to each integral term:

$$\int_T \phi_i \, \phi_j dA = \frac{A_T}{3} \left(\phi_i(\mathbf{x_1})\phi_j(\mathbf{x_1}) + \phi_i(\mathbf{x_2})\phi_j(\mathbf{x_2}) + \phi_i(\mathbf{x_3})\phi_j(\mathbf{x_3}) \right), \tag{2.53}$$

where A is the area of the triangle T and we use uniform weighting ($\frac{1}{3}$) evaluating at locations $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$, which are simply the midpoints of the edges of T. Note that our formula is exact because $\phi_i \, \phi_j$ is a quadratic polynomial [Felippa 2004].

Because we know ϕ_i are simple linear functions over each triangle we may further evaluate this formula arriving at:

$$\frac{A_T}{3} \left(\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} + 0 \cdot 0 \right) = \frac{A_T}{6} \text{ if } i = j \tag{2.54}$$

$$\frac{A_T}{3} \left(\frac{1}{2} \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} + \frac{1}{2} \cdot 0 \right) = \frac{A_T}{12} \text{ otherwise.} \tag{2.55}$$

Finally, putting the two triangle integral terms together we have:

$$\mathbf{M}_{ij} = \sum_{T \in N(i) \cap N(j)} \begin{cases} \frac{A_T}{6} & \text{if } i = j \\ \frac{A_T}{12} & \text{otherwise.} \end{cases} \tag{2.56}$$

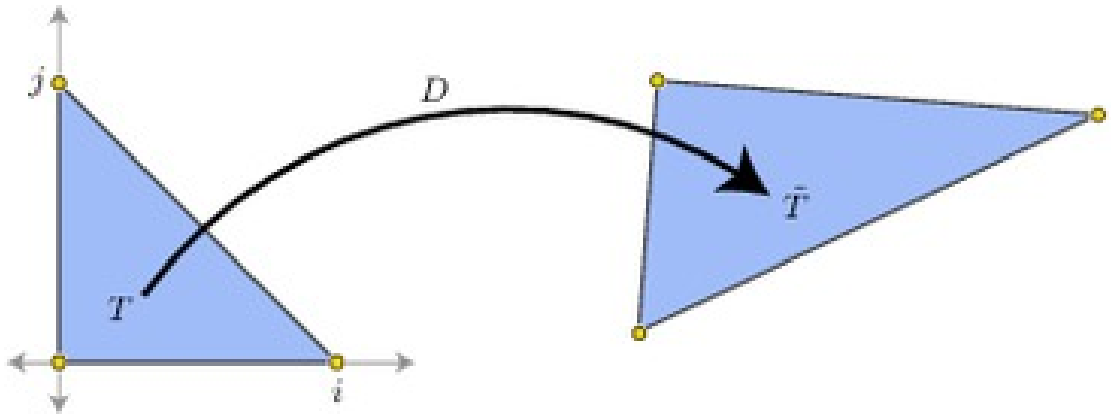


Figure 2.6: *The reference element method in $\mathbb{R}^2$ defines integral with respect to a simple triangle T (left) and an affine map D to any arbitrary triangle $\tilde{T}$ (right).*

Reference element

Though the previous derivation using quadrature rules suffices for assembly of the mass matrix, it is interesting to consider another approach which defines the integral over the triangle in equation Equation (2.53) as a function of the integral defined over a *reference triangle*. To do this we first consider a right triangle in the unit square as depicted in Figure 2.6. Treating T in Equation (2.53) as this triangle, we may explicitly write:

$$\int_T \phi_i \, \phi_j dA = \int_0^{1-x} \int_0^1 \phi_i \left(x, y\right) \phi_j \left(x, y\right) dxdy \tag{2.57}$$

$$= \begin{cases} \int_0^{1-x} \int_0^1 x^2 \, dxdy = 1/12 & \text{if } i = j \\ \int_0^{1-x} \int_0^1 xy \, dxdy = 1/24 & \text{otherwise.} \end{cases} \tag{2.58}$$

We can deform this reference triangle T to any arbitrary triangle $\tilde{T}$ using an affine transformation D. Employing integration by substitution we have that:

$$\int_T f dA = \det\left(D\right) \int_{\tilde{T}} f d\tilde{A} = 2\tilde{A} \int_{\tilde{T}} f d\tilde{A}, \tag{2.59}$$

where $\tilde{A}$ is the area of $\tilde{T}$. Applying this to our integral we arrive at the same assembly as in Equation (2.56). Similar derivations and further discussions of the stiffness matrix and mass matrix occur in many books (e.g. [Sayas 2008]).

We may also consider the mass matrix defined for a tetrahedral mesh in $\mathbb{R}^3$. Both the reference element and quadrature rule derivations extend analogously with a larger matrix D or an additional quadrature point, respectively.

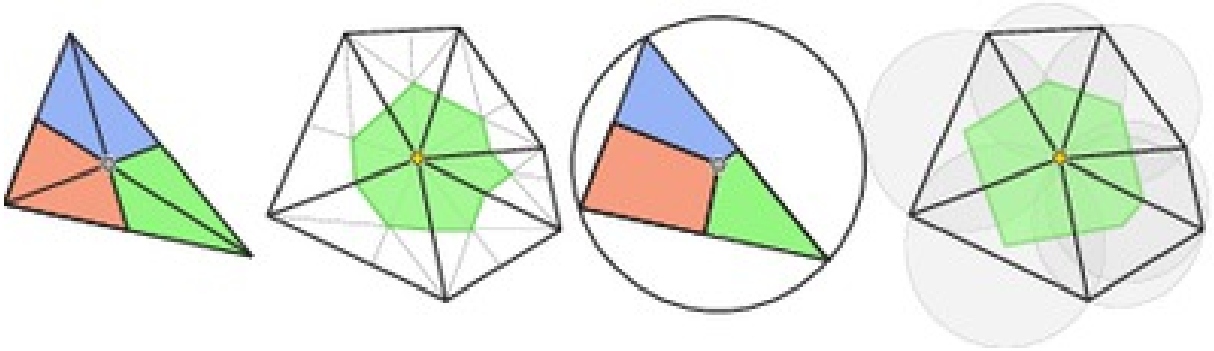


Figure 2.7: *Left to right: The barycenter splits a triangle into three equal area parts. Summing the areas of parts incident on a triangle mesh vertex equals its barycentric mass matrix entry. The Voronoi center splits the triangle area into unequal parts: the areas correspond to the portion of the triangle closer to one vertex than the others. The sum of these areas equals a vertex's Voronoi mass matrix entry.*

Lumping

The *full* mass matrix $\mathbf{M}$ just described is often replaced by a *lumped* mass matrix, i.e. a diagonal matrix $\mathbf{M}^d$. The lumping step is mathematically sound:

$$\int_\Omega uudA \approx \int_\Omega \sum_{i\in I} u_i\,\phi_i \sum_{j\in I} u_j\,\phi_j dA = \mathbf{u}^\mathsf{T}\mathbf{M}\mathbf{u} \approx \mathbf{u}^\mathsf{T}\mathbf{M}^d\mathbf{u}, \tag{2.60}$$

where our hat functions ϕ_i are built over a mesh with average edge length h. The convergence rate with respect to h is not damaged if the quadrature rule has accuracy $O(h)$, which is satisfied by using vertices as quadrature points [Ciarlet 1978, Brezzi and Fortin 1991]. This leads to the diagonal mass matrix $\mathbf{M}^d$. The sum of the (unnormalized) diagonal entries $\mathbf{M}^d_i$ will equal the surface area of Ω. There are two relevant methods for determining how this area is distributed along the diagonal, corresponding to the so-called barycentric or Voronoi-mixed mass matrices. Both schemes have been shown to have linear convergence, but the Voronoi-mixed approach has, in general, much smaller error norms and less mesh dependence [Meyer et al. 2003]. This is also considered later in the context of higher order PDEs (see Section 3.5).

In the case of irregular meshes, the distribution of area should correspond to the vertex density. The barycentric mass matrix achieves this in the most straightforward way. For each triangle, its area is split into three quadrilaterals by connecting the triangle's barycenter to its edge midpoints (see Figure 2.7 left). By definition these areas are equal regardless of the shape of the triangle. We distribute $A/3$ to each incident vertex, and the entry $\mathbf{M}^d_i$ is the one third the sum of the areas of incident triangles on vertex i.

Another, also natural, choice is to use the Voronoi area around each vertex. Because we deal with piecewise-linear domains, we may again consider each triangle individually (see Figure 2.7 right). First we find the circumcenter, the center of the circumscribing circle of the triangle, equidistant to vertices on the triangle. Connect this point to the midpoints along the edges of the triangle, again forming three quadrilaterals. The Voronoi mass matrix entry $\mathbf{M}^d_i$ for vertex i is the sum of its corresponding quadrilaterals from all incident triangles. For acute triangles, each quadrilateral represents the portion of the triangle closest to one of the vertices. For obtuse

triangles, the circumcenter lies outside of the triangle, resulting in negative signed areas entering the summation. Thus, [Meyer et al. 2003] propose to use a mixed or hybrid approach. For acute triangles, the Voronoi areas are computed according to the circumcenter. For obtuse triangles, the circumcenter is snapped to the midpoint along the edge opposite the obtuse angle. Thus the obtuse angled vertex receives $A/2$ and the others $A/4$.

We now have all necessary pieces for numerically solving Poisson equations subject to Dirichlet or Neumann boundary conditions discretized using linear FEM over a triangle or tetrahedral mesh. Despite the long derivations, the machinery necessary to manifest these pieces in code is quite compact and efficient. In particular, we can compute all quantities and matrix elements in a way that relies only on the intrinsic description of a given mesh. So instead of requiring the geometry (i.e. vertex positions) and connectivity information of a mesh we need only the simplex edge lengths. This will become important when considering tiled parameterizations (see Section 3.8.1) or manifolds embedded in higher dimensions (see Section 4.8).

2.2 Constrained optimization

Many chapters in this thesis follow the scientific principle of variations. The desired behavior for the particular problem will be described as the minimum of some cost or energy function. The first challenge in each case will be to describe these energy functions in mathematical terms. But even with a mathematical description, realizing optimal solutions on a computer poses problems. In the previous Section 2.1 we discussed the machinery necessary to leverage FEM to solve certain types of PDEs corresponding to solutions of energy functions. Many times it suffices to describe a problem using fixed boundary conditions of these systems. But often we will model our desired behavior not just as energy-minimizing but also subject to certain, general constraints. This section reviews a few important notions in constrained optimization. We will highlight a few special cases (e.g. quadratic and conic programming).

In the most general case we consider, the problem of discrete quadratic programming is to optimize a quadratic function:

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{l} + c, \tag{2.61}$$

subject to linear inequality constraints:

$$\mathbf{A}_{\text{ie}}\mathbf{x} \leq \mathbf{b}_{\text{ie}}, \tag{2.62}$$

where $\mathbf{x}$ is a vector of unknowns, $\mathbf{Q}$ is a matrix of quadratic coefficients (usually positive definite and often sparse), $\mathbf{l}$ is a vector of linear coefficients, c is a constant scalar term, $\mathbf{A}_{\text{ie}}$ is the linear inequality *constraints* matrix, and $\mathbf{b}_{\text{ie}}$ is a vector of linear inequality constraints right-hand side values.

Immediately notice that special constraints such as linear *equalities* or constant *inequalities* can be expressed in the general form of Equation (2.62) by repeating constraints with opposite signs or using rows of the identity matrix respectively.

2.2.1 Constant equality constraints

Constant equality constraints fix certain values in $\mathbf{x}$:

$$\mathbf{x}_f = \mathbf{g}_f, \tag{2.63}$$

where $\mathbf{x}_f$ denotes the *sliced* vector of unique values in $\mathbf{x}$ we would like to constrain and $\mathbf{g}_f$ denotes the corresponding fixed values.

Imposing such constraints in Equation (2.61) is simple. We separate the $\mathbf{x}$ into known values $\mathbf{x}_f$ and unknown values $\mathbf{x}_u$ and rearrange the quadratic, linear and constant terms with respect to the unknowns:

$$E(\mathbf{x}_u) = \frac{1}{2}\begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix}^\mathsf{T}\mathbf{Q}\begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix} + \begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix}^\mathsf{T}\mathbf{l} + c \tag{2.64}$$

$$= \frac{1}{2}\begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix}^\mathsf{T}\begin{pmatrix}\mathbf{Q}_{uu} & \mathbf{Q}_{uf}\\ \mathbf{Q}_{fu} & \mathbf{Q}_{ff}\end{pmatrix}\begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix} + \begin{pmatrix}\mathbf{x}_u\\ \mathbf{x}_f\end{pmatrix}^\mathsf{T}\begin{pmatrix}\mathbf{l}_u\\ \mathbf{l}_f\end{pmatrix} + c \tag{2.65}$$

$$= \frac{1}{2}\mathbf{x}_u^\mathsf{T}\mathbf{Q}_{uu}\mathbf{x}_u + \mathbf{x}_u^\mathsf{T}\left(\mathbf{l} + \frac{1}{2}\left(\mathbf{Q}_{uf} + \mathbf{Q}_{fu}^\mathsf{T}\right)\mathbf{x}_f\right) + \frac{1}{2}\mathbf{x}_f^\mathsf{T}\mathbf{Q}_{ff}\mathbf{x} + c \tag{2.66}$$

$$= \frac{1}{2}\mathbf{x}_u^\mathsf{T}\tilde{\mathbf{Q}}\mathbf{x}_u + \mathbf{x}_u^\mathsf{T}\tilde{\mathbf{l}} + \tilde{c}. \tag{2.67}$$

If the resulting $\tilde{\mathbf{Q}}$ is positive definite then we may solve this by taking partial derivatives with respect to $\mathbf{x}_u^\mathsf{T}$, setting each to zero and solving the system of linear equations. In practice this may be achieved efficiently using Cholesky decomposition and triangular back-substitution.

2.2.2 Linear equality constraints

Optimizing Equation (2.61) subject to linear *equality* constraints is similarly straight forward. These constraints are of the form:

$$\mathbf{A}_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}}. \tag{2.68}$$

We may enforce these constraints using Lagrange multipliers and converting our minimization problem into a saddle search of a Lagrangian:

$$\Lambda(\mathbf{x}, \lambda) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{l} + c + \lambda^\mathsf{T}\left(\mathbf{A}_{\text{eq}}\mathbf{x} - \mathbf{b}_{\text{eq}}\right) \tag{2.69}$$

$$= \frac{1}{2}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix}^\mathsf{T}\begin{pmatrix}\mathbf{Q} & \mathbf{A}_{\text{eq}}{}^\mathsf{T}\\ \mathbf{A}_{\text{eq}} & *\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix} + \begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix}^\mathsf{T}\begin{pmatrix}\mathbf{l}\\ -\mathbf{b}_{\text{eq}}\end{pmatrix} + c \tag{2.70}$$

$$= \frac{1}{2}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix}^\mathsf{T}\tilde{\mathbf{Q}}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix} + \begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix}^\mathsf{T}\tilde{\mathbf{l}} + c, \tag{2.71}$$

where λ is a *vector* of *unknown* Lagrange multipliers. So long as $\mathbf{A}$ is full row rank then the constraints are linearly independent. We can then find the saddle point of Λ by taking partial derivatives with respect to both $\mathbf{x}^\mathsf{T}$ and λ^T, setting these to zero and solving the resulting system of linear equations. In general we will not be able to use Cholesky decomposition because —

even if $\mathbf{Q}$ is positive definite — $\tilde{\mathbf{Q}}$ will not be. Thus we must resort to general LU decomposition [Davis 2004], LDL decomposition [Duff 2004], or employ a trick to reduce LU decomposition for this special case to two Cholesky decompositions (see Section 2.2.2).

The values of the Lagrange multipliers λ are often not needed, however, they have a very useful interpretation. If we consider our unconstrained solution $\mathbf{x}$ as steady-state solution or equilibrium w.r.t. the *internal force* $\frac{\partial E}{\partial \mathbf{x}^\mathsf{T}}$, then the Lagrange multipliers λ are intuitively the signed magnitudes of the minimal *external* forces necessary to *pull* the unconstrained solution $\mathbf{x}$ to the feasible region (set of solutions which satisfy the given constraints). Here we are careful to keep track of the sign as this will become useful when dealing with inequalities (Section 2.2.3).

If $\mathbf{x}$ lives in $\mathbb{R}^n$ then each row in Equation (2.68) represents a hyperplane in $\mathbb{R}^n$. The Lagrange multiplier method limits our optimization to the intersection of those hyperplanes. Another method is to use Equation (2.68) to build a parameterization description of that intersection. Here we find a matrix Γ such that $\mathbf{x}_{\text{proj}} = \Gamma\mathbf{y}$ where $\mathbf{y}$ parameterizes all values of $\mathbf{x}$ such that $\mathbf{A}_{\text{eq}}\mathbf{x} - \mathbf{b}_{\text{eq}} = 0$. That is, it parameterizes the null space of the constraints. The projection matrix Γ can be found efficiently using QR decomposition. Then $\Gamma\mathbf{y}$ is directly substituted into Equation (2.61), where minimizing results in a linear system with a solution $\mathbf{y}*$. The primary solution is recovered by computing $\mathbf{x}* = \Gamma\mathbf{y}*$. The null space method has the advantage that it can robustly cope with linearly dependent constraints. The disadvantage is that substitution may result in a dense problem even if the original matrices $\mathbf{Q}$ and $\mathbf{A}_{\text{eq}}$ are sparse. Nonetheless, this has been employed successfully in graphics (e.g. [Bouaziz et al. 2012]).

The constant equality constraints in Section 2.2.1 are a special case of linear equality constraints. We could use the null space method or the Lagrange multiplier method to enforce them. For constant constraints, the rows of $\mathbf{A}_{\text{eq}}$ are simply the rows of the identity matrix corresponding to $\mathbf{x}_f$. Thus the null space method would directly accomplish substitution resulting in Equation (2.67). The Lagrangian method would introduce a set of multipliers λ_f, but since the resulting set of linear equations will be simply these rows of the identity matrix, one may easily invert them to solve for $\mathbf{x}_f$ and substitute into the rows above. If we ignore the values of λ_f we arrive again at Equation (2.67).

LU decomposition via two Cholesky decompositions

Minimizing quadratic energies with and without linear equality constraints results in a system of linear equations:

$$\mathbf{Q}\mathbf{x} = \mathbf{l}\,,\ \mathbf{x} = \mathbf{Q}^{-1}\mathbf{l}, \tag{2.72}$$

where $\mathbf{Q}$ is the, typically sparse, system matrix (aka quadratic coefficients matrix or Hessian), $\mathbf{x}$ is a vector of unknowns and $\mathbf{l}$ is the right-hand side (aka vector of linear coefficients). We may solve this equation by inverting $\mathbf{Q}$. We will rarely ever want to compute $\mathbf{Q}^{-1}$ explicitly: it is usually dense even if $\mathbf{Q}$ is sparse, it is slow to compute, and it may be numerically unstable to do so. Rather we find a factorization of $\mathbf{Q} = \mathbf{L}\mathbf{U}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular (hopefully sparse) matrices respectively. If $\mathbf{Q}$ is positive definite then we may use a Cholesky decomposition $\mathbf{Q} = \mathbf{L}\mathbf{L}^\mathsf{T}$. However, in constrained optimization using Lagrange multipliers for

linear equality constraints, we typically have a system matrix that looks like:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix}, \tag{2.73}$$

where $\mathbf{A}$ are the quadratic coefficients of our primary quadratic energy, $\mathbf{C}$ is the linear equality constraints matrix, and $*$ represents a block of zeros. The $\mathbf{A}$ and $\mathbf{C}$ blocks may be sparse and as a whole the matrix is symmetric (so long as $\mathbf{A}$ is), but it is generally indefinite: the equation identifies the critical point (saddle) of a quadratic Lagrangian, rather than the global minimum of a convex quadratic function. In this case Cholesky decomposition cannot be used, so one may resort to generic LU decomposition. However, if $\mathbf{A}$ is positive definite (as is often the case in quadratic optimization), then we can design a special case LU decomposition for matrices $\mathbf{Q}$ of the type described in Equation (2.73).[2]

Our problem is to find an LU decomposition of $\mathbf{Q}$. Just like our matrix $\mathbf{Q}$, we can divide our LU decomposition into four blocks each:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix} = \begin{pmatrix} ? & * \\ ? & ? \end{pmatrix} \begin{pmatrix} ? & ? \\ * & ? \end{pmatrix}, \tag{2.74}$$

where $?$ represent the so far unknown blocks.

Because $\mathbf{A}$ is positive definite, we can compute a Cholesky decomposition $\mathbf{A} = \mathbf{L}\mathbf{L}^\mathsf{T}$ and insert accordingly into the appropriate corners of the lower and upper triangular parts:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ ? & ? \end{pmatrix} \begin{pmatrix} \mathbf{L}^\mathsf{T} & ? \\ * & ? \end{pmatrix}. \tag{2.75}$$

This now means that we need $\mathbf{C} = \mathbf{L}?$, but since $\mathbf{L}$ is lower triangular it is easy to compute $\mathbf{M} = \mathbf{L}^{-1}\mathbf{C}$. It is important to note at this point that $\mathbf{M}$ will likely be dense, regardless of whether $\mathbf{A}$ and $\mathbf{C}$ are sparse. Thus, we see immediately that this method is only going to be useful when $\mathbf{C}$ has a small number of columns (i.e. for optimization problems with a small number of linear equality constraints). We may place $\mathbf{M}$ into our factorization:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ \mathbf{M}^\mathsf{T} & ? \end{pmatrix} \begin{pmatrix} \mathbf{L}^\mathsf{T} & \mathbf{M} \\ * & ? \end{pmatrix}. \tag{2.76}$$

We are almost done. All that is left is to reproduce the zeros in the bottom right corner on the left hand side of $\mathbf{Q}$. This means we need the two missing pieces when multiplied to produce $-\mathbf{M}^\mathsf{T}\mathbf{M}$. But again this is made easy via Cholesky factorization: $\mathbf{M}^\mathsf{T}\mathbf{M}$ is clearly symmetric and is positive semidefinite: $\mathbf{x}^\mathsf{T}\mathbf{M}^\mathsf{T}\mathbf{M}\mathbf{x} = \left(\mathbf{x}^\mathsf{T}\mathbf{M}^\mathsf{T}\right)(\mathbf{M}\mathbf{x}) \geq 0,\ \forall \mathbf{x}$. We factorize $\mathbf{M}^\mathsf{T}\mathbf{M} = \mathbf{K}\mathbf{K}^\mathsf{T}$ where $\mathbf{K}$ is lower triangular. Finally this gives us:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ \mathbf{M}^\mathsf{T} & \mathbf{K} \end{pmatrix} \begin{pmatrix} \mathbf{L}^\mathsf{T} & \mathbf{M} \\ * & -\mathbf{K}^\mathsf{T} \end{pmatrix}. \tag{2.77}$$

The final LU decomposition *almost* has the property that $\mathbf{U} = \mathbf{L}^\mathsf{T}$. We may exploit this by only storing $\mathbf{L}$ and flipping the sign on the bottom right $\mathbf{K}$ block as necessary during back substitution.

[2] Ilya Baran brought this method to my attention, but the original source is unknown.

In the end we only need one sparse Cholesky factorization, one sparse back substitution, one dense (but small) matrix-matrix multiplication, and one dense (but small) Cholesky decomposition. The sparse Cholesky factorization corresponding to the primary quadratic coefficients $\mathbf{A}$ can be precomputed and constraints in $\mathbf{C}$ can be dynamically added, edited and removed at low cost.

We are not quite done. It is well known that without reordering both Cholesky and LU decompositions for sparse matrices can result in effectively dense factors. When reordering we compute Cholesky decompositions as $\mathbf{S}^\mathsf{T}\mathbf{A}\mathbf{S} = \mathbf{L}\mathbf{L}^\mathsf{T}$, where $\mathbf{S}$ is a permutation matrix. Computing these factorizations with reordering is faster, stabler and results in much sparser factors than typical decomposition. To take advantage of this in our trick above we will have to redefine our LU decomposition to also include reordering. Thus our problem at the beginning is now:

$$\begin{pmatrix} ? & * \\ * & ? \end{pmatrix}\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix}\begin{pmatrix} ? & * \\ * & ? \end{pmatrix} = \begin{pmatrix} ? & * \\ ? & ? \end{pmatrix}\begin{pmatrix} ? & ? \\ * & ? \end{pmatrix}. \tag{2.78}$$

Now, we can go through similar steps as before. First utilize $\mathbf{S}^\mathsf{T}\mathbf{A}\mathbf{S} = \mathbf{L}\mathbf{L}^\mathsf{T}$:

$$\begin{pmatrix} \mathbf{S}^\mathsf{T} & * \\ * & ? \end{pmatrix}\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix}\begin{pmatrix} \mathbf{S} & * \\ * & ? \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ ? & ? \end{pmatrix}\begin{pmatrix} \mathbf{L}^\mathsf{T} & ? \\ * & ? \end{pmatrix}. \tag{2.79}$$

This means that we need $\mathbf{L}? = \mathbf{S}^\mathsf{T}\mathbf{C}?$, which again we can solve for using back subtitutition: $\mathbf{M}?^\mathsf{T} = \mathbf{L}^{-1}\mathbf{S}^\mathsf{T}\mathbf{C}$. Because we used reordering, $\mathbf{L}$ will be very sparse. Computing $\mathbf{M}?^\mathsf{T}$ will be a fast back substitution, but it will still, in general, end up dense. The missing $?$ block comes from the permutation matrices on the other side, let us declare it $\mathbf{T}$. Then we have $\mathbf{M}\mathbf{T}^\mathsf{T} = \mathbf{L}^{-1}\mathbf{S}^\mathsf{T}\mathbf{C}$:

$$\begin{pmatrix} \mathbf{S}^\mathsf{T} & * \\ * & \mathbf{T}^\mathsf{T} \end{pmatrix}\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix}\begin{pmatrix} \mathbf{S} & * \\ * & \mathbf{T} \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ \mathbf{T}\mathbf{M}^\mathsf{T} & ? \end{pmatrix}\begin{pmatrix} \mathbf{L}^\mathsf{T} & \mathbf{M}\mathbf{T}^\mathsf{T} \\ * & ? \end{pmatrix}. \tag{2.80}$$

Finally we want to create the zeros in the lower right corner of the system. To do this, we again use a reordering Cholesky factorization: $\mathbf{T}\mathbf{M}^\mathsf{T}\mathbf{M}\mathbf{T}^\mathsf{T} = \mathbf{K}\mathbf{K}^\mathsf{T}$.

$$\begin{pmatrix} \mathbf{S}^\mathsf{T} & * \\ * & \mathbf{T}^\mathsf{T} \end{pmatrix}\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & * \end{pmatrix}\begin{pmatrix} \mathbf{S} & * \\ * & \mathbf{T} \end{pmatrix} = \begin{pmatrix} \mathbf{L} & * \\ \mathbf{T}\mathbf{M}^\mathsf{T} & \mathbf{K} \end{pmatrix}\begin{pmatrix} \mathbf{L}^\mathsf{T} & \mathbf{M}\mathbf{T}^\mathsf{T} \\ * & -\mathbf{K}^\mathsf{T} \end{pmatrix}. \tag{2.81}$$

The result is a LU decomposition of our original system conjugated by a permutation matrix. Again we have a lot of repeated blocks, so potential savings on storage is high.

In practice this trick is only useful when there are a small number of constraints that are changing dynamically. Otherwise, LDL decomposition is typically faster.

Weak constraints via quadratic penalty functions

Another way to impose linear equality constraints is to enforce them weakly by appending quadratic penalty functions to original energy in 2.61. This results in new quadratic energy of

the form:

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{l} + c + (\mathbf{A}_{\text{eq}}\mathbf{x} - \mathbf{b}_{\text{eq}})^\mathsf{T}\mathbf{W}(\mathbf{A}_{\text{eq}}\mathbf{x} - \mathbf{b}_{\text{eq}})^\mathsf{T} \tag{2.82}$$

$$= \frac{1}{2}\mathbf{x}^\mathsf{T}\left(\mathbf{Q} + 2\mathbf{A}_{\text{eq}}^\mathsf{T}\mathbf{W}\mathbf{A}_{\text{eq}}\right)\mathbf{x} + \mathbf{x}^\mathsf{T}\left(\mathbf{l} - 2\mathbf{A}_{\text{eq}}^\mathsf{T}\mathbf{W}\mathbf{b}_{\text{eq}}\right) + c + \mathbf{b}_{\text{eq}}^\mathsf{T}\mathbf{W}\mathbf{b}_{\text{eq}} \tag{2.83}$$

$$= \frac{1}{2}\mathbf{x}^\mathsf{T}\tilde{\mathbf{Q}}\mathbf{x} + \mathbf{x}^\mathsf{T}\tilde{\mathbf{l}} + \tilde{c}, \tag{2.84}$$

where $\mathbf{W}$ is a diagonal matrix with positive diagonal entries $\mathbf{W}_i$ representing the *weight* on row i in our linear equality constraints. Intuitively, these weights represent how badly we would like to satisfy each constraint. Theoretically, setting these weights to ∞ results in the strong or strict equality constraints as discussed in Section 2.2.2. In practice, exceptionally large weights must be used with care as they may introduce floating-point errors.

Determining the values for $\mathbf{W}$ is a problem-specific and domain-specific issue. When dealing with discrete, mesh-based energies weak constraints are often used to impose constant constraints on function values associated with mesh vertices. Simply using uniform weights $\mathbf{W} = w\mathbf{I}$ will result in uneven enforcement on irregular meshes. The mesh density should also be taken into account when dealing with approximations to continuous properties. This ensures discretization independence and facilitates convergence as resolution increases. A safe choice then is to use a uniformly weighted lumped mass matrix $\mathbf{W} = w\mathbf{M}^d$ (see Section 2.1.3).

Weak constraints are more difficult to control, but admittedly easier to set up and employ. Linearly dependent constraints are simply enforced *stronger*, but do not cause algebraic singularities. Contradictory constraints are similarly handled. And if $\mathbf{Q}$ is positive semidefinite, then $\tilde{\mathbf{Q}}$ will be, too, so Cholesky decomposition may be used.

2.2.3 Linear inequality constraints and the active set method

Having considered the special cases where only equality constraints exist, we now turn back to the general case and deal with linear inequality constraints. Without loss of generality we assume we are dealing with constraints of the form:

$$\mathbf{A}_{\text{ie}}\mathbf{x} \leq \mathbf{b}_{\text{ie}}. \tag{2.85}$$

There are many methods to solve such general QPs [Andersen and Andersen 2000, Byrd et al. 2006, MATLAB 2012]. Instead of describing the inner workings of all these methods, we will instead explore a straightforward technique that exploits the now well-understood Lagrangian method for quadratic optimization with linear equality constraints. This method iteratively determines which inequality constraints should be enforced as hard equality constraints. At any given time these constraints are called the "active set".

Before describing the algorithm, let us consider what inequality constraints mean intuitively. Our quadratic function in Equation (2.61) measures any potential solution $\mathbf{x} \in \mathbb{R}^n$. Assuming the energy is positive definite, if we viewed a plot of this function in $\mathbb{R}^{n+1}$ we would see a hyper-parabola. Viewed as a topographical map, we would we see concentric hyper-ellipses centered around the global minimum $\mathbf{x}^*$. When we have a linear *equality* constraint we are restricting the feasible set of solutions to a hyperplane cutting through $\mathbb{R}^n$. When we have a linear

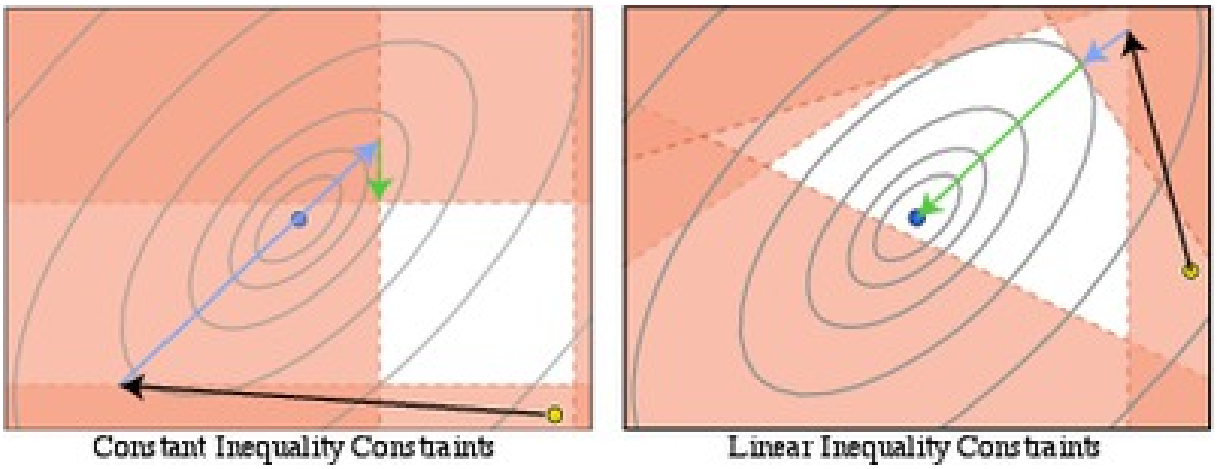


Figure 2.8: *Iterations of the active set method for constant (left) and linear (right) inequality constraints.*

inequality constraint we restrict to a halfspace of $\mathbb{R}^n$. The boundary of this space is determined by the hyperplane described by treating the constraint as an equality. The important intuition to draw on is that the intersection of the halfspaces corresponding to multiple linear inequality constraints forms a convex subspace in $\mathbb{R}^n$. There are two options: the global minimum of the *unconstrained* energy in Equation (2.61) lies inside this convex subspace or it lies outside and a unique minimum within the subspace lies on the boundary of the subspace. Corollarily there are no other local minima inside this subspace.

The goal of the active set method is to determine which, if any, inequality constraints this unique minimum is *touching*. These are the active set of constraints at the solution and at convergence of this algorithm.

We begin the algorithm by assuming we have a current solution $\mathbf{x}_i$ and current set of active constraints $\mathbf{A}_i, \mathbf{b}_i$. We now compute an updated solution $\mathbf{x}_{i+1}$ by minimizing Equation (2.61) subject to:

$$\mathbf{A}_i \mathbf{x}_{i+1} = \mathbf{b}_i. \tag{2.86}$$

We enforce these as strict equality constraints using Lagrange multipliers λ_{i+1}. The signs of λ_{i+1} reveal whether each constraint is pulling $\mathbf{x}_{i+1}$ away from the energy minimum into or out of the feasible region. Each negative entry in λ_{i+1} means that if we resolve without that individual constraint the inequality constraint will still hold (and the energy will be smaller). Thus we set $\mathbf{A}_{i+1}, \mathbf{b}_{i+1}$ to be a subset of the constraints with positive λ_{i+1} entries. The updated solution $\mathbf{x}_{i+1}$ will in general violate some inactive constraints. Thus a subset of these violated constraints are *activated* and appended to $\mathbf{A}_{i+1}, \mathbf{b}_{i+1}$. This process is repeated until convergence.

In the special case of constant inequality or box constraints:

$$\mathbf{b}_l \leq \mathbf{x} \leq \mathbf{b}_u, \tag{2.87}$$

then substitution may be employed for faster linear solves at each iteration. The corresponding Lagrange multipliers values may be recovered *post facto*. If our active constant constraints are $\mathbf{x}_f = \mathbf{b}_f$ then the after solving according to Equation (2.67) the corresponding Lagrange multiplier values are $\lambda_f = -\mathbf{Q}_{uf}\mathbf{x}_u + \mathbf{Q}_{ff}\mathbf{b}_f - \mathbf{l}_f$.

A few iterations of two hypothetical optimization problems constraints are shown in Figure 2.8. We visualize a quadratic function (e.g. $5x^2 - 6y + 5y^2$) as a topographical map (elliptic isolines) with a global minimum (blue dot). In the left image we show the feasible region (white region inside red overlay) defined by constant inequality or *box* constraints of the form $\mathbf{b}_l \leq [x; y] \leq \mathbf{b}_u$. Given a, generally infeasible, initial guess (yellow dot) we may optimize by first enforcing the violated constraint as an equality constraint (black arrow). This generates a new, generally infeasible point. The previous constraint is inactivated, a new constraint is activated, and we resolve (blue arrow). Now the previous constraint cannot be inactivated so we only append a new violated constraint and resolve, reaching convergence (green arrow). The unconstrained global minimum is not feasible so the active set is identified at the unique feasible minimum. In the right image, a similar step-through is shown for a problem with general linear constraints of the form: $\mathbf{A}_{\text{ie}}[x; y] \leq \mathbf{b}_{\text{ie}}$. This time the global minimum is found to be feasible.

This method is robust and accurate, but not the fastest method for solving QPs in practice. For large, sparse QPs, we have found that interior point solvers are faster, as implemented by MOSEK [Andersen and Andersen 2000] or in recent versions of the `quadprog` routine of [MATLAB 2012]. Often a QP can be converted to a different formalization like non-negative least-squares or conic programming. If the reformatting can utilize extra geometric notions (like the Laplacian being a sparse square root of the bi-Laplacian) then there may be a lot of performance gains (see Section 2.2.4).

On the other hand, the active set method is simpler to implement and more accurate. In order to provide high accuracy results at reasonable performance, the commercial KNITRO solver begins optimizing with an interior point method similar to MOSEK to get a quick approximate initial guess and finishes with an active set method to get a highly accurate solution [Byrd et al. 2006]. By construction the active set method terminates with a list of the inequality constraints which are being exactly satisfied with equality. For certain problems analyzing these active constraints provides useful geometric insights. For example in Chapter 4, analyzing the active set of the constant bound constraints may be used to empirically verify the claim of local support.

2.2.4 Conic programming

Often, quadratic functions arise in the form of least-squares energies, where we may have a set of linear formulas:

$$\mathbf{F}\mathbf{x} - \mathbf{f}, \tag{2.88}$$

which measure some discrete quantities (one for each row in $\mathbf{F}, \mathbf{f}$) of $\mathbf{x}$. For example, using the stiffness and mass matrices of Sections 2.1.2 & 2.1.3 then $\mathbf{M}^{-1}\mathbf{L}\mathbf{x} - \mathbf{0}$ measures the Laplacian of some function $\mathbf{x}$ at each point on a mesh. If these all measure zero, then the function is called *harmonic*. We can easily construct a quadratic energy which measures these quantity vectors

with respect to a L^2 norm or least-squares sense[3]:

$$E_{\text{LS}}(\mathbf{x}) = \frac{1}{2}\|\mathbf{F}\mathbf{x} - \mathbf{f}\|^2. \tag{2.89}$$

So if measuring zero meant the function was called *something*, then we say that minimizers of these energies are as-*something*-as-possible. Thus minimizers of $\|\mathbf{M}^{-1}\mathbf{L}\mathbf{x} - \mathbf{0}\|^2$ are as-*harmonic*-as-possible.

When such energies appear in QPs then we can take advantage of their special formulation. The matrix $\mathbf{F}$ in Equation (2.89) is a (generally rectangular) matrix square root of the quadratic coefficients matrix $\mathbf{Q}$ in Equation (2.61). If we require that $\mathbf{Q}$ is positive semidefinite then such a square root always exists, but it may not always be possible to derive it analytically or efficiently for a given problem. For example, consider the discrete bi-Laplacian operator for a triangle mesh $\mathbf{Q} = \mathbf{L}^\mathsf{T}\mathbf{M}^{-1}\mathbf{L}$. This matrix has on average ≈ 18 nonzero entries per row. A naive square root of this matrix using Cholesky decomposition $\mathbf{Q} = \mathbf{K}\mathbf{K}^\mathsf{T}$ produces a square matrix $\mathbf{K}$ with hundreds of nonzeros per row. Reordering helps reduce this, but to perhaps ≈ 25 nonzeros per row. However, if we know where this bi-Laplacian comes from, then we know the Laplacian operator is a square root of the bi-Laplacian $\mathbf{Q} = \left(\mathbf{M}^{-1}\mathbf{L}\right)^\mathsf{T}\mathbf{M}\left(\mathbf{M}^{-1}\mathbf{L}\right) = \left(\sqrt{\mathbf{M}^{-1}}\mathbf{L}\right)^\mathsf{T}\left(\sqrt{\mathbf{M}^{-1}}\mathbf{L}\right)$. The Laplacian will have on average only seven nonzeros per row and can be constructed procedurally without expensive decomposition or reordering.

If this square root matrix $\mathbf{F}$ is sparse then this information is especially useful for interior point solvers like MOSEK [Andersen and Andersen 2000]. The MOSEK solver can optimize non-negative least squares problems (i.e. constant bound constraints $\mathbf{x} \geq 0$) much faster than treating it as a general QP.

Even more interesting is that, when this square root matrix $\mathbf{F}$ is available, we may transform any QP with arbitrary linear inequality constraints into a second-order conic programming problem. These problems minimize *linear* energies:

$$E_{\text{L}}(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{l}, \tag{2.90}$$

subject to linear equality constraints (see Equation (2.62) and cone constraints:

$$\mathbf{x} \in \mathcal{C}, \tag{2.91}$$

where $\mathcal{C}$ is a convex cone.

Many derivations of the following conversion exist, notably in the documentation of MOSEK. We illustrate here not only as a reference but to give intuition to how such cones arise from least-squares QP optimization.

[3] Often in the case of mesh-based quantities the L^2 norm is replaced with a more mesh-sensitive norm. If we have quantities living at each mesh vertex, then measuring $\|\mathbf{F}\mathbf{x}-\mathbf{f}\|^2 = (\mathbf{F}\mathbf{x} - \mathbf{f})^\mathsf{T}(\mathbf{F}\mathbf{x} - \mathbf{f})$ treats each vertex value as having equal weight regardless of its associated Voronoi area on the mesh. Instead we can *integrate* according to the Voronoi-mixed mass matrix described in Section 2.1.3: $(\mathbf{F}\mathbf{x} - \mathbf{f})^\mathsf{T}\mathbf{M}(\mathbf{F}\mathbf{x} - \mathbf{f})$. This is less mesh-dependent when approximating continuous functions. This relates analogously to the earlier discussion of quadratic penalties for weak constraints (see Section 2.2.2).

Conversion from quadratic to conic program

To convert to a conic problem, let us separate the quadratic, linear and constant terms of Equation (2.89):

$$E_{\text{LS}}(\mathbf{x}) = \frac{1}{2}\|\mathbf{F}\mathbf{x} - \mathbf{f}\|^2 \tag{2.92}$$

$$= \frac{1}{2}\|\mathbf{F}\mathbf{x}\|^2 - \mathbf{x}^\mathsf{T}\mathbf{F}^\mathsf{T}\mathbf{f} + \frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{f} \tag{2.93}$$

$$= \frac{1}{2}\|\mathbf{F}\mathbf{x}\|^2 + \mathbf{x}^\mathsf{T}\mathbf{l} + c. \tag{2.94}$$

Now, introduce a *vector* of auxiliary variables $\mathbf{t}$ and rewrite our energy and linear inequality constraints in Equation (2.62):

$$\begin{aligned} \underset{\mathbf{x},\mathbf{t}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{t}\|^2 + \mathbf{x}^\mathsf{T}\mathbf{l} + c \\ \text{subject to} \quad & \mathbf{F}\,\mathbf{x} - \mathbf{t} = 0, \\ & \mathbf{A}_{\text{ie}}^\mathsf{T}\mathbf{x} \leq \mathbf{b}_{\text{ie}}. \end{aligned} \tag{2.95}$$

Using an auxiliary scalar variable v we convert into conic form:

$$\begin{aligned} \underset{\mathbf{x},\mathbf{t},v}{\text{minimize}} \quad & v + \mathbf{x}^\mathsf{T}\mathbf{l} + c \\ \text{subject to} \quad & \mathbf{F}\mathbf{x} - \mathbf{t} = 0, \\ & \mathbf{A}_{\text{ie}}^\mathsf{T}\mathbf{x} \leq \mathbf{b}_{\text{ie}}, \\ & 2v \geq \mathbf{t}^\mathsf{T}\mathbf{t}. \end{aligned}$$

where the inequality constraint on v forces its value to be inside the cone described by the coordinates of $\mathbf{t}$.

Putting all variables in a column vector, we can write this in matrix form, as we supply it to the MOSEK solver:

$$\begin{aligned} \underset{\begin{bmatrix}\mathbf{x}^\mathsf{T} & \mathbf{t}^\mathsf{T} & v\end{bmatrix}}{\text{minimize}} \quad & \begin{bmatrix}\mathbf{x}^\mathsf{T} & \mathbf{t}^\mathsf{T} & v\end{bmatrix}\begin{bmatrix}\mathbf{l}\\ \mathbf{0}\\ 1\end{bmatrix} + c \\ \text{subject to} \quad & \begin{bmatrix}\mathbf{F} & -\mathbf{I} & *\\ \mathbf{A}_{\text{ie}}^\mathsf{T} & * & *\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{t}\\ v\end{bmatrix} \geq \begin{bmatrix}\mathbf{0}\\ -\infty\end{bmatrix}, \\ & \begin{bmatrix}\mathbf{F} & -\mathbf{I} & *\\ \mathbf{A}_{\text{ie}}^\mathsf{T} & * & *\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{t}\\ v\end{bmatrix} \leq \begin{bmatrix}\mathbf{0}\\ \mathbf{b}_{\text{ie}}\end{bmatrix}, \\ & 2v \geq \sum_i t_i^2. \end{aligned}$$

Similar conversions also exist for treating general QPs as nonnegative least-squares problems, which enjoy specialized solvers such as [Brand and Chen 2011]. However, this conversion via the Lagrangian duality requires computing $\mathbf{A}_{\text{ie}}\mathbf{Q}^{-1}\mathbf{A}_{\text{ie}}^\mathsf{T}$, which is likely dense even for sparse $\mathbf{Q}$ and $\mathbf{A}_{\text{ie}}$.

3

Mixed finite elements for variational surface modeling

Many problems in geometric modeling can be described using variational formulations that define the smoothness of the shape and its behavior w.r.t. the posed modeling constraints. For example, high-quality C^2 surfaces that obey boundary conditions on positions, tangents and curvatures can be conveniently defined as solutions of high-order geometric PDEs. The advantage of such a formulation is its conceptual representation-independence. In practice, solving high-order problems efficiently and accurately for surfaces approximated by meshes is notoriously difficult. Classical FEM approaches require high-order elements which are complex to construct and expensive to compute. Recent discrete geometric schemes are more efficient, but their convergence properties are hard to analyze, and they often lack a systematic way to impose boundary conditions. In this chapter, we present an approach for discretizing common PDEs on meshes using mixed *finite elements, where additional variables for derivatives in the problem are introduced. Such formulations use first-order derivatives only, allowing a discretization with simple linear elements. Various boundary conditions can be naturally discretized in this setting. We formalize continuous* region constraints*, and show that these seamlessly fit into the mixed framework. We demonstrate mixed FEM in the context of diverse modeling tasks and analyze its effectiveness and convergence behavior.*

3.1 Introduction

A variety of geometric modeling problems are solved using energy minimization or geometric PDEs: defining a smooth surface interpolating fixed points or curves, filling holes and connecting pieces of geometry, deformations, and cut-and-paste operations. Such energies and PDEs

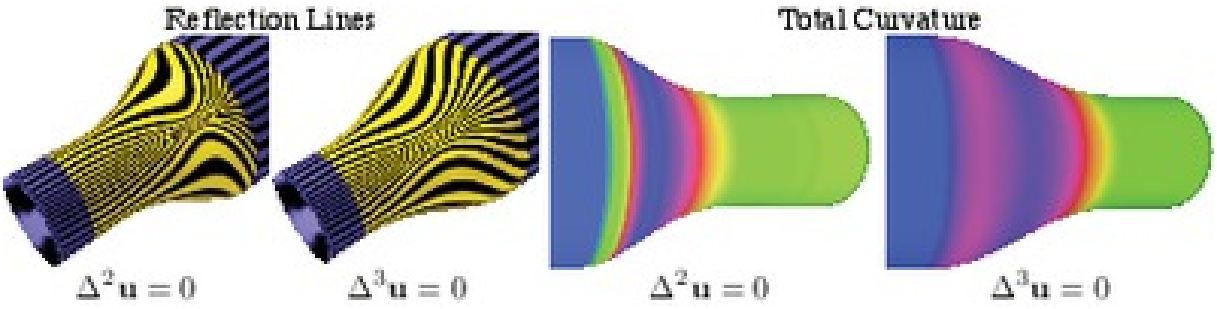


Figure 3.1: *Blending two cylinders of different diameters using triharmonic and biharmonic equations.*

are defined in terms of differential quantities such as tangents, curvatures and curvature derivatives. Boundary conditions for these problems play an important role, for instance, ensuring that surfaces are joined smoothly. Higher-order boundary conditions can only be imposed if the variational problems are of high order: C^1 and C^2 conditions can be imposed for PDEs of order at least four and six, respectively.

While the PDEs of interest have high order, they usually need to be solved for surfaces approximated by meshes, and numerical schemes that produce mesh-independent solutions in the limit of fine meshes and remain efficient need to be devised. Many reliable finite element discretizations are available for fourth-order problems, yet all standard conforming and non-conforming elements require additional degrees of freedom associated with derivatives at vertices and edge midpoints, increasing the overall number of variables and complicating implementation. Furthermore, sixth-order problems, which are highly important in geometric modeling applications, rarely appear in finite-element literature.

A currently preferred alternative in many geometry processing applications is to formulate discrete analogs of variational problems using discrete-geometric operators, such as discrete Laplacians. These approaches are simple, robust and perform well in practice, but lack the mesh-independence guarantees provided by the finite elements designed to be convergent, although mesh-independent and convergent behavior was observed experimentally for biharmonic problems on a range of mesh types [Grinspun et al. 2006]. An additional important complication is a systematic treatment of boundary conditions: while satisfactory heuristics were designed for a number of cases (most importantly, for *region* conditions, see Section 3.3), other types of conditions, such as direct specification of tangents or curvature along a boundary, are more difficult and typically require ad hoc mesh extension.

In this chapter, we present a discretization for two common PDEs on meshes using *mixed* finite elements, a well-established finite-element technique. The main idea is to introduce additional variables for the derivatives in the problem, along with additional constraint equations relating these variables to the original ones, thereby reducing a high-order equation to a low-order system which can be discretized with linear elements. We show that the intuitive and commonly used discretization of biharmonic and triharmonic equations introduced in [Botsch and Kobbelt 2004] can be viewed as a transformation of a mixed element discretization, and is directly related to the Ciarlet-Raviart discretization of the biharmonic equation. For the triharmonic equation, we show that a standard mixed FEM discretization may lead to a singular system, but an alternative formulation with region constraints does not suffer from this

problem. We show how different types of boundary conditions commonly used for geometric problems can be discretized with mixed elements. We formalize the notion of continuous *region constraints*, which are common in geometric modeling but rarely considered in finite element literature, and demonstrate that these also naturally fit into the mixed framework.

We explore the convergence behavior of mixed discretizations for a variety of mesh types and evaluate the degree of mesh dependence of the discretization. framework in geometric modeling, such as interactive shape editing, hole filling, blending (see Figure 3.1), and surface patch construction from boundary curves. To this end, this work is a direct continuation of that begun in [Tosun 2008].

3.2 Previous work

Many works in graphics and geometric modeling have considered surface design based on PDEs. In the following we briefly describe the most relevant literature, classifying it by the type of PDEs addressed, the employed discretization and the possible boundary conditions.

In the special case of simple domains and analytically specified boundary conditions, PDEs can be solved in closed form without any discretization [Bloor and Wilson 1990]. In more general settings, the equations need to be discretized and solved numerically. Moreton and Séquin [1992] model shapes that minimize the curvature variation energy, where positions, tangents and normal curvatures are specified along curves, and Bézier patches are used to approximate the surface. Welch and Witkin [1994] also use curvature variation to interpolate curve networks and points, but on general triangular meshes; they compute the required differential quantities with local quadratic fits and finite differences.

Local quadratic fits may suffer from instabilities, unless sufficiently large number of vertices are used, which results in lower performance. To gain speed and accuracy (by using much finer meshes) many recent approaches adopt discrete differential operators such as the discrete Laplacians. Taubin [1995] used the uniform-weight discrete Laplacian for fair surface design and proposed constraining point positions or discrete Laplacian values at vertices. Later works use the more accurate, discrete cotangent Laplacian [Pinkall and Polthier 1993], closely related to FEM discretization [Wardetzky et al. 2007a, Reuter et al. 2009].

Mixed finite elements are based on factoring a higher-order problem into a system of lower-order problems. This idea is also used in the context of discrete differential operators when boundary conditions allow this, e.g. Schneider and Kobbelt [2000, 2001] factor a PDE involving the Laplacian of mean curvature $\Delta H = 0$ into two second-order equations. In the FIBERMESH system [Nealen et al. 2007], the same approach is used to construct fair surfaces that interpolate a set of arbitrary curves, with no tangents or curvatures fixed. Similarly, [Joshi and Carr 2008] decompose $\Delta^2 \mathbf{x} = 0$ for curve inflation, fixing $\mathbf{x}$ and $\Delta \mathbf{x}$ to boundary curves with prescribed position and mean curvature normal values. A more complete survey of discrete variational techniques can be found in [Botsch and Sorkine 2008]. A priori, these approaches lack convergence guarantees for general meshes. In absence of convergence, the degree of mesh dependence is hard to predict, and adaptive refinement techniques are hard to apply. In [Grinspun et al. 2006], it is demonstrated that the discretization of the Laplacian en-

ergy (equivalently, the biharmonic equation) based on the cotangent weights does not satisfy a version of the patch test (a standard test used for verifying convergence of finite elements in engineering), yet empirically it exhibits convergent behavior. The mixed FEM point of view described in this thesis provides a different approach for analysis of this discretization.

The closest work to ours is the method of Clarenz et al. [2004] that applies the Willmore flow (fourth-order geometric PDE) to fair surface design by using FEM with auxiliary variables $\mathbf{y} = \Delta\mathbf{x}$. Their formulation allows prescribing $\mathbf{x}$ values and co-normals on the boundary. We demonstrate that this and other discrete approaches can be viewed as an application of the mixed finite element discretization, and that a variety of discrete boundary conditions (region conditions) used in [Botsch and Kobbelt 2004], [Clarenz et al. 2004] and [Xu et al. 2006] can be derived from a continuous formulation.

Finite elements offer a consistent way of solving geometric PDEs and treating various boundary conditions; however, relatively complex higher-order elements are required for conforming discretization of higher-order PDEs. Non-conforming elements such as DKT, are widely used in engineering for fourth-order problems but are difficult to extend to higher orders and require many additional derivative-related degrees of freedom. Some of the early work applied standard engineering elements to surface modeling [Celniker and Gossard 1991], yet most methods, especially for interactive applications, used simpler discretizations for reasons of efficiency and implementation ease. Mixed finite elements for fourth-order problems offer an alternative approach that allows using piecewise linear basis functions. Although additional variables are introduced, they have a natural interpretation and in most cases can be eliminated inexpensively. A mixed formulation was introduced for the biharmonic equation in [Ciarlet and Raviart 1974] and its variations are considered in [Falk 1978, Monk 1987, Amara and Dabaghi 2001] and many other papers. Convergence of mixed discretization for linear elements was shown in [Scholz 1978]. In contrast to biharmonic equations, sixth-order equations (especially triharmonic) did not receive much attention, as few physical problems result in such equations. One exception is [Bramble and Falk 1985], deriving theoretical convergence estimates for a mixed formulation of polyharmonic equations. While piecewise-linear discretizations extend naturally to the case of non-flat domains, convergence questions require special treatment; the Laplace-Beltrami equation and Willmore flow discretizations were analyzed in [Dziuk 1988, Dziuk 1990, Deckelnick and Dziuk 2006].

3.3 Model problems

We consider two important examples: Laplacian and Laplacian gradient energies, leading to biharmonic and triharmonic equations (Figure 3.1). We use the notation $\langle f, g\rangle_X = \int_X fg\, dA$ for the L^2 inner product of two functions on a domain X (an area or a curve in the plane or on a surface). When the subscript is omitted, the domain Ω is implied. For vector-valued functions, denoted by bold letters, the product in the integral is replaced by dot product.

In the first problem we compute a deformation $\mathbf{u}$ of a planar sheet occupying an area Ω_0 in the plane so that the Laplacian energy is minimized:

$$E_B = \tfrac{1}{2}\langle \Delta\mathbf{u}, \Delta\mathbf{u}\rangle_{\Omega_0} \;\rightarrow\; \min, \tag{3.1}$$

where $\mathbf{u} : \Omega_0 \to \mathbb{R}^3$ is the deformation. The related Euler-Lagrange equation is the *biharmonic equation* $\Delta^2 \mathbf{u} = 0$.

The second example is the *Laplacian gradient* energy

$$E_T = \tfrac{1}{2}\langle \nabla\Delta\mathbf{u}, \nabla\Delta\mathbf{u}\rangle_{\Omega_0} \;\to\; \min, \tag{3.2}$$

with boundary conditions on second derivatives, important for modeling curvature-continuous surfaces. The Euler-Lagrange equation for this problem is the sixth order *triharmonic equation*: $\Delta^3 \mathbf{u} = 0$.

To solve this problem numerically we need to answer three main questions:

- How do we discretize the functional involving second derivatives using a triangle mesh which is only C^0 continuous?
- How do we impose different types of boundary conditions?
- What guarantees do we have that the answer remains reasonably consistent for any mesh we choose for Ω_0?

For example, a common answer to the first question in the case of (3.1) is to use a discrete-geometric analog of the Laplacian, such as the well-known cotangent formula. It is less clear how to apply this formula near the boundary, and how to impose boundary conditions; most commonly, constrained vertices are added outside the domain. The validity of such discretization remains questionable if mesh-independence is desired at least for finer meshes: the cotangent formula does not converge pointwise, so, strictly speaking, it yields only a mesh-dependent analog of the Laplacian energy.

3.3.1 Low-order decomposition

A general systematic approach to discretization of problems involving high-order derivatives is to convert an unconstrained optimization problem like (3.1) into a lower-order constrained optimization problem with additional variables. Instead of the Euler-Lagrange equations for (3.1), we solve the constrained problem

$$\tfrac{1}{2}\langle \mathbf{v}, \mathbf{v}\rangle_{\Omega_0} \to \min, \;\; \text{s.t.} \;\; \Delta\mathbf{u} = \mathbf{v}. \tag{3.3}$$

Similarly, instead of minimizing energy E_T, we solve

$$\tfrac{1}{2}\langle \nabla\mathbf{v}, \nabla\mathbf{v}\rangle_{\Omega_0} \to \min, \;\; \text{s.t.} \;\; \Delta\mathbf{u} = \mathbf{v}. \tag{3.4}$$

For the Laplacian gradient energy, we no longer need to discretize third derivatives. Furthermore, in both cases one can also eliminate second derivatives by using Green's identity:

$$L_B = \tfrac{1}{2}\langle \mathbf{v}, \mathbf{v}\rangle_{\Omega_0} + \langle \lambda, \Delta\mathbf{u} - \mathbf{v}\rangle_{\Omega_0} \tag{3.5}$$

$$= \tfrac{1}{2}\langle \mathbf{v}, \mathbf{v}\rangle_{\Omega_0} - \langle \lambda, \mathbf{v}\rangle_{\Omega_0} + \langle \lambda, \tfrac{\partial \mathbf{u}}{\partial n}\rangle_{\partial\Omega} - \langle \nabla\lambda, \nabla\mathbf{u}\rangle_{\Omega_0}. \tag{3.6}$$

The partial derivative $\partial/\partial n$ is w.r.t. the normal of the boundary curve. A similar transformation yields a formulation with first-order derivatives for the minimization of E_T. Similar considerations apply for weak formulations of high-order PDEs in general. However, in this case,

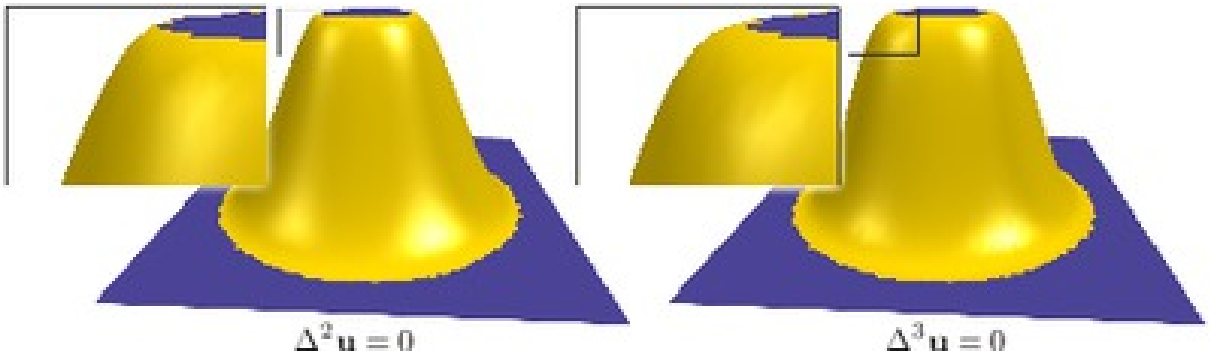


Figure 3.2: *Region conditions: the blue areas (Ω_f) are fixed and the yellow part (Ω) is solved for. The circular region in the middle was lifted upwards. Solving $\Delta^k\mathbf{u} = 0$ with region boundary conditions. The triharmonic solution is smoother near the region boundaries.*

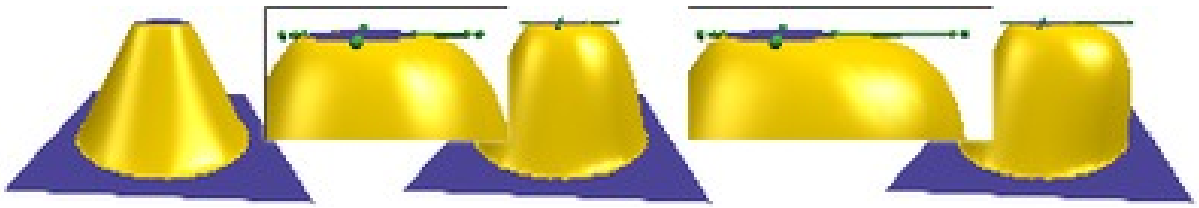


Figure 3.3: *Prescribing tangent control on boundary curves. The leftmost image shows a biharmonic surface with $\partial\mathbf{u}/\partial n = 0$. Tangents can be explicitly manipulated by the user (see also Figure 3.6).*

one needs to consider a *weak* formulation, (e.g. $\langle \Delta u, v \rangle = 0$ for any function v for which the integral is defined, instead of $\Delta u = 0$), and apply integration by parts to reduce the order of the equation. For the transformed problems, piecewise-linear elements can be used for all quantities, as it is commonly done for second-order problems. This idea is the foundation of the *mixed* finite element discretizations (Section 3.4). The constraint-based formulation has an additional significant advantage: it allows us to treat *region* boundary conditions in a systematic way, as explained below.

3.3.2 Boundary condition types

We define various boundary conditions in the continuous case. While for some applications (e.g. deformations of a fixed mesh) the problem can be studied entirely in the discrete domain, we are concerned with cases when a common reference point is needed for meshes with different connectivity and resolution that approximate the same shape.

In geometric modeling literature, the boundary conditions for problems on meshes are often formulated by assuming that a part of the mesh outside the solution domain is available or by adding vertices on the boundaries of the mesh using a heuristic. In contrast, in FEM discretizations, it is typically assumed that the explicit conditions on boundary derivatives are given, and the mesh does not extend beyond the boundary of the solution domain. Both types of conditions are relevant for geometric problems.

A boundary condition can be associated with subsets of the domain Ω_0 of different dimensions:

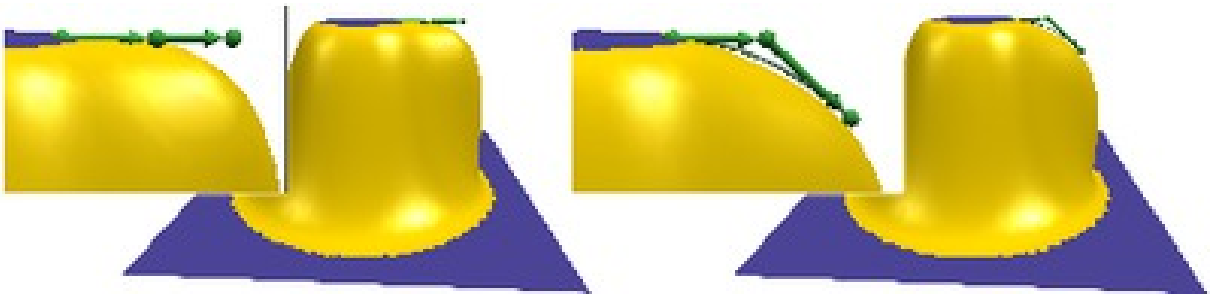


Figure 3.4: *Prescribing curvature control on boundary curves. A triharmonic surface with boundary curvature manipulated via a Bézier control widget.*

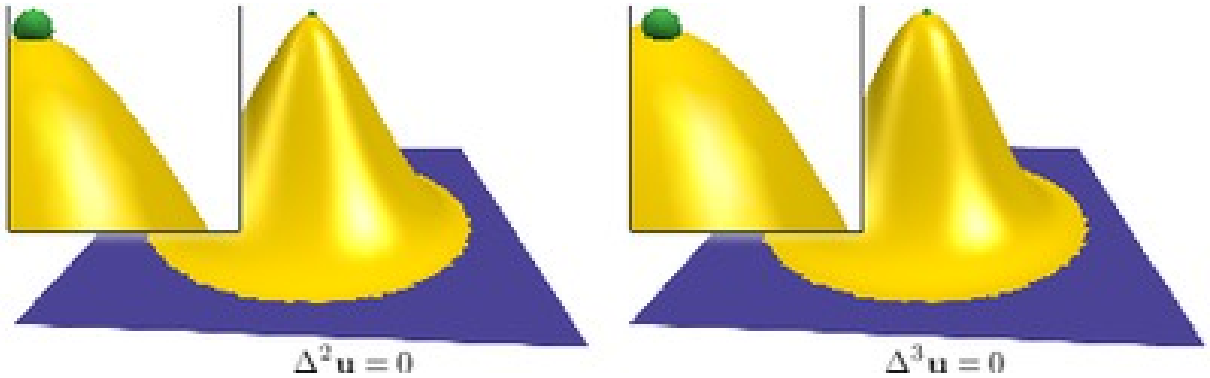


Figure 3.5: *Point boundary conditions: an isolated point is highlighted by the green dot. The triharmonic surface is smoother around the boundary (C^2).*

- *region* boundary conditions on open domains $\Omega_f \subset \Omega_0$;
- *curve* boundary conditions on curves $C \subset \Omega_0$;
- *point* boundary conditions on isolated points $P \in \Omega_0$.

Fixed region boundary conditions require that, given a domain Ω_f and a fixed function $\mathbf{u}_f$,

$$\mathbf{u}|_{\Omega_f} = \mathbf{u}_f, \quad \text{and } \mathbf{u} \text{ is } C^k \text{ on } \Omega, \text{ for some } k \leq 2.$$

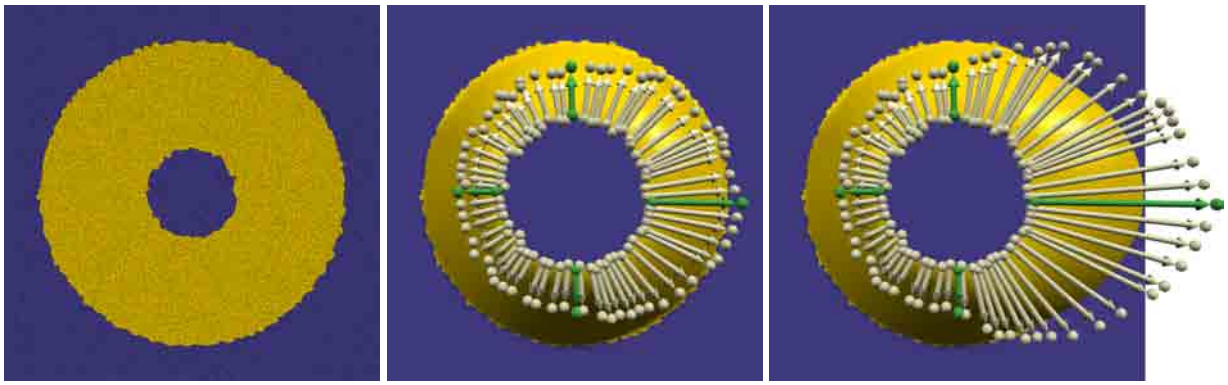
These boundary conditions are illustrated in Figure 3.2. This type of conditions can be viewed as the continuous analog of conditions defined directly for meshes in [Clarenz et al. 2004, Xu et al. 2006].

Fixed curve boundary conditions are most commonly considered in the FEM setting. On a curve C (typically a part of $\partial\Omega$, but possibly in the interior of Ω_0), we require that

$$\left.\frac{\partial^i \mathbf{u}}{\partial n^i}\right|_C = \mathbf{b}_i, \quad i \leq k \text{ for } k \leq 2.$$

Figures 3.3 & 3.6 demonstrates the usage of this condition for explicit tangent control on a biharmonic surface. For triharmonic surfaces, we may specify both tangents and curvature (second derivatives) at the boundary, as shown in Figure 3.4.

Free (natural) curve boundary conditions are naturally obtained from energy-based problems; they are needed at a boundary with no Dirichlet constraints in order for the energy to be minimal.



meshed domain tangent constraints (see Figure 3.3)

Figure 3.6: *The mesh domain used in our examples for region and curve boundary conditions. Tangent vectors can be explicitly manipulated by the user (green vectors in the right two images). The rest of the tangents along the boundary are automatically interpolated (light brown).*

They can also be applied to PDEs not originating from an energy, yielding similar boundary behavior.

For the Laplacian energy, integration by parts leads to free boundary conditions $\partial\mathbf{u}/\partial n = \partial\Delta\mathbf{u}/\partial n = 0$. For the Laplacian gradient energy, the condition $\partial\Delta^2\mathbf{u}/\partial n = 0$ needs to be added. However, no special effort is needed to enforce them when the discretization is derived from the energy, hence we do not consider them in detail. Figure 3.3 (left) shows the effect of these boundary conditions for the biharmonic solutions. These conditions are used extensively in the following Chapters 4 & 5. In these chapters, we define functions which may meet user given constraints only in the interior of the domain. Thus, the boundary of the domain (boundary curve in $\mathbb{R}^2$ or a solid's surface in $\mathbb{R}^3$ is from the user's point of view left unconstrained. Effectively these natural boundary conditions (and their higher order cousins) are enforced implicitly (see also Section 4.7).

One may also consider other combinations of conditions for curves. For a biharmonic surface we may fix any pair of $\mathbf{u}$, $\partial\mathbf{u}/\partial n$, $\Delta\mathbf{u}$, and $\partial\Delta\mathbf{u}/\partial n = 0$ [Helenbrook 2003]. These possibilities should be obvious when examining the biharmonic equation in factored form: two Poisson equations, one in $\mathbf{u}$ and one in $\mathbf{v} = \Delta\mathbf{u}$. Thus we may fix either Dirichlet or Neumann conditions for each. Note, however, that fixing only $\Delta\mathbf{u}$, and $\partial\Delta\mathbf{u}/\partial n = 0$ is not a possible combination as it will not uniquely determine $\mathbf{u}$. Other pairs, such as Dirichlet constraints and $\Delta\mathbf{u} = \mathbf{v}$ may be useful (e.g. [Joshi and Carr 2008]), however we focus on directly controlling first derivatives (tangents), which we argue is a more intuitive interface.

Point boundary conditions are identical to region boundary conditions, except instead of a function defined on Ω_f, we use a set of values defined at isolated points $\mathbf{p}_i$ of Ω_0. An example is shown in Figure 3.5.[1]

[1] Point constraints remain a curiosity in the continuous case. For the harmonic equation, they result in discontinuous indicator functions [Crane 2012]. In the discrete case, fixing isolated points is tantamount to fixing a mesh dependent sub-discretization-level curve of Dirichlet conditions around the point. The situation becomes more interesting for higher order systems, where discontinuous appear (only) in derivatives.

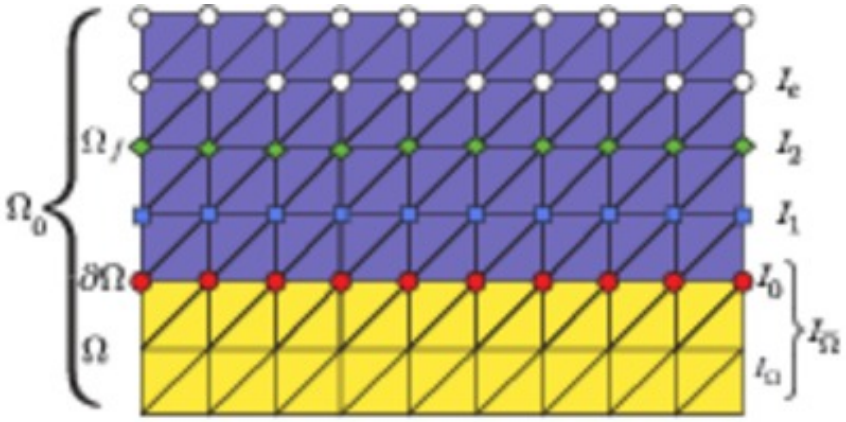


Figure 3.7: *Notation for different areas of the mesh. For fourth-order equations, we include I_2 into I_e.*

3.4 Mixed finite element discretization

In this section, we demonstrate, using the two model examples discussed in Section 3.3, how low-order decomposition can be used to discretize such problems with piecewise-linear elements only, and how the different types of boundary conditions can be imposed.

Our main focus is on the *region* boundary conditions, which lack a systematic treatment in finite element context. We demonstrate the relationship between the classical Ciarlet-Raviart mixed-element discretization of the biharmonic equation [Ciarlet and Raviart 1974], discrete-geometric discretization of [Botsch and Kobbelt 2004] and our region constraint formulation.

Notation. Please refer to Figure 3.7. We assume that the domain Ω_0 has polygonal boundary and is meshed (the discretizations we describe, with some restrictions, are applicable to domains with curved boundaries approximated by polygonal domains Ω_0^h for each resolution). The parameter h denotes the average edge length of the mesh. The domain Ω is defined as the subset of Ω_0 excluding the union of Ω_f with constrained curves and points, and the complement of Ω in Ω_0 is denoted by Ω_e.

The set of vertex indices in the interior of Ω is denoted by I_Ω, the set of vertex indices on the boundary of Ω is I_0, the sets of indices in Ω_e in two layers outside the boundary are I_1 and I_2. The set of all vertex indices in Ω_0 is I, and the set of vertices in Ω_e is I_e. In a matrix $\mathbf{S}$ with rows and columns corresponding to vertex indices, we use subscripts Ω, 0, 1, 2 and e, to define *sliced* submatrices with rows and columns coming from corresponding index sets. We also use subscripts like 01 to denote $I_1 \cup I_0$, and $\bar{\Omega}$ for $I_\Omega \cup I_0$, i.e., Ω with its boundary included.

3.4.1 Laplacian energy & biharmonic equation

We start with the very well studied example of Laplacian energy (3.1), whose Euler-Lagrange equation is the biharmonic equation. It reveals the essential relations between conventional mixed finite element discretizations, the region constraint discretization that we propose, and the discrete-geometric approach of [Botsch and Kobbelt 2004].

Region conditions require an additional term in the Lagrangian, constraining the solution to coincide with a given function on Ω_f:

$$\tfrac{1}{2}\langle \mathbf{v}, \mathbf{v}\rangle_{\Omega_0} - \langle \lambda, \mathbf{v}\rangle_{\Omega_0} + \langle \lambda, \tfrac{\partial \mathbf{u}}{\partial n}\rangle_{\partial\Omega} - \langle \nabla\lambda, \nabla\mathbf{u}\rangle_{\Omega_0} + \langle \mu, \mathbf{u}_f - \mathbf{u}\rangle_{\Omega_f}, \tag{3.7}$$

where μ is the Lagrange multiplier function for the region constraint. It is important to note that the first constraint, $\Delta\mathbf{u} = \mathbf{v}$, is enforced over *all* of Ω_0, which ensures C^1 continuity across the boundary of Ω.

A similar Lagrangian, with the first term replaced with $\langle \nabla\mathbf{v}, \nabla\mathbf{v}\rangle_{\Omega_0}$ is obtained for the triharmonic problem (3.2):

$$\tfrac{1}{2}\langle \nabla\mathbf{v}, \nabla\mathbf{v}\rangle_{\Omega_0} - \langle \lambda, \mathbf{v}\rangle_{\Omega_0} + \langle \lambda, \tfrac{\partial \mathbf{u}}{\partial n}\rangle_{\partial\Omega} - \langle \nabla\lambda, \nabla\mathbf{u}\rangle_{\Omega_0} + \langle \mu, \mathbf{u}_f - \mathbf{u}\rangle_{\Omega_f}. \tag{3.8}$$

Discretization with piecewise linear elements. We use piecewise-linear approximations for $\mathbf{u}$, $\mathbf{v}$, λ and μ, all of the form $\sum_{i\in I} a_i\varphi_i$. For a quantity a, we denote the piecewise linear approximation by a^h.

Substituting these approximations into the Lagrangian (3.7), and differentiating w.r.t. all free variables $\mathbf{v}_i$, $\mathbf{u}_i$, λ_i, μ_i, defined at vertices $i \in I \setminus I_{\partial\Omega}$, we obtain the following system of equations:

$$\begin{aligned}
\sum_{j\in I} (\mathbf{v}_j - \lambda_j)\langle \varphi_i, \varphi_j\rangle &= 0,\\
\sum_{j\in I} -\lambda_j\langle \nabla\varphi_i, \nabla\varphi_j\rangle - \sum_{j\in I} \mu_j\langle \varphi_j, \varphi_i\rangle_{\Omega_f} &= 0,\\
\sum_{j\in I} -\mathbf{u}_j\langle \nabla\varphi_i, \nabla\varphi_j\rangle + \sum_{j} \mathbf{v}_j\langle \varphi_j, \varphi_i\rangle &= 0,\\
\sum_{j\in I} (\mathbf{u}_j - \mathbf{u}_j^f)\langle \varphi_j, \varphi_i\rangle_{\Omega_f} &= 0.
\end{aligned} \tag{3.9}$$

The first equation allows us to eliminate λ_i immediately as it has to coincide with $\mathbf{v}_i$, leaving variables $\mathbf{u}$, $\mathbf{v}$ and μ. One can observe that the coefficients of the system mostly come from the discrete Laplacian matrix $\mathbf{L}_{ij} = -\langle \nabla\varphi_i, \nabla\varphi_j\rangle$, and the mass matrix $\mathbf{M}_{ij}^{full} = \langle \varphi_i, \varphi_j\rangle$ (see Sections 2.1.2 & 2.1.3 respectively).

Lumped mass matrices. The mass matrix $\mathbf{M}^{full}$ is often replaced by a *lumped* mass matrix, i.e. a diagonal matrix $\mathbf{M}^d$. The lumping step is mathematically sound: replacing $\langle \mathbf{v}^h, \mathbf{v}^h\rangle = \mathbf{v}^{hT}\mathbf{M}\mathbf{v}^h$ with $\mathbf{v}^{hT}\mathbf{M}^d\mathbf{v}^h$ does not affect the convergence rate for the solution, if the quadrature rule has accuracy $O(h)$, which is satisfied by using vertices as quadrature points [Ciarlet 1978, Brezzi and Fortin 1991], leading to a diagonal mass matrix. From now on, we assume that the mass matrix in (3.9) was replaced by a diagonal matrix $\mathbf{M}^d$ whose diagonal entries are denoted by $\mathbf{D}_i$. Two approaches to computing $\mathbf{D}_i$ are discussed in Section 2.1.3 and briefly compared for our model problems in Section 3.5. With a diagonal mass matrix, the

system reduces to:

$$\begin{aligned} \sum_{j\in I} \mathbf{v}_j \mathbf{L}_{ij} + \mathbf{D}_i \mu_i &= 0 \text{ if } i \in I_e, \\ \sum_{j\in I} \mathbf{v}_j \mathbf{L}_{ij} &= 0 \text{ if } i \notin I_e, \\ \sum_{j\in I} \mathbf{L}_{ij}\mathbf{u}_j - \mathbf{D}_i \mathbf{v}_i &= 0, \\ \sum_{j\in I} \mathbf{u}_j - \mathbf{u}_j^f &= 0 \text{ if } i \in I_e. \end{aligned} \tag{3.10}$$

As the values of the Lagrange multiplier μ are generally not of interest to us, we can eliminate the first set of equations defining μ_i in terms of $\mathbf{v}_i$, since μ_i are not present in other equations. The last set of equations indicates that for i on I_f, $\mathbf{u}_i$ can be replaced by the known values $\mathbf{u}_i^f$. Finally, we can compute the values of $\mathbf{v}_i$ for $i \in I_e$ using $\sum_{j\in I} \mathbf{L}_{ij}\mathbf{u}_j = \mathbf{D}_i\mathbf{v}_i$, as for entries away from the boundary, all $\mathbf{u}_j$ with nonzero $\mathbf{L}_{ij}$ are fixed values $\mathbf{u}_j^f$.

Eliminating values of $\mathbf{u}_j$ and $\mathbf{v}_j$ in fixed regions from the system after swapping equations and moving known quantities to the right-hand side, yields in matrix form:

$$\begin{bmatrix} -\mathbf{M}^d & \mathbf{L}_{\bar{\Omega},\Omega} \\ \mathbf{L}_{\Omega,\bar{\Omega}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\bar{\Omega}} \\ \mathbf{u}_\Omega \end{bmatrix} = \begin{bmatrix} -\mathbf{L}_{\bar{\Omega},0}\mathbf{u}_0^f - \mathbf{L}_{\bar{\Omega},1}\mathbf{u}_1^f \\ 0 \end{bmatrix}. \tag{3.11}$$

The system can be simplified even further, if we observe that $\mathbf{M}^d$ can be easily inverted, and $\mathbf{v}_{\bar{\Omega}}$ can be eliminated from the system, yielding

$$\mathbf{L}_{\Omega,\bar{\Omega}}(\mathbf{M}^d)^{-1}\mathbf{L}_{\bar{\Omega},\Omega}\mathbf{u}_\Omega = -\mathbf{L}_{\Omega,\bar{\Omega}}(\mathbf{M}^d)^{-1}\mathbf{L}_{\bar{\Omega},01}\mathbf{u}_{01}^f, \tag{3.12}$$

leaving only $\mathbf{u}_\Omega$ as the unknown.

Observe that *point* conditions are enforced using the same system of equations: for an isolated point constraint, assuming it is at a mesh vertex, I_1 is empty, so the second term of the right-hand side is absent.

Fixed curve conditions. Fixed-curve boundary conditions of the form $\mathbf{u} = \mathbf{b}_0$ and $\partial\mathbf{u}/\partial n = \mathbf{b}_1$ are commonly used in mixed discretizations of the biharmonic equation; we refer to [Ciarlet and Raviart 1974, Amara and Dabaghi 2001] for details. An approach similar to the one outlined above can be used, with the Lagrangian in (3.6) restricted to $\bar{\Omega}$. This yields the system of equations

$$\begin{bmatrix} -\mathbf{M}^\Omega & \mathbf{L}_{\bar{\Omega},\Omega} \\ \mathbf{L}_{\Omega,\bar{\Omega}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\bar{\Omega}} \\ \mathbf{u}_\Omega \end{bmatrix} = \begin{bmatrix} -\mathbf{L}^\Omega_{\bar{\Omega},0}\mathbf{b}_0 - \mathbf{N}^{\partial\Omega}_{\bar{\Omega},0}\mathbf{b}_1 \\ 0 \end{bmatrix}, \tag{3.13}$$

where the matrix $\mathbf{N}_{ij}^{\partial\Omega} = \langle\varphi_j, \varphi_i\rangle_{\partial\Omega}$, and $\mathbf{L}^\Omega$ and $\mathbf{M}^\Omega$ are the discrete Laplacian and mass matrices with integration performed over Ω only, as $\langle\cdot\rangle_\Omega = \langle\cdot\rangle_{\Omega_0} - \langle\cdot\rangle_{\Omega_e}$, $\mathbf{L}^\Omega = \mathbf{L} - \mathbf{L}^{\Omega_e}$. This is exactly the Ciarlet-Raviart discretization for piecewise-linear elements with convergence established in [Scholz 1978].

Comparing (3.13) and (3.11), we observe two differences: first, the right-hand side term $\mathbf{L}_{\bar{\Omega},1}\mathbf{u}_1^f$ is replaced with $-\mathbf{L}_{\bar{\Omega},1}^{\Omega_e}\mathbf{u}_1^f + \mathbf{N}_{\bar{\Omega},0}^{\partial\Omega}\mathbf{b}_1$; second, the full mass matrix is retained (in this case, lumped matrices are not required to eliminate variables). A more detailed analysis presented in the Section 3.7 shows that the difference between the solutions of the two systems converges to zero as $h \to 0$. Replacing $\mathbf{M}^{\Omega}$ with diagonal $\mathbf{M}^{\Omega,d}$ yields a system matrix similar to (3.11), except the integration is over Ω only; the same elimination procedure can be applied to obtain a system for $\mathbf{u}$ alone.

Our observations are summarized in Proposition 1:

Proposition 1. *The systems* (3.11) *and* (3.12) *are equivalent to the discretization presented in [Botsch and Kobbelt 2004], and, up to a perturbation of the right-hand side and restricting mass-matrix integration to* Ω*, to the Ciarlet-Raviart system* (3.13)*.*

We note that adjusting $\mathbf{b}_1$ offers a degree of control over how smoothly the surface approaches the boundary, similar to the control offered by the λ parameter in [Botsch and Kobbelt 2004] but without modifying the system matrix (which leads to more efficient computations) and with more direct interpretation.

Convergence. As we discuss in more detail in the Section 3.7, this connection between (3.11) and (3.13) can be used to apply available theory for (3.11) and establish convergence. Note that the cotangent formula discretization of the Laplacian energy discussed in [Grinspun et al. 2006] exactly matches (3.11). While it fails the consistency part of the version of the patch test discussed there, this, by itself, does not preclude convergence, which in the case of (3.11) is established by reduction to Ciarlet-Raviart system.

We emphasize that using a conforming FEM discretization in the case of constrained problems like (3.11) does not guarantee convergence. In fact, the more general technique for analysis of mixed elements based on the LBB condition [Brezzi and Fortin 1991] cannot be applied to any discretization of the biharmonic equation. All known estimates are suboptimal, in a sense that the rate of convergence of the solution is lower than $O(h^2)$ approxmiation power of piece-wise linear finite elements. The H^1 norm estimates in [Scholz 1978] and [Amara and Dabaghi 2001], combined with Poincare-Friedrichs inequality lead to L^2 norm convergence rates $h^{7/4}(\log h)^{3/2}$.

3.4.2 Laplacian gradient energy & triharmonic equation

The similarity between the Lagrangians (3.6) and (3.8) allows deriving a piecewise-linear discretization of the triharmonic system in a similar way: the main steps remain the same, namely *(i)* use piece-wise linear approximations for all functions; *(ii)* replace mass matrices with lumped mass matrices; *(iii)* eliminate unneeded variables.

Starting with (3.8),

$$\langle \nabla\mathbf{v}, \nabla\mathbf{v}\rangle_{\Omega_0} - \langle \lambda, \mathbf{v}\rangle_{\Omega_0} + \langle \lambda, \tfrac{\partial \mathbf{u}}{\partial n}\rangle_{\partial\Omega} - \langle \nabla\lambda, \nabla\mathbf{u}\rangle_{\Omega_0} + \langle \mu, \mathbf{u}_f - \mathbf{u}\rangle_{\Omega_f}, \tag{3.14}$$

we obtain the following system, the analog of (3.10):

$$\begin{aligned}
\sum_{j\in I} \mathbf{L}_{ij}\lambda_j + \mathbf{D}_i\mu_i &= 0 \text{ if } i \in I_e,\\
\sum_{j\in I} \mathbf{L}_{ij}\lambda_j &= 0 \text{ if } i \notin I_e,\\
\sum_{j\in I} \mathbf{L}_{ij}\mathbf{v}_j - \mathbf{D}_i\lambda_i &= 0,\\
\sum_{j\in I} \mathbf{L}_{ij}\mathbf{u}_j - \mathbf{D}_i\mathbf{v}_i &= 0,\\
\sum_{j\in I} \mathbf{u}_i - \mathbf{u}_i^f &= 0 \text{ if } i \in I_e.
\end{aligned} \tag{3.15}$$

An important distinction is that the Lagrange multiplier λ can no longer be eliminated, due to a more complex form of the equation obtained by differentiating with respect to $\mathbf{v}_i$. From direct examination of the system one observes that *the Lagrange multiplier λ corresponds to* $\mathbf{w}$ *in the low-order factorization of the triharmonic equation of the form* $\Delta\mathbf{u} = \mathbf{v}$, $\Delta\mathbf{v} = \mathbf{w}$, $\Delta\mathbf{w} = 0$.

Because of the different form of the first equation, we no longer can eliminate all variables corresponding to vertices outside the one-row neighborhood of $\bar{\Omega}$, and two rows are needed, as expected from the discrete-geometric discretizations of triharmonic equations. We obtain the following system in the end:

$$\begin{bmatrix} \mathbf{L}_{\bar{\Omega}\bar{\Omega}} & 0 & -\mathbf{M}^d_{\bar{\Omega}\bar{\Omega}} \\ 0 & 0 & \mathbf{L}_{\Omega\bar{\Omega}} \\ -\mathbf{M}^d_{\bar{\Omega}\bar{\Omega}} & \mathbf{L}_{\bar{\Omega}\Omega} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\bar{\Omega}} \\ \mathbf{u}_{\Omega} \\ \lambda_{\bar{\Omega}} \end{bmatrix} = \begin{bmatrix} -\mathbf{L}_{\bar{\Omega},1}\mathbf{v}_1 \\ 0 \\ -\mathbf{L}_{\bar{\Omega},0}\mathbf{u}_0^f - \mathbf{L}_{\bar{\Omega},1}\mathbf{u}_1^f, \end{bmatrix} \tag{3.16}$$

where

$$\mathbf{v}_1 = -\mathbf{L}_{1,0}\mathbf{u}_0^f - \mathbf{L}_{1,1}\mathbf{u}_1^f - \mathbf{L}_{1,2}\mathbf{u}_2^f. \tag{3.17}$$

Again, as it was observed for the biharmonic equation, one can also eliminate $\mathbf{v}$ and λ entirely by inverting the mass matrices, and obtaining a system in terms of $\mathbf{u}$ alone, which coincides with the system of [Botsch and Kobbelt 2004].

Curve boundary conditions. For curve boundary conditions, the discretization can be obtained in a similar way, however an important difference arises. In this case, the values $\mathbf{v}_i$ are fixed for boundary vertices, unlike in the case of the region boundary condition system (3.16), where these remain as free variables. This results in the system matrix

$$\begin{bmatrix} \mathbf{L}_{\Omega\Omega} & 0 & -\mathbf{M}^{\Omega,d}_{\Omega\bar{\Omega}} \\ 0 & 0 & \mathbf{L}_{\Omega\bar{\Omega}} \\ -\mathbf{M}^{\Omega,d}_{\bar{\Omega}\Omega} & \mathbf{L}_{\bar{\Omega}\Omega} & 0 \end{bmatrix}. \tag{3.18}$$

Somewhat surprisingly, although the discretization follows the same pattern as well-established discretizations for biharmonic equations, the resulting systems are often overdetermined and cannot be practically used. The following proposition describes simple local mesh configurations leading to singularity, but one can observe more complex global dependencies, resulting from the fact that there are "too many" degrees of freedom fixed on the boundary, while the equations corresponding to boundary vertices are retained.

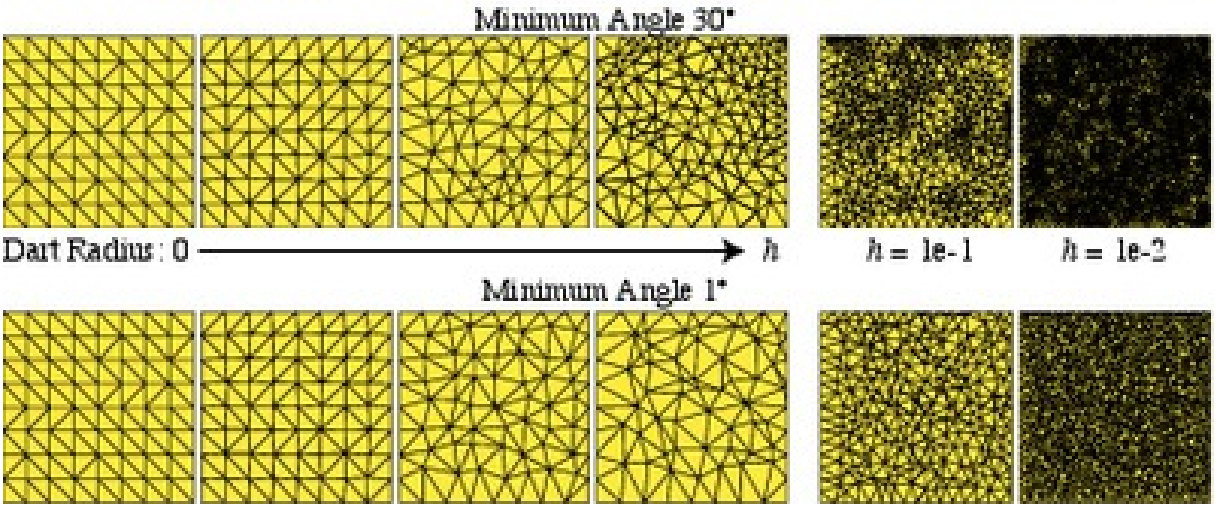
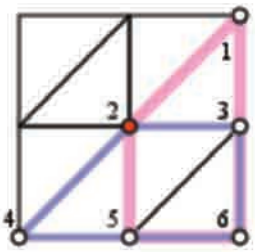


Figure 3.8: *We generate irregular meshes by perturbing vertices of a regular grid by an amount less than a specified "dart radius" (increasing from 0 to average edge length h, four columns far left to right). We consider such meshes as resolutions increase (e.g. two right most columns). We also run Delaunay triangulation with Steiner points to ensure minimal angle constraints (compare top and bottom rows).*

Proposition 2. *The system* (3.18) *is singular if two vertices in I_1 have exactly one edge-adjacent vertex in I_Ω.*

Indeed, observe that the system with lumped mass contains the equations of the form $\mathbf{L}_{52}\mathbf{u}_2 = \mathbf{D}_5\mathbf{v}_5 - \mathbf{L}_{54}\mathbf{u}_4 - \mathbf{L}_{55}\mathbf{u}_5 - \mathbf{L}_{56}\mathbf{u}_6 - \mathbf{L}_{53}\mathbf{u}_3$, and $\mathbf{L}_{52}\mathbf{u}_2 = \mathbf{D}_3\mathbf{v}_3 - \mathbf{L}_{55}\mathbf{u}_5 - \mathbf{L}_{56}\mathbf{u}_6 - \mathbf{L}_{53}\mathbf{u}_3 + \mathbf{L}_{51}\mathbf{u}_1$. Each equation has exactly one variable $\mathbf{u}_2$; the remaining components of $\mathbf{u}$ and $\mathbf{v}$ are on the boundary and are fixed. In general, these equations are not compatible. Similar observation holds for the full mass matrix, but a slightly larger neighborhood needs to be considered.



Reduction of fixed curve boundary conditions. An alternative approach to discretizing curve boundary conditions is to reduce them to the discretization of the type (3.16). This is achieved by inferring the values of $\mathbf{u}_i$ for $i \in I_{-1}$ where I_{-1} are the vertices in the interior of Ω, edge-adjacent to the boundary, from the boundary values of $\partial\mathbf{u}/\partial n = \mathbf{b}_1$.

For each triangle in Ω with 2 vertices on the boundary, the value of the interior vertex can be easily determined independently, as the $\partial\mathbf{u}/\partial n$ and the boundary values completely specify the gradient of $\mathbf{u}$. If several such triangles have a common vertex v in the interior (the situation that leads to overdetermined systems), we simply average all values obtained for v from these triangles, since the difference in these values should decrease at least as $O(h^2)$ as the mesh is refined. Algebraically, this procedure eliminates a part of the overdetermined system by exploiting the fact that the values of $\mathbf{u}$ on I_{-1} can be solved for in least-squares sense, independently of the rest of the variables.

As a result, we obtain a discrete system of equations with two rows of values of $\mathbf{u}$ fixed, and values of $\mathbf{v}$ on the boundary. The rest of the derivation proceeds as before, by substituting

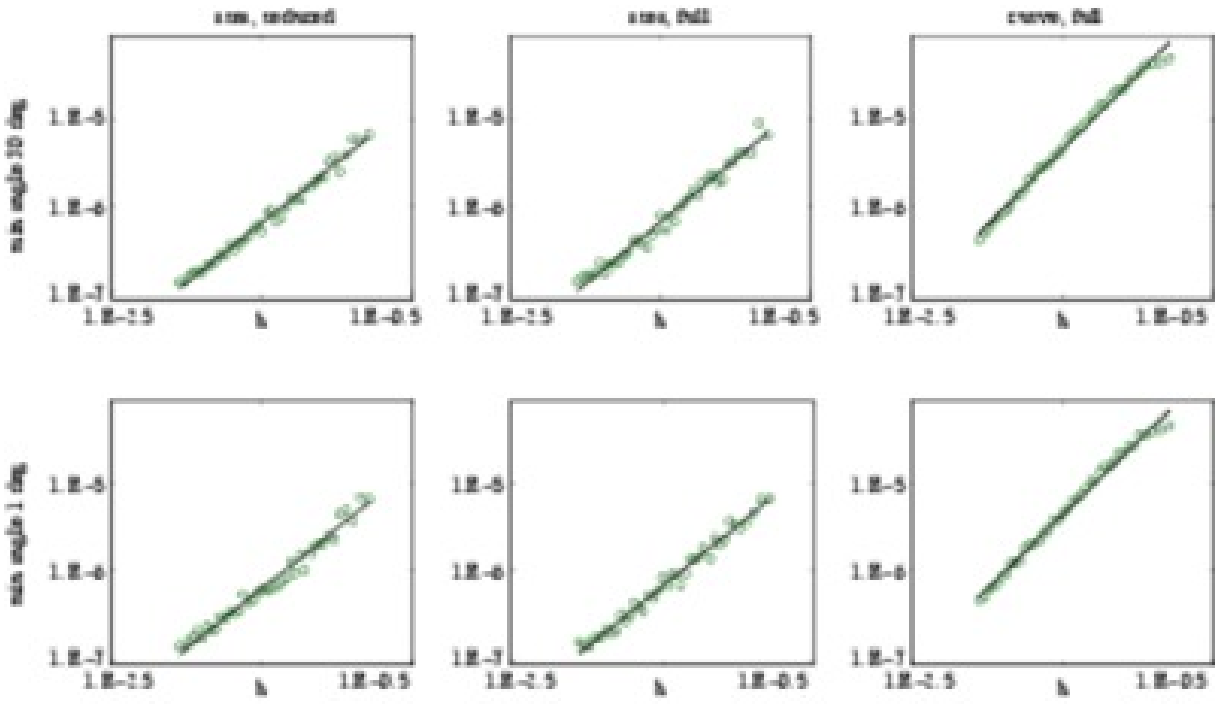


Figure 3.9: *Dependence of the error on average edge length for the Laplacian energy, randomized meshes.*

piecewise linear expressions for $\mathbf{u}$ (with two rows fixed), $\mathbf{v}$ (with one row fixed) and λ (with no rows fixed). The system matrix is identical to (3.16), if one row on the boundary is removed from Ω, and renamed I_0, and I_0 is renamed I_1. The right-hand side has the same form, but instead of $\mathbf{v}_1$ given by (3.17), the boundary value of $\mathbf{v}_1$ appears directly.

Convergence. While many versions of theoretical analysis are available for the biharmonic and other fourth-order problems, much less work was done on higher-order equations. Unfortunately, the triharmonic problem suffers from the same diffuculty as biharmonic: general theorems based on the LBB (*inf-sup*) condition do not apply for similar reasons. Mixed elements for polyharmonic problems are considered in [Bramble and Falk 1985]; however, error estimates are obtained under the assumption that the discretization in the domain Ω is asymptotically finer than the discretization on the boundary.

3.5 Evaluation and applications

Implementation. In our implementations, we solve the system using either UMFPACK's sparse LU factorization [Davis 2004] or MATLAB's LDL decomposition [Duff 2004]. Though our systems are symmetric, they are in general not positive definite. Therefore, we may use a Cholesky factorization only when the system is reduced to a single variable. We observe that the timings for solving the system with auxiliary variables do not differ much from timings for solving the reduced system: while the number of variables is smaller, the system is much less sparse. As only the right-hand side depends on the boundary conditions, the system can be

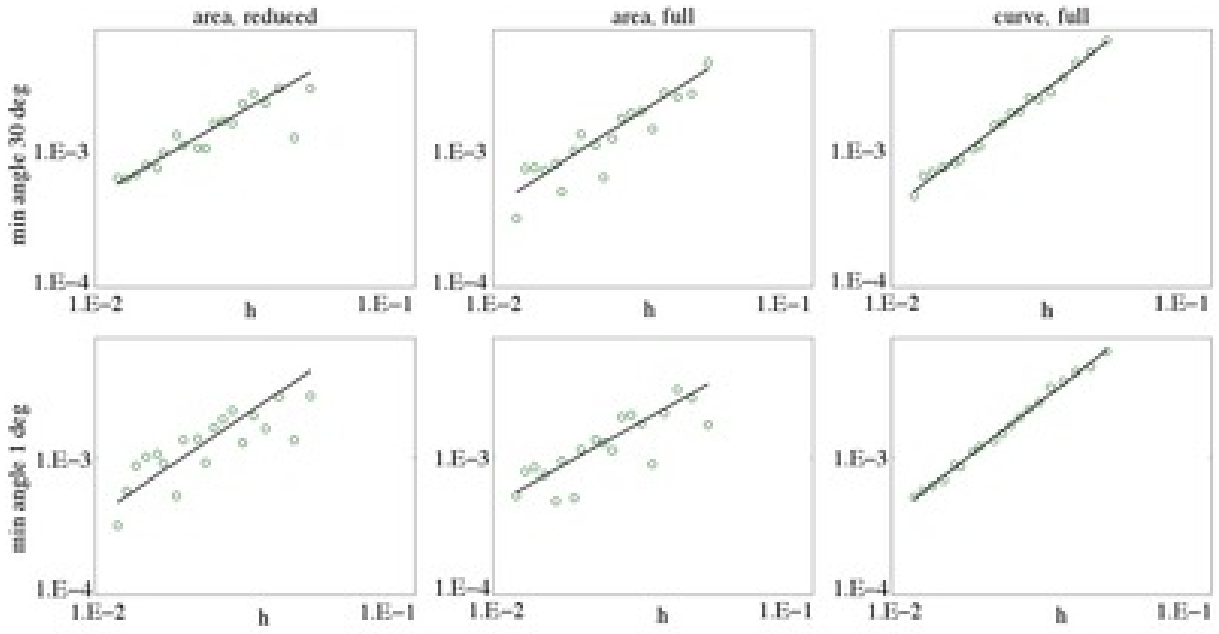


Figure 3.10: *Dependence of the error on average edge length for the Laplacian gradient energy, randomized meshes.*

prefactored when these conditions are manipulated.

Convergence and mesh dependence. We start with a study of the errors of the region and curve boundary conditions. We characterize meshes by the average edge length h. We have used several sequences of meshes from [Grinspun et al. 2006] (see Figure 3.11). We also used several sets of randomized meshes, obtained by perturbing the positions of points on a regular grid and running Delaunay triangulation on the resulting vertices, with Steiner points added to satisfy a constraint on the minimal angle (see Figure 3.8). The errors are calculated by comparing our solution with an analytic solution. We choose an arbitrary function u^t as the target analytic solution, sample boundary conditions from this function and use the right-hand side $g = \Delta^k u^t$ for $k = 2$ and $k = 3$. We have tried a number of test functions with similar results.

Figures 3.9 and 3.10 show how the L^2 error changes with average edge length for randomized meshes with minimal angle of 30 and 1 degrees, and Figure 3.12 shows the same dependence for the meshes from Figure 3.11. The plots read right to left as h decreases. We observe that the convergence rate for the Laplacian energy (biharmonic equation) is consistent with the error estimate of [Scholz 1978]. There is no significant difference between the behaviors of region and curve boundary condition formulations, and there is good consistency between different connectivities with the same average edge length. At the same time, more regular mesh patterns result in greater dependence on connectivity (Figure 3.12, top). There is no significant difference in numerical solutions obtained from the reduced formulation (3.12) and the equivalent two-variable formulations (3.11) and (3.13).

The behavior for the Laplacian gradient is quite different (Figs. 3.10 and 3.12, bottom). The observed average convergence rate is substantially slower for region conditions, and the devia-

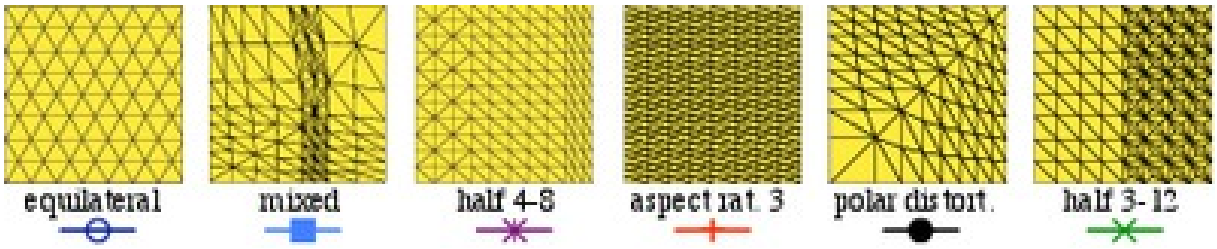


Figure 3.11: *Mesh connectivities used in tests, from [Grinspun et al. 2006].*

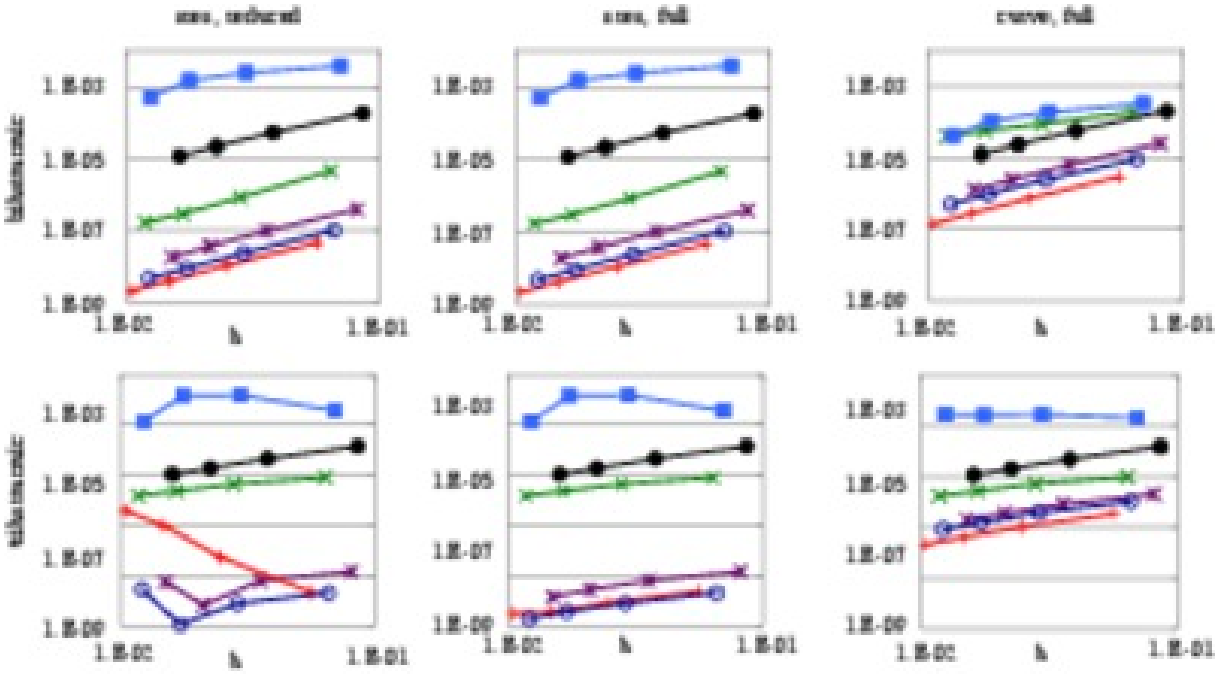


Figure 3.12: *Dependence of the error on average edge length for the Laplacian energy (left), and Laplacian gradient energy (right); test mesh connectivities shown in Figure 3.11.*

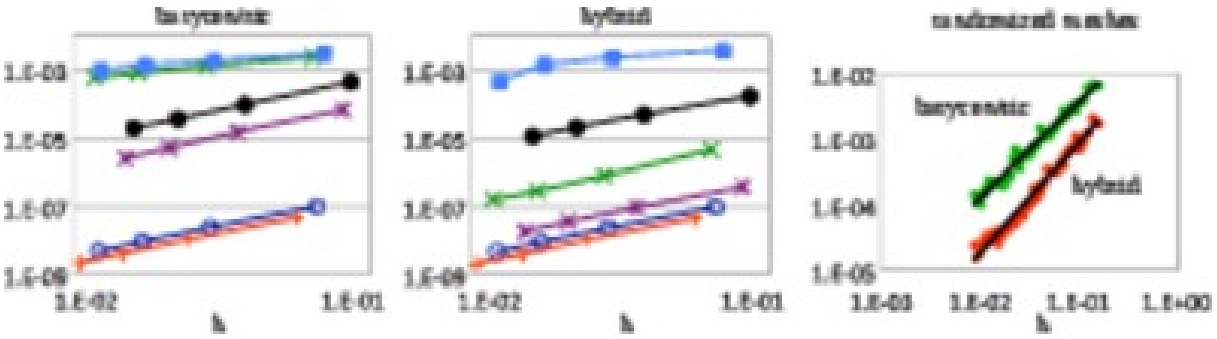


Figure 3.13: *Comparison of barycentric and hybrid [Meyer et al. 2002] lumped mass matrices.*

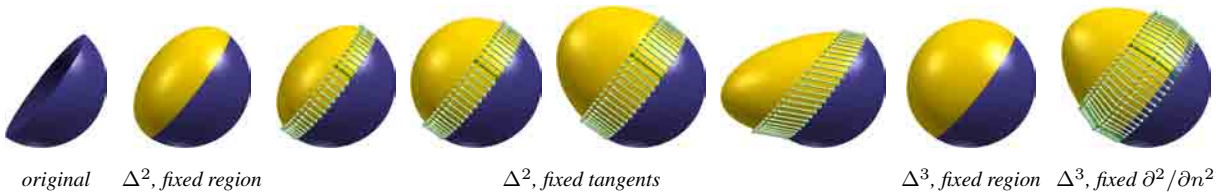


Figure 3.14: *Hole filling: the half-sphere mesh was completed to close the "hole". The left images show biharmonic reconstruction using region constraints and curve constraints with different prescribed tangents. The two rightmost images show the triharmonic reconstruction using region constraints and curve constraints with user-prescribed curvatures.*

tion from the average error for a given mesh resolution is higher. The most significant observed effect is that by using curve conditions with explicitly defined second derivative, we obtain substantially better results, both in terms of error magnitude and less mesh dependence. Reduced single-variable systems of the type (3.12) produce stronger mesh dependence and in some cases inferior convergence behavior compared to the full mixed element system.

Mass matrix lumping. While the convergence rate and mesh dependence are not strongly affected by the choice of the mass matrix lumping strategy, there is a substantial difference in error magnitudes, which is consistent with the observations in the literature. Figure 3.13 shows a comparison of "barycentric" lumping, with each diagonal entry obtained as $1/3$ of the sum of areas of triangles incident at the vertex, and the "hybrid" approach, using Voronoi areas for non-obtuse triangles [Meyer et al. 2002] (see Section 2.1.3 for details).

Applications. Figures 3.2, 3.5, 3.3, and 3.4 show examples of interactive editing of biharmonic and triharmonic surfaces, with different user-defined boundary conditions. The user is free to manipulate single points, curves or regions in the edited shape. Tangents can be prescribed along curve boundaries for biharmonic surfaces; triharmonic surfaces additionally admit second derivative control along the direction perpendicular to the boundary. Tangents are explicitly specified by the user at a sparse set of points along the boundary curve using a simple vector widget, and the tangents at the remaining points are computed by interpolation (see Figure 3.6). Second derivatives are set using the Bézier widget (see Figures 3.4 & 3.18): the control triangle of a quadratic Bézier curve whose derivatives are used in the boundary conditions for $\partial/\partial n$ and $\partial^2/\partial n^2$.

Smooth detail-preserving deformations (Figure 3.18) are achieved by solving the bi- or triharmonic equation for the *displacement* function $\mathbf{u}$. The user can interactively manipulate the boundary conditions for the displacement field, namely positions, first and second derivatives. A more complex approach involving rotations [Botsch and Sorkine 2008] can also be formulated in the mixed-element framework.

Another application of PDE-based surfaces is smooth hole filling and blending between shapes. In hole filling, a surface with a boundary loop (a hole) is given, whereas blending implies two or more loops that need to be connected by a surface. One can use the region constraints to ensure C^1 or C^2 continuity with the rest of the surface, or curve boundary constraints to specify tangents and/or curvatures at the hole-border directly (Figure 3.14). Blending between

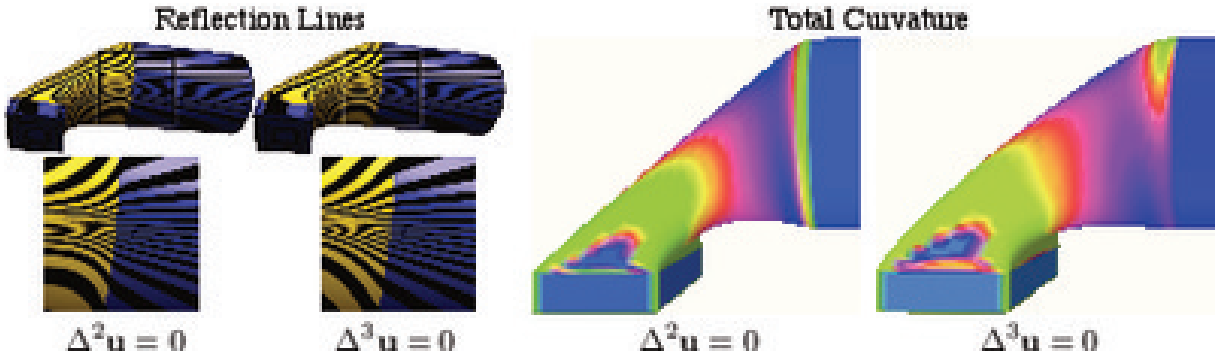


Figure 3.15: *Blending between surfaces (cylinders with a square and circular cross-sections) using bi- and triharmonic equations and region boundary conditions. Note the smoother behavior of reflection lines in the triharmonic case.*

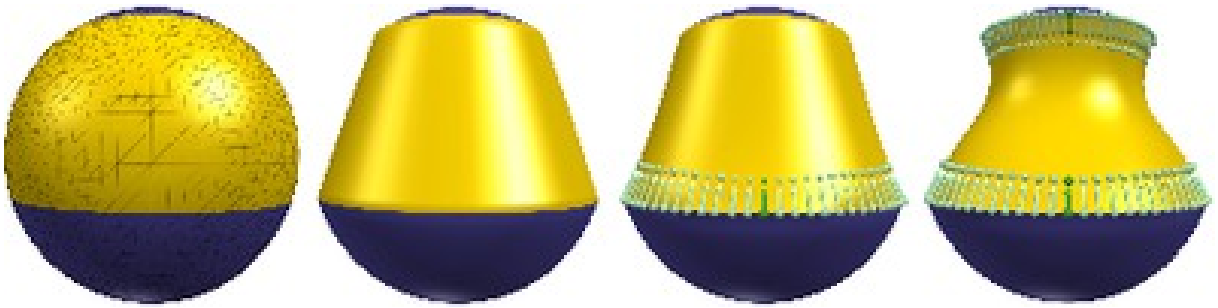


Figure 3.16: *Blending between two spherical caps, with controllable sharp features introduced using tangent conditions.*

two shapes using region boundary conditions is shown in Figure 3.15; the higher smoothness of the triharmonic surface is evident from the reflection lines. Figure 3.17 shows two curves interpolated with a variational patch, with the shape of the patch controlled by tangents and second derivatives specified along the curves (we map both curves to two opposite boundaries of a rectangular area in the plane, and periodic conditions are imposed at the other two boundaries).

3.6 Conclusions

The technique that we have presented allows one to discretize a broad variety of functionals and PDEs with different types of boundary conditions using only piecewise linear elements. The main components of the approach include (1) factorization of the original equations into low-order equations by introducing additional variables; (2) using constraints to impose region conditions; (3) lumping mass matrices to eliminate unneeded variables.

While experimental evidence shows that the method is converging for both forth- and sixth-order problems, convergence is significantly slower for the latter, and mesh dependence is stronger. We observe that using high-order interpolation to estimate the second-order boundary condition somewhat improves the situation.

One potential direction for improvement is, instead of factoring this system into three second-

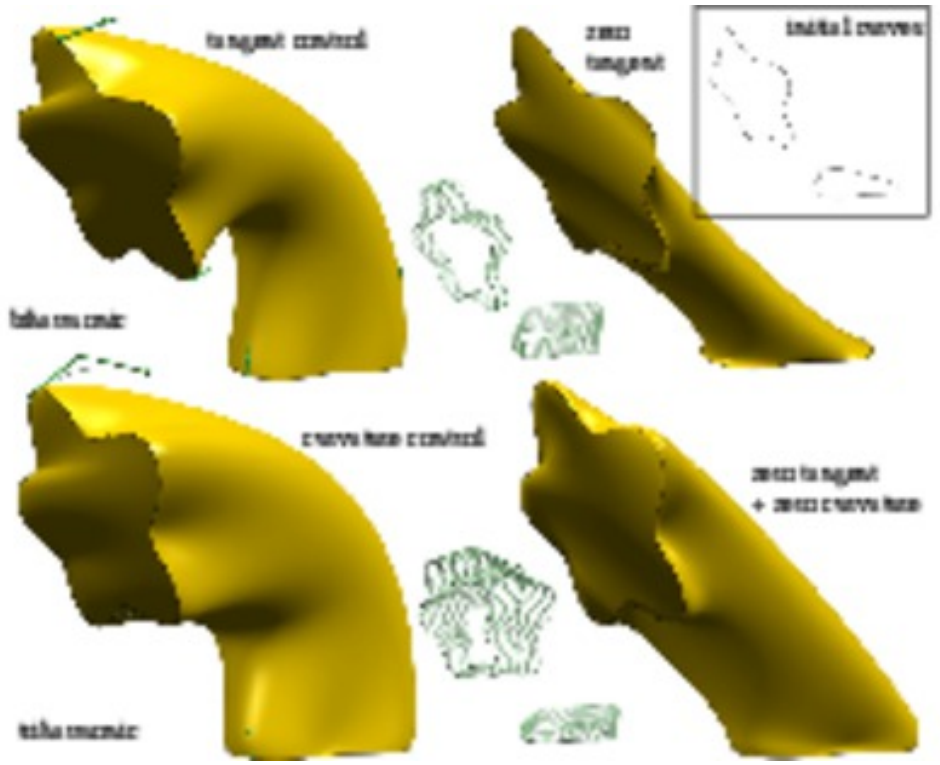


Figure 3.17: *Filling in a patch between surfaces. Tangents or tangents plus second derivatives can be specified at curves to obtain the desired shape.*

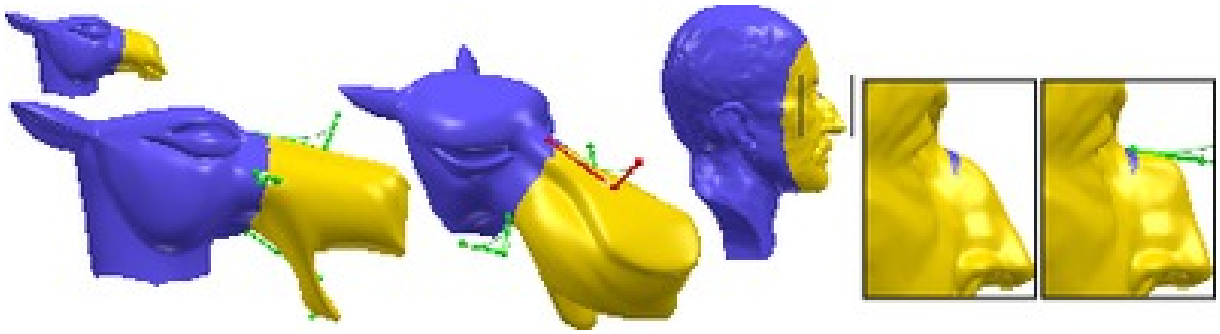


Figure 3.18: *Controlling deformations using tangents and second derivatives. We solve the triharmonic equation* $\Delta^3\mathbf{u} = 0$ *for the* displacement *function (the edited surface is then* $\mathbf{x} + \mathbf{u}$*). The* Camel*'s nose is lengthened and the mouth opened by manipulating both the first and the second normal derivatives of the displacement field; the curvature of the* Max Planck*'s nose is altered by interacting with the second derivatives via the Bézier widget.*

order systems, to use a fourth- and a second-order system, with quadratic elements for the former (e.g. the ones used in [Grinspun et al. 2006]).

In the simpler case (biharmonic equation) existing theory can be used to establish convergence guarantees; much less is known for sixth-order systems, and the experimentally observed convergence rates are significantly lower. Another important direction for exploration is the effect of the non-flat metric. The discretizations we describe should still apply with no significant changes.

3.7 Appendix: Ciarlet-Raviart discretization and region boundary conditions

We outline the connection between solutions to the systems (3.13) and (3.11) here; a full rigorous treatment would require detailing assumptions on the smoothness spaces for boundary data and is beyond the scope of this thesis. To simplify consideration, we assume that the solutions are classical solutions, i.e., $\mathbf{u}$ is four times differentiable in Ω. In this case, solutions of continuous problems with curve boundary conditions and region boundary conditions are identical, as long as the boundary conditions for the curve problem are sampled from $\mathbf{u}^f$ for the region problem.

We consider a simplified situation with homogeneous Dirichlet conditions, i.e., we assume that for (3.11), $\mathbf{u}^f = 0$, and for (3.13), $\mathbf{b}_0 = 0$ (this is a standard reduction for Dirichlet conditions, using substitution $\mathbf{u} = \mathbf{u}^{orig} - \mathbf{u}^D$ where $\mathbf{u}^D$ satisfies the Dirichlet condition [Braess 2002]). This reduction requires introducing a right-hand side for the second equation in the system: $\Delta\mathbf{u} = \mathbf{v}$, $\Delta\mathbf{v} = \Delta^2\mathbf{u}^D = \mathbf{g}$.

The Ciarlet-Raviart system (3.13) with lumped mass matrix with solution $(\mathbf{v}^*_{\bar{\Omega}}, \mathbf{u}^*_{\Omega})$ can be rewritten in the form

$$\begin{bmatrix} -\mathbf{M}^d & \mathbf{L}_{\bar{\Omega},\Omega} \\ \mathbf{L}_{\Omega,\bar{\Omega}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\bar{\Omega}} \\ \mathbf{u}_{\Omega} \end{bmatrix} = \begin{bmatrix} -\mathbf{N}^{\partial\Omega}_{\bar{\Omega},0}\mathbf{b}_1 - \mathbf{M}^{d,\Omega^e}_{\bar{\Omega},0}\mathbf{v}^*_0 \\ \mathbf{g}^{\Omega} \end{bmatrix}, \tag{3.19}$$

where we have subtracted $\mathbf{M}^{d,\Omega^e}_{\bar{\Omega},0}\mathbf{v}^*_0$ from both sides, to obtain the same left-hand side as in (3.11), and $\mathbf{g}^{\Omega}_j = \langle \mathbf{g}, \varphi_j \rangle_{\Omega}$. In comparison, the right-hand side of (3.11) is $[0, \mathbf{g}^{\Omega_0}]$, with $\mathbf{g}^{\Omega_0}_j = \langle \mathbf{g}, \varphi_j \rangle_{\Omega_0}$. As shown in [Scholz 1978], $(\mathbf{v}^*)^h$ converges to $\mathbf{v}$ in L^2-norm, and $\mathbf{v}$ is at least continuous on $\bar{\Omega}$. It can be extended by zero to all of Ω_0, consistently with $\Delta\mathbf{u}^f$. On the other hand, the j-th component of $-\mathbf{N}^{\partial\Omega}_{\bar{\Omega},0}\mathbf{b}_1$, $-\langle \mathbf{b}_1, \varphi_j \rangle_{\partial\Omega} = \langle \nabla u^f, \nabla\varphi_j \rangle_{\Omega_e} = -\langle \mathbf{b}, \varphi_j \rangle_{\partial\Omega} = \langle \mathbf{v}, \varphi_j \rangle_{\Omega_e}$. Combining L^2 convergence of solutions $(\mathbf{v}^*)^h$ of (3.11) to $\mathbf{v}$, and the fact that components of $\mathbf{M}^{d,\Omega^e}_{\bar{\Omega},0}\mathbf{v}^*_0$ are quadrature approximations of $\langle \mathbf{v}^*, \varphi_j \rangle_{\Omega_e}$, we observe that the r.h.s. of (3.19) converge in L^2 norm to the the r.h.s. of the system obtained for the (3.13) formulation in L^2 norm, so the difference in the solutions also converges as shown in [Brezzi and Fortin 1991].

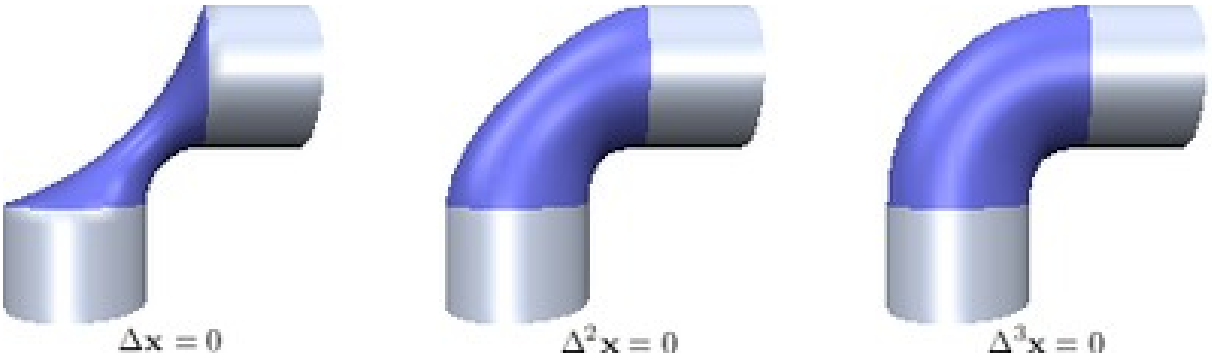


Figure 3.19: *The famous* Three Pipes. *The original pipes from [Botsch and Kobbelt 2004] suggest a favorable parameterization, perhaps the quarter torus itself.*

Low resolution images used under fair use for criticism and commentary.

3.8 Appendix: Reproducing the *Three Pipes*

Fair surface design, curve completion and hole filling are instances of *boundary constraint modeling*. As we have just read in this chapter, one paradigm to solve these problems is to compute parametric surfaces which minimize an intrinsic energy approximating the aesthetic notion of fairness. Botsch and his colleagues pioneered this field in the early 2000s, focusing on three relevant energies. Minimizers of the Dirichlet energy are harmonic functions and corresponding minimal surfaces are membrane, soap-film-like. Minimizers of the Laplacian energy (or linearized thin-plate energy) are curvature minimizing, biharmonic functions. Finally, minimizers of the Laplacian-gradient energy (or linearized curvature-variation energy) are triharmonic functions. Botsch et al. have often illustrated the difference between these functions with a figure of such surfaces connecting two equal-radius, cylindrical pipes positioned at right angles [Botsch and Kobbelt 2004, Botsch 2005, Botsch et al. 2006b, Botsch et al. 2007a, Botsch et al. 2008, Botsch et al. 2010, Botsch 2011]. Figure 3.19 shows the sequence of increasing smoothness where the interior meets the boundary conditions: C^0, C^1, then C^2. The triharmonic solution appears to approximate a quarter torus.

Reproducing these images proves difficult with no extra knowledge. We must carefully consider the choices of surface parameterization and boundary conditions to the k-harmonic equations responsible for these solutions. In this section, we will explore the space these choices span in the context of this example as a case study. This is not just an exercise in scientific reproduction, but also an exploration and examination of the care necessary when crafting surfaces from such functions.

As noted in [Botsch 2005, Botsch and Sorkine 2008], the parameterization of the Δ^k operators has a large impact on the quality of the linearized solutions. Despite this strong dependence, to the best of our knowledge in all reproductions of the *Three Pipes* image the respective parameterization has not been mentioned. Though most instances are accompanying linearized solutions, implying that the pipes are a result of solving a single set of linear equations, a few uses appear in the context of nonlinear solutions [Botsch et al. 2007a, Botsch et al. 2008][2]. Here

[2] Although this is likely not the case at least in the minimal solution to $\Delta \mathbf{x} = 0$, which (depending on the exact boundary conditions) would be a discontinuous, piecewise-planar surface as in Figure 3.20 (second left) and as described in [Pinkall and Polthier 1993]

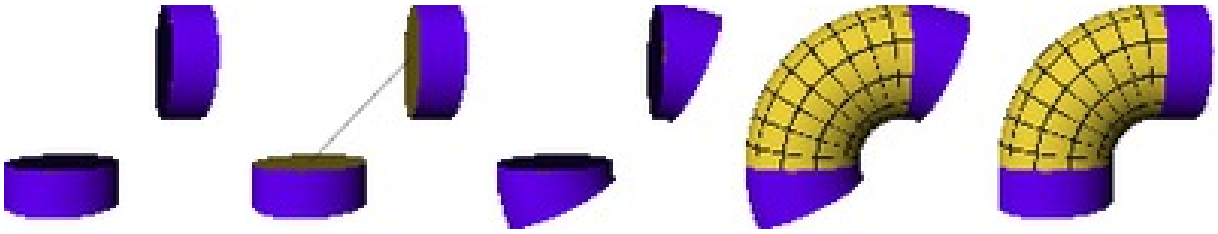


Figure 3.20: *Problem input, incorrect topology solution, possible alternative boundary conditions, gold-standard torus solution, glued to original input.*

we will limit our discussion to linear solutions, recognizing that if the parameterization were defined upon the *unknown* solution then the linear and nonlinear solutions would coincide.

Let us first back up and define the problem we are trying to solve as it might appear in the practical application of surface completion. As input we have two disjoint regions of a cylindrical pipe. The topology of the input and the solution is assumed known: let the inputs be "capped" so that they are disk topology. Then the solution should join the input into single sphere-topology component. This immediately restricts the parameterizations we will use to describe this unknown surface to those that are annulus topology (homeomorphic to a finite, "uncapped" cylinder). We expect the solutions to be continuous so we are not interested in solutions like the second-to-left column in Figure 3.20, whose discontinuity effectively allows a non-sphere final topology.

3.8.1 Parametric domain

The surfaces in question are solutions to:

$$\Delta \mathbf{x} = 0, \tag{3.20}$$
$$\Delta^2 \mathbf{x} = 0, \tag{3.21}$$
$$\Delta^3 \mathbf{x} = 0, \tag{3.22}$$

which correspond to minimizers of the Dirichlet, Laplacian, Laplacian gradient energies:

$$E_D(\mathbf{x}) = \int_\Omega \|\nabla \mathbf{x}\|^2 d\mathbf{x}, \tag{3.23}$$
$$E_L(\mathbf{x}) = \int_\Omega \|\Delta \mathbf{x}\|^2 d\mathbf{x}, \tag{3.24}$$
$$E_{LG}(\mathbf{x}) = \int_\Omega \|\nabla \Delta \mathbf{x}\|^2 d\mathbf{x}, \tag{3.25}$$

respectively. We will assume that the boundary conditions are imposed as *region conditions* in a mixed finite elements discretization (see Section 3.3.2). The precise choice of the boundary values will be discussed later.

The gradient and Laplace operators here are defined with respect to the underlying parameterization. A perfectly isometric parameterization (e.g. the unknown surface itself) would be ideal for some applications as it implies minimal surfaces, minimal curvature surfaces and minimal

curvature-variation surfaces respectively. This is often not feasible in practice and we rely on a known parameterization instead. Unless otherwise noted, these minimal notions and continuity are always with respect to the chosen parameterization and its imposed metric.

We consider three reasonable choices for this parameterization. The simplest possible choice is to lay out our parameterization on the unit square, with periodic tiling vertically (so we have a topological annulus). This choice seems most practical in scenarios where no knowledge of the unknown surface is known in advance beyond the topology or when the boundary conditions are too complicated to imply any meaningful information. Such a parameterization was used in Figure 3.17. Next we consider using a straight cylinder with radius equal to those of the boundary loops and height equal to the length of a circular arc between the boundary regions[3] This parameterization seems reasonable in the surface completion application when some prior knowledge is available (e.g. connecting equal radius pipes). Finally we consider parameterizing using the quarter torus itself, which exactly matches the values and normals of the boundary regions. The practicality of this parameterization seems unlikely as the quarter torus could be subjectively seen as the gold-standard solution to the original problem. However this parameterization is useful for analyzing this functions in terms of their *reconstruction* capabilities [Tosun 2008].

We compare these different parameterizations in the Figure 3.21. In this case we use region boundary conditions sampled from the straight cylinders at a right angle (top three rows). All solutions suffer shrinkage between the pipes and the outer silhouette tends to straightens out. The cylinder and torus parameterizations are subjectively *better* as they appear to have less unnecessary oscillation.

Other considerations of the parametric domain such as mesh density and quality are ignored here, but examined closely in [Tosun 2008] (e.g. see her Figure 4.14).

3.8.2 Boundary conditions

As described in [Botsch and Kobbelt 2004] and also subsequently in the context of our mixed FEM discretization, region conditions are a simply way to impose boundary conditions for higher-order PDEs. For k-harmonic functions we simply provide function values for k rings of boundary vertices. If the function values are sampled from some existing surface then this ensures C^{k-1} continuity of the coordinate functions of the solution, and thus the solution surface as a whole. However, we may still use region conditions to impose *other* functions values. For Figure 3.17, we used a system to use interactive splines to control derivatives along boundary curves. Instead, we could simply sample k-rings of values from any arbitrary function or surface. Relevant to this example, we could sample values of the region handles from a *different* surface, one other than the straight cylinders at a right angle. In particular, we could sample boundary values as if the regions lay on a torus (see Figure 3.20 middle).

We compare using the straight cylinders as boundary conditions and using the torus as boundary conditions (compare the top and bottom three rows in Figure 3.21). Here we show both the cylinder and torus parameterization as described above. Notice that the straight cylinders

[3] The cylinder is developable, thus this parameterization is equivalent to using correspondingly stretched and tiled rectangle on the plane.

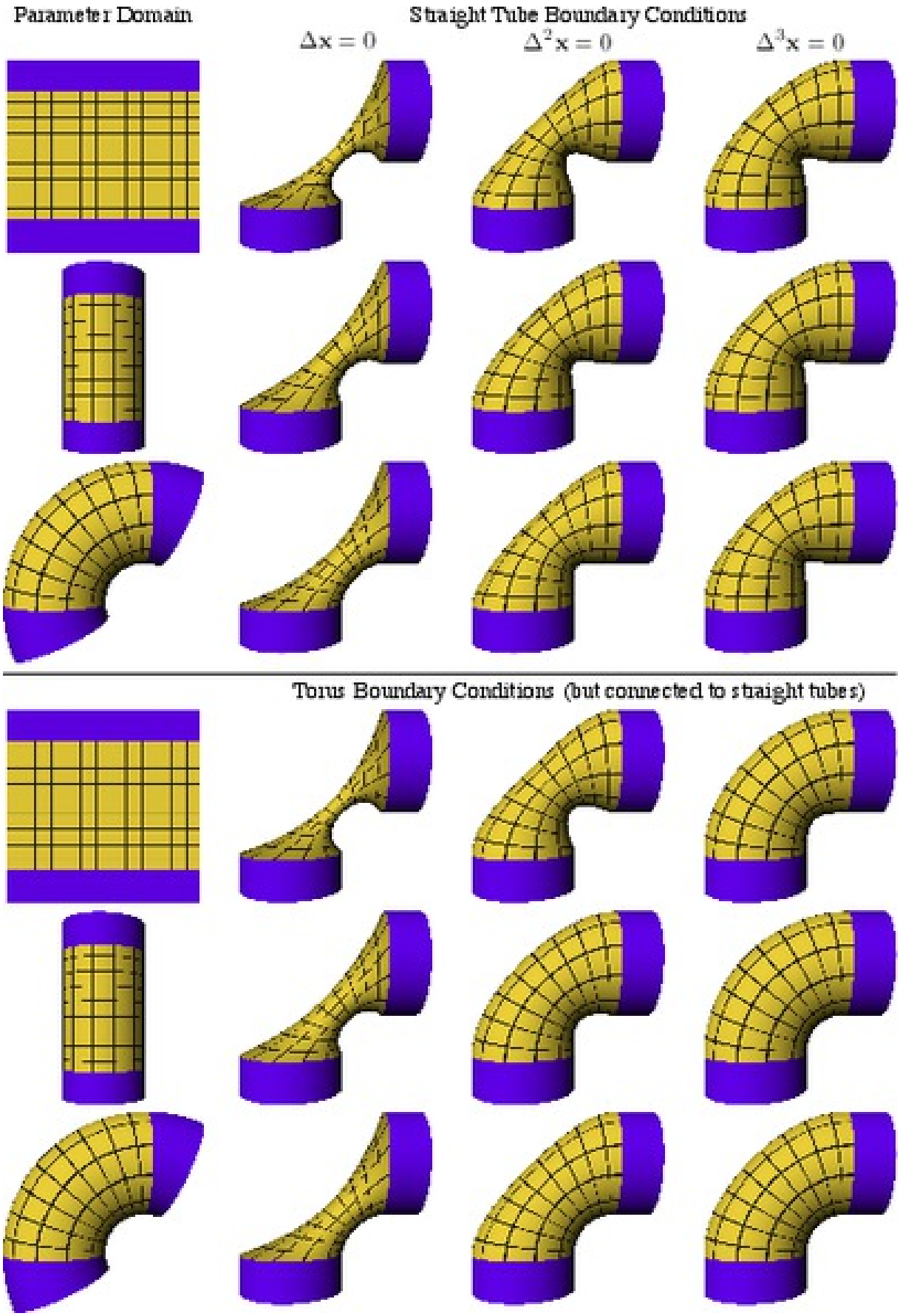


Figure 3.21: *Difference choices of domain parameterization and boundary conditions span a wide space of solutions.*

imply Neumann condition *vectors* aligned with the cylinder axes and scaled uniformly around the boundary curves. For the sampled torus boundary conditions we visualize the result showing the straight cylinder regions in place. Though a bit confusing, we do this on purpose to highlight that such an operation is possible: different surfaces may be used to impose boundary conditions than the input surface which is to be *completed*. It is also useful to analyze the continuity and impact of the implied Neumann conditions in this case. We have chosen this torus so that Dirichlet values agree on the immediate boundary of the unknown surface and regions (this ensures C^0 continuity). While the effective first derivative Neumann conditions of the torus agree in direction with that of the straight cylinders, the scales vary around the boundary curves. This ensures that the surface is G^1, meaning *geometrically* continuous rather than *parametrically* continuous. This looser condition implies that surface is C^1 under *some* parameterization or intuitively that first derivatives match up to scale [DeRose 1985]. In contrast, the effective higher-order Neumann conditions for the triharmonic surface vary around the curve, not just in scale, but also direction. Further manipulations of boundary conditions are considered [Tosun 2008] (see her Figures 4.16 & 4.17).

Combining these choices of boundary conditions with the *best* parameterizations above it is not surprising to find that the *best* choice of boundary conditions is to sample the torus rather than the straight tubes. Combining the torus parameterization and the torus boundary conditions also seem to most closely match the images of Botsch et al.[4] This seems to imply that continuity (the tirharmonic solutions are only G^1, rather than C^2) is perhaps less important than other higher-level notions of fairness such as *roundness* and volume preservation. These appear to be incorporated in the parameterization and boundary conditions choices, though perhaps not intuitively controllable outside simple examples such as this.

[4]Private correspondence with Mario Botsch reveals that the true parameterization in use was closest to the far right image in Figure 3.20: a quarter torus in the interior and straight cylinders on either end, a surface which is merely G^1 to begin with.

4

Bounded biharmonic weights for real-time deformation

Object deformation with linear blending dominates practical use as the fastest approach for transforming raster images, vector graphics, geometric models and animated characters. Unfortunately, linear blending schemes for skeletons or cages are not always easy to use because they may require manual weight painting or modeling closed polyhedral envelopes around objects. Our goal is to make the design and control of deformations simpler by allowing the user to work freely with the most convenient combination of handle types. We develop linear blending weights that produce smooth and intuitive deformations for points, bones and cages of arbitrary topology. Our weights, called bounded biharmonic weights, minimize the Laplacian energy subject to bound constraints. Doing so spreads the influences of the controls in a shape-aware and localized manner, even for objects with complex and concave boundaries. The variational weight optimization also makes it possible to customize the weights so that they preserve the shape of specified essential object features. We demonstrate successful use of our blending weights for real-time deformation of 2D and 3D shapes.

4.1 Introduction

Interactive space deformation is a powerful approach for editing raster images, vector graphics, geometric models and animated characters. This breadth of possibilities has led to an abundance of methods seeking to improve interactive deformation with real-time computation and intuitive use. Real-time performance is critical for both interactive design, where tasks require exploration, and interactive animation, where deformations need to be computed repeatedly, often sixty or more times per second. Among all deformation methods, linear blending and its

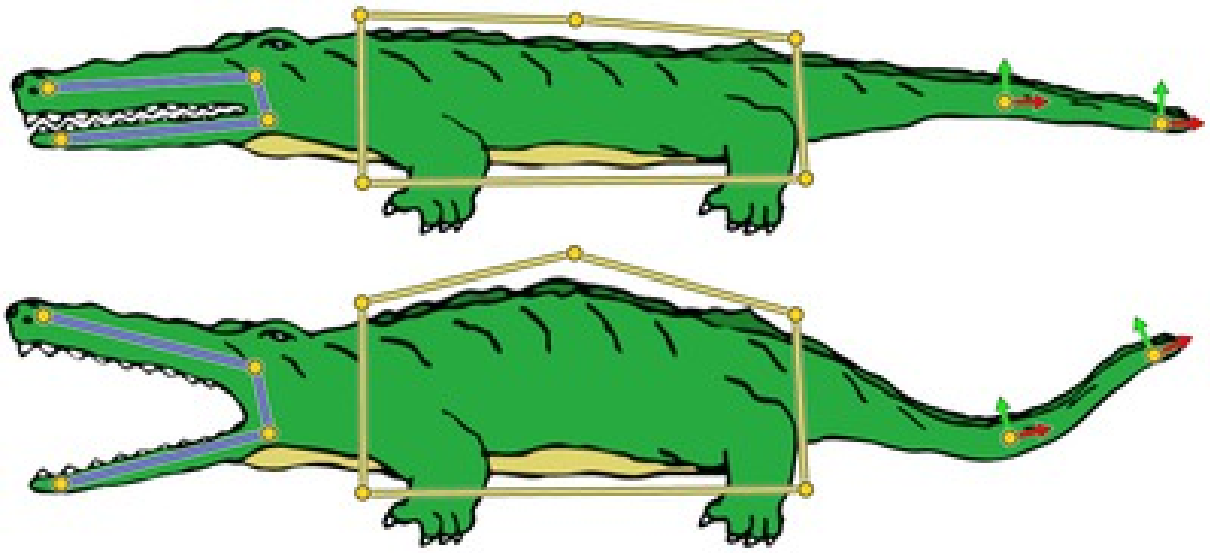


Figure 4.1: *Bounded biharmonic blending supports points, bones, and cages arranged in an arbitrary configuration. This versatility makes it possible to choose the right tool for each subtask: bones to control rigid parts, cages to enlarge areas and exert precise control, and points to transform flexible parts. The weight computation is done at bind time so that high-quality deformations can be computed in real time with low CPU utilization. In this and other figures, affine transformations specified at point handles are illustrated by colored frames. They are omitted when the transformation is just a translation.*

variants dominate practical usage thanks to their speed: each point on the object is transformed by a linear combination of a small number of affine transformations.

In a typical workflow, the user constructs a number of handles and the deformation system binds the object to these handles; this is termed the *bind time*. The user then manipulates the handles (interactively or programmatically) and the system deforms the shape accordingly; this is the *pose time*. Unfortunately, linear blending schemes are not always easy to use. The user must choose the handle type a priori and different types have different advantages (Figure 4.2). Free-form deformations rely on a lattice of handles, but the requirement for regular structure complicates control of concave objects. Skeleton-based deformations offer natural control for rigid limbs, but are less convenient for flexible regions. Generalized barycentric coordinates provide smooth weights automatically, but require construction of closed or nearly closed cages that fully encapsulate transformed objects and can be tedious to manipulate. In contrast, variational techniques support arbitrary handles at points or regions, but at a greater pose-time cost.

Real-time object deformations would be easier with support for all handle types above: points, skeletons, and cages. Points are quick to place and easy to manipulate. They specify local deformation properties (position, rotation and scaling) that smoothly propagate onto nearby areas of the object. Bones make some directions stiffer than others. If a region between two points appears too supple, bones can transform it into a rigid limb. Cages allow influencing a significant portion of the object at once, making it easier to control bulging and thinning in regions of interest.

Our goal is to supply weights for a linear blending scheme that produce smooth and intuitive

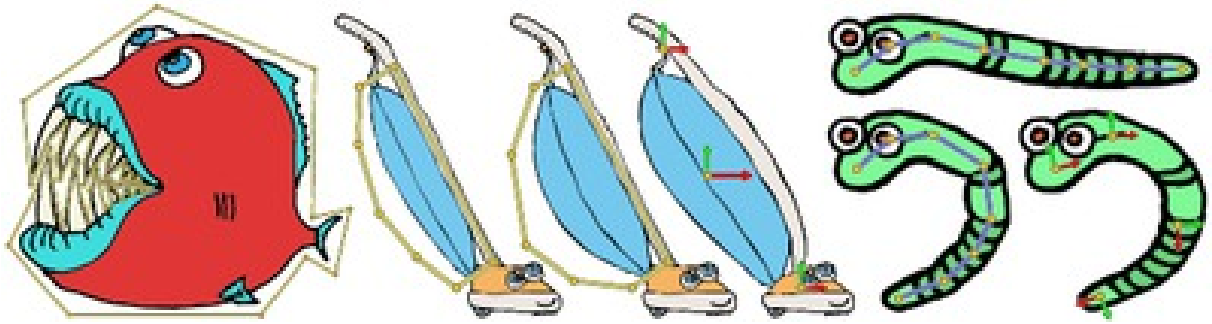


Figure 4.2: *Left to right: Although cages allow flexible control, setting up a closed cage can be both tedious and unintuitive: the* Pirahna*'s jaws require weaving around the teeth. In the case of the* Vacuum*, points can provide crude scaling effects, while cages provide precise scaling articulation. Point handles can provide loose and smooth control, while achieving the same effect with a skeleton results in an overly complex armature.*

deformation for handles of arbitrary topology (Figure 4.1). We desire real-time interaction for deforming high-resolution images and meshes. We want smooth deformation near points and other handles, so that they can be placed directly *on* animated surfaces and warped textures. And, we seek a local support region for each handle to ensure that its influence dominates nearby regions and disappears in parts of the object controlled by other handles.

Our solution computes blending weights automatically by minimizing the Laplacian energy subject to upper and lower bound constraints. Because the related Euler-Lagrange equations are biharmonic, we call these weights *bounded biharmonic weights* and the resulting deformation *bounded biharmonic blending*. The weights are computed once at bind time. At pose time, points on the object are transformed in real time by blending a small number of affine transformations. Our examples demonstrate that bounded biharmonic blending produces smooth deformations and that points, bones, and cages have intuitive local influences, even on objects with complex and concave boundaries. Our weight computation requires space discretization (see Chapter 8) and optimization (see Section 2.2), which could be a drawback in some applications. In any case, the generality of our formulation also makes it possible to provide additional control over the energy minimization, for example to define weights that preserve the shape of specified essential object features (see Section 4.3.3).

4.2 Previous work

Variational methods are known to compute high-quality shape-preserving deformations for arbitrary handles on a surface [Igarashi et al. 2005, Botsch et al. 2006a, Sorkine and Alexa 2007, Botsch and Sorkine 2008] and some variational methods work with bones [Weber et al. 2007] or can be extended to other off-surface handles [Botsch et al. 2007b]. The primary drawback of these techniques is that they rely on optimization at pose time. Although system matrices can be prefactored and back-substitution can be implemented on a GPU [Naumov 2011, Naumov 2012], it is not an embarrassingly parallel problem like linear blend skinning, and is therefore much slower. Even with significant performance tuning [Shi et al. 2007] or model reduction [Der et al. 2006, Sumner et al. 2007, Au et al. 2007], pose-time optimization is too

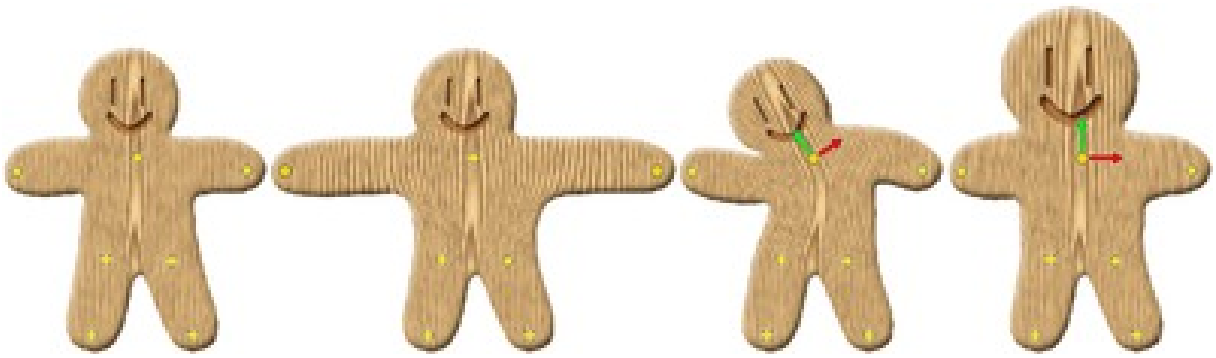


Figure 4.3: *Each handle specifies an affine transformation and these propagate smoothly throughout the object via our blending weights.*

slow to deform high-resolution objects at high framerate, as necessary, for example, for video games. Variational harmonic maps [Ben-Chen et al. 2009] are faster (though not as fast as linear blending), but they restrict the degrees of freedom to harmonic deformations of a (usually manually) specified cage.

Most methods that are fast at pose time compute the transformation at each object point by using a weighted blend of handle transformations. To perform the blending, some methods use moving least squares [Schaefer et al. 2006], some use dual quaternions [Kavan et al. 2008], but most use linear blend skinning (LBS) [Magnenat-Thalmann et al. 1988]. With LBS, the affine transformations of the handles are linearly averaged with different weights to transform each vertex. Although linearly blending rotations leads to well-known artifacts, LBS has been a popular technique for skeletal animation for over two decades because it is simple, predictable, and the pose-time computation can be implemented very efficiently on a GPU. In addition to skeletal animation, most cage-based deformation methods [Floater 2003, Ju et al. 2005, Joshi et al. 2007, Lipman et al. 2007, Hormann and Sukumar 2008] are effectively LBS, where the handle (cage vertex) transformations are restricted to be translations and the focus is choosing the weights. The so called *higher order* barycentric coordinate deformations of [Langer and Seidel 2008] are equivalent to generic LBS (see Section 4.6). Additionally, the reduced-model variational shape deformation methods mentioned above use LBS to go from the reduced model to the full model.

The choice of weights for LBS determines whether the affine transformations of the handles affect the shape intuitively. In some cases, weights that have a closed form in terms of the handle structure have been used [Shepard 1968, Ju et al. 2005, Lipman et al. 2008], but more often they rely on precomputation at bind time or they are painted by hand. In Section 4.3.1 we formulate the desirable properties of LBS weights and in Section 4.3.2, we discuss previous weight choice schemes in the context of these properties.

4.3 Bounded biharmonic weights

Our goal is to define smooth deformations for 2D or 3D shapes by blending affine transformations at arbitrary handles. Let $\Omega \subset \mathbb{R}^2$ or $\mathbb{R}^3$ denote the volumetric domain enclosed by the

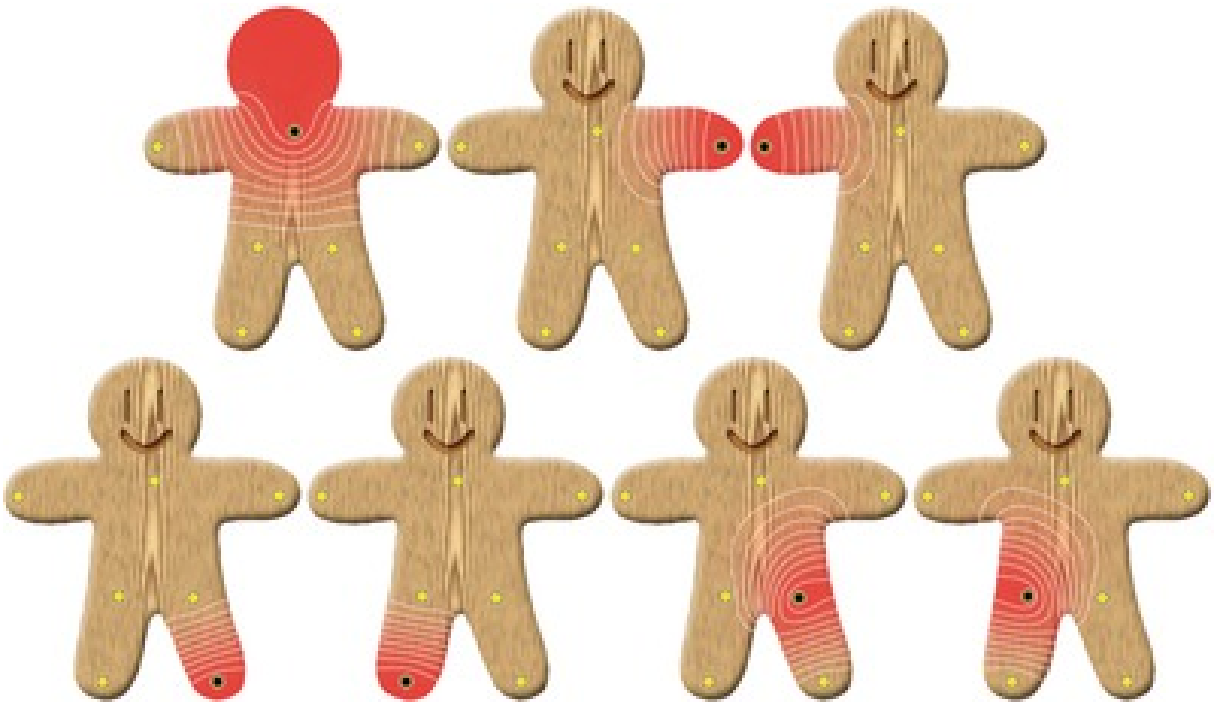


Figure 4.4: *Bounded biharmonic weights are smooth and local: the blending weight intensity for each handle is shown in red with white isolines. Each handle has the maximum effect on its immediate region and its influence disappears in distant parts of the object.*

union of the given shape $\mathcal{S}$ and cage controls (if any). We denote the (disjoint) control handles by $H_j \subset \Omega,\ j = 1, \ldots, m$. A handle can be a single point, a region, a skeleton bone (such that H_j consists of all the points on the bone line segment) or a vertex of a cage. The user defines an affine transformation T_j for each handle H_j (see Figure 4.3), and all points $\mathbf{p} \in \Omega$ are deformed by their weighted combinations :

$$\mathbf{p}' = \sum_{j=1}^{m} w_j(\mathbf{p})\, T_j \mathbf{p}, \tag{4.1}$$

where $w_j : \Omega \to \mathbb{R}$ is the weight function associated with handle H_j.

Note that cages are generally understood as closed polygons in 2D or polyhedra in 3D containing $\mathcal{S}$ or part of it, but our framework is agnostic to the cage topology and treats a cage simply as a collection of simplices, with the requirement that these simplices transform linearly as the cage vertices are translated. Hence, open cages are possible (see Figure 4.16). In contrast to so called *complex* barycentric coordinates [Weber et al. 2009, Weber et al. 2011, Weber et al. 2012], we do not consider cage facets (line segments in 2D or triangles in 3D) as handles; they receive linear weights, as we will see in Section 4.3.1. Note also that for skeleton bones connected by joints, we formally include each joint point in one single bone of those that share it (we assume that the skeleton is never torn apart, i.e., that all bones sharing a joint transform the joint to the same location). In practice, we constrain the weights at shared points to be equally distributed between the overlapping bones to maximize the symmetry of our weights.

4.3.1 Formulation

We propose to define the weights w_j as minimizers of a higher-order shape-aware smoothness functional, namely, the Laplacian energy (see Section 3.3), subject to constraints that enforce interpolation of the handles and several other desirable properties:

$$\underset{w_j,\ j=1,\dots,m}{\arg\min} \sum_{j=1}^{m} \frac{1}{2} \int_{\Omega} (\Delta w_j)^2 \, dV \tag{4.2}$$

$$\text{subject to: } w_j|_{H_k} = \delta_{jk}, \tag{4.3}$$

$$w_j|_F \text{ is linear} \qquad \forall F \in \mathcal{F}_{\mathcal{C}}, \tag{4.4}$$

$$\sum_{j=1}^{m} w_j(\mathbf{p}) = 1 \qquad \forall \mathbf{p} \in \Omega, \tag{4.5}$$

$$0 \le w_j(\mathbf{p}) \le 1, \ \ j = 1, \dots, m, \qquad \forall \mathbf{p} \in \Omega, \tag{4.6}$$

where $\mathcal{F}_{\mathcal{C}}$ is the set of all cage facets and δ_{jk} is Kronecker's delta. Figure 4.4 shows an example of w_j computed for point handles.

Let us discuss several properties possessed by our weight functions w_j that allow for intuitive and high-quality deformations.

Smoothness: Lack of smoothness at the handles causes visible artifacts in 2D textured shapes (Figure 4.13) and prevents placing handles directly *on* 3D shapes. Note that by calculus of variations, minimizing the Laplacian energy (4.2) amounts to solving the Euler-Lagrange equations, which are the biharmonic PDEs in this case: $\Delta^2 w_j = 0$. Equivalently, we could formulate our blending weights as minimizers of the linearized thin-plate energy, as it leads to the same biharmonic PDE (see e.g. [Botsch and Sorkine 2008]). The bounded biharmonic weights are C^1 as they approach zero and one (i.e. near handles) and C^∞ everywhere else, provided that the posed boundary conditions are smooth. This is always the case, with the exception of skeletal joints and cages vertices: for bones connected by joints, the weights must have a discontinuity at the joints since w_j on bone H_j is 1 and it must be zero on the adjacent bone. However, this does not lead to smoothness problems for the actual deformations because the joints are always transformed to the same location by all emanating bones.

For cages, explicit linear interpolation constraints (4.4) on cage facets are required to achieve expected behavior, since otherwise the cage facets would not deform linearly when translating cage vertices. These linear constraints on cage facets preclude smoothness of *any* weights at the cage vertices. Therefore our deformations are not smooth at cage vertices, but they are smooth everywhere else, including across cage facets.

Non-negativity: Negative weights lead to unintuitive handle influences, because regions of the shape with negative weights move in the opposite direction to the prescribed transformation (Figure 4.5). We explicitly enforce non-negativity in (4.6), since otherwise biharmonic functions (as in [Botsch and Kobbelt 2004]) are often negative, even if all boundary conditions are non-negative.

Shape-awareness: Informally, shape-awareness implies intuitive correspondence between the handles and the domain Ω. The influence of the handles should conform to the features of the

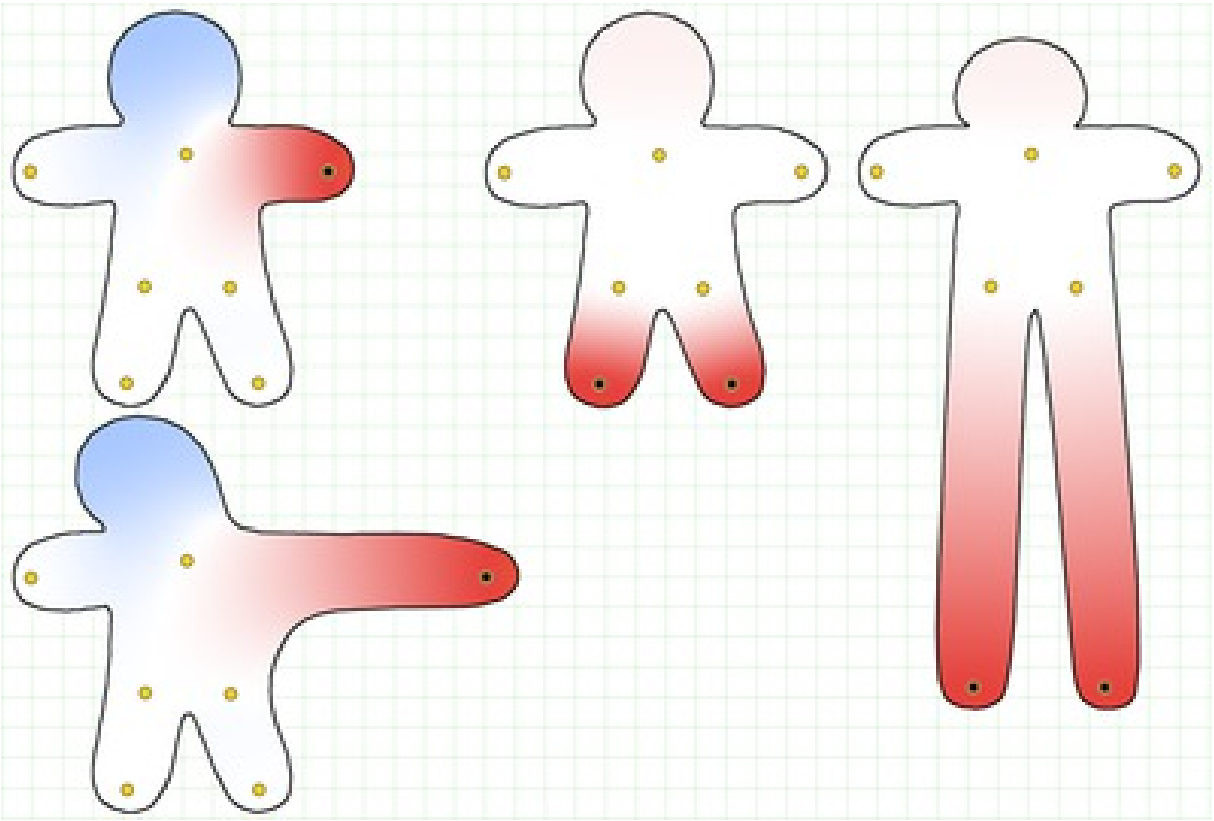


Figure 4.5: *Weights like unconstrained biharmonic functions that have negative weights (left) and extraneous local maxima (right) lead to undesirable and unintuitive behavior. Notice the shrinking of the head on the right.*

shape and diminish with geodesic (as opposed to Euclidean) distance. The best shape-aware behavior one can hope for is when the weights w_j depend on the metric of Ω alone and do not change for any possible embedding of Ω. Our weights are shape-aware since the bi-Laplacian operator is determined solely by the metric.

Partition of unity: This classical property (also seen in e.g. Bézier or NURBS) ensures that if the same transformation T is applied to all handles, the entire object will be transformed by T. We enforce this property explicitly in (4.5) since non-negative biharmonic weights do not sum to 1, unlike unconstrained biharmonic weights.[1]

Locality and sparsity: Each handle should mainly control a shape feature in its vicinity, and each point in Ω should be influenced only by a few closest handles. Specifically, if every locally shortest path (in a shape-aware sense) from a point $\mathbf{p}$ to H_j passes near some other handle, then H_j is "occluded" from $\mathbf{p}$ and $w_j(\mathbf{p})$ should be zero. We observed this property of our weights

[1] Unconstrained biharmonic weights partition unity because they are a linear function of the boundary conditions in (4.3) & (4.4), which also partition unity. Why bounded biharmonic weights do not is made clear by considering them as solved by the active set method (see Section 2.2.3). If we drop the partition of unity constraints then each weight function may be optimized independently. In general, each will find a different active set. The final iteration of the active set method treats these active sets as Dirichlet values, hence the weights are truly biharmonic functions in the region(s) bounded by the active set. Biharmonic functions are also unique. If we consider the sum of our weights inside the union of all these active sets, there is no reason to expect these values to sum to one. Hence, we may not expect the weights to sum to one elsewhere.

Property	Our method	1	2	3	4
Smoothness	Y	-	Y	Y	-
Non-negativity	Y	Y	-	Y	Y
Shape-awareness	Y	Y	Y	-	-
Partition of unity	Y	Y	Y	Y	Y
Locality and sparsity	Y*	-	-	-	Y
No local maxima	$\sim$	Y	-	-	Y

1 = [Joshi et al. 2007, Baran and Popović 2007]
2 = [Botsch and Kobbelt 2004]
3 = [Shepard 1968]
4 = [Sibson 1981]
Y* only experimental confirmation
$\sim$ often, but not always

Table 4.1: *A summary of the properties of five methods for choosing blending weights. Our method appears to satisfy all of the necessary properties. We have empirical evidence but no formal proof for locality and sparsity or the lack of local maxima in our weights.*

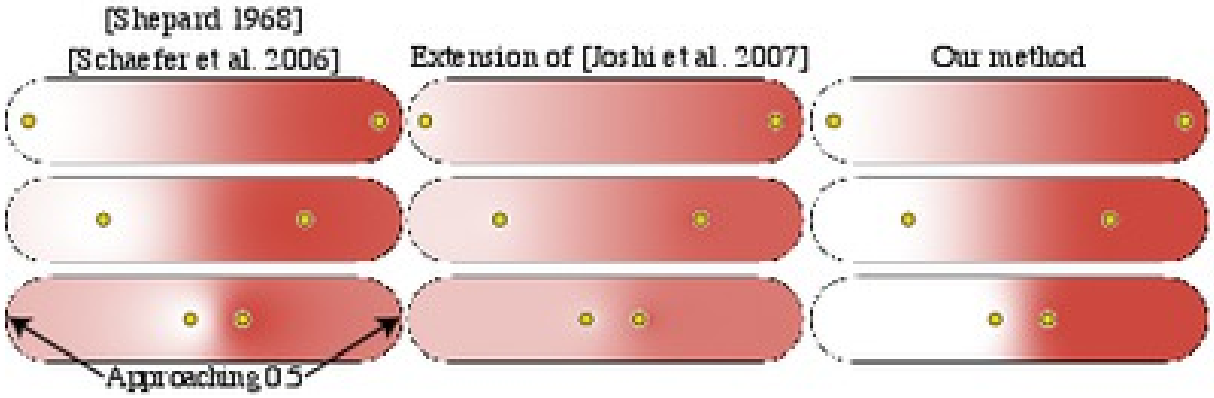


Figure 4.6: *Fall-off effect: Inverse-distance (left) and harmonic weights (center) are not local. Weights are reasonable if two control points are at either ends of this cigar (top). But placing them closer together reveals non-locality (bottom). Our weights block one another (right).*

in all our experiments.

No local maxima: Each w_j should attain its global maximum (i.e., value of 1) on H_j and should have no other local maxima. This property provides monotonic decay of a handle's influence and guarantees that no unexpected influences occur away from the handle. This property was experimentally observed often in our tests. Likely, it is facilitated by imposing the bound constraints (4.6). Without these constraints, the biharmonic functions in general do not necessarily achieve maxima at the handles and cause deformation artifacts (Figure 4.5). While it is true that bounded biharmonic weights often do not have local extrema it is certainly not the case that they are always monotonic. In fact, it is relatively easy to get local extrema to appear on shapes with long appendages geodesically equidistant from handles (e.g. Figure 5.1). Dealing with this non-monotonic behavior is complicated and requires a dedicated optimization strategy and thus a dedicated chapter in this thesis (Chapter 5).

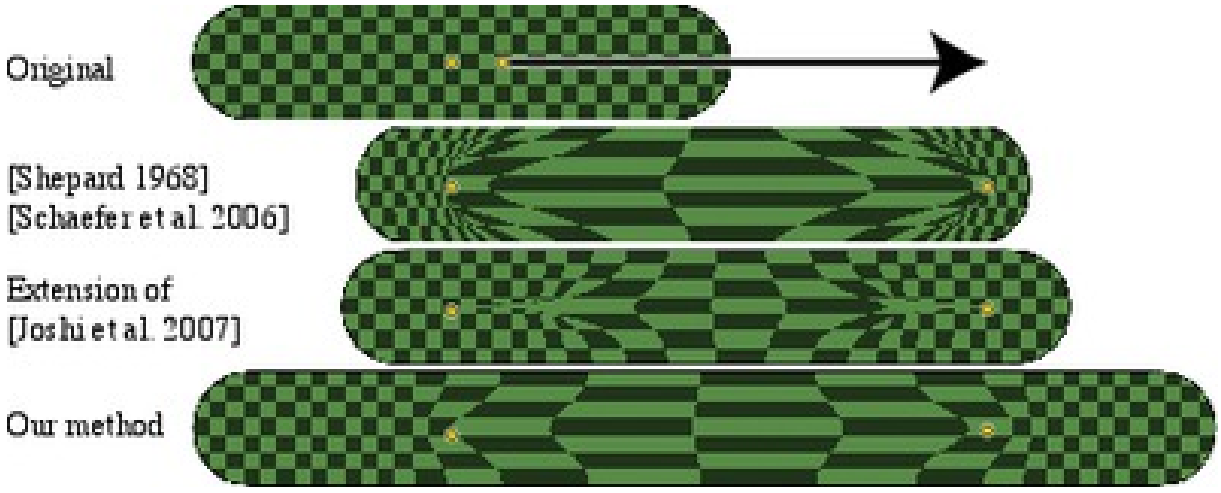


Figure 4.7: *Deformation reveals ill behavior of weights suffering from the* fall-off effect.

4.3.2 Comparison to existing schemes

Existing schemes formulate and satisfy subsets of these properties, but not all. For example, Shepard's [1968] and similar weights (used in embedded deformation [Sumner et al. 2007] and moving least squares image deformation [Schaefer et al. 2006]) are dense and not shape-aware. Further, these methods (as well as [Joshi et al. 2007, Baran and Popović 2007]) are not local (see Figures 4.6, 4.7, 6.12 and 6.13). Other schemes do not support arbitrary handles: for example, extending harmonic coordinates [Joshi et al. 2007] to handles in the cage interior results in a lack of smoothness and locality (see Figure 4.13). Heat diffusion weights [Baran and Popović 2007] suffer from the same problem, albeit to a lesser degree. Natural neighbor interpolation [Sibson 1981] is one of the few schemes that guarantees locality, but it is also not smooth at handles. Biharmonic weights without constraints (cf. [Botsch and Kobbelt 2004]) are smooth, but can be negative (or greater than one), frequently have local maxima away from handles and result in non-local influences. Notably, our weight functions coincidentally satisfy all the axioms formulated for higher order barycentric coordinates [Langer and Seidel 2008]. Table 4.1 shows the properties satisfied by several methods.

A number of recent methods focus on locally preserving or prescribing angles [Lipman et al. 2008, Weber et al. 2009, Weber and Gotsman 2010]. While they have elegant formulations in terms of complex analysis, the methods by Weber et al. are restricted to 2D, while Green Coordinates [Lipman et al. 2008] are only defined for polyhedral cages.

4.3.3 Shape preservation

The energy minimization framework supports incorporating additional energy terms and constraints to customize the weight functions. One example of a useful addition is making all points of a specified region $\Pi \subset \Omega$ undergo the same transformation, i.e., have all the weight functions be constant on Π ($\nabla w_j|_\Pi = 0$). Since typically, we only prescribe translations, rotations and uniform scales at handles, this implies that Π will undergo a similarity transformation in 2D and

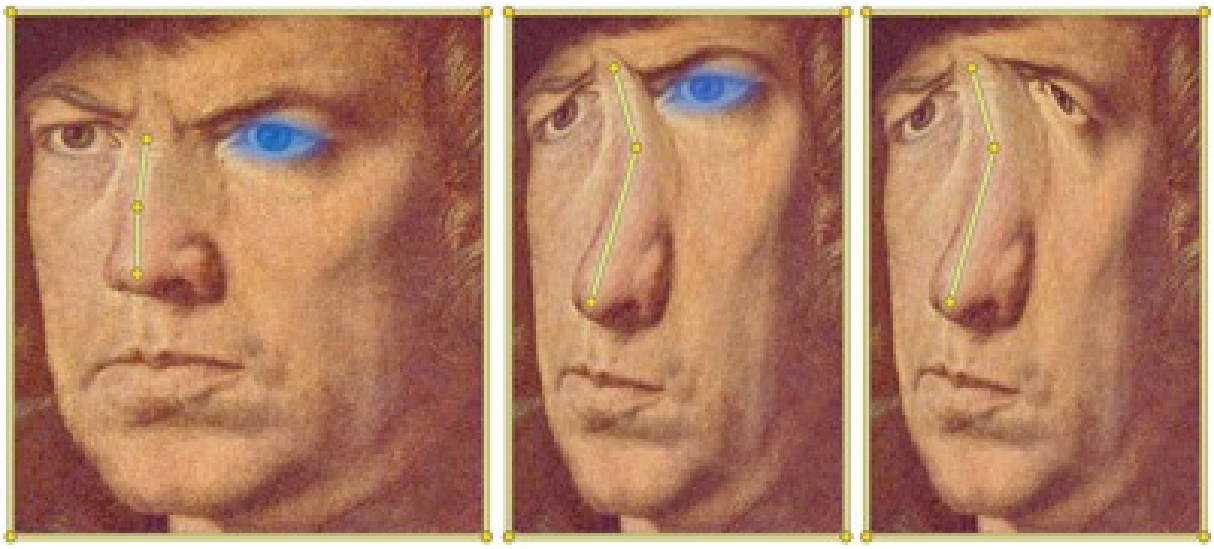


Figure 4.8: *The generality of our optimization framework makes it possible to compute weights that respect salient object features. In this example, marking a region preserves the shape of an eye (middle), which would otherwise be distorted (right). Interaction time remains fast and fluid because deformations are still the result of a weighted combination of prescribed transformations.*

an affine transformation in 3D, so that the shape of Π will be preserved. Similar to the rigidity brush in [Igarashi et al. 2005], the user can paint Π with a (possibly soft) brush, creating a mask $\rho : \Pi \to \mathbb{R}^+$; we then add a least-squares term to our energy minimization:

$$\sum_{j=1}^{m} \frac{1}{2} \int_{\Pi} \rho \|\nabla w_j\|^2 dV. \tag{4.7}$$

See Figure 4.8, where the shape-preservation brush helped retain the shape of the man's eye while deforming the nose. Note that this is different from placing a handle because no explicit transformation needs to be prescribed by the user; the painted region just follows the transformation from other handles (cf. the *free* handles in Chapter 7).

4.3.4 Implementation

We discretize our constrained variational problem (4.2) using linear finite elements in order to solve it numerically with quadratic programming (we use the flattened mixed FEM formulation for fourth-order problems, see Equation (3.12). Assuming that the object $\mathcal{S}$ is given as a 2D polyline or triangle mesh in 3D, we sample vertices on all provided skeleton bones and cage facets, and mesh the domain Ω in a way compatible with all of the handles and the vertices of $\mathcal{S}$. The result is a triangle/tetrahedral mesh $\mathcal{M}$ whose vertices $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ include all discretized H_j's and the object itself. The weights become piecewise-linear functions whose vertex values we are seeking; we denote them by column vectors $\mathbf{w}_j = (w_{1,j}, w_{2,j}, \ldots, w_{n,j})^T$.

The Laplacian energy (4.2) is discretized using the standard linear FEM Laplacian $\mathbf{M}^{-1}\mathbf{L}$ where $\mathbf{M}$ is the lumped mass matrix (with Voronoi area/volume of vertex $\mathbf{v}_i$ on each diagonal entry

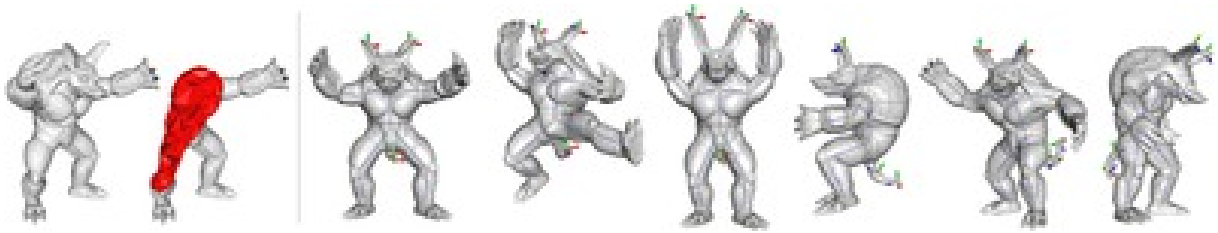


Figure 4.9: *This example uses a human skeleton embedded in the* Armadillo. *The skeleton does not control the tail or the ears, but points are easily added to control them for additional expressiveness in each pose. Left: a cutaway of the* Armadillo *shows the graded tet mesh produced by TetGen. The inner tetrahedra are much larger than the surface ones, which keeps the discretization complexity reasonable.*

$\mathbf{M}_i$, see Section 2.1.3) and $\mathbf{L}$ is the symmetric stiffness matrix (i.e., the cotangent Laplacian in 2D and its equivalent in 3D, see Section 2.1.2):

$$\begin{aligned}\sum_{j=1}^{m} \frac{1}{2} \int_{\Omega} \|\Delta w_j\|^2 dV &\approx \sum_{j=1}^{m} \frac{1}{2} (\mathbf{M}^{-1}\mathbf{L}\mathbf{w}_j)^T \mathbf{M} (\mathbf{M}^{-1}\mathbf{L}\mathbf{w}_j) \\ &= \frac{1}{2} \sum_{j=1}^{m} \mathbf{w}_j^T \left(\mathbf{L}\mathbf{M}^{-1}\mathbf{L}\right) \mathbf{w}_j. \end{aligned} \tag{4.8}$$

We impose the constraints (4.3)-(4.6) using the discretized handles. To discretize the additional shape-preservation energy term (4.7), we employ the linear FEM gradient operator $\mathbf{G}$ (see its derivation in [Botsch et al. 2010]). $\mathbf{G}\mathbf{w}_j$ is a vector of stacked gradients, one gradient per element (triangle in 2D and tetrahedron in 3D; since we deal with linear elements, the gradient over an element is constant). Let $\mathbf{R}$ be a diagonal matrix containing the integrals of the user brush ρ over each element (the brush value on an element is zero if the user did not paint on it) and let $\mathbf{D}$ be the per-element diagonal mass matrix (i.e., for each triangle/tet i, $\mathbf{D}_{ii}$ contains its its area/volume repeated for each dimension). Then the energy term in (4.7) is discretized as

$$\sum_{j=1}^{m} \frac{1}{2} \int_{\Pi} \rho \|\nabla w_j\|^2 dV \approx \sum_{j=1}^{m} \frac{1}{2} \mathbf{w}_j^T \left(\mathbf{G}^T\mathbf{R}\mathbf{D}\mathbf{G}\right) \mathbf{w}_j. \tag{4.9}$$

Note that the matrix $\mathbf{G}^T\mathbf{R}\mathbf{D}\mathbf{G}$ is a kind of weighted linear FEM Laplacian, and therefore its sparsity pattern is a subset of the main energy matrix $\mathbf{L}\mathbf{M}^{-1}\mathbf{L}$, creating no new non-zeros. Hence adding this energy term does not increase the optimization complexity.

We use Triangle [Shewchuk 1996] for 2D constrained Delaunay meshing and TetGen [Si 2003] for constrained tetrahedral meshing to create the discretized domains. In 2D, we configure Triangle to create triangles of near uniform size and shape. For all our 2D examples, Triangle takes less than a second, even for detailed images which require pixel-size triangles. In 3D, we configure TetGen to create rather graded tet meshes to reduce complexity (Figure 4.9); for the *Armadillo* mesh of 43,234 vertices and 120 vertices sampled internally along bones, the resulting tet mesh has 46,898 vertices. For the *Armadillo* and all our 3D examples, TetGen takes a few seconds.

The energy terms in (4.8) and (4.9) are quadratic in the unknowns $\mathbf{w}_j$ and convex, and (4.3)-(4.6) are linear equality and inequality constraints. We use MOSEK [Andersen and Andersen 2000]

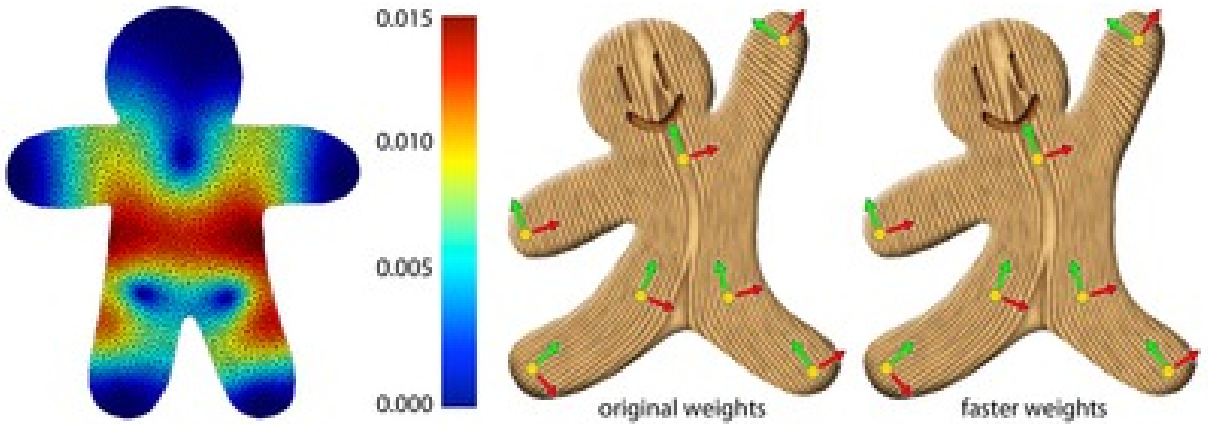


Figure 4.10: *Dropping the partition of unity constraint (4.5) greatly optimizes the precomputation of our weights without losing quality. Left: the mean absolute difference (in pixels) between the original and faster weights at each vertex, over all handle weights at that vertex. Deformations with the same handle configuration using our original (middle) and faster (right) weights are visually indistinguishable.*

as a sparse quadratic programming solver to compute the weights for all of the handles simultaneously.

Since the time necessary to solve the quadratic program is superlinear in the number of unknowns, dividing it into several smaller subproblems allows for a significant speedup. Notice that the optimization of each handle is independent from the rest if we drop the partition of unity constraint (4.5). We have implemented this strategy, solving for each $\mathbf{w}_j$ separately and then normalizing the weights for each vertex in a postprocess. We have observed mostly negligible average differences between this faster solution and the original one, often resulting in visually indistinguishable deformations (see Figure 4.10). Larger differences occasionally occur far from handles, but the weights have the same qualitative behavior: smoothness and observed local support/lack of spurious local maxima. For the *Gargoyle*, for example, computing the weights for 7 handles separately is 50 times faster than simultaneously. We report the timings as well as the difference between the original and these faster weights in Section 4.4.

Once the weights are computed, the deformation itself is real-time even for very large meshes, since it is computed with a GPU implementation of linear blend skinning (4.1).

Specifying handle transforms. In the cage-based systems of the various barycentric coordinates methods, the only inputs are the translations of the cage vertices. In our system, the user provides a full affine transformation at each handle. Depending on the application the user may choose to specify only translations, i.e., identity rotations and scales. However, non-trivial rotations are often necessary to achieve a desired effect, and these could be tedious to specify manually. We found it easier to have rotations inferred from the user-provided translations. To do this, the user supplies a set of *pseudo-edges* between point handles (Figure 4.11). When the user translates a handle, its rotation is computed automatically as the average of the smallest rotations that take each pseudo-edge incident on the handle from its rest orientation to its pose orientation. In 2D these rotations are averaged as signed angles, in 3D as quaternions. Note that the pseudo-edges are in no way related to the computation of bounded biharmonic weights.

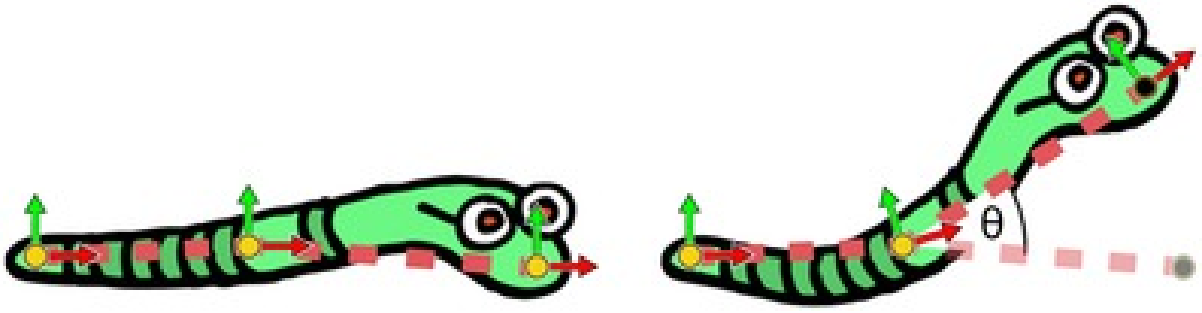


Figure 4.11: *User-provided pseudo-edges between point handles (left) allow rotations to be inferred automatically from translations. When the user translates handles, these pseudo-edges define rotations between their rest and pose orientations (right). Each handle receives the average of rotations defined by incident pseudo-edges.*

They are merely user interface devices that assist the specification of rotations for a set of point handles. This method is simply and efficient, but like other forms of inverse kinematics it does not choose transformations based on their effect on the *quality* of the shape's deformation. This idea of choosing subsets of the LBS degrees of freedom automatically is explored in more detail in Chapter 7.

4.4 Results

Bounded biharmonic blending combines intuitive interaction with real-time performance. Its controls unify three different interaction metaphors so that simple tasks remain simple and complex tasks become easier to achieve.

Experiments. Points are a particularly elegant metaphor for manipulating flexible objects [Igarashi et al. 2005]. Although similar transformations could be accomplished with bones, Figures 4.12 and 4.18 illustrate the simplicity of direct point manipulation of supple regions and highlight the inappropriateness of using rigid bones for the same task.

In contrast to previous techniques [Igarashi et al. 2005, Joshi et al. 2007], our approach deforms shapes smoothly even when the handle transformations are large. Figure 4.13 illustrates the importance of smoothness to minimize texture tearing.

We have observed our weights to be local in all examples tested. Figure 4.14 compares the support regions of our weights to the biharmonic functions of [Botsch and Kobbelt 2004], which are globally supported and contain many local extrema.

Some tasks are more easily accomplished by controlling both points and lines. Figure 4.15 demonstrates our weights with smooth point-based warping while the external cage maintains or resizes the image boundary. Cages are ideally suited for precise area control. In Figure 4.16, we use an arbitrary collection of open and closed lines to manipulate the shape and orientation of the tower. These deformations and fine adjustments, needed to account for perspective distortions, are difficult to achieve with points or lines alone.

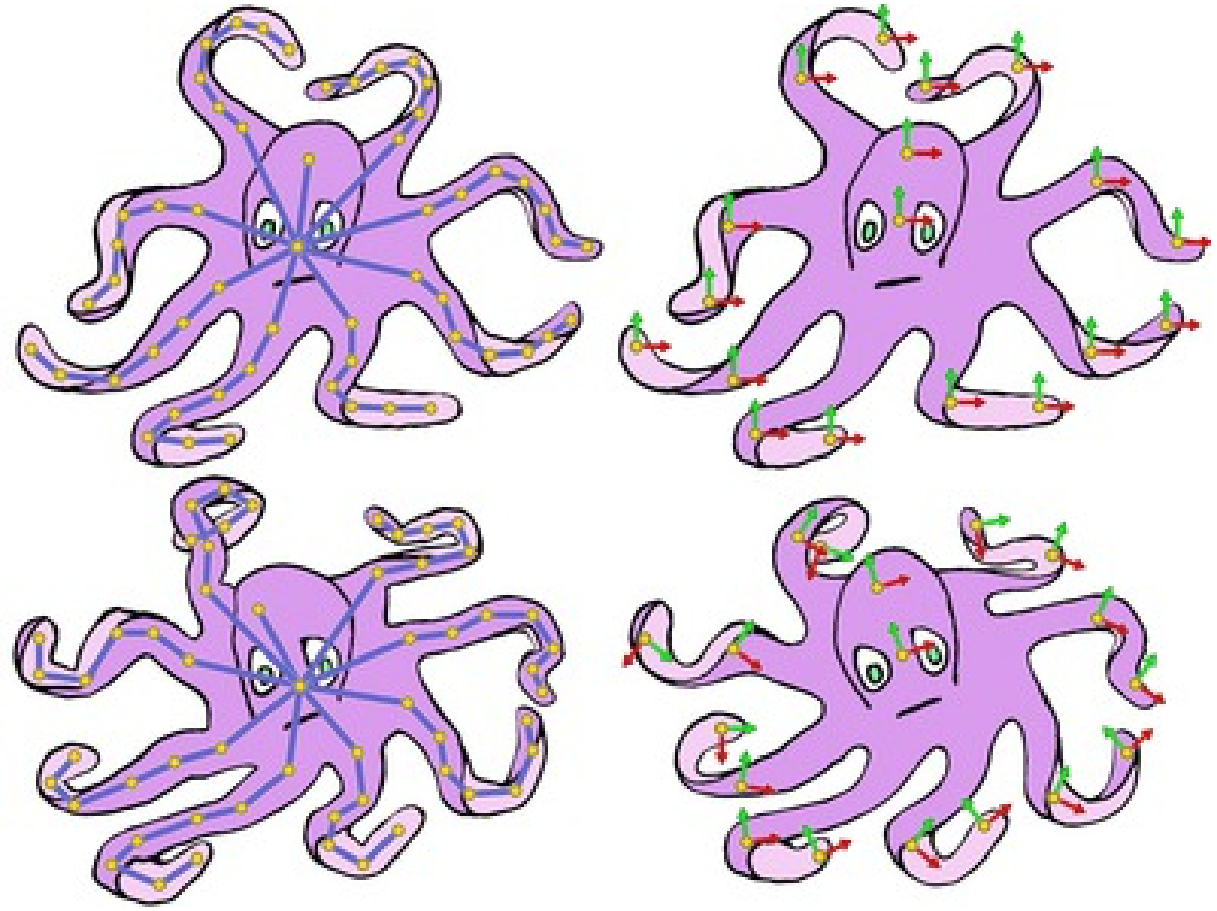


Figure 4.12: *Flexible objects could be deformed with bones, but points are quicker and easier to specify. At best, supple deformation requires a skeleton with many bones, but these can be difficult to control even with inverse kinematics and are often still too rigid.*

Figure 4.13: *Weights must be smooth everywhere, especially at internal handles, which are likely to correspond to important features. Weights that have discontinuities at handles, like Harmonic Coordinates (center), introduce tearing artifacts with even slight changes in the handles. Our weights are smooth, as shown on the right.*

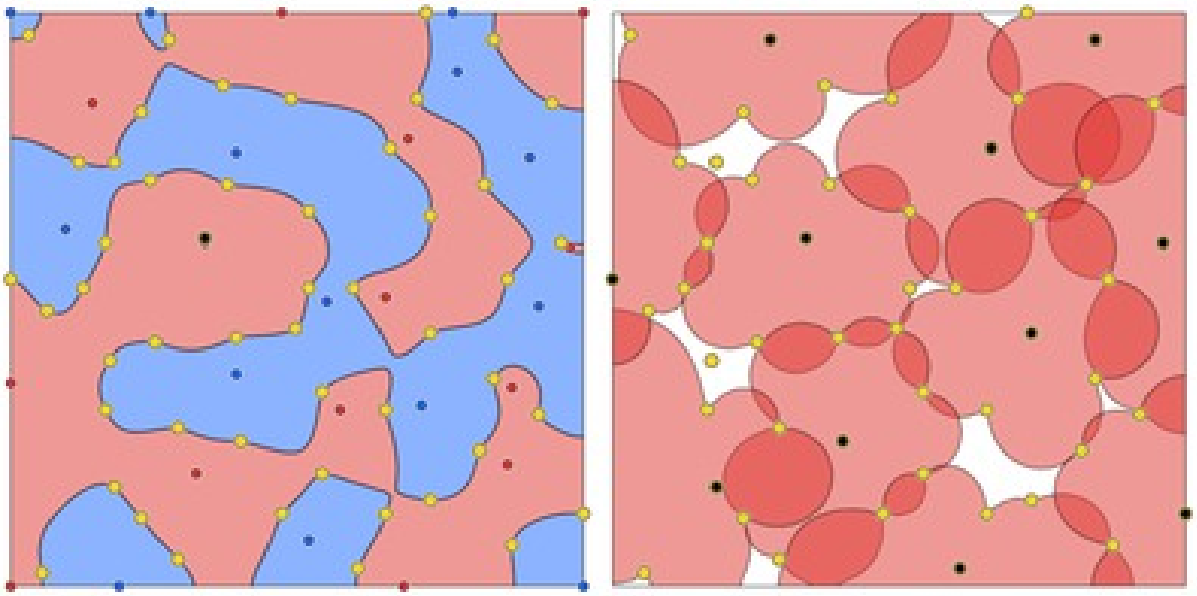


Figure 4.14: *Fifty point handles (black and yellow) are randomly placed in a square domain. Left: the sign for the black handle's* unbounded *biharmonic weight (red for positive and blue for negative regions). The local maxima and minima are shown as red and blue dots, respectively. Right: the support regions for bounded biharmonic weights of the black handles. In this and all other tested examples, the weights are local.*

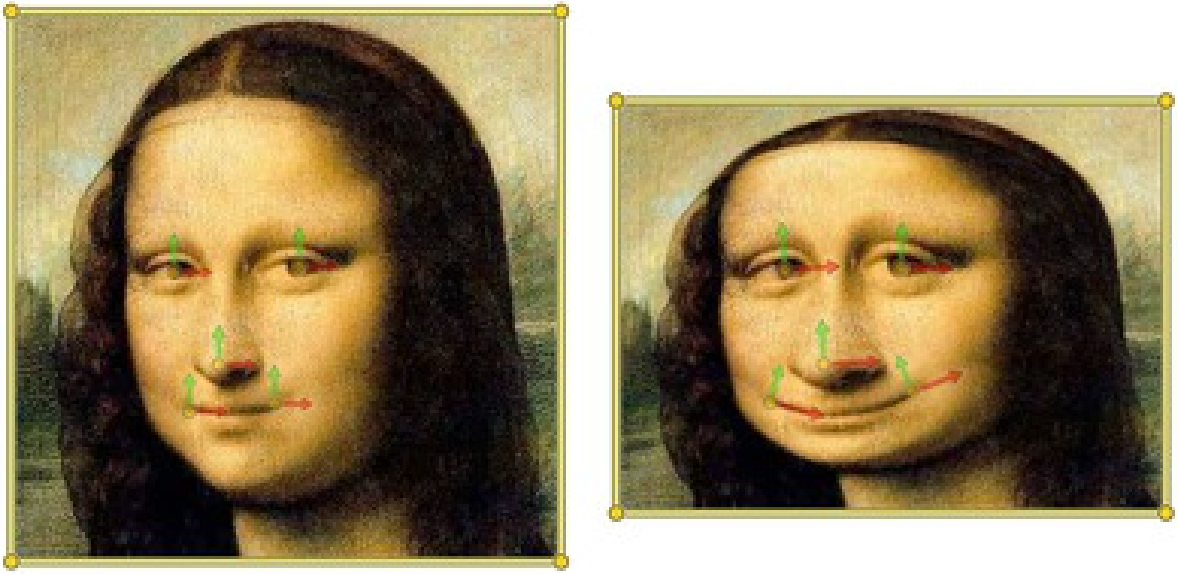


Figure 4.15: *Points handles deform the image by blending the affine transformations specified at each point. The cage on the boundary maintains the rectangular image shape or allows its resizing.*

Figure 4.16: *Deformation of the leaning tower (original is shown left). Cages provide more exact control over area than other handle types.*

Figure 4.17: *We smoothly blend depth values given for each point handle (pseudocolor plot and 3D rendering). We can interactively reorder these values (right).*

When deforming cartoons in 2D, creating believable 2.5D by layering is essential [Igarashi et al. 2005]. Bounded biharmonic weights may used to not only blend transformations but also depth values for layering 2D deformations (Figure 4.17).

Our approach generalizes naturally to 3D. At bind time, the optimization distributes the weights over the volume so that linear blending delivers smooth deformation at run time. This scheme ensures real-time performance and low CPU utilization even for high-resolution meshes. We note that cages can be even more tedious to setup in 3D than in 2D, particularly when they are required to envelop objects fully. For tasks such as hand manipulation shown in Figure 4.19 (left), skeletons are easier to embed and use to manipulate a 3D object. Skeletons still suffer from joint-collapse problems and lack the precise volume control offered by cages and our approach supports and simplifies the combined use of bones and cages. In particular, our approach supports partial cages that control parts of the object but are not required to surround it fully. Figure 4.19 (right) shows a simple cage used to enlarge the belly of the *Mouse*. As always, partial cages can be combined with points and bones, and this combined use of all three metaphors is often the most powerful.

Figure 4.9 shows combined use of points and skeletons. We create a sequence of poses of the *Armadillo* by embedding a human skeleton. The skeleton does not control the tail or the ears so their deformations are adjusted by directly by attaching a few points. Direct surface manipulation makes it easy to bend the tail into a more realistic pose and expressively curl the ears. Likewise in Figure 4.21, point handles are a natural and simple choice of control for stretching and bending the wings of the *Gargoyle*. Bounded biharmonic weights combine the motion of the skeleton and configuration of these points to yield smooth deformations.

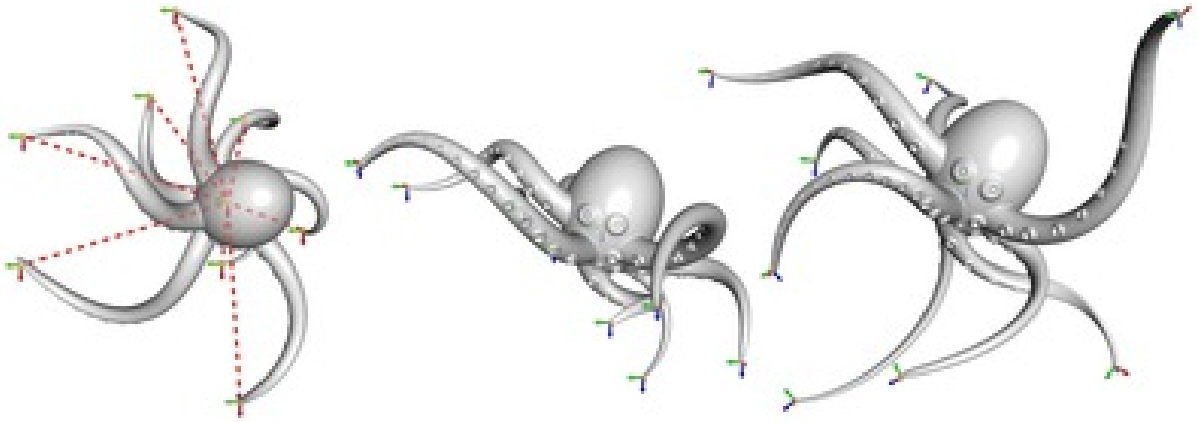


Figure 4.18: *Using point handles to manipulate the flexible* Octopus. *Pseudo-edges are shown in their rest state on the left.*

	$\lvert\mathcal{S}\rvert$	$\lvert\Omega\rvert$	BT/h	E_{mean}	E_{max}
Gingerman		5,040	0.1397	4.3e-3	5.8e-2
Frowny		5,442	0.0906	4.5e-3	9.0e-2
Alligator		7,019	0.1779	1.3e-3	5.5e-2
Pisa		12,422	0.3174	2.5e-3	6.0e-2
Mona Lisa		32,258	1.2417	5.0e-3	1.1e-1
Gargoyle	20,000	46,003	1.1939	4.3e-3	1.8e-1
Hand	28,692	51,263	3.1268	2.0e-3	3.7e-1
Mouse	26,294	112,355	8.4464	4.1e-3	1.1e-1
Armadillo	86,442	142,073	12.0870	4.1e-3	4.0e-1

Table 4.2: *Statistics for the various examples in this chapter. $\lvert\mathcal{S}\rvert$ is the number of triangles of the input 3D model, $\lvert\Omega\rvert$ is the number of elements in the discretization of Ω,* BT/h *is the bind time per handle in seconds. E_{mean} and E_{max} are respectively the mean and max absolute difference between our original weights with (4.5) enforced explicitly and our faster weights where each handle's weights are solved independently and then normalized. Mean and max values are taken over both handles and vertices.*

Discussion. We have tested our method on a MacPro Quad-Core Intel Xeon 2.66GHz computer with 8GB memory. The bind time measurements of our unoptimized code are reported in Table 4.2. One limitation of our solution is the optimization time needed to compute the weights at bind time. We discretized the problem using linear FEM, although other choices may be more efficient, such as the multiresolution framework used in e.g. [Botsch et al. 2007b, Joshi et al. 2007]. Generating bounded biharmonic weights in 3D requires a discretization of the volume (see Chapter 8). Note that once a volume is computed, an arbitrary embedded object (e.g. polygon soup) may be deformed without regard for its topology. The auxillary weights computed at internal volume vertices may be discarded.

Our bounded biharmonic weights do not have the linear precision property, i.e., they do not necessarily reproduce linear functions. This property is necessary for cage-based deformations (e.g. [Ju et al. 2005, Joshi et al. 2007]) that apply the deformation solely by interpolating the positions of cage vertices because otherwise, they would distort the shape when the cage is

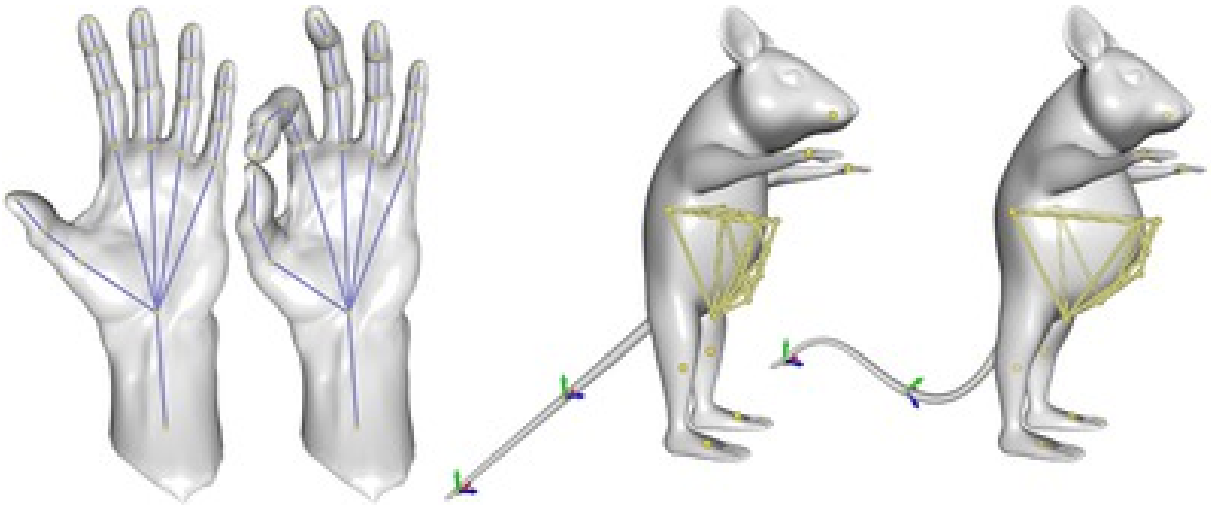


Figure 4.19: *Cages that fully envelop 3D objects such as the* Hand *are difficult to setup. Skeletons are often easier to embed and manipulate. When cages are needed for precise volume control, our scheme makes them easier to use by allowing partial cages that only overlap parts of the object.*

rotated. In contrast, our approach allows arbitrary transformations to be supplied at the handles and blends them over the shape; we therefore do not have to rely on linear precision to be able to work with rotations. A comparison for translational deformation between the linearly-precise Harmonic Coordinates [Joshi et al. 2007] and our cages is shown in Figure 4.20 with a rectangular image.

We have only experimented with linear blending deformations based on Equation (4.1) in this chapter; however, our weights are also useful when combined more advanced methods of transformation blending, such as dual quaternion skinning [Kavan et al. 2008]. When blending rigid motions this way, the result remains a rigid motion for a fixed set of weights. Among other advantages, this will cause the regions painted for shape-preservation (Section 4.3.3) to move rigidly, while currently we only obtain similarity or affine transformation there. This combination and a new skinning technique are presented in Chapter 6.

4.5 Conclusion

We have shown how to unify all popular types of control armatures for intuitive design of real-time blending-based deformations. This allows users to freely choose the most convenient handles for every task and relieves them from the burden of manually painting blending weights.

In future work, we would like to optimize the efficiency of both bind- and pose-time of our deformations. Aside from looking at alternative discretizations and numerical approaches, further analysis will help to reduce the dimensionality of the quadratic program: the observed locality property implies that the weights vanish on significant portions of the domain, which could thus be removed from the minimization. It would also be interesting to apply the weight reduction technique in [Landreneau and Schaefer 2010]. Though, given their similar optimization, encoding sparsity into our original optimization would be more direct.

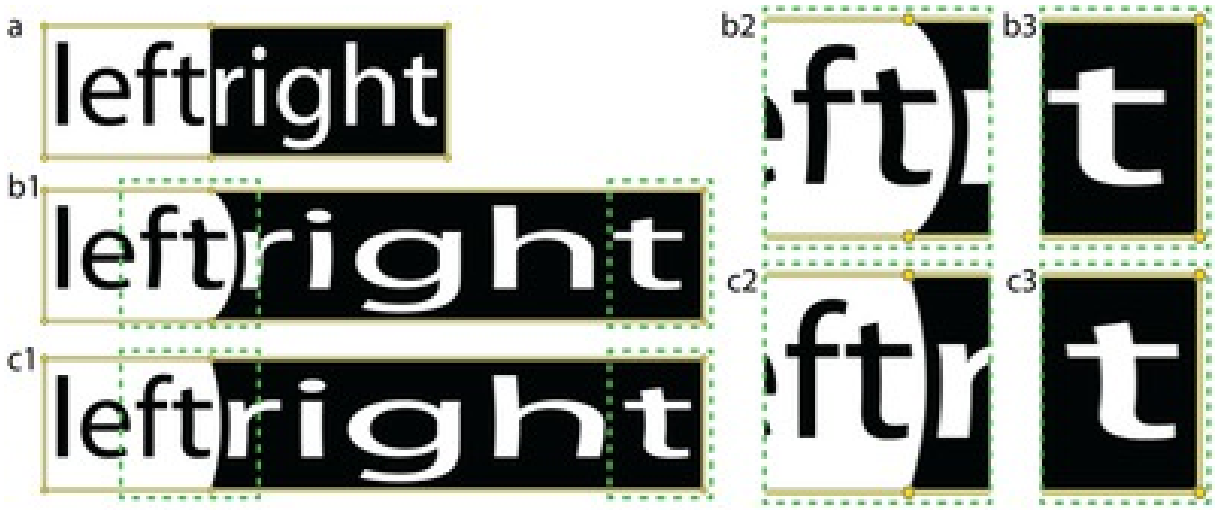


Figure 4.20: *We show the trade-off between locality and linear precision within a cage. The rest pose of an image of text (a) is stretched horizontally with a cage using Harmonic Coordinates (b1), and our bounded biharmonic weights (c1). A closer look shows that Harmonic Coordinates' deformation has a more global response to the handle translation (b2), whereas our weights are more local (c2). On the other hand, Harmonic Coordinates maintain the vertical lines in letter T near the deformed handles (b3), while our weights reveal their lack of linear precision (c3).*

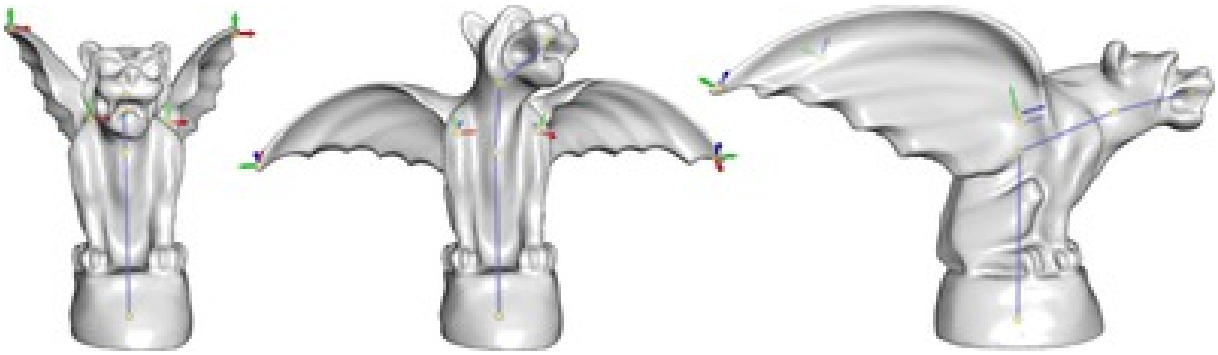


Figure 4.21: *The* Gargoyle *is deformed using an internal skeleton and point handles.*

Skinning-based deformations are prone to foldovers and self-intersections, as the deformation mapping is not always injective. Our method is no exception. Interestingly, we will soon see in Section 4.9 that in the special case of barycentric coordinate deformation *no choice* of weight functions exists which will create bijective deformations. It remains to show this for general LBS, but our hypothesis is that a similar counter example exists. Building a reduced model with our weights for simulation and contact handling is worth exploring in this context. We also plan to study the mathematical properties of the bounded biharmonic weights to determine the necessary conditions for which the observed locality and maximum principle hold.

4.6 Appendix: Equivalence of higher order barycentric coordinates and LBS

We quickly show that the higher order barycentric coordinate deformations of [Langer and Seidel 2008] are equivalent to linear blend skinning. Recall, LBS deformation is written:

$$\mathbf{v}' = \sum_{i=1}^{m} w_i(\mathbf{v})\mathbf{T}_i\mathbf{v} \tag{4.10}$$

$$= \sum_{i=1}^{m} w_i(\mathbf{v}) \left(\mathbf{L}_i\mathbf{v} + \mathbf{t}_i\right), \tag{4.11}$$

where the rest position of a point $\mathbf{v}$ is deformed to a new position $\mathbf{v}'$ according to a sum of affine transformations $\mathbf{T}_i$ (composed of linear and translation parts $\mathbf{L}_i$ and $\mathbf{t}_i$) applied to $\mathbf{v}$ and weighted by a spatially varying weight function $w_i(\mathbf{v})$. Barycentric coordinates may also be used to create deformations:

$$\mathbf{v}' = \sum_{i=1}^{m} w_i(\mathbf{v})\mathbf{c}_i', \tag{4.12}$$

where $\mathbf{c}'$ are the deformed positions of the control polygon. These deformations may then be described as a restricted form of LBS. Because w_i are barycentric *coordinates*, we know by definition that:

$$\mathbf{v} = \sum_{i=1}^{m} w_i(\mathbf{v})\mathbf{c}_i, \tag{4.13}$$

where $\mathbf{c}_i$ are the rest positions of the control polygon. Substituting this identity into Equation (4.12) and rewriting to match Equation (4.11) with $\mathbf{L}_i = \mathbf{I}$ and $\mathbf{t}_i = \mathbf{c}_i' - \mathbf{c}_i$, we have:

$$\mathbf{v}' = \sum_{i=1}^{m} w_i(\mathbf{v})(\mathbf{c}_i' + \mathbf{c}_i - \mathbf{c}_i) \tag{4.14}$$

$$= \mathbf{v} + \sum_{i=1}^{m} w_i(\mathbf{v})(\mathbf{c}_i' - \mathbf{c}_i) \tag{4.15}$$

$$= \sum_{i=1}^{m} w_i(\mathbf{v})(\mathbf{I}\mathbf{v} + \mathbf{c}_i' - \mathbf{c}_i), \tag{4.16}$$

assuming that $\sum_{i=1}^{m} w_i(\mathbf{v}) = 1$ which is typically the case as otherwise affine invariance of the barycentric coordinates is lost. In this way we see barycentric coordinates is an special case of LBS. It is restricted in two ways: first, the transformations $\mathbf{T}_i$ are restricted to translations; and second, the weight functions $w_i(\mathbf{v})$ have the unique property of being *coordinates*. This second quality *helps* alleviate the first restriction in the sense that deformations retain affine invariance, however, the restriction on the weight function prohibits other uses. We discuss derivative control here, but locality and sparsity also become an issue.

Written as a special case of LBS, barycentric coordinate deformation clearly lacks control of derivatives. Simply consider identity translations. With barycentric coordinates, by setting translations to zero, all degrees of freedom are exhausted and the deformation is forced

to be the identity transformation. With general LBS the linear part $\mathbf{L}_i$ of each transformation remains allowing such control. Noticing and exploiting this was the contribution of [Langer and Seidel 2008]. Their derivation is conducted outside the context of linear blend skinning. We will now show that their "higher order barycentric coordinate" deformations are in fact equivalent to LBS with an extra restriction that the first derivatives of the weight functions vanish at each control point: $\nabla w_i = \mathbf{0}$.

In [Langer and Seidel 2008], an interpolatory function is introduced of the form:

$$\mathbf{v}' = \sum_{i=1}^{m} w_i(\mathbf{v}) \left(\mathbf{c}_i' + \mathbf{D}_i(\mathbf{v} - \mathbf{c}_i)\right), \tag{4.17}$$

where $\mathbf{D}_i$ "are the linear functions (usually represented as matrices) which specify the derivatives at the" $\mathbf{c}_i$. We may rewrite our linear blend skinning formula in Equation (4.11) substituting $\mathbf{t}_i = -\mathbf{L_i c}_i + \mathbf{c}_i$:

$$\mathbf{v}' = \sum_{i=1}^{m} w_i(\mathbf{v}) \left(\mathbf{L}_i\mathbf{v} - \mathbf{L_i c}_i + \mathbf{c}_i\right) \tag{4.18}$$

$$= \sum_{i=1}^{m} w_i(\mathbf{v}) \left(\mathbf{c}_i' + \mathbf{L}_i(\mathbf{v} - \mathbf{c}_i)\right), \tag{4.19}$$

or equivalently reading from right to left, $\mathbf{v}$ is first translated to the rest position of the control point $\mathbf{c}$, here the linear (derivative) terms are applied (often just a rotation) and then it is translated *back* to the deformed position. Written in this way it is clear that higher order barycentric coordinate deformations are equivalent to the usual LBS deformations. What is interesting now, is how do the weights proposed by [Langer and Seidel 2008] fair as skinning weights. Unfortunately, they fair rather poorly. Their scheme is to apply a simple filter of to existing barycentric coordinate methods which were not originally designed to be good skinning weights. The filtered harmonic coordinates [Joshi et al. 2007] are promising, but we know these functions to be too globally supported (see Figure 4.7). The "axioms" of [Langer and Seidel 2008] turn out to be not quite all that we desire in skinning weights. Indeed we want interpolation, partition of unity and zero-gradient at handles, but we also want shape-awareness, locality, non-negativity, monotony, and smoothness. In addition, the notion of "coordinates" is lost after the filtering of [Langer and Seidel 2008]. There is no apparent gain in having the input of the filter be *coordinate* functions beyond the fact that they satisfy the interpolation property.

4.7 Appendix: Relationship to precomputed bases

By now we are familiar with how linear blend skinning weight functions can be computed using energy optimization. In the case that weights are minimizers of *unconstrained* quadratic energies, then they are solutions to linear systems. That is to say they are a linear function of the boundary conditions (e.g. the ones or zeros at handle locations). This bears a close resemblance to deformations computed directly as minimizers of quadratic energies [Botsch and Sorkine 2008]. These deformations are also linear functions of their boundary conditions (e.g. positional constraints on selected vertices). If these vertices are grouped

into *region handles*, each deformed with an affine transformation, then we may precompute a few *basis functions* which fully describe the linear deformation [Botsch and Kobbelt 2004, Sorkine et al. 2005]. When the same energies are used for defining LBS weight functions and linear variational deformation, it is tempting to declare that the resulting basis functions and weight functions and the deformations they imply are equivalent [Jacobson et al. 2012b]. Interestingly, this is not the case.

The biharmonic equation and corresponding Laplacian energy (a.k.a. as-harmonic-as-possible energy [Finch et al. 2011], Hessian energy [Weber et al. 2012] or linearized thin-plate energy [Botsch et al. 2010]) is at the heart of many linear (and some *almost linear*) variational techniques [Botsch and Sorkine 2008]. The Laplacian energy is also the active ingredient in our bounded biharmonic weights for LBS. For the sake of clarity we now forget the bounds, and consider *unconstrained* biharmonic weights w_j, which are unique minimizers of:

$$E_B(w_j) = \frac{1}{2}\int_\Omega (\Delta w_j)^2 dV \tag{4.20}$$

$$\text{subject to: } w_j|_{H_k} = \delta_{jk},\ k = 1 \ldots m, \tag{4.21}$$

where here H_k are always isolated regions of the domain[2].

Similarly for linear variational deformation we consider biharmonic *displacement* functions $\mathbf{d} : \Omega \to \mathbb{R}^3$, such that the new positions of some point $\mathbf{p}$ are $\mathbf{p}' = \mathbf{p} + \mathbf{d}$. These displacements are solutions to:

$$E_B(\mathbf{d}) = \frac{1}{2}\int_\Omega \|\Delta \mathbf{d}\|^2 dV \tag{4.22}$$

$$\text{subject to: } \mathbf{d}|_{H_k} = \hat{\mathbf{d}}, \tag{4.23}$$

where $\hat{\mathbf{d}}$ are the (potentially arbitrary) displacements inside handle H_k. Notice that $\mathbf{d}$ is vector-valued, and optimization treats each coordinate function $\mathbf{d}_x$, $\mathbf{d}_y$, and $\mathbf{d}_z$ independently. Thus each coordinate of $\mathbf{d}$ is a biharmonic function. These displacements are identical to those used in [Sorkine et al. 2004] if linearized rotations are omitted. Ostensibly the formulas in [Sorkine et al. 2004] are different: their derivations are fully discrete, but more noticeably, they optimize for new positions $\mathbf{p}'$ directly, minimizing:

$$E_{LSE}(\mathbf{p}') = \frac{1}{2}\int_\Omega \|\Delta \mathbf{p}' - \Delta \mathbf{p}\|^2 dV \tag{4.24}$$

$$\text{subject to: } \mathbf{p}'|_{H_k} = \hat{\mathbf{p}}, \tag{4.25}$$

where $\Delta\mathbf{p}$ are dubbed the "differential coordinates" of the input mesh and $\hat{\mathbf{p}}$ are the new positions of H_k. This is easily shown to be equivalent to Equation (4.22) by linearity of the Laplace operator:

$$\Delta\mathbf{p}' - \Delta\mathbf{p} = \Delta\left(\mathbf{p}' - \mathbf{p}\right) = \Delta\mathbf{d} \longrightarrow E_{LSE}(\mathbf{p}') = E_B(\mathbf{p}' - \mathbf{p}) = E_B(\mathbf{d}). \tag{4.26}$$

Analogous displacements were also defined in [Botsch and Kobbelt 2005] (though replacing biharmonic with triharmonic, and defining quantities in space rather than on the surface).

[2] In all examples here the unconstrained biharmonic weights *coincide* with our bounded biharmonic weights.

As noted in [Botsch and Sorkine 2008], energy minimizing displacements are closely related to energy minimizing surfaces, such as biharmonic *surfaces*:

$$\arg\min_{\mathbf{p}'} \frac{1}{2} \int_{\Omega} \|\Delta \mathbf{p}'\|^2 dV \tag{4.27}$$

$$\text{subject to: } \mathbf{p}'|_{H_k} = \hat{\mathbf{p}}. \tag{4.28}$$

Such surfaces are not directly usable for deformation as they smooth away any high-frequency detail. They are also heavily dependent on the parameterization of the Laplace(-Beltrami) operator Δ employed [Botsch 2005, do Carmo 1976] (see Section 3.8). Nonetheless, these surfaces are combined with a detail restoration technique to achieve deformation in [Kobbelt et al. 1998], and — more interestingly for this conversation — by [Botsch and Kobbelt 2004], where precomputed bases are also defined. Similar bases were applied by [Sorkine et al. 2005] for point handles. Though designed for solutions to Equation (4.27), the construction of bases in [Botsch and Kobbelt 2004] is also directly applicable to Equation (4.22). We restrict our consideration to bases for biharmonic displacement, since it is most directly comparable to LBS with biharmonic weights.

As in Section 3.3, we may discretize Equation (4.22) for a mesh with vertices V using the mixed finite elements method for region constraints[3]. We substitute the known displacements at selected vertices $\mathbf{d}_H \in \mathbb{R}^{|H|\times 3}$ and take partial derivatives with respect to the unknown displacements $\mathbf{d}_I^\mathsf{T} \in \mathbb{R}^{3\times|V\setminus H|}$. Setting the right-hand side to zero produces a set of linear equations, whose solution is the minimum of Equation (4.22):

$$\frac{dE_B}{d\mathbf{d}^\mathsf{T}} = 2\mathbf{Q}_{I,I}\mathbf{d}_I + 2\mathbf{Q}_{I,H}\hat{\mathbf{d}}_H = 0, \tag{4.29}$$

where $\mathbf{Q} = \mathbf{L}^\mathsf{T}\mathbf{M}^{-1}\mathbf{L}$ is the reduced system matrix in the mixed FEM discretization composed of the cotangent stiffness matrix $\mathbf{L}$ and (diagonal) mass matrix $\mathbf{M}$, see Equation (3.12). Alternatively, one can forget one's continuous roots and just view $\mathbf{Q}$ as a discrete bilaplace operator [Botsch and Kobbelt 2004]. We use subscripts $\mathbf{Q}_{X,Y}$ to indicate matrix slicing according to rows X and columns Y. Here H refers to selected vertices and $I = V \setminus H$ interior vertices. Moving the known terms to the right-hand side gives:

$$\mathbf{Q}_{I,I}\mathbf{d}_I = -\mathbf{Q}_{I,H}\hat{\mathbf{d}}_H, \tag{4.30}$$

and the unknowns may be solved by computing:

$$\mathbf{d}_I = -\mathbf{Q}_{I,I}^{-1}\left(\mathbf{Q}_{I,H}\hat{\mathbf{d}}_H\right), \tag{4.31}$$

where in practice we never compute $\mathbf{Q}_{I,I}^{-1}$ explicitly but rather precompute a Cholesky decomposition, $\mathbf{Q}_{I,I} = \mathbf{K}^\mathsf{T}\mathbf{K}$, where $\mathbf{K}$ is lower triangular. To solve interactively at runtime, we change

[3] Technically, any open boundaries will receive first and third order implicit Neumann conditions: $\partial\mathbf{d}/\partial\mathbf{n} = 0$ and $\partial\Delta\mathbf{d}/\partial\mathbf{n} = 0$ [Helenbrook 2003]. This behavior is seen in all three columns of Figures 4.22 and 4.23. If any regions degenerate into curves then first-order implicit Neumann conditions apply. If any regions degenerate into points, then first-order implicit Neumann conditions should also apply. However, in practice this is not observed, presumably due to discretization error (perhaps implicit third-order Neumann conditions are activated instead). Point constraints are particularly disturbing as they imply discontinuous solutions/derivatives in the smooth setting. Investigating point constraints properly is an interesting area of future work.

$\hat{\mathbf{d}}_H$, compute $\mathbf{Q}_{I,H}\hat{\mathbf{d}}_H$ using sparse matrix-vector multiplication and finally conduct 3 (number of dimensions and columns in $\mathbf{d}$) sparse back-substitutions [Lipman et al. 2004].

At this point, we may notice as [Botsch and Kobbelt 2004] that although $\hat{\mathbf{d}}_H$ represents many mesh vertex displacements they are usually dependent on only a *few* region handles being transformed by the user. If the transformation of each region H_j and thus all corresponding displacements in $\hat{\mathbf{d}}_H$ are determined by a single affine transformation $\mathbf{T}_j \in \mathbb{R}^{3\times(3+1)}$, then we may follow the proposal in [Botsch and Kobbelt 2004] to maximize precomputation[4] by writing $\hat{\mathbf{d}}_H$ in terms of $\mathbf{T}_j$:

$$\hat{\mathbf{d}}_H = \sum_{j=1}^{m} \delta_{jk} \left(\mathbf{T}_j \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} - \mathbf{p} \right) \tag{4.32}$$

$$= \sum_{j=1}^{m} \delta_{jk} \left(\mathbf{T}_j - \mathbf{I} \right) \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} \tag{4.33}$$

$$= \mathbf{A}\tilde{\mathbf{T}}, \tag{4.34}$$

where $\mathbf{I} \in \mathbb{R}^{3\times(3+1)}$ is the identity transformation, $\tilde{\mathbf{T}} \in \mathbb{R}^{(3+1)m\times 3}$ stacks transposed matrices $(\mathbf{T}_j - \mathbf{I})^{\mathsf{T}}$, and $\mathbf{A} \in \mathbb{R}^{|H|\times 3m}$ contains stacked rest homogeneous positions of $\mathbf{p}|_H$, with column blocks:

$$\mathbf{A}_j = \begin{pmatrix} \delta_j(\mathbf{p}|_H) \otimes (\mathbf{p}|_H & \mathbf{1}) \\ \delta_j(\mathbf{p}|_H) \otimes (\mathbf{p}|_H & \mathbf{1}) \\ \delta_j(\mathbf{p}|_H) \otimes (\mathbf{p}|_H & \mathbf{1}) \end{pmatrix}, \tag{4.35}$$

where $\delta_j(\mathbf{p}|_H)$ is 1 for all $\mathbf{p}|_H \in H_j$ and 0 otherwise and $\otimes$ indicates a row-wise, Hadamard-like product[5].

Substituting Equation (4.34) into Equation (4.31) produces:

$$\mathbf{d}_I = -\mathbf{Q}_{I,I}^{-1} \left(\mathbf{Q}_{I,H}\mathbf{A}\tilde{\mathbf{T}} \right) \tag{4.36}$$

$$= \tilde{\mathbf{Q}}_I\tilde{\mathbf{T}}, \tag{4.37}$$

where the columns of the *dense* matrix $\tilde{\mathbf{Q}}_I \in \mathbb{R}^{|I|\times(3+1)m}$ are called the *precomputed basis functions*. By appending the necessary rows of the identity matrix above, we may write an equation which also includes the known displacements on the left-hand side:

$$\mathbf{d} = \tilde{\mathbf{Q}}\tilde{\mathbf{T}}. \tag{4.38}$$

To deform interactively we simply updated $\tilde{\mathbf{T}}$ and compute $\tilde{\mathbf{Q}}\tilde{\mathbf{T}}$ via dense matrix-vector multiplication. Written in matrix form it is apparent that linear variational displacement with affine transformations at region handles is an instance of Animation Space deformation [Merry et al. 2006], itself a special case of [Wang and Phillips 2002]. Compared with LBS,

[4] Whether the resulting dense matrix multiplication is actually faster than performing three sparse back substitutions at runtime probably comes down to implementation. Both operations are in theory $O(n)$, n being the number of unknown mesh vertex coordinates. Indeed, sparse back substitution is more difficult to parallelize, though not impossible [Naumov 2011, Naumov 2012].

[5] $\mathbf{a} \otimes (\mathbf{b}, \mathbf{c}, \mathbf{d}, \dots) = (\mathbf{a} \circ \mathbf{b}, \mathbf{a} \circ \mathbf{c}, \mathbf{a} \circ \mathbf{d}, \dots)$, where $\circ$ is the usual Hadamard (element-wise) vector-vector product.

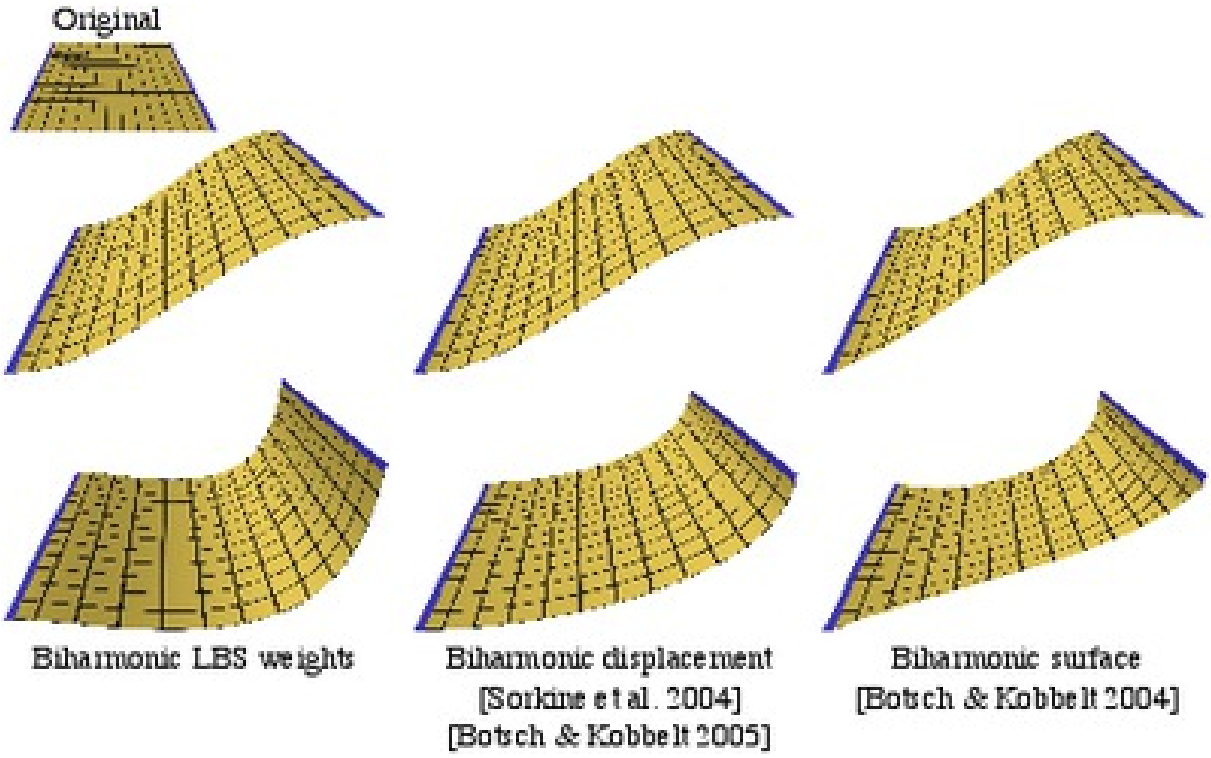


Figure 4.22: *Top: a plane is deformed with various biharmonic methods by applying a translation (top) and a rotation (bottom) to a single handle.*

Animation Space assigns a vector of four weights per vertex instead of a single scalar. These weights multiply against the vertex's rest position in homogeneous coordinates, meaning it effectively blends an arbitrary rest shape per handle. Every four columns in our $\tilde{\mathbf{Q}}$ represents one such rest shape. However, since the original rest positions are not explicitly taken into account, one would need to divide these columns by $\begin{pmatrix}\mathbf{v}^\mathsf{T} & 1\end{pmatrix}$ to recover the Animation Space weights.

We are now in a position to directly compare the basis matrix $\tilde{\mathbf{Q}}$ with our biharmonic weight functions $\mathbf{w}$. In particular, by the definition of $\mathbf{A}$, every fourth column of $\tilde{\mathbf{Q}}$ corresponds exactly to a weight function $\mathbf{w}_j$. These columns correspond to the 1 from the homogeneous coordinates representation and respond to the translation components in $\mathbf{T}$. It is not surprising then that LBS with biharmonic weight functions and biharmonic displacement deformations agree exactly when handle transformations are limited to translations (see top rows in Figures 4.22 and 4.23).

However, if transformations contain rotations or any other non-trivial linear part, then the behavior can be drastically different (see bottom rows in Figures 4.22 and 4.23). Intuition for this follows from noticing that LBS is trilinear in the weight functions, transformation matrix elements, and the point's rest position. Notice that biharmonic displacement is linear with respect to the bases, the transformations, and the rest positions of only the selected vertices in each *handle*. The rest positions of interior vertices are not involved in the basis functions except implicitly in the energy coefficients.

Another way to gain intuition is to look at the individual coordinates of the biharmonic displacements. Each of these are biharmonic functions and so are uniquely defined by their boundary conditions, namely the displacements of the transformed handles. In particular, consider ro-

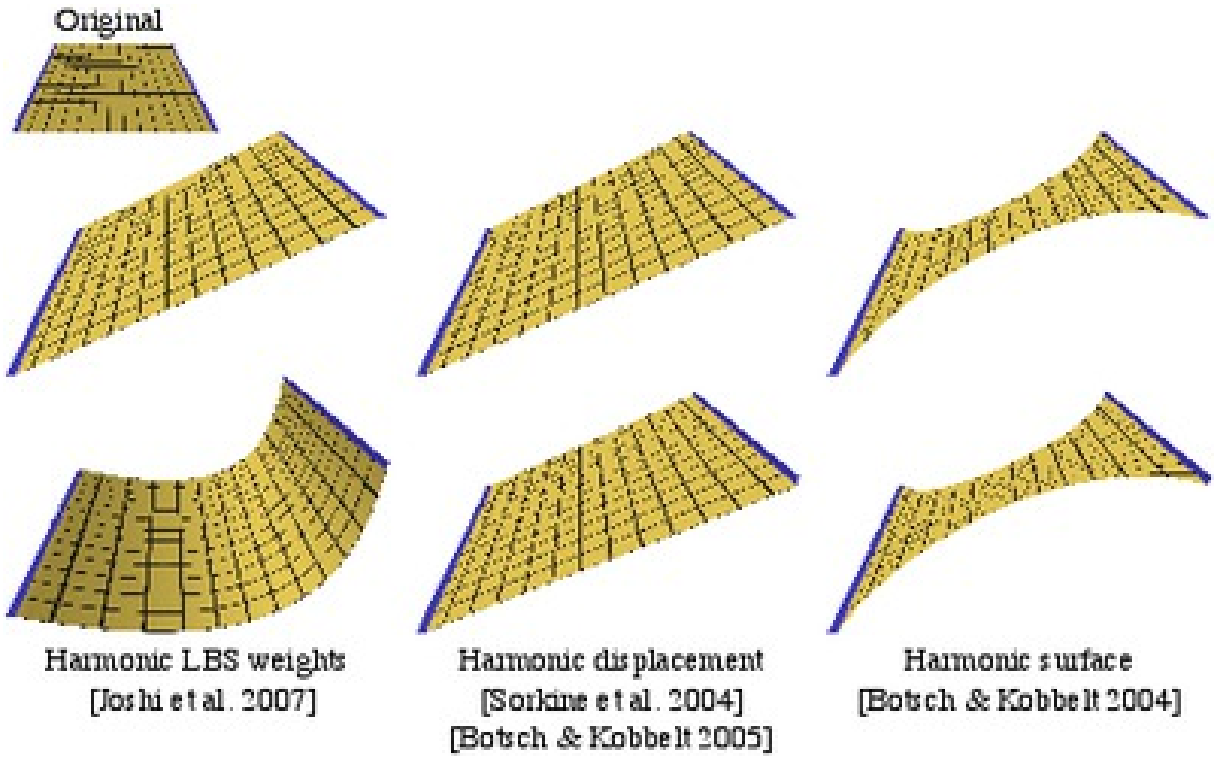


Figure 4.23: *Top: a plane is deformed with various harmonic methods by applying a translation (top) and a rotation (bottom) to a single handle.*

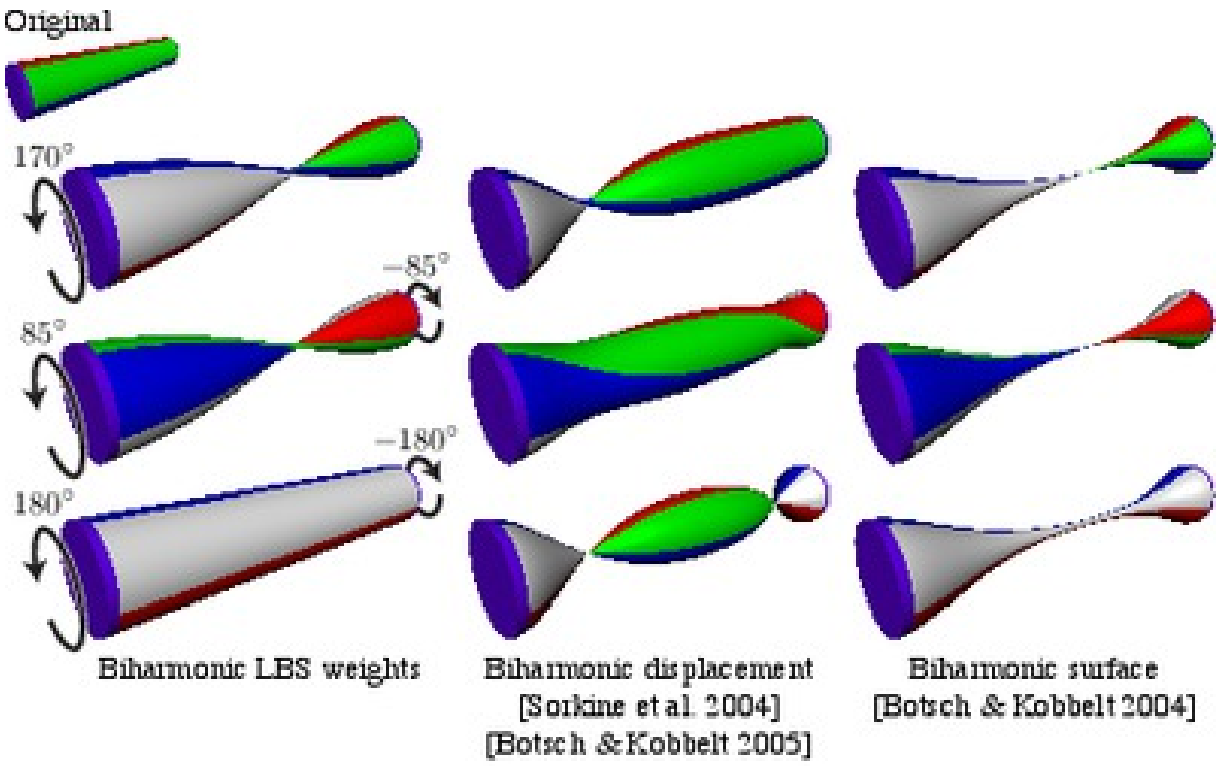


Figure 4.24: *Rotations applied to region handles controlling a cylinder, produce very different deformations using various biharmonic methods.*

tating one handle by 170 degrees in the middle column of Figure 4.24. The displacements of this region are vectors pointing through the axis of rotation. The y-coordinate of the displacements are a sine function going around the handle. If the other handle remains fixed then all its displacement coordinates are zero. The y-coordinate of the displacement in the interior is thus *the* biharmonic function which interpolates these boundary values (see Figure 4.25). Visualized over a parameterization to the plane, we can clearly see that this function is not symmetric along the x-axis. The boundary conditions are also not, so why should it be? Instead we see that the *influence* of the zero boundary conditions on the left spread farther than the sine function on the right. Combined with similar behavior of the z-coordinate, this results in a twist that is asymmetrically placed along the cylinder. In contrast, blending these transformations with biharmonic weights results in a symmetric deformation: the weights are symmetric so anything they blend will also be (see left column in Figure 4.24).

The asymmetry of biharmonic displacements is exacerbated if we rotate each handle 85 degrees in opposite directions instead. This produces two smaller twists with a bump in the middle. To understand this, we again look at an individual coordinate function. Now our y-coordinate is *the* biharmonic function which interpolates two phase-shifted sine functions (middle row, middle column in Figure 4.24). The oscillations quickly die out creating a near-constant 0 function in the middle (middle row, middle column in Figure 4.25, corresponding to the unrotated bump in the middle of the deformed result. In contrast, LBS with biharmonic weights is affine invariant so the resulting LBS deformation is just a rotated version of what we had before.

Rotating each handle even more exaggerates these effects. LBS is incapable of blending rotations differing more than 180° so the result is just a global rotation. In each of the cases so far, we see the *candy-wrapper effect*: collapsing shrinkage caused by blending rotations linearly. This is a long well-known artifact of LBS [Weber 2000], but also a studied issue with linear variational methods [Botsch and Sorkine 2008]. For the sake of comparison we show the corresponding biharmonic surfaces for the cases discussed above (see third columns in Figures 4.24 and 4.25). Here the shrinkage due to linearity is amplified by the shrinkage inherent in minimal curvature surfaces.

The C^1 continuity of biharmonic displacements confuses this comparison a bit because, like LBS with smooth weights, derivatives of each handle's transformation are interpolated. But, this does not promise equivalence (second row in Figure 4.22). Switching to C^0 harmonic weights and harmonic displacements (minimizers of Dirichlet energy) makes the difference clear (Figure 4.23). In neither case do we expect/see derivative continuity, but harmonic displacements do not blend rotations of the rigid transformations at all and the deformation is equivalent to that of applying translations. LBS with harmonic weights on the other hand produce very different deformations depending on whether rotations or just translations are interpolated.

LBS with biharmonic weights is symmetric, detail-preserving, respectful of handle transformations and affine invariant. This seems to imply superiority over biharmonic displacements or reintroducing details atop biharmonic surfaces. We need to be careful about our claims. If we allow general affine transformations and not just translations then we cannot claim that the LBS displacements or surface are "biharmonic" or "minimizing the Laplacian energy". Instead, we must grasp at the notion of the *blending* being biharmonic and energy minimizing. This has a strong relationship to the local frame blending of [Lipman et al. 2005, Zayer et al. 2005], discussed in [Botsch and Sorkine 2008]. Its difficult to make claims about the smoothness or fair-

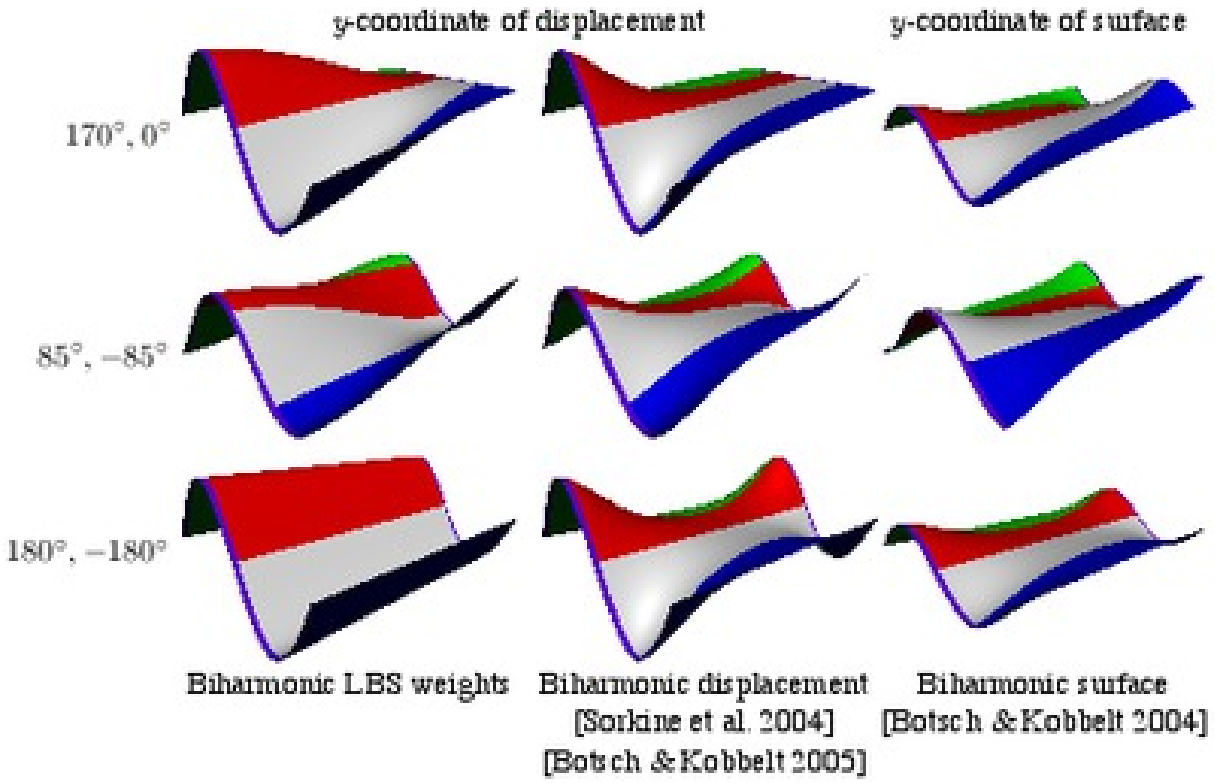


Figure 4.25: *The y-coordinate of the displacements of the biharmonic deformations in Figure 4.24 visualized as a height field over a planar parametric domain.*

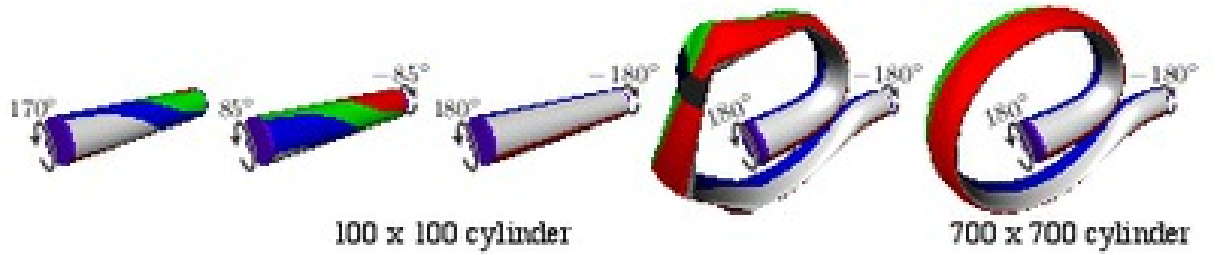


Figure 4.26: *A cylinder is deformed using biharmonic weights by rotating the region handles. Dual quaternion skinning prevents collapse during twisting, but it is unstable. A small change in the rotation and the shape takes on a bizarre form, ultimately shown with a finer tessellation.*

ness of the surface beyond simple extrapolation from the smoothness or fairness of the weight functions themselves. If the weights are smooth the deformation will be smooth. Once satisfied with the notion of biharmonic blending, one may consider other, even nonlinear, blends such as dual quaternion skinning [Kavan et al. 2008] (see Figure 4.26) or even blending with two sets of biharmonic weights (see Chapter 6).

4.8 Appendix: A cotangent Laplacian for images as surfaces

By embedding images as surfaces in a high dimensional coordinate space defined by each pixel's Cartesian coordinates and color values, we directly define and employ cotangent-based, discrete differential-geometry operators. These operators define discrete energies useful for image segmentation and colorization.

Many image processing techniques rely on differential operators defined in terms of some metric adapted to image content. For example, discrete Laplacians with stencils weighted by a function of pixel locations and color values define energies whose minima intuitively propagate tone map adjustments [Lischinski et al. 2006] or sparse color values [Levin et al. 2004]. Other techniques have overlain triangle meshes atop images to reduce computation complexity (e.g. for image warping [Karni et al. 2009]), while simultaneously manifesting the ability to employ discrete differential-geometry operators common in computer graphics [Meyer et al. 2003].

The mesh-based Laplacians enjoy well-studied properties: convergence with respcet to mesh resolution, positive semi-definiteness when defining Dirichlet energies, and symmetric, locally-supported stencil weights [Wardetzky et al. 2007b]. Though the content-adaptive stencils used by [Levin et al. 2004, Lischinski et al. 2006] imply discrete Laplacians, they are rarely labeled as such. As a result they appear to be less studied in this regard and most likely only enjoy a subset of these properties. However, they depend on the image color values, and are thus tantamount to defining a Laplacian in terms of some content-adaptive metric. As of yet, the planar triangle meshes previously used in image processing incorporate only the Cartesian coordinates of mesh vertices and are thus defined solely by the Euclidean image-plane metric, ignorant of image content.

We define an *image surface* by first overlaying a triangle mesh atop the image. We place vertices at pixel centers and Delaunay-triangulate them. The mesh is then *lifted* into higher dimensional space by appending each pixel's color values as coordinates to the corresponding mesh vertex. Thus the pixel i at location (x_i, y_i) is lifted to (x_i, y_i, I_i), $(x_i, y_i, R_i, G_i, B_i)$, or $(x_i, y_i, L_i, A_i, B_i)$ in a grayscale, RGB, or LAB color model respectively.

Now our triangle mesh lives as a disk-topology surface embedded in a higher dimension. If only a single color channel is used then the surface lives in $\mathbb{R}^3$, and typical discrete differential operators common in 3D mesh processing may be immediately applied. If we use more

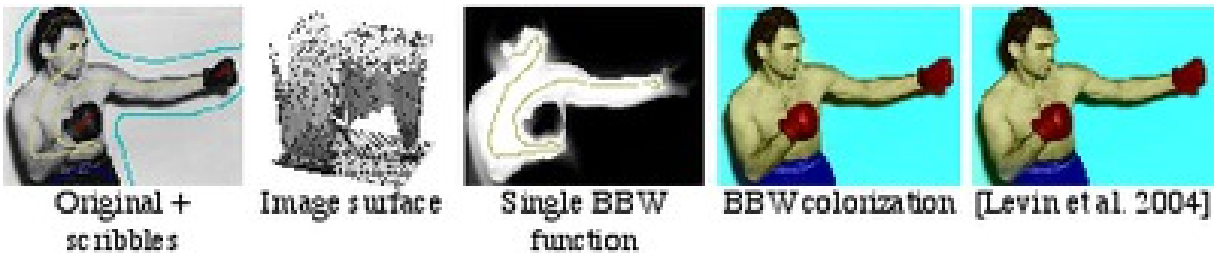
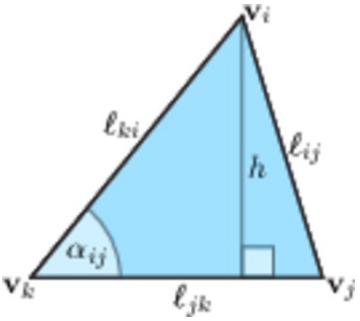


Figure 4.27: *Left to right: an image with color scribbles is fit with a mesh and lifted into $\mathbb{R}^3$ according to intensity values. Bounded Biharmonic Weights computed for each scribble (shown for body) are used to colorize the image. Compare to colorization by [Levin et al. 2004].*

channels then our surface lives in $\mathbb{R}^5$ or possibly higher. At first glance it may seem difficult to define the usual operators in higher dimensions. However, the building blocks of standard operators, e.g. the discrete Laplace-Beltrami operator, are the triangle areas and cotangents of each triangle corner angle [Meyer et al. 2003], and these may be defined *intrinsically* based solely on triangle edge lengths (rather than using cross products as one might in $\mathbb{R}^3$).



Consider a triangle with vertices $\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k \in \mathbb{R}^d$. The triangle area A_{ijk} is defined intrinisically by [Heron 60]:

$$A_{ijk} = \sqrt{r(r-\ell_{ij})(r-\ell_{jk})(r-\ell_{ki})}, \tag{4.39}$$

where l_{ij} is the length of the edge between $\mathbf{v}_i$ and $\mathbf{v}_j$, and r is the semi-perimeter $\frac{1}{2}(\ell_{ij}+\ell_{jk}+\ell_{ki})$.

We may similarly define the cotangent of the angle opposite each edge. First we can derive the cosine and sine. Recall the law of cosines:

$$\ell_{ij}^2 = \ell_{jk}^2 + \ell_{ki}^2 - 2\ell_{jk}\ell_{ki}\cos\alpha_{ij} \;\rightarrow\; \cos\alpha_{ij} = \frac{-\ell_{ij}^2 + \ell_{jk}^2 + \ell_{ki}^2}{2\ell_{jk}\ell_{ki}}. \tag{4.40}$$

For sine, we employ the familiar area formula treating the $\overline{\mathbf{v}_j\mathbf{v}_k}$ as base:

$$A_{ijk} = \frac{1}{2}\ell_{jk}\ell_{ki}\sin\alpha_{ij} \;\rightarrow\; \sin\alpha_{ij} = \frac{2A_{ijk}}{\ell_{jk}\ell_{ki}}.$$

Finally putting these together we have:

$$\cot\alpha_{ij} = \frac{\cos\alpha_{ij}}{\sin\alpha_{ij}} = \frac{-\ell_{ij}^2 + \ell_{jk}^2 + \ell_{ki}^2}{2\ell_{jk}\ell_{ki}}\frac{\ell_{jk}\ell_{ki}}{2A_{ijk}} = \frac{-\ell_{ij}^2 + \ell_{jk}^2 + \ell_{ki}^2}{4A_{ijk}}.$$

Note that a similar intrinsic derivation is given in Equations (7) and (13) of [Meyer et al. 2003] and may be extracted from our derivation of the planar Laplacian (Section 2.1.2). Derivation of an equivalent intrinsic formulation of the discrete cotangent Laplacian for 3-manifolds lurks in Section 2.1.2, but is left as an exercise to the reader.

With cotangents in hand, we may employ operators like the discrete Laplacian to solve Poisson equations over an image surface. Consider the colorization problem. We wish to propogate the colors of sparse user scribbles to the rest of the image in a localized, smooth and content-aware manner. [Levin et al. 2004] pose this as a discrete Poisson equation. Despite their published formulas, private discussion one of the authors and their published code agree that they solve a second- (not fourth-) order system, whose system matrix is a content-dependent, nonsymmetric Laplacian. Though similar to that of [Lischinski et al. 2006], it is not positive semi-definite. Thus its solutions do not correspond to minimizers of a modified Dirichlet, (or Laplacian) energy.[6]

The resulting system of the colorization problem is linear: the final colors are just a weighted linear combination of each scribble's color. In this light, we may acknowledge the connection between the colorization problem and the handle-based linear shape deformation problem,

[6]This implies that [Wang et al. 2011] and possibly others are also solving a second-order PDE, but (accidentally) writing otherwise.

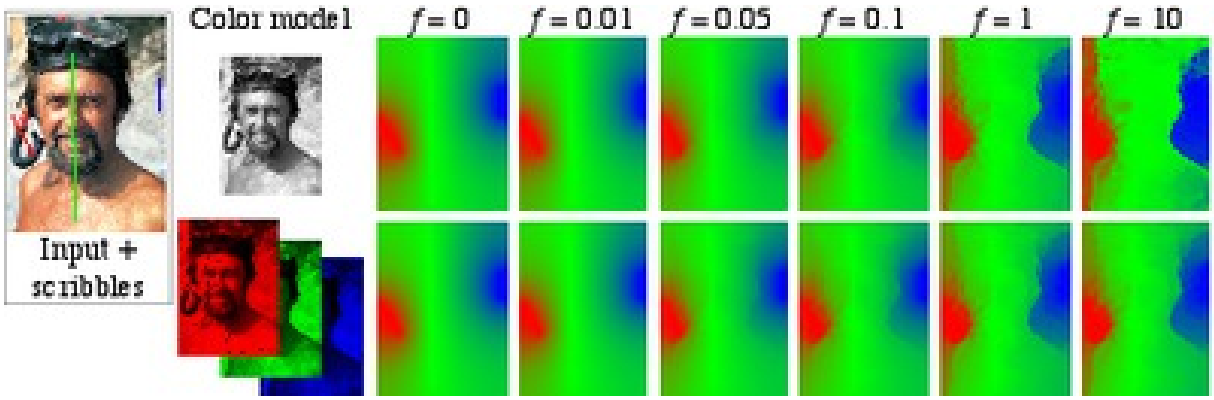


Figure 4.28: *The original input image with colored user scribbles (left inset) defines an* image surface *in* $\mathbb{R}^3$ *using intensity values (top row), or in* $\mathbb{R}^5$ *using RGB values (bottom row). Varying the scale factors (columns) on the appended coordinates affects the discrete Laplace equation used to produce these soft-segmentations.*

where correspondence weights are computed for each handle and each point on the surface. If we replace the Laplacian used by [Levin et al. 2004] with the cotangent Laplacian of the image surface, the resulting system would then be analagous to the Harmonic Coordinates of [Joshi et al. 2007]. The previously described bounded biharmonic weights show numerous advantages over Harmonic Coordinates in the realm of deformation. Because bounded biharmonic weights also optimize an energy involving the cotangent Laplacian, we may similarly compute them on image surfaces and use them for colorization (see Figure 4.27).

One missing element is the choice of scale relationship between the Cartesian coordinates of a pixel and its appended color coordinates. For a pixel i at location $x_i, y_i \in [0, \max(w, h)]$ where w and h are the width and height measured in pixels, let its color values be $I_i, R_i, G_i, B_i, \ldots \in [0, 1]$. Then we parameterize the *amount* of content-adaptiveness desired in our operators by scaling each color coordinate by a constant factor when we lift the image surface. For a pixel i in a grayscale image, the embedded coordinates are $(x_i, y_i, f_I I_i)$ and for an RGB image $(x_i, y_i, f_R R_i, f_G G_i, f_B B_i)$. The effect of tweaking these parameters is shown in Figure 4.28. For this example we consider Harmonic Coordinates defined on the image surface as a soft segmentation for three user scribbles. The top row shows the segmentation in pseudo-color for various scale factors, considering only the intensity channel. The bottom row considers the RGB channels (with $f_R = f_G = f_B$).

This idea was successfully employed by [Yücer et al. 2012] for image registration. In future work, we would like to explore different color models and the ideal weighting of each color coordinate. It would also be interesting whether a similar embedding can be defined for cyclical color spaces like HSV.

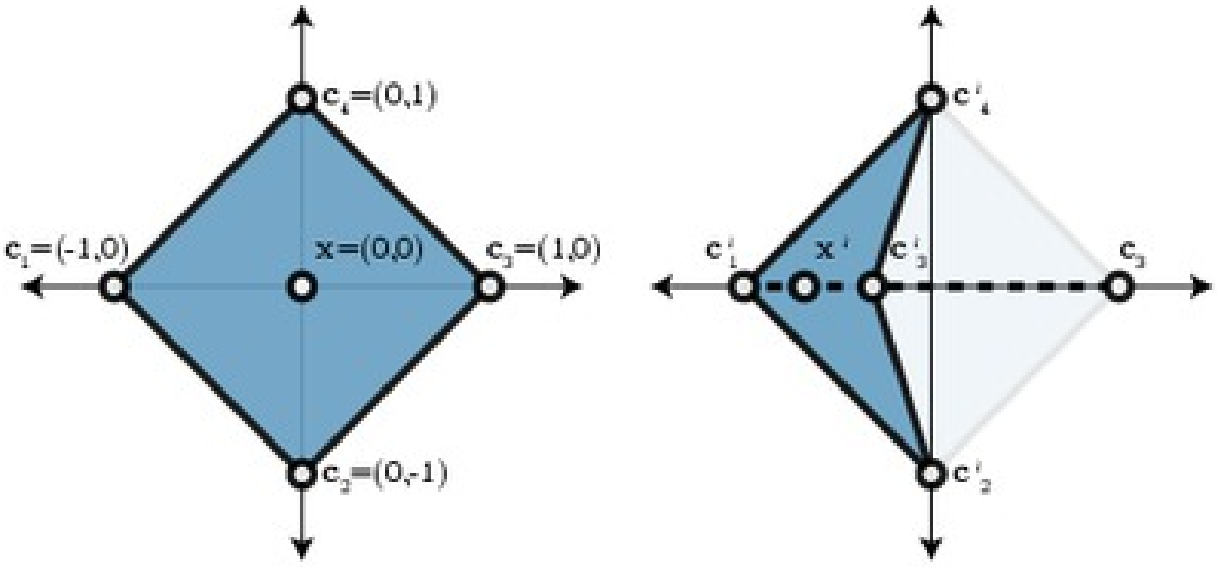


Figure 4.29: *Consider the center of a square. Moving an arbitrary corner toward its opposite corner, we may show that a mapping produced by barycentric coordinates is either non-bijective or the coordinate corresponding to this corner is $1/2$. Because the choice of corner was arbitrary and there are four corners, this contradicts the partition of unity property.*

4.9 Appendix: Bijective mappings with barycentric coordinates — a counterexample

Many recent works attempt to generalize barycentric coordinates to arbitrary polygons. We construct a counterexample proving that no such generalization will produce purely bijective mappings in the plane provided the coordinates meet the Lagrange, reproduction, and partition of unity properties. The proof concerns generalized barycentric coordinates in a square, but trivially generalizes to arbitrary polygons with degree greater than three.

Barycentric coordinates in a triangle boast a list of favorable properties making them useful in a number of important tasks across fields (e.g. scattered data interpolation). A recent resurgence of work attempts to generalize barycentric coordinates to arbitrary polygons. Typically, these works maintain the *basic* properties of coordinates (Lagrange, reproduction, and partition of unity) and vary in the degree to which they support other properties: closed form expression [Floater 2003, Manson and Schaefer 2010], smoothness [Joshi et al. 2007], positivity [Lipman et al. 2007], and so on. However, triangular barycentric coordinates possess an elusive property, so far unobtained by generalizations: the ability to produce bijective mappings in the plane. When used to define a planar map, barycentric coordinates produce an affine mapping which is trivially bijective so long as the map does not degenerate (the original triangle should map to a non-degenerate triangle). In the rest of this document we show by counterexample how such bijective mappings are unobtainable by any arbitrary generalized barycentric coordinates in a square. The counterexample trivially extends to any polygon of degree greater than three.

Consider a square S with corners $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4\}$ at respective Cartesian coordinates $\{(-1,0),(0,-1),(1,0),(0,1)\}$. Define a *barycentric coordinates mapping* inside this square

as a map:

$$\mathcal{M}(\mathbf{x}) : S \to \mathbb{R}^2 = \sum_{i=1}^{4} w_i(\mathbf{x})\mathbf{c}'_i, \tag{4.41}$$

where $\mathbf{c}'_i$ are the *new* or *deformed* positions of $\mathbf{c}_i$ and w_i are scalar barycentric coordinate functions defined for each point $\mathbf{x}$ in and on the square, obeying the following properties:

- **Lagrange:** $w_i(\mathbf{c}_j) = \delta_{ij}$ and thus $\mathcal{M}(\mathbf{c}_i) = \mathbf{c}'_i$,
- **Reproduction:** $\sum_{i=1}^{4} w_i(\mathbf{x})\mathbf{c}_i = \mathbf{x}$, and
- **Partition of unity:** $\sum_{i=1}^{4} w_i(\mathbf{x}) = 1$.

We call an arrangement of $\mathbf{c}'_i$ *non-degenerate* if the quadrilateral $\{\mathbf{c}'_1, \mathbf{c}'_2, \mathbf{c}'_3, \mathbf{c}'_4\}$ remains simple.

Theorem: For any such functions w_i, there exists a non-degenerate arrangement of $\mathbf{c}'_i$ such that the mapping is not injective. That is, there always exists some $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{x} \neq \mathbf{y}$ but $\mathcal{M}(\mathbf{x}) = \mathcal{M}(\mathbf{y})$.

Sketch: The idea of the proof is that as we bring $\mathbf{c}'_3$ toward $\mathbf{c}_1$, one of three things will happen:

1. $\mathcal{M}(\mathbf{0})$ will reach $\mathbf{c}_1$ before $\mathbf{c}'_3$ does,
2. $\mathbf{c}'_3$ will overrun $\mathcal{M}(\mathbf{0})$ before $\mathbf{c}'_3$ reaches $\mathbf{c}_1$, or
3. $w_3(\mathbf{0}) = 1/2$, contradicting partition of unity.

Proof: Applying the reproduction property, we may rewrite the mapping formula as:

$$\mathcal{M}(\mathbf{x}) = \sum_{i=1}^{4} w_i(\mathbf{x})(\mathbf{c}'_i - \mathbf{c}_i + \mathbf{c}_i) \tag{4.42}$$

$$= \mathbf{x} + \sum_{i=1}^{4} w_i(\mathbf{x})(\mathbf{c}'_i - \mathbf{c}_i). \tag{4.43}$$

Consider the mapping of the origin $\mathcal{M}(\mathbf{0})$ as we move $\mathbf{c}'_3$ along the x-axis from $\mathbf{c}_3 = (1, 0)$ toward $\mathbf{c}_1 = (-1, 0)$ while keeping $\mathbf{c}'_i = \mathbf{c}_i$ for $i \neq 3$. Until $\mathbf{c}'_3$ finally reaches $\mathbf{c}_1$ then the mapping is non-degenerate. Immediately we may write

$$\mathcal{M}(\mathbf{0}) = \sum_{i=1}^{4} w_i(\mathbf{0})(\mathbf{c}'_i - \mathbf{c}_i) \tag{4.44}$$

$$= w_3(\mathbf{0})(\mathbf{c}'_3 - \mathbf{c}_3). \tag{4.45}$$

By noticing that $\mathbf{c}_3 = -\mathbf{c}_1$ we may write

$$\mathcal{M}(\mathbf{0}) = w_3(\mathbf{0})(\mathbf{c}'_3 + \mathbf{c}_1). \tag{4.46}$$

The remainder of the proof will deal only with the x-coordinate of the relevant points. We will use a nonbold font to refer to the scalar x-coordinate, so that c'_3 is the x-coordinate of $\mathbf{c}'_3$.

Assume for any choice of c'_3 between c_1 and c_3 we have an injective map. We now draw a contradiction for every possible weight $w_3(\mathbf{0})$.

Case 1: Consider the case when $w_3(\mathbf{0}) > 1/2$ and let $\mathcal{M}(\mathbf{0}) = \mathbf{c}_1$:

$$w_3(\mathbf{0})(\mathbf{c}'_3 + \mathbf{c}_1) = \mathbf{c}_1 \tag{4.47}$$

Because $c'_3 + c_1 \leq 0$ and $1/2 < w_3(\mathbf{0})$ we may solve for c'_3:

$$\frac{1}{2}(c'_3 + c_1) > w_3(\mathbf{0})(c'_3 + c_1) = c_1 \tag{4.48}$$

$$c'_3 + c_1 > 2c_1 \tag{4.49}$$

$$c'_3 > c_1 \tag{4.50}$$

which means that $\mathcal{M}(\mathbf{0}) = \mathbf{c}_1 = \mathcal{M}(\mathbf{c}_1)$ before degeneracy. So if we want an injective mapping then $w_3(\mathbf{0}) \leq 1/2$.

Case 2: Consider $w_3(\mathbf{0}) < 1/2$, and let $\mathcal{M}(\mathbf{0}) = \mathbf{c}'_3$.

$$w_3(\mathbf{0})(c'_3 + c_1) = c'_3 \tag{4.51}$$

and now since $w_3(\mathbf{0}) < 1/2$ we solve again for c'_3:

$$c_1 < c'_3 \tag{4.52}$$

So once again, $\mathcal{M}(\mathbf{0}) = \mathbf{c}'_3 = \mathcal{M}(\mathbf{c}_3)$ before degeneracy. Together with Case 1, if we want an injective mapping, then $w_3(\mathbf{0}) = 1/2$.

Case 3: We now know that $w_3(\mathbf{0}) = 1/2$, but by rotational symmetry of our problem we may repeat our logic above showing that $w_i(\mathbf{0}) = 1/2$ for $i = 1, 2, 4$. Thus $\sum_{i=1}^{4} w_i(\mathbf{0}) = 2$ which is a contradiction to the partition of unity property. ■

Extending this counter example to other polygons simply involves considering a map from the original polygon to the given square (e.g. by mapping four corners to the square's corners and the remaining corners to somewhere along the respective sides). Then one may follow the same logic above.

This counterexample shows that no *real-valued* generalized barycentric coordinates induce mappings that are always bijective. It would be interesting to consider whether so-called *complex* barycentric coordinates [Weber et al. 2009] could provide such mappings. These coordinates often relax the Lagrange property so it is not clear to what extent these may be considered "coordinates". Nonetheless, pursuit of an analogous counterexample is interesting and, we hypothesize, fruitful[7].

[7] I am grateful to Kai Hormann and Ofir Weber for illuminating discussions which lead to developing this counterexample.

5

Smooth shape-aware functions with controlled extrema

Functions that optimize Laplacian-based energies have become popular in geometry processing, e.g. for shape deformation, smoothing, multiscale kernel construction and interpolation. Minimizers of Dirichlet energies, or solutions to Laplace equations, are harmonic functions that enjoy the maximum principle, ensuring no spurious local extrema in the interior of the solved domain occur. However, these functions are only C^0 at constrained values. For this reason, many applications optimize higher-order Laplacian energies, resulting in biharmonic or triharmonic functions. Their minimizers exhibit increasing orders of continuity but lose the maximum principle and show oscillations. In this work, we identify characteristic artifacts caused by spurious local extrema, and provide a framework for minimizing quadratic energies on manifolds while constraining the solution to obey the maximum principle in the solved region. Our framework allows the user to specify locations and values of desired local maxima and minima, while preventing any other local extrema. We demonstrate our method on the smoothness energies corresponding to popular polyharmonic functions and show its usefulness for fast handle-based shape deformation, controllable color diffusion, and topologically-constrained data smoothing.

5.1 Introduction

Smooth, shape-aware functions have become a cornerstone of geometry processing. They are often obtained by minimizing discrete differential energies, and are used in a wide range of applications, e.g. detail-preserving shape deformation, data interpolation on manifolds, multiscale shape analysis, mesh segmentation and even image processing. Some applications require

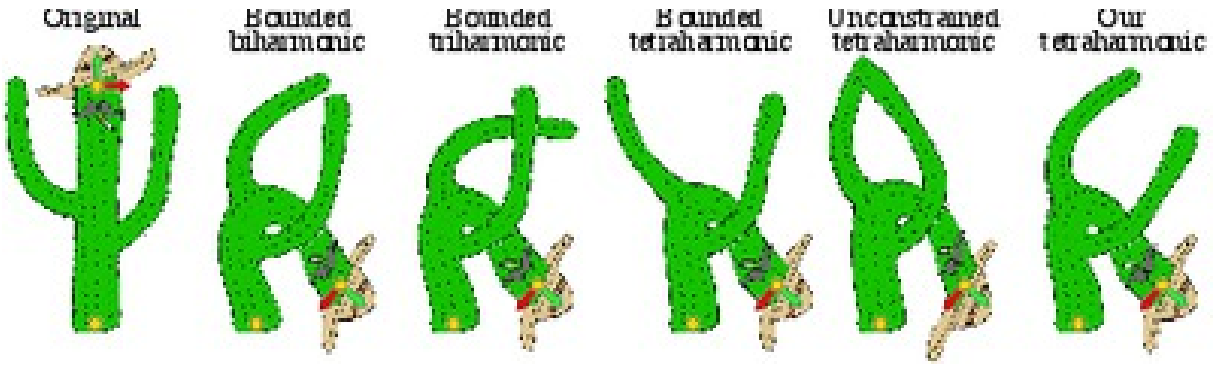


Figure 5.1: *Shape deformation: Recent works emphasize the importance of bounded control, but simply adding constant bounds to shape-aware smoothness energies of increasing order encourages more and more oscillation. Our framework efficiently optimizes such high-order energies as $\int_{\mathcal{M}} \|\nabla^4 f\|^2 dA$ while ensuring against spurious local extrema.*

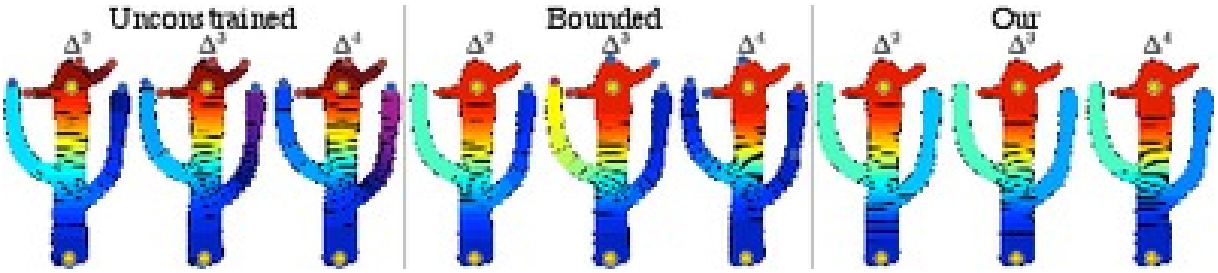


Figure 5.2: *Various weight functions for the control point in the head of the cactus used in Figure 5.1, with highlighted local minima (blue dots) and maxima (red dots). Our framework prohibits local extrema during optimization, wrangling the oscillations.*

functions with high-order smoothness or sophisticated control of the boundary conditions, in which case energies involving high-order differential quantities are used.

Energies associated with polyharmonic partial differential equations are particularly popular, since they are relatively easy to define on meshes using discrete gradient and Laplace operators [Meyer et al. 2003], whose properties are well-studied [Wardetzky et al. 2007b]. Solutions of polyharmonic equations exhibit increasing smoothness at fixed values, but unfortunately also show increasingly wild oscillations that are difficult or impossible to control with boundary values alone. The previous Chapter 4 investigated weight functions for propagating handle transformations in real-time shape deformation; it was shown that minimizing the Laplacian energy with constant bounds produces functions that capture the good qualities of biharmonic functions while avoiding negative values that lead to unintuitive deformation results. The bounds effectively dampen the natural oscillations of the biharmonic solution while maintaining smoothness.

Bounds by definition prevent function values outside the intuitive range, but do not prevent all unintuitive oscillations. In Figure 5.1, the *Cactus* is deformed using bounded biharmonic functions, but the top of his left arm *unintuitively* stays put when the control point in the head is moved. This is due to a local minimum in the associated weight function (see Figure 5.2). Increasing the order of the smoothness energy and keeping the constant bounds only makes the oscillations worse to the detriment of the final deformation. We show how the local extrema introduced by these oscillations occur frequently even if constant bounds are in place.

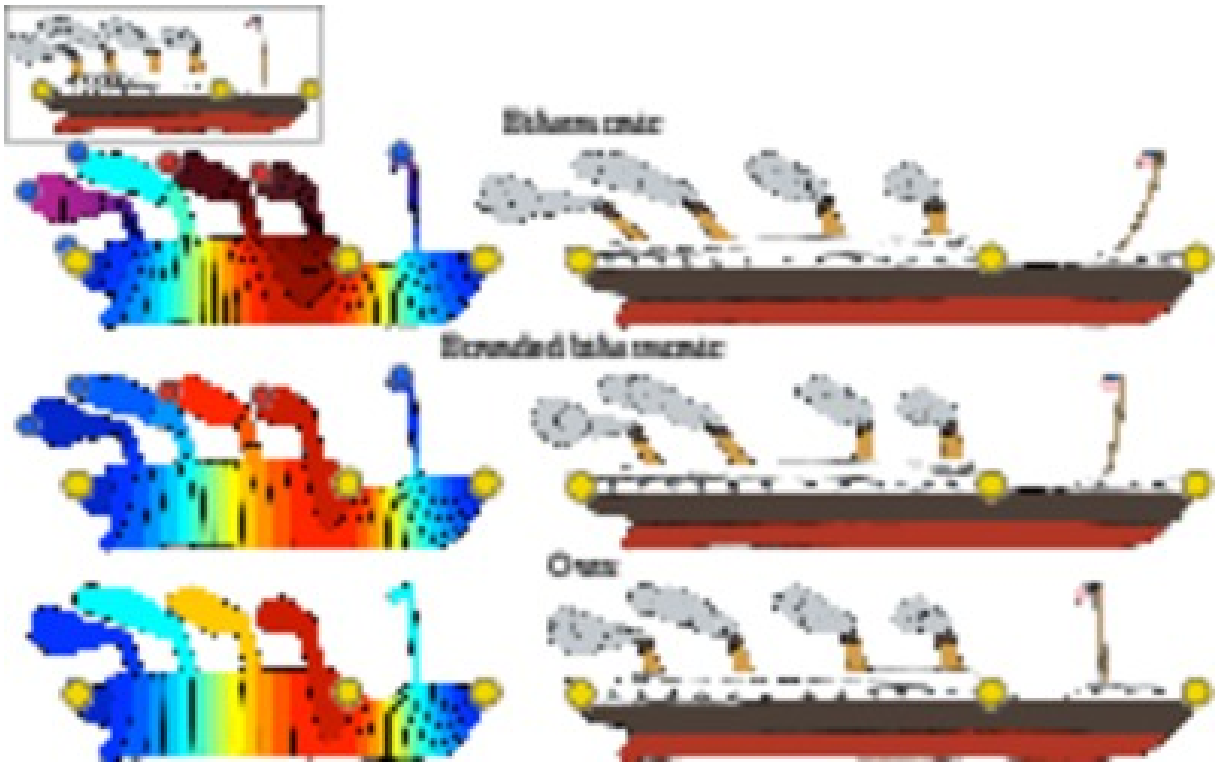


Figure 5.3: *The* Titanic *(inset) is stretched with 3 control points. Biharmonic and bounded biharmonic weights introduce extrema in the chimneys and flagpole, causing unintuitive response. Our solution (right) is smooth and intuitively deforms the appendages as if rigidly attached to the shape. The weight function for the middle control point is shown for each (left).*

In this chapter, we formally define the ideal problem we would like to solve: minimize high-order, shape-aware smoothness energies with guarantees on the locations and values of local extrema. The resulting nonlinear optimization problem turns out to be impractically difficult due to nonlinear, non-convex inequality constraints. Accordingly, we provide a framework to simplify the constraints in a way that not only ensures we find a feasible solution, but converts the problem into a computationally tractable one.

Our robust optimization allows us to consider smoothness energies associated with higher-order PDEs. We demonstrate the usefulness of guarantees on the absence of new extrema in the context of weight-based shape deformation, color diffusion and data smoothing.

5.2 Background

We survey a representative selection of works that utilize smooth, shape-aware functions. Especially relevant to us are works that comment on the oscillatory nature of polyharmonic functions and/or impose topological constraints.

Shape deformation. Smooth, shape-aware influence functions are profusely used for shape deformation. A basic way of deforming the geometry $\mathbf{x}$ of a 2D or 3D shape $\mathcal{M}$ is by combining the propagated influences of affine transformations T_j provided at user-controlled deformation

handles (LBS):

$$\mathbf{x}'_i = \sum_{j=1}^{H} f_j(\mathbf{x}_i)\, T_j\, \mathbf{x}_i, \tag{5.1}$$

where $\mathbf{x}_i$ are vertices of the mesh $\mathcal{M}$. An influence function f_j should attain maximum value of 1 in the shape region that is most affected by the handle j, and decay towards 0 away from that region (reaching exactly 0 on points fully associated with other handles). If f_j has negative values, the deformed shape will unintuitively move in the "opposite direction" to the prescribed T_j. Thus, f_j should be *bounded* within $[0, 1]$. Moreover, as we illustrate in this work, if the influence functions have additional local extrema (besides at the constrained handle regions), the deformation behavior is unintuitive as well, causing parts of the shape to "lag behind" or move too fast relative to neighboring parts. We colloquially refer to such behavior as *non-monotonic*.

Equation (5.1) is common in real-time skinning deformations [Magnenat-Thalmann et al. 1988, Kavan et al. 2008], its execution is parallel and extremely fast on the GPU. Several works proposed automatic computation of skinning weights f_j. [Weber et al. 2007] use harmonic functions that are provably monotonic and bounded but have only C^0 smoothness near constrained boundary. [Baran and Popović 2007, Wareham and Lasenby 2008] solve a Poisson equation whose right-hand side is not guaranteed to be monotonic and thus might produce non-monotonic weight functions. The bounded biharmonic weights (BBW) of the previous chapter use biharmonic functions, which are smooth at the handles, and constrain them to be bounded within $[0, 1]$. Although in [Jacobson et al. 2011] we claim to observe a lack of spurious maxima in our weights, we now show a number of examples where this is not the case: especially in shapes with appendages (e.g. Figures 5.1, 5.3).

As discussed in Section 4.7, linear variational surface-based deformation methods [Botsch and Sorkine 2008] can be expressed in form similar (but not equivalent) to (5.1)[1] When all mesh vertices belonging to one handle are assigned the same affine transformation, the influence functions f_j are then analogous to the columns of the inverted system matrix stemming from the variational optimization (referred to as "bases" in [Botsch and Kobbelt 2004, Sorkine et al. 2005]). Specifically, [Botsch and Kobbelt 2004, Sorkine et al. 2005] and others define families of bases that are the solutions of polyharmonic PDEs $\Delta^k f = 0$, providing C^{k-1} continuity at the constrained handles. In Chapter 3, we refined this approach to allow direct control over boundary derivatives for the biharmonic and the triharmonic cases. Alternatively, triharmonic radial basis functions can be used [Botsch and Kobbelt 2005], replacing sparse large system solves by small and dense ones. In all these cases, the functions are often not monotonic nor even bounded, as discussed in Chapter 4.

Nonlinear deformation methods often employ model reduction based on skinning, such that the transformations T_j become the degrees of freedom in the nonlinear optimization. The weight functions play an important role in the design of the reduced model and greatly affect the final deformation quality. For example, [Huang et al. 2006] uses generalized barycentric coordinates and [Au et al. 2007] employs harmonic functions (using their isolines as reduced-space handles). In Chapter 7, we will consider a similar reduction using LBS.

[1] In our corresponding publication [Jacobson et al. 2012b], we erroneously claimed equivalence.

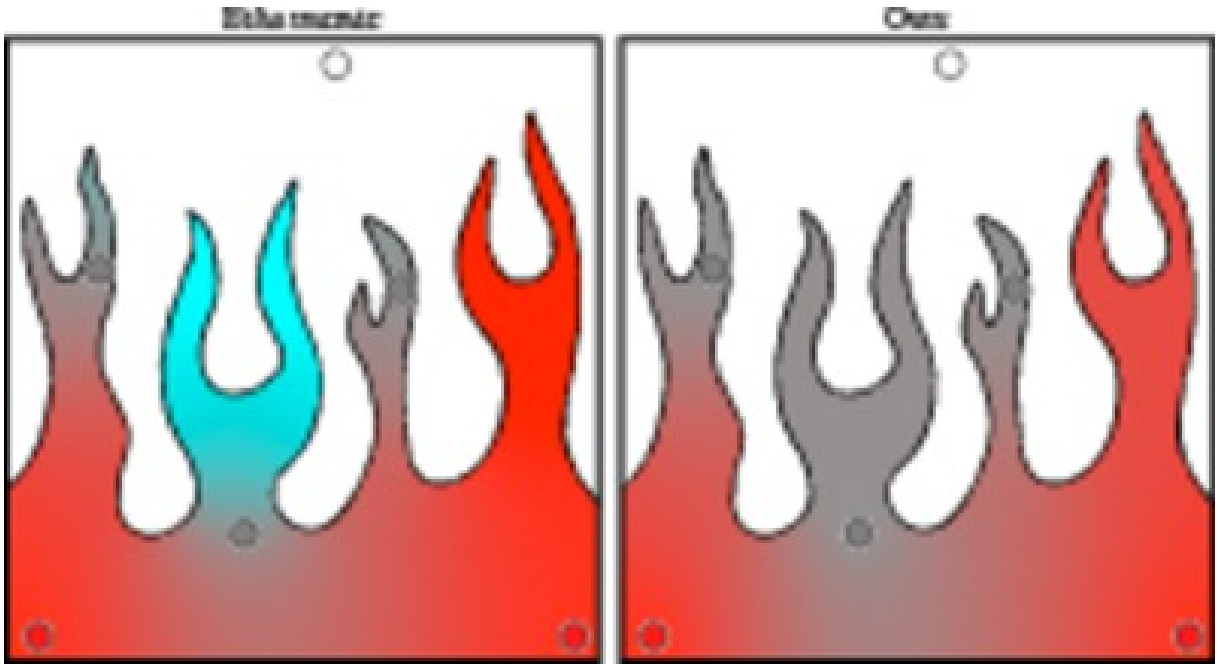


Figure 5.4: *Colors (specified at the small circles) are diffused using the biharmonic functions of [Finch et al. 2011], but these are unbounded and extrapolate colors not present in the constraints. Our interpolation explicitly prohibits local extrema and implicitly imposes bounds (cf. Figure 3 in [Finch et al. 2011]).*

When the control handles are vertices of a cage mesh enclosing the shape to be deformed, and the transformations T_j are just translations, the weight functions f_j are called generalized barycentric coordinates (see e.g. [Ju et al. 2005, Joshi et al. 2007, Hormann and Sukumar 2008, Manson and Schaefer 2010]). Significant attention has been devoted to the boundedness and locality problems [Lipman et al. 2007, Joshi et al. 2007], but we argue the same problems of unintuitive control arise with local extrema. While simple filters may be used to induce boundedness [Langer and Seidel 2008], such filters will at best downplay spurious local extrema, but will not in general remove them.

A large class of weight functions are constructed to decay with the (geodesic) distance from their handle [Schaefer et al. 2006, Zhu and Gortler 2007]. Deformations with such functions suffer from the "fall-off" effect, characterized by extraneous minima away from the handles (see Figures 4.7 & 6.11). These functions are unable to form large regions where one handle's weight function is significantly dominant. These effective plateaus in weight functions are necessary for naturally-looking deformations. In contrast, our method allows dominant regions and we demonstrate that through our choice of smoothness energy we may encourage such behavior, leading to intuitive deformation control.

Boundary value interpolation. Replacing deformed vertex positions $T_j\mathbf{x}$ in Equation (5.1) with general values, we obtain an interpolation problem, which has numerous uses in graphics:

$$\mathbf{v}(\mathbf{x}) = \sum_{j=1}^{H} f_j(\mathbf{x})\, \mathbf{v}_j \,. \tag{5.2}$$

Smoothness and monotonicity of the interpolation basis functions f_j remains impor-

tant. For example, if $\mathbf{v}_j$'s are RGB colors, one obtains color diffusion, useful in image colorization [Levin et al. 2004], seamless cloning [Pérez et al. 2003] and vector graphics design [Orzan et al. 2008]. The mentioned approaches employ harmonic functions f_j; [Georgiev 2004] cites possible use of triharmonic and tetraharmonic[2] functions for seamless image cloning, but settles on biharmonic functions. Finch et al. [2011] define biharmonic Diffusion Curves to enable diverse control of the color gradients and in particular smoothness at the provided user constraints. Their biharmonic functions can be negative and have prevalent local extrema, leading to unexpected results (see Figure 5.4). Their proposal to use hard clamping when final color values exceed the displayable range hurts smoothness and does not necessarily avoid local extrema, which are present even when imposing explicit bounds (see Figure 5.10).

Data smoothing. Scalar data smoothing is often required for subsequent processing, effective visualization and analysis. Polyharmonic RBFs are employed as low-pass filters for smoothing and reconstruction of scattered data, such as range images [Carr et al. 2001, Carr et al. 2003]; however, no explicit control over the resulting topology, e.g. guarantees of monotonicity, locations and values of extrema, is provided. Carr et al. [2004] work with the contour tree of the input data set, which enables them to remove small topological features. Gingold and Zorin [2006] prevent the formation of new topological features by controlling the data isocontours while using common iterative filtering methods, such as Laplacian smoothing or anisotropic diffusion.

The two most related approaches to our method in the data smoothing context are [Bremer et al. 2004] and [Weinkauf et al. 2010]. Bremer et al. iteratively simplify the Morse-Smale complex of the data and produce a corresponding scalar function after each cancellation step by Laplacian smoothing of the data within each Morse-Smale cell, until no interior critical points are left. Their iterative procedure is time-consuming and produces only C^0 continuity along separatrices. Weinkauf et al. first perform persistence-based simplification of the input Morse-Smale complex, and then fit a C^1 function that adheres to the complex by constrained minimization of a weighted sum of the Laplacian energy and a data term. They optimize each cell subject to nonlinear monotonicity constraints, extracted from a harmonic function computed in the cell. Our use of the maximum principle of harmonic functions is conceptually similar, but we devise a sparse set of linear inequality constraints and convert our energy optimization to a conic programming problem. As a result, our optimization is roughly $1000\times$ faster (see Section 5.4). Both methods [Bremer et al. 2004] and [Weinkauf et al. 2010] fully constrain the entire Morse-Smale complex, which may be beneficial for visualization, but could be too restrictive in cases where the precise locations or values of saddle points and separatrices are not relevant for the application. Our method constrains locations and values of the extrema, the locations of the saddles, and the combinatorial connectivity between these points in the Morse-Smale complex. The values of saddles and the locations of separatrices may be enforced in our framework but we find it useful to leave these free, allowing for a wider range of feasible solutions.

Mesh and image processing. Smooth, shape-aware (and in particular polyharmonic) functions find many additional uses, such as mesh segmentation [Zheng and Tai 2010], design of

[2] Functions satisfying the eighth-order equation $\Delta^4 f = 0$ have been previously referred to as *quatraharmonic*, *quadraharmonic*, *quadriharmonic* or *quadharmonic*, but *tetraharmonic* seems to be the prevailing terminology. This agrees with the Greek roots of *harmonic*, but contradicts the Latin numerical prefix of the well established *biharmonic*.

fair Morse functions [Ni et al. 2004] or multiscale kernels [Rustamov 2011]. Chen et al. [2011] used topological constraints to control the number of connected component in an image segmentation. Our framework is general and applicable in any such contexts.

5.3 Method

Let our domain be the triangle mesh $\mathcal{M}$ and let $\mathcal{N}(i)$ denote the 1-ring neighbors of vertex i. Our goal is to find a piecewise linear function $f : \mathcal{M} \to \mathbb{R}$, which interpolates values at specified minima locations $\mathbf{p}_i\,, i \in \mathcal{K}_{\text{min}}$ and maxima locations $\mathbf{p}_i\,, i \in \mathcal{K}_{\text{max}}$, with corresponding fixed values $g_i \in \mathbb{R}$. We also want f to be *monotonic*, i.e., no other extrema in $\mathcal{M}$.

Many functions fulfill those conditions. As often done in data interpolation, we introduce an energy functional $E(f)$ which measures the quality of f for a given application. Laplacian-based energies are often employed as a smoothness regularization term (e.g. [Botsch and Kobbelt 2004]) and are of the form[3]:

$$E_{L^k}(f) = \int_{\mathcal{M}} \|\nabla^k f\|^2 dA \text{ for } k = 2, 3, \ldots \tag{5.3}$$

Details of discretization may be found in Section 2.1 and Chapter 3 or [Botsch et al. 2010].

For applications like data smoothing, we may also introduce a data energy term E_D. In the simplest form, E_D measures in a least-squares sense the deviation from some data function $h : \mathcal{M} \to \mathbb{R}$:

$$E_D(f) = \sum_{i \in \mathcal{M}} \|f_i - h_i\|^2. \tag{5.4}$$

In general, we consider any energy functional E, but typically we are concerned with combinations of a smoothness term and possibly a data term:

$$E(f) = \gamma_L\, E_L(f) + \gamma_D\, E_D(f). \tag{5.5}$$

where γ_L and γ_D balance the influences of the energies.

[3] We write $\|\cdot\|$ to indicate the Euclidean norm operating on vectors if k is odd and scalars otherwise.

5.3.1 Ideal optimization

We may formulate the ideal problem as an energy minimization with nonlinear, non-convex[4], non-differentiable inequality constraints:

$$\arg\min_{f} \; E(f) \tag{5.6}$$

$$\text{subject to:} \quad f_i = g_i \qquad \forall i \in \mathcal{K}_{\min} \cup \mathcal{K}_{\max}, \tag{5.7}$$

$$f_j > f_i \qquad \forall j \in \mathcal{N}(i),\, \forall i \in \mathcal{K}_{\min}, \tag{5.8}$$

$$f_j < f_i \qquad \forall j \in \mathcal{N}(i),\, \forall i \in \mathcal{K}_{\max}, \tag{5.9}$$

$$f_i > \min_{j \in \mathcal{N}(i)} f_j \qquad \forall i \notin \mathcal{K}_{\min} \cup \mathcal{K}_{\max}, \tag{5.10}$$

$$f_i < \max_{j \in \mathcal{N}(i)} f_j \qquad \forall i \notin \mathcal{K}_{\min} \cup \mathcal{K}_{\max}. \tag{5.11}$$

The constant equality constraints (5.7) simply enforce that the values of the known extrema are interpolated. The linear inequality constraints (5.8) and (5.9) ensure that the prescribed extremal points are local minima and maxima, respectively. The nonlinear inequality constraints (5.10) and (5.11) enforce that all unknown values are greater than their minimum neighbor and smaller than their maximum neighbor.

We assume that the given set of extrema does not contradict the Morse inequalities. For example, a non-constant function on a topological disk must have at least one minimum. We also assume that prescribed extrema are not immediate neighbors and there exists one minimum smaller than all maxima and vice-versa.

Given a quadratic energy E, this optimization problem would be practical to solve if not for the nonlinear inequality constraints. The other constraints are linear and at worst produce a quadratic programming problem. Trying to optimize the ideal problem directly with commercial "black-box" nonlinear optimization software [MATLAB 2012] shows discouraging convergence. Often feasible solutions are not found, and even if the solver does converge, the dismal performance renders this option useless for most applications. For example, on a 16-vertex mesh, the optimization takes 3 seconds to converge, and only does so when provided with a generously near-optimal and feasible initial guess.

Many nonlinear optimization methods allow a "constraint violation tolerance" parameter. However, our topological constraints are highly sensitive. Even the slightest violation can allow oscillations preferred by the unconstrained energy, producing potentially unbounded spurious hills and valleys.

5.3.2 Constraint simplification

In light of our inability to solve the ideal optimization problem in (5.6) in a reasonable amount of time, we propose a method for simplifying the nonlinear inequality constraints (5.10) and (5.11)

[4]It is straightforward to show that in general constraints (5.10 & 5.11) are not convex. Simply consider fixing all variables but a small subset. Vary the values in this subset linearly. We know the feasible region is not convex if the evaluation becomes feasible, then infeasible, then feasible again along this line.

Figure 5.5: *We illustrate with the space of all solutions that satisfy the user's constraints (white). Ideally we would consider all monotonic solutions directly (green), but this problem is impractical. Instead we find a monotonic representative u and optimize in the subspaces of solutions whose gradients match a sufficient subset of directions of $\nabla u/\|\nabla u\|$ (red and blue).*

into a larger set of *linear* inequality constraints. Suppose we have a *representative* function $u : \mathcal{M} \to \mathbb{R}$ which satisfies constraints (5.8)-(5.11). We replace the nonlinear inequality constraints (5.10) and (5.11) with linear inequality constraints requiring the *direction* (but not magnitude) of ∇f be aligned with $\nabla u/\|\nabla u\|$. In essence we enforce the *monotonicity* or, loosely, the topology of u onto our optimized solution f. As long as we choose u intelligently and construct our linear constraints from $\nabla u/\|\nabla u\|$ carefully, the optimization in (5.6) becomes convex and thus efficiently solvable, and finds an acceptable solution (see Figure 5.5).

Given a monotonic representative function u as described above, we reduce the optimization to:

$$\arg\min_{f} \; E(f) \tag{5.12}$$

$$\text{subject to:} \quad f_i = g_i \qquad \forall i \in \mathcal{K}_{\min} \cup \mathcal{K}_{\max}, \tag{5.13}$$

$$(f_i - f_j)(u_i - u_j) > 0 \qquad \forall (i,j) \in \mathcal{E}, \tag{5.14}$$

where $\mathcal{E}$ is a subset of the edges of the domain $\mathcal{M}$. The constraints (5.14) require that the direction of decrease across every edge of $\mathcal{E}$ in our optimal solution f should match that of the known function u. If $\mathcal{E}$ contains all edges, this is enforced everywhere and the choice of u greatly restricts the shape of f. This is useful if we have high confidence in the representativeness of u (e.g. when it is derived from existing input data). In other situations we only want to capture the monotonicity of u, hence we would like the smallest possible edge set $\mathcal{E}$. To guarantee that our constraints prohibit local extrema, $\mathcal{E}$ must include for each vertex $i \in \mathcal{M} \setminus (\mathcal{K}_{\min} \cup \mathcal{K}_{\max})$ *at least* one edge (i,j) where $u_j < u_i$ and another where $u_j > u_i$. This guarantees that each vertex value f_i in our solution will have one neighbor $f_j < f_i$ and another $f_j > f_i$.

5.3.3 Choice of representative function

We provide two methods for constructing a valid representative function u satisfying (5.8)-(5.11). The first method shows that such a function always exists, ensuring a feasible solution.

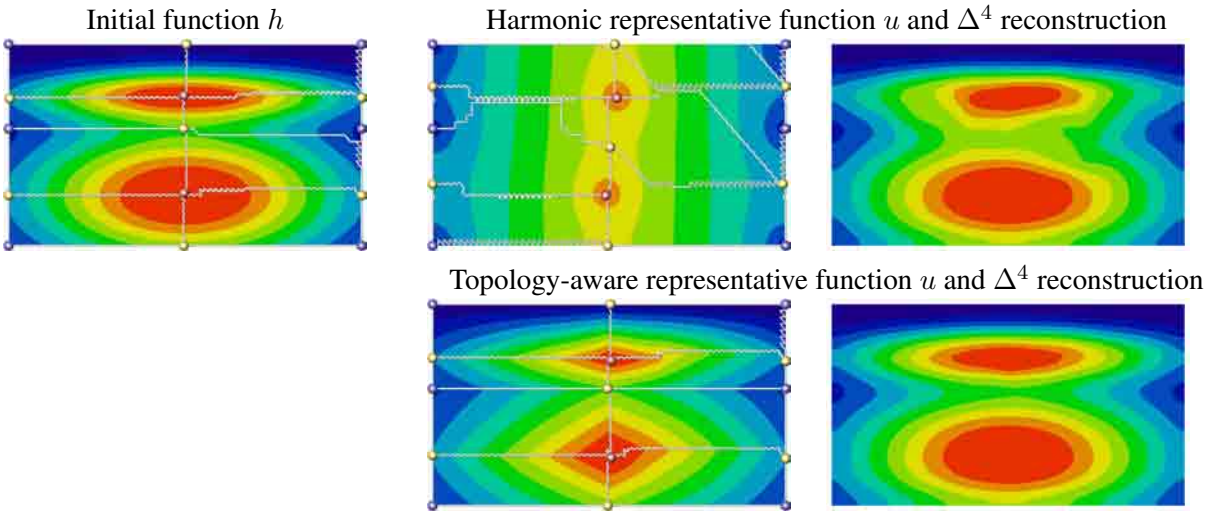


Figure 5.6: *Two different representative functions are compared for the application of smoothing an initially given function. The harmonic function (middle) is visualized together with its Morse-Smale complex, which is clearly different to the original topology. The constraints derived from this fight against the data term during the reconstruction, which leads to a poor result. The topology-aware representative function (right) leads to good reconstruction results, since it respects the original topology.*

The second method takes advantage of situations when initial data is present.

No initial data. Consider a situation where no data is available besides the domain $\mathcal{M}$ and the locations and values of extrema in $\mathcal{K}_{\text{max}}$ and $\mathcal{K}_{\text{min}}$. This is useful, for example, in the context of shape deformation and color interpolation. Taking advantage of the strong maximum principle of harmonic functions, we can always construct a valid representative function u by solving the following Dirichlet problem:

$$\arg\min_{u} \int_{\mathcal{M}} \|\nabla u\|^2 dA \tag{5.15}$$

$$\text{subject to: } u|_{\mathcal{K}_{\text{min}}} = 0, \tag{5.16}$$

$$u|_{\mathcal{K}_{\text{max}}} = 1. \tag{5.17}$$

Minimizers of the Dirichlet energy are harmonic functions. Their maximum principle guarantees that, when choosing the Dirichlet boundary conditions $u|_{\mathcal{K}_{\text{min}}} = 0$ and $u|_{\mathcal{K}_{\text{max}}} = 1$, the locations in $\mathcal{K}_{\text{min}}$ and $\mathcal{K}_{\text{max}}$ become minima and maxima, respectively. Thanks to the uniqueness of harmonic functions, u contains no other extrema inside $\mathcal{M}$. This fulfills the necessary conditions for u to be a valid representative function.

Initial data. In other situations, e.g. scalar field smoothing, our input will contain some initial data function h and the objective is to remove some of its extrema while keeping others. This is useful in the context of topologically-constrained smoothing [Weinkauf et al. 2010], where the Morse-Smale complex of h is simplified based on Forman's discrete Morse theory [Forman 1998] such that only the critical points above a user-defined persistence [Edelsbrunner et al. 2002] threshold remain. To reconstruct a smooth function f based on the remaining topology and as close as possible to the initial data function h, we set $\mathcal{K}_{\text{min}}$ and $\mathcal{K}_{\text{max}}$

	$\lvert\mathcal{M}\rvert$	k	Time/f_j	Total Time
Cactus	2403	2	0.1246	0.2493
Cactus	2403	3	1.3135	2.6271
Cactus	2403	4	0.2324	0.4649
Colored J	8229	2	0.2306	0.9225
Hummingbird	14636	3	86.393	259.18
Mouse	26294	2	6.1345	6.1345
Dino	28136	2	2.4564	7.3693
Combustor	29021	2	5.1707	5.1707
Beetle	38656	2	6.0263	6.0263

Table 5.1: *Statistics for various examples in this chapter. $\lvert\mathcal{M}\rvert$ is the number of triangles in the discretized domain,* k *is the order of the corresponding polyharmonic operator,* Time/f_j *is the average optimization time per function in seconds, and* Total Time *is the total optimization time.*

to contain the minima/maxima of the simplified Morse-Smale complex and construct the representative function u in the same fashion as [Weinkauf et al. 2010] builds its "preview" function: the critical points are fixed to their original values $u(\mathbf{p}_i) = h(\mathbf{p}_i)$, and all vertices on each separatrix are fixed to a linearly interpolated value between its end points, i.e., a saddle and an extremum. All fixed vertices serve as Dirichlet boundary conditions for solving $\Delta u = 0$ on the domain. The solution u is a harmonic function that has no interior critical points in the domain: since the boundary conditions are monotonic, we obtain a valid scalar field that obeys the prescribed topology.

Note that the solver of [Weinkauf et al. 2010] required that the values along the separatrices remain fixed. This mandated smoothing the locations and values of the separatrices before building another harmonic function and finally optimizing the interiors of the Morse cells. Our framework does not require fixing separatrix values and may optimize the entire domain at once.

Figure 5.6 compares this topology-aware representative function to the harmonic function created by (5.15)–(5.17). The latter places the central saddle point in the middle between the two peaks, thereby disregarding the size and shape of the peaks in the original function. In this example, this creates areas where the gradients of the harmonic and the original function are perpendicular to each other (visible in the isolines). This leads to a poor reconstruction. The topology-aware representative function is built from the original topology, leading to a favorable reconstruction.

Finally, with a valid representative function we may take a minimally sufficient edge set $\mathcal{E}$. For each vertex i in $\mathcal{M}$, we include the edges $\{i, j\}$ and $\{i, k\}$ where u_j and u_k are the smallest and greatest of the neighbors of i. To ensure the topological conditions of the user constraints are met, we add all edges incident on any i in $\mathcal{K}_{\text{min}}$ or $\mathcal{K}_{\text{max}}$.

5.3.4 Implementation

One could solve (5.12) with any sparse quadratic programming solver, but we saw major performance improvements ($\approx 100\times$) when converting our problem to conic programming and using MOSEK [Andersen and Andersen 2000], a sparse, conic programming solver. See Section 5.7

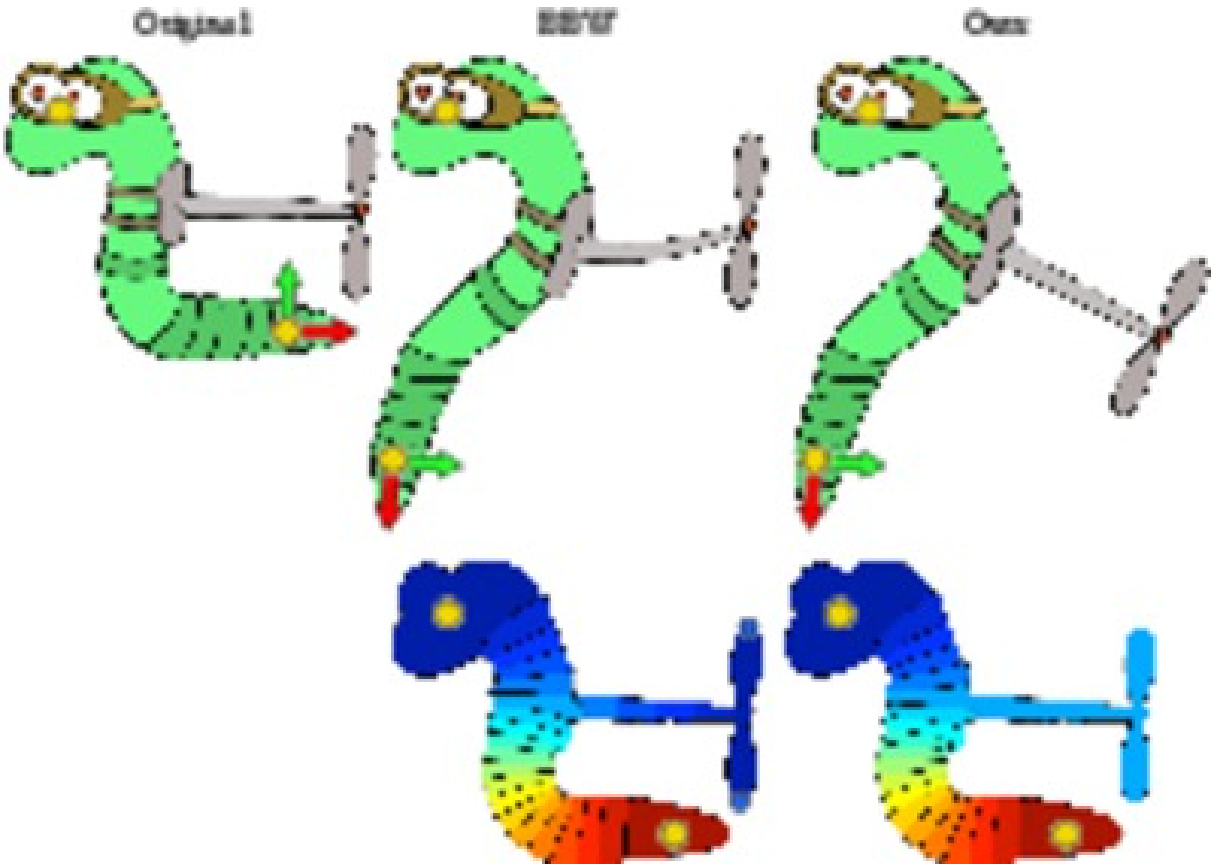


Figure 5.7: *The* Propeller Worm *in its rest pose (left) is deformed using bounded biharmonic weights (BBW, middle). A local maximum leaves its propeller behind when its tail is bent, also causing the rod to unintuitively squish. Our solution (right) attaches the propeller to the body, giving it near constant weights (visualized below).*

and Section 2.2.4 for details of this conversion. For 2D image deformation and color diffusion we use Triangle [Shewchuk 1996] to triangulate the interior of the shape outline.

When solving for shape-deformation or interpolation weight functions in (5.1) and (5.2), we may append additional linear equality constraints to ensure weights across handles sum to one for every vertex in $\mathcal{M}$. This would tether the optimizations of the weight functions together into one larger problem. However, we observe the same behavior of our weight functions as in the previous Chapter 4 (see Figure 4.10): dropping the partition of unity constraints and normalizing *post-hoc* has little effect on the final weights and thus also the deformation. Thus, in all our experiments, we compute each weight function independently and normalize *post facto*. We report the average times for computing each function in Table 5.1.

It is worth noting that this normalization technically invalidates any guarantees of monotonicity in the final functions, but we argue that the artifacts arising from non-monotonicity are involved with low-frequency oscillations present in the original functions before normalization. In any case, our contribution is a general framework which easily incorporates creating guaranteed monotonic functions with partition of unity constraints if so desired, just at additional precomputation costs. One would simply add the appropriate linear equality constraints and compute all functions simultaneously as also described in Equation 4.5.

Figure 5.8: *The* Dino *in its rest pose (left) is deformed using the biharmonic weights of [Botsch and Kobbelt 2004] (middle). In this example the weights are between* $[0, 1]$ *but a local minimum (blue dot) leaves the tail connected to the feet, giving the impression that it is glued to the ground when bending. Our solution finds more intuitive, monotonic weights.*

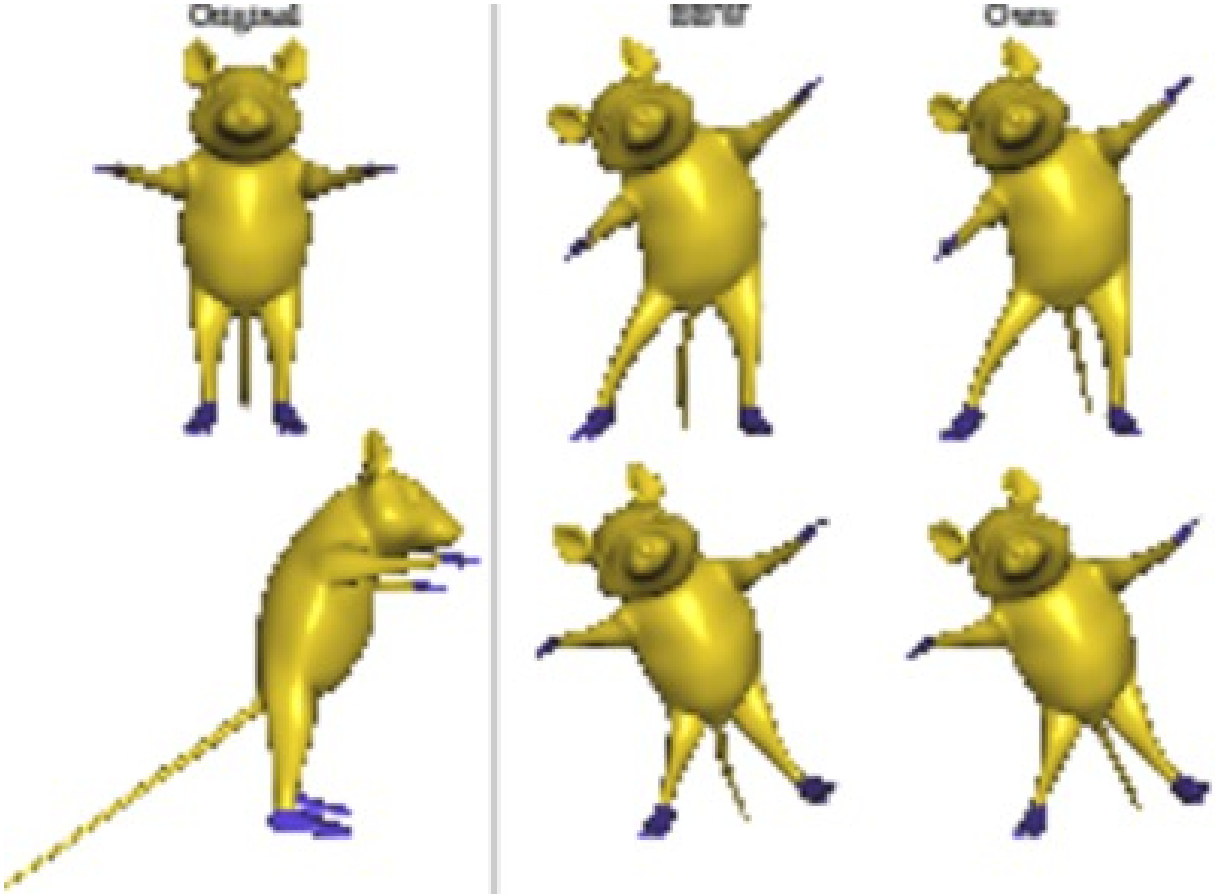


Figure 5.9: *The* Mouse *in its rest pose (left) is deformed using the biharmonic weights (BBW, middle), which each have an extrema in the tail causing it to wiggle when the handles are deformed. Our weights have no spurious extrema and keep the tail stiff.*

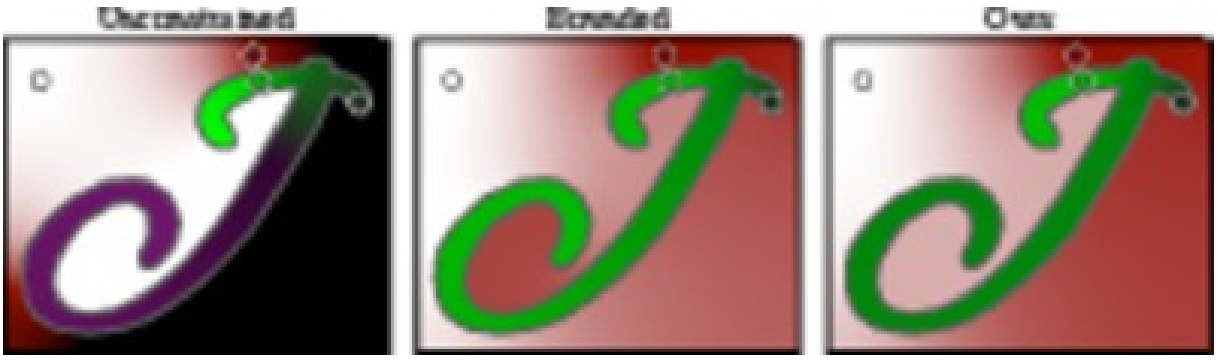


Figure 5.10: *Colors are diffused using the unconstrained biharmonic functions of [Finch et al. 2011], resulting in extrapolated (purple) and clipped (black) regions. Placing constant bounds helps, but oscillations with local extrema are still visible. Our solution provides a smooth diffusion without wild oscillations.*

5.4 Experiments and results

We experimented with quadratic energies corresponding to biharmonic, triharmonic and tetraharmonic equations with and without least-squares data energy terms. We tested our method on an iMac Intel Core i7 3.4GHz computer with 16GB memory. The computation time measurements of our code are reported in Table 5.1. The optimization time required depends heavily on the order of the polyharmonic operator in play. For the Laplacian energy, $k = 2$, we show timings on the same order of magnitude as Section 4.4, where solve a simpler quadratic programming problem with only constant bounds. For $k = 3$, our conversion to conic programming results in a rectangular coefficients matrix, which is in turn not as efficiently optimized. Somewhat surprisingly, increasing the order to the next even power, $k = 4$, returns to much faster computation time: the square root of discrete tetraharmonic operator is again square.

We use dual quaternion skinning [Kavan et al. 2008] to deform the *Cactus* in Figure 5.1. This avoids distracting shrinkage artifacts present in linear blend skinning. This example demonstrates that increasing the smoothness operator under our framework does not result in wilder oscillations.

We have found that unconstrained and bounded polyharmonic solutions often struggle with long appendages, placing extrema in the propeller in Figure 5.7 and the chimneys in Figure 5.3. During exploration of the deformation, local extrema in these appendages are immediately apparent and distracting during interaction. The tail of the *Dino* in Figure 5.8 feels as if it is glued to the ground when the head is transformed. The tail of the *Mouse* in Figure 5.9 wiggles when deforming the hands and feet, rather than staying stiff as one might expect. Applying scaling exaggerates the unintuitive nature of local extrema in weight functions: the geodesically distant sousaphone bell is sheared when scaling the head in Figure 5.13. Whether the additional local extrema are maxima or minima seems largely unpredictable; the oscillations may flip due to slight changes in the boundary definition. In Figure 5.12, the unconstrained and bounded weights of the *Hummingbird*'s beak introduce a local minimum in the tail, whereas ours stay monotonic (top row, right). The bottom row shows the chaotic nature of the unconstrained and bounded weights' oscillations. Slightly larger wing handles reverse the oscillations, now producing a local maximum in the tail. We constrain the solution to be monotonic and thus

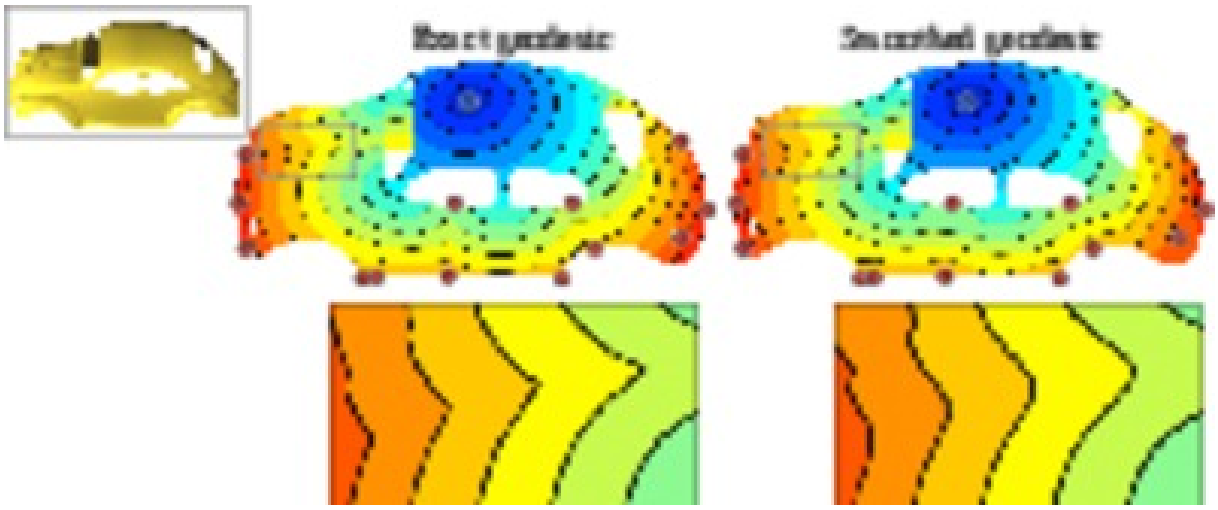


Figure 5.11: *Sharp cusps appear in exact geodesic distances (left) computed for a point on the top of the* Beetle *(inset). Our framework smooths this field while guaranteeing that all maxima maintain their location and value (right). No new extrema are created.*

avoid such chaotic oscillations. In contrast, the monotonic weight functions resulting from our method allow intuitive deformations in all these cases. Please refer to the supplemental video of [Jacobson et al. 2012b] to get a better impression of the deformation behavior.

Long features are also difficult for previous methods when blending colors. Oscillations common to biharmonic functions used by [Finch et al. 2011] lead to interpolation weights outside of $[0, 1]$ and may extrapolate colors not present in the user's constraints: red - grey = blue in Figure 5.4. Placing bounds on these weight functions helps, but local extrema are still present, having the effect that colors fade out and then suddenly reappear somewhere else in the domain (see Figure 5.10).

In Figure 5.14, we smooth exchange rate data which contains a pronounced, sharp spike at the global maximum. To ensure that the C^0 nature of this spike is not smoothed away, we take advantage of the fact that the parameterized blends of different polyharmonic operators in [Botsch and Kobbelt 2004] may be reexpressed as discrete quadratic energies. We may then optimize using our method, such that only specified extrema are present in the result. In this example, we choose our blend parameters (λ in Section 3 of [Botsch and Kobbelt 2004]) such that we create a C^0 at the global maximum and C^1 elsewhere.

Smoothing geodesic distance fields is often a tricky balance between achieving desired smoothness and loosing control of the linear spacing of isolines and placement and values of maxima (farthest points). Our framework reconstructs a smoothed geodesic distance field which has the original maxima and no others (see Figure 5.11). Smoothing noisy data similarly requires care, or original features may be lost while new ones are introduced during smoothing. In Figure 5.15, we reproduce the topology-based smoothing results of [Weinkauf et al. 2010] (compare to Figure 10c in their paper) with an optimization time that, for this example, is $1000\times$ faster.

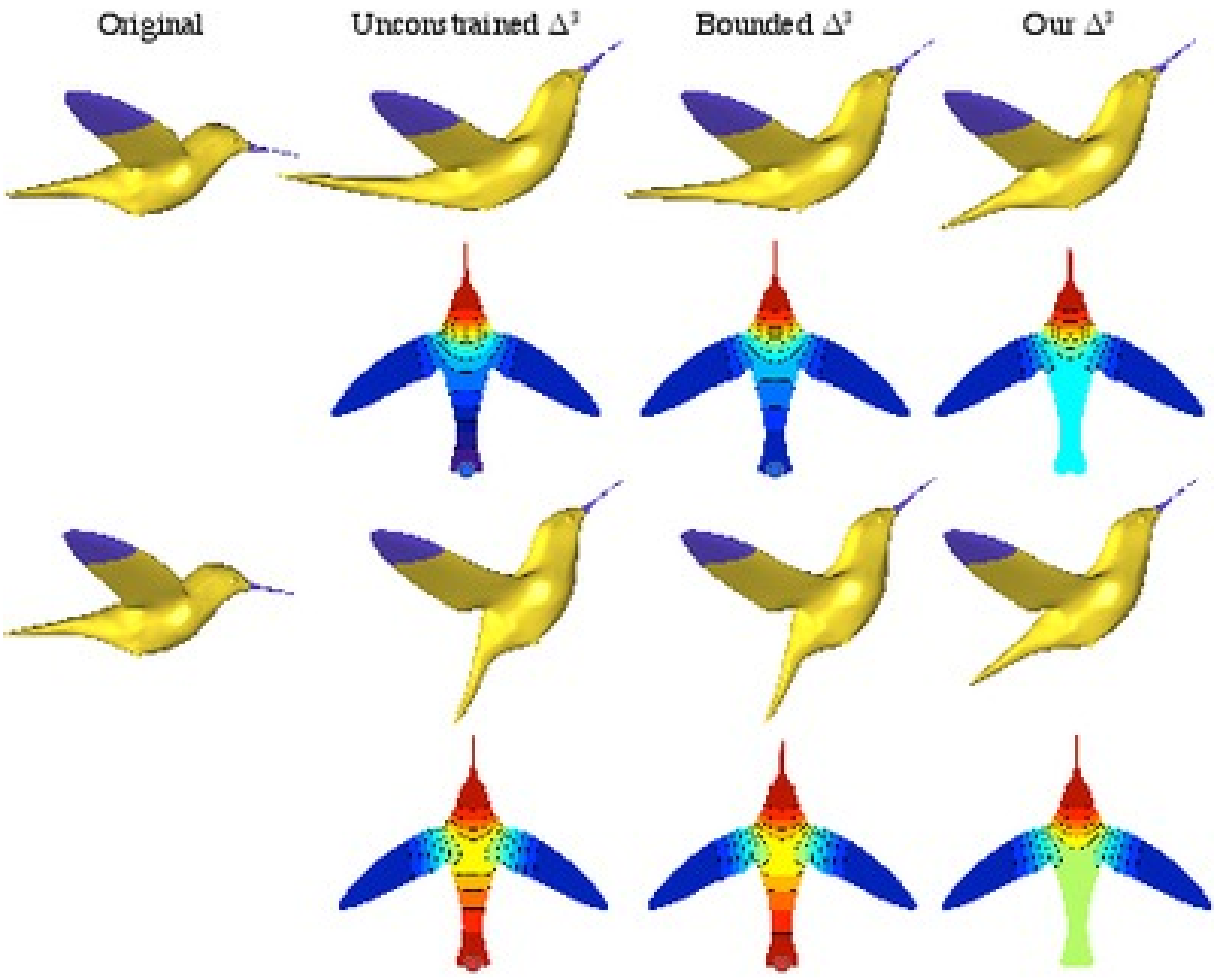


Figure 5.12: *The* Hummingbird *in her rest pose is deformed using the unconstrained, bounded and our triharmonic weights.*

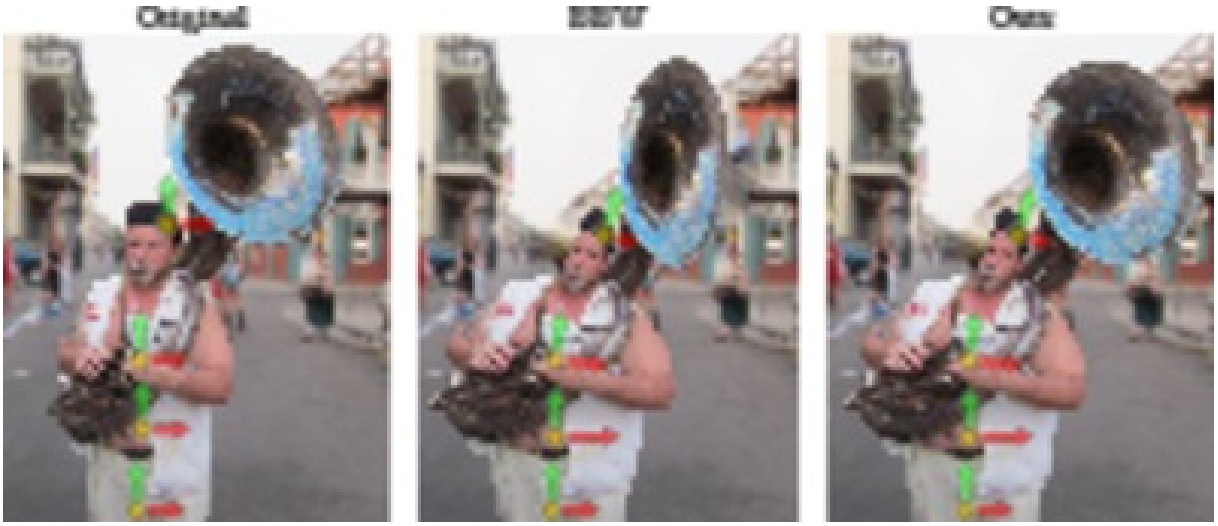


Figure 5.13: *The* Sousaphonist *in his rest pose (left) is deformed using bounded biharmonic weights (BBW, middle). Due to local extrema the horn's bell gets an uneven deformation, whereas our local-extrema-free deformation maintains its shape.*

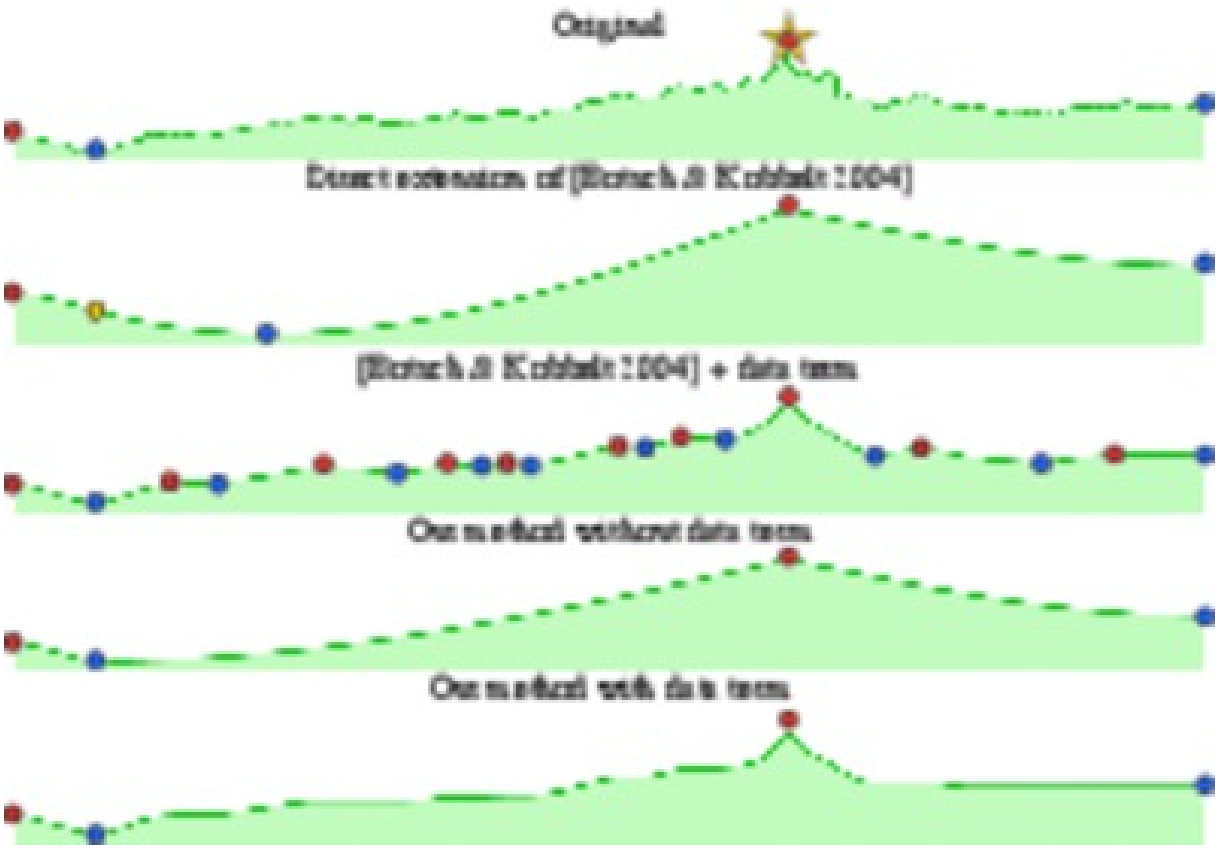


Figure 5.14: *Top to bottom: Currency exchange data with hundreds of extrema including a sharp, global maximum (star). Blended polyharmonic energies of [Botsch and Kobbelt 2004] can reproduce the spike, constraining only the values of the global minimum, global maximum, and endpoint values. But this produces new extrema, changing the global min (from yellow dot). Adding a data term introduces even more extrema. Our formulation prohibits new extrema, and with a data term provides a smooth, monotonic representation of the data.*

5.5 Limitations and future work

Due to the large number of linear inequality constraints, computation can still be expensive, compared to solving a linear system (e.g. [Botsch and Kobbelt 2004, Finch et al. 2011]), or to optimizing a quadratic energy with constant bounds, as in Chapter 4. But it is important to keep in mind that the ideal problem in Equation (5.6) is far more difficult. We would like to investigate other optimization methods that could take advantage of a warm start or other constraint simplification possibilities.

We take advantage of the maximum principle of harmonic functions, but it is well known that obtuse angles may cause weights in a cotangent Laplacian to be negative, thus nullifying the guarantee of the maximum principle [Wardetzky et al. 2007b]. We sometimes observe this problem *locally* in construction of our representative functions u. While this could potentially lead to actually enforcing local extrema in our final solution, we observe that globally u captures the correct gradient information we need, and problems due to poor discretization can often be safely ignored. Of course, another option is to use a discrete Laplacian with positive weights, but this may come at the cost of other convenient properties [Wardetzky et al. 2007b].

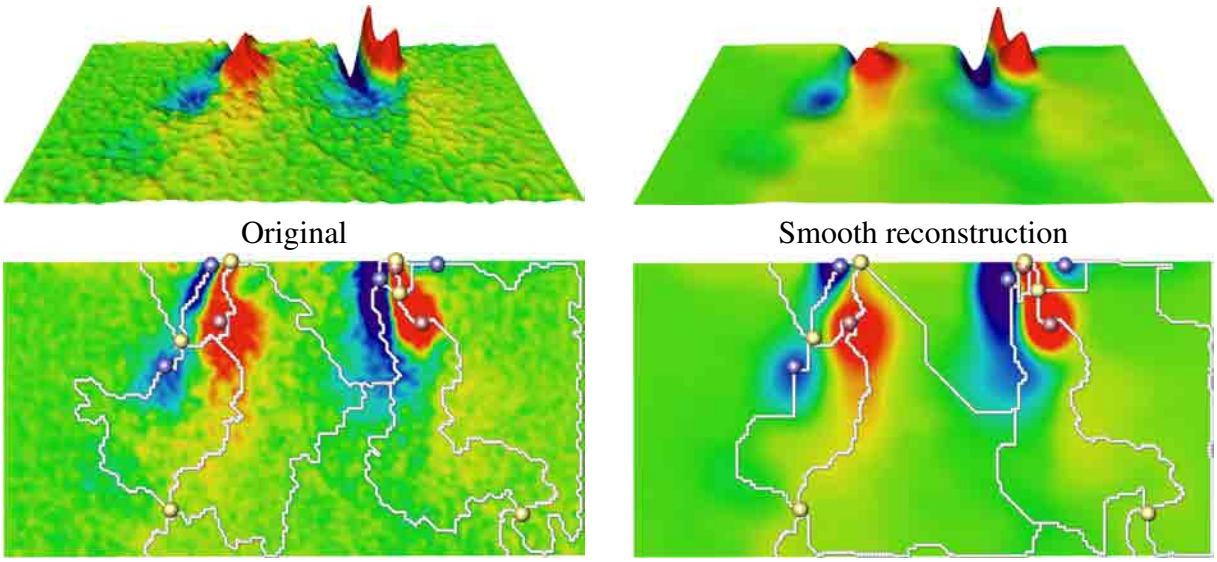


Figure 5.15: *By adding a data energy term and using a topology-aware representative function, we may use our framework to smooth noisy data. Left: vorticity magnitude derived from an optically measured flow at the outlet of a combustion chamber, with thousands of local extrema. Using persistence-based simplification, we isolate the most important extrema: 9 minima, 4 maxima (blue and red dots). We then smooth the data guaranteeing that these and only these extrema occur in the solution (right). Data courtesy of A. Lacarelle (TU Berlin) [Lacarelle et al. 2009].*

Crane et al. [2012] use the normalized gradients of solutions to a Poisson equation to define smooth geodesic distances. It would be interesting to relate this new method to our approach.

Finally, our per-edge, linear inequality constraints are inherently discrete. For data smoothing this means, an asymmetric meshing combined with a strong data term may produce unintuitively asymmetric reconstructions. Without a data term (e.g. for deformation or color interpolation) this appears to be a non-issue: the constrained optimization subspace is still quite large.

5.6 Conclusion

We have shown a framework for constructing smooth, shape-aware functions on 2D and 3D surfaces with guarantees on the placement and values of extrema. We also highlight the typical problematic situations for which our method succeeds over previous work. We believe our work will help promote the continued study of topological constraints in connection with geometry processing applications like deformation and interpolation on manifolds.

5.7 Appendix: Conversion to conic programming

We use MOSEK [Andersen and Andersen 2000] to efficiently solve sparse, quadratic progamming problems. Its documentation strongly recommends converting convex quadratic energy

minimization with linear inequality constraints, like Equation (5.12), into linear energy minimization with conic constraints. We found this to be especially advantageous for our problem. Without loss of generality we assume our energy is of the form: $E(f) = \int_{\mathcal{M}} (\nabla^k f)^2 dA$, which can be discretized as

$$E(\mathbf{f}) = \frac{1}{2} \mathbf{f}^\mathsf{T} \left(\mathbf{L}\mathbf{M}^{-1}\right)^{k-1} \mathbf{L}\,\mathbf{f}, \tag{5.18}$$

where $\mathbf{L}$ and $\mathbf{M}$ are the familiar cotangent Laplacian and normalized, diagonalized mass matrix, respectively [Meyer et al. 2003]. We may write $\mathbf{L} = \mathbf{G}^\mathsf{T}\mathbf{A}\mathbf{G}$ where $\mathbf{G}$ is the per-element gradient operator and $\mathbf{A}$ has triangle areas repeated along the diagonal [Botsch et al. 2010]. For odd k, Equation (5.18) becomes:

$$E(\mathbf{f}) = \frac{1}{2} \| \sqrt{\mathbf{A}}\mathbf{G} \left(\mathbf{M}^{-1}\mathbf{L}\right)^{\frac{k-1}{2}} \mathbf{f} \|^2, \tag{5.19}$$

and for even k:

$$E(\mathbf{f}) = \frac{1}{2} \| \sqrt{\mathbf{M}}^{-1} \left(\mathbf{L}\mathbf{M}^{-1}\right)^{\frac{k}{2}-1} \mathbf{L}\mathbf{f} \|^2. \tag{5.20}$$

This allows us to write $E(f) = \frac{1}{2}\|\mathbf{F}\,\mathbf{f}\|^2$ and Equation (5.12) becomes convertible to conic form via Section 2.2.4.

5.8 Appendix: Iterative *convexifiction*

In the previously described method, we computed a single feasible representative function u by exploiting the maximum principle of harmonic functions. Then, like [Weinkauf et al. 2010], we optimize Equation 5.12 and return a solution $\mathbf{f}$. We may now notice that this solution f is again a valid representative function. Treating this f as u, we repeat the process of convexifying the ideal, nonlinear constraints around u with linear constraints, optimizing the resulting QP for f and setting u to f.

We may repeat this process until convergence, which is guaranteed because the energy of our iterative solutions $E(f_i)$ is monotonically decreasing. Given an initial solution f_i, we linearize our constraints around f_i. Because the constraints are constructed according to f_i, we know the feasible set they describe is nonempty: it at least contains f_i. Optimizing the resulting QP for the next solution f_{i+1} guarantees that $E(f_{i+1}) \leq E(f_i)$ because f_{i+1} is the unique global minimum in the feasible set.

Figure 5.16 illustrates a few steps of this process. In the upper-left image we visualize a quadratic function, x^2+4y^2, with a topographical map (concentric, elliptic isolines) and a global minimum (blue dot). We define a feasible region (white region inside red overlay). Given an initial guess (yellow dot) we can use our heuristic to determine a convex subset of the feasible region around this point (formed by intersecting halfspaces shown in green overlays). Within this region the global minimum is found by solving a QP. The process is then repeated *around* this point: the heuristic is applied again, defining a new convex subset, etc. Our heuristic in general will be a poor approximation of the complex, non-convex feasible region. Still for a given problem this *moving convex window* shows improvements at each iteration, making it of course superior to running just one convexification and solving one QP.

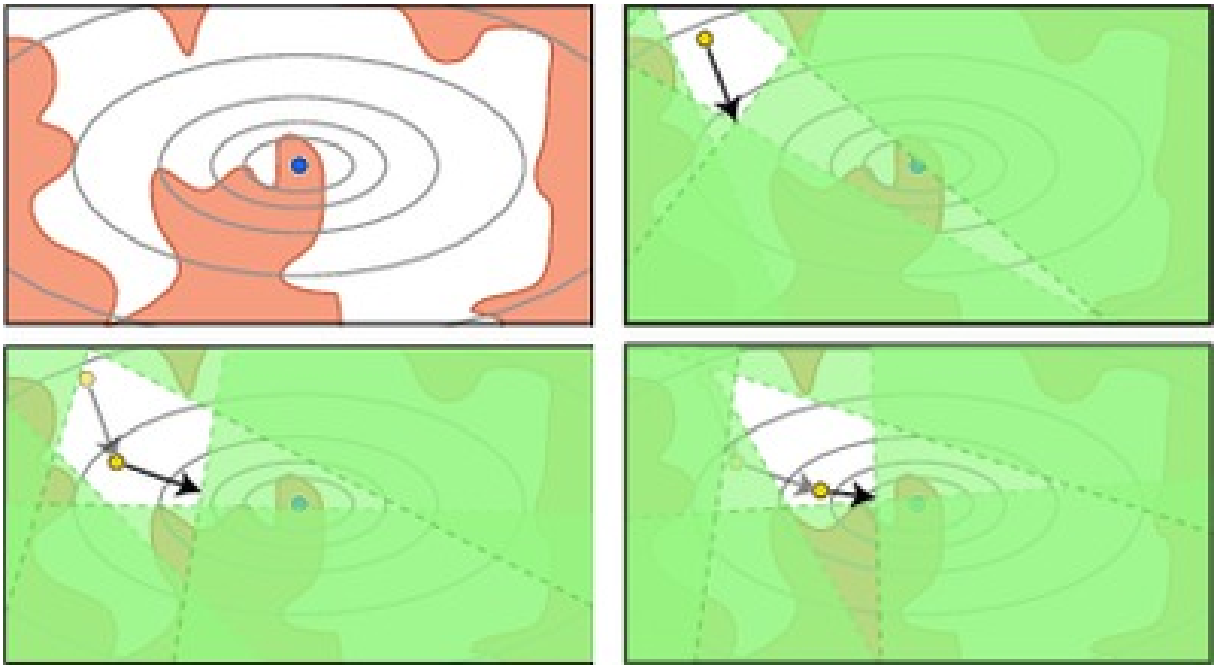


Figure 5.16: *Illustration of the nonlinear optimization problem Equation (5.6) and our scheme to iteratively decrease the energy of the solution by repeated convexification of the feasible region using the linear constraint Equation (5.14).*

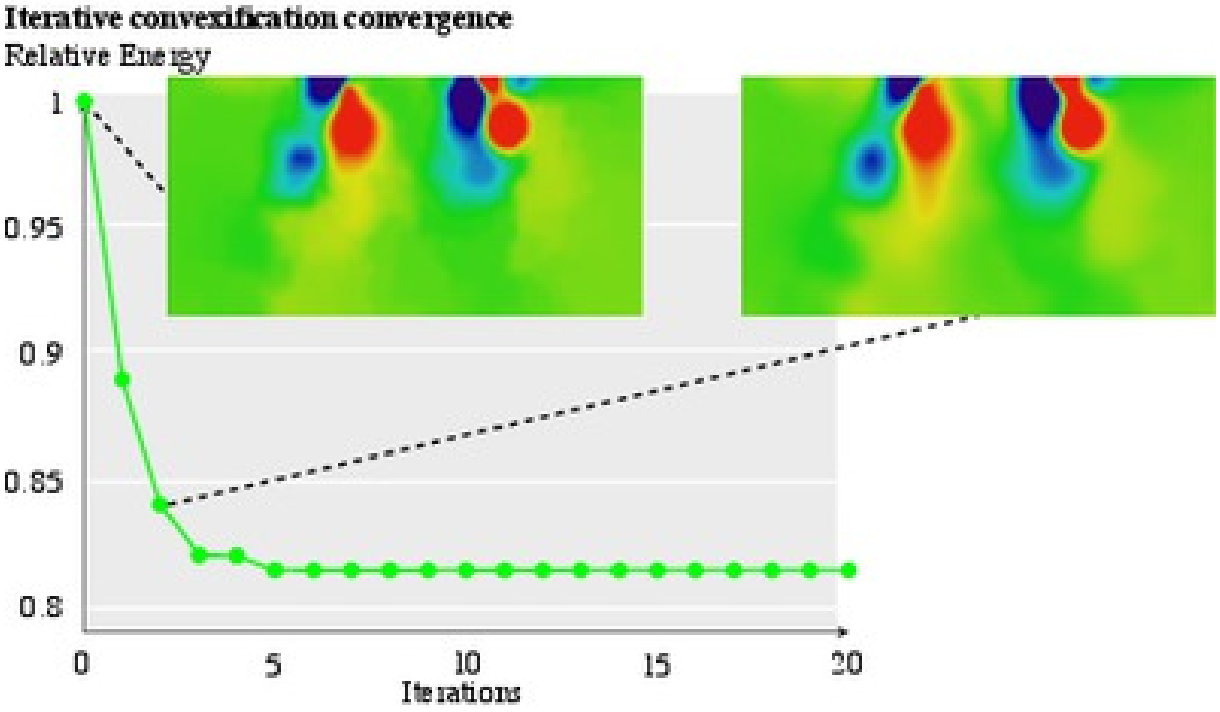


Figure 5.17: *Convergence with respect to relative error on the* Combustor *data set (cf. Figure 5.15).*

While we are not guaranteed to converge to the global minimum, or even a local minimum, of the ideal problem in Equation 5.6, we see enormous energy reduction in the first few iterations with diminishing returns Figure 5.17.

Our process of iteratively redefining the convex feasible set and solving QPs is related to Sequential Quadratic Programming (SQP). However, generic SQP assumes all nonlinear inequality constraints are at least twice continuously differentiable [Nocedal and Wright 2006].

6

Stretchable and twistable bones for skeletal shape deformation

Skeleton-based linear blend skinning (LBS) remains the most popular method for real-time character deformation and animation. The key to its success is its simple implementation and fast execution. However, in addition to the well-studied elbow-collapse and candy-wrapper artifacts, the space of deformations possible with LBS is inherently limited. In particular, blending with only a scalar weight function per bone prohibits properly handling stretching, where bones change length, and twisting, where the shape rotates along the length of the bone. We present a simple modification of the LBS formulation that enables stretching and twisting without changing the existing skeleton rig or bone weights. Our method needs only an extra scalar weight function per bone, which can be painted manually or computed automatically. The resulting formulation significantly enriches the space of possible deformations while only increasing storage and computation costs by constant factors.

6.1 Introduction

Skinning and skeletal deformation remain standard for character animation because the associated deformation metaphor is directly intuitive for many situations: most characters are creatures or humans who ought to behave as if a skeleton was moving underneath their skin. The motion capture pipeline, for example, explicitly relies on this metaphor to build a subspace representation of human motion [Anguelov et al. 2005]. At the cost of performance, some applications demand physical accuracy, ensuring preservation of volume or simulating muscles [Teran et al. 2005]. Other applications, such as video games, crowd simulation and interactive animation editing, cannot afford to compromise real-time performance, so they trade

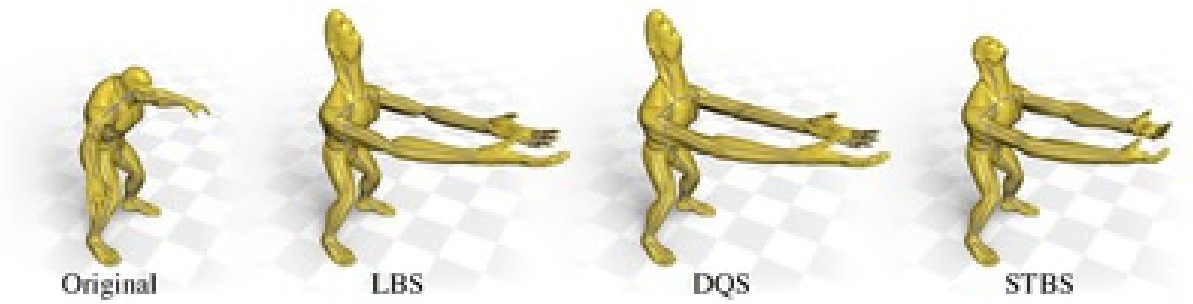


Figure 6.1: *Left to right: the* Beast *model is rigged to a skeleton in its rest pose. The neck is stretched and the arms are twisted and stretched using linear blend skinning. LBS relies solely on per-bone scalar weight functions, resulting in the explosion of the head and hands. The candy-wrapper artifact of LBS is also noticeable at the elbows. The dual quaternion skinning (DQS) solution [Kavan et al. 2008] correctly blends rotations, avoiding the candy-wrapper artifact, but reliance on bone weights alone unnaturally concentrates the twisting near the elbows. DQS also does not alleviate the stretching artifacts. Our solution, stretchable, twistable bones skinning (STBS), uses an extra set of weights per bone, allowing stretching without explosions and smooth twisting along the entire length of each arm.*

accuracy for speed and often adopt the simplest and most efficient implementation of skeletal deformation.

The time-tested standard for real-time skeletal deformation method is linear blend skinning (LBS), also known as skeletal subspace deformation or enveloping [Magnenat-Thalmann et al. 1988, Lewis et al. 2000]. In a typical workflow, a trained rigging artist manually constructs and fits a skeleton of rigid bones within the target shape. The skeleton is bound to the shape by assigning a set of correspondence weights for each bone, a process which can be tedious and labor-intensive. To deform the shape, animators assign transformations to each skeleton bone, either directly or with the assistance of an inverse kinematics engine or motion capture data. These transformations are propagated to the shape by blending them linearly as matrix operations according to the bone weights.

Linearly blending matrix transformations with scalar weight functions has a number of limitations. Many improvements of LBS focus on the problems arising from linearly blending rotations as matrices, which results in shape collapses near joints. Multi-weight enveloping (MWE) [Wang and Phillips 2002, Merry et al. 2006] and Dual Quaternions [Kavan et al. 2008] have been proposed as alternative rotation blending methods. However, a different set of limitations arises from the fact that using only a single scalar weight function per bone limits the space of possible deformations. We show that neither LBS nor its improvements properly handle stretching, where bones change length, nor twisting, where the skin twists along the length of a bone, as in the human forearm (see Figure 6.1).

Our goal is to expand the space of deformations possible with skinning to include stretching and twisting. Incorporating these two actions into real-time skinning greatly increases the space of deformations. For example, stretching helps facilitate exaggerated actions (see Figure 6.2). Exaggeration has long been held as a cornerstone principle of traditional and computer animation [Thomas and Johnston 1987, Lasseter 1987].

We achieve stretching and twisting by using an additional set of weights for each bone. We

Figure 6.2: *Examples of stretching used to exaggerate animations in cartoons and feature films.*

Low resolution images used under fair use for criticism and commentary.

call them *endpoint weights* as they reveal correspondences between the shape and the endpoints of each bone[1]. These additional weights allow us to modify standard skinning formulas — in particular, LBS and dual quaternion skinning (DQS) — to explicitly expose stretching and twisting of bones. We hence term our method "stretchable, twistable bones skinning" (STBS). Deformation computation remains embarrassingly parallel, and the involved extra storage and computation costs are minimal. Unlike other methods that employ additional weights, our endpoint weights have a clear and intuitive geometric meaning and thus may be painted manually. Capitalizing on recent methods like [Baran and Popović 2007] that build skeletons and compute bone weights automatically, our endpoint weights may similarly be computed automatically, keeping pipelines fully automatic if desired.

Many artists will be hesitant to change to a new skinning scheme. Our solution complements the typical skinning environment without changing the rigid skeleton metaphor or interfering with existing controls: if bones are not stretched or twisted, the deformation remains the same as defined by the underlying skinning method. We also take advantage of existing skinning rigs, allowing users to opt in without modifying their existing skeletons or bone weights.

Problem context. Skeletal skinning with bone weights works well because bones as deformation handles properly capture the natural rigidity of body parts. Linearly blending bone transformations via bone weight functions efficiently expresses rigidity along bones during bending, packing smooth transitions near joints, where the weights briefly overlap [Magnenat-Thalmann et al. 1988]. Unfortunately, bone weights that produce natural bending are insufficient for producing plausible stretching and twisting, because they are poor at controlling the subspace *along* the bone.

Works like [Wang and Phillips 2002, Merry et al. 2006] improve LBS by supplying additional weights per bone. These extra weights are additional degrees of freedom which can alleviate joint collapse, and perform twisting better. However, these additional weights do not retain an immediate or intuitive geometric meaning, since they essentially correspond to individual transformation matrix entries. As a result, they cannot be easily painted or adjusted manually, but only computed automatically via fitting example poses of the target shape. Often such

[1] Alternatively, consider these weights as a parameterization of the shape *along* each bone segment.

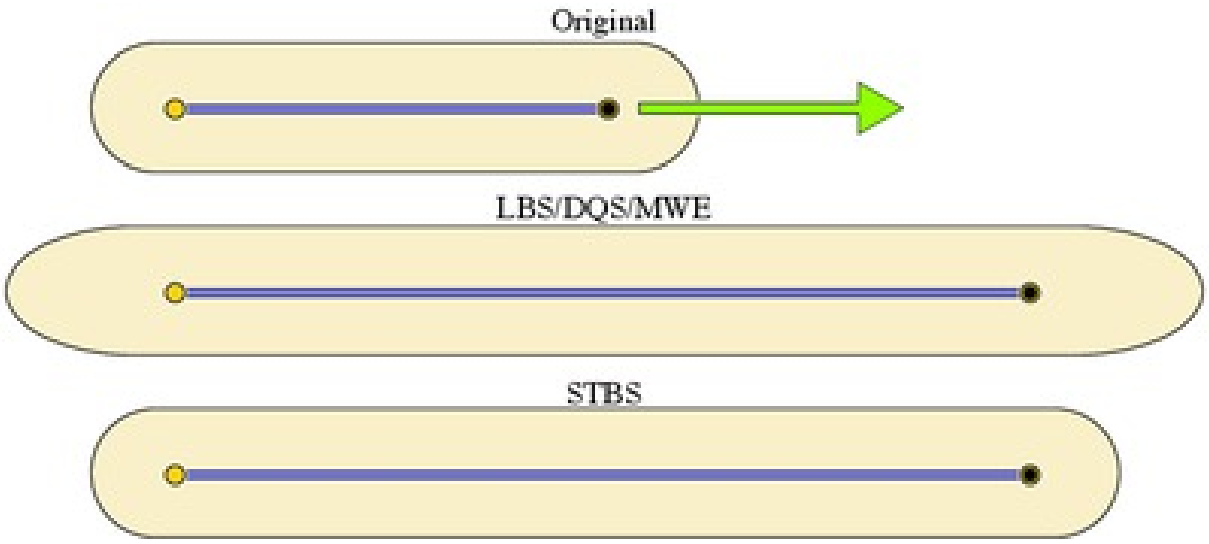


Figure 6.3: *A single bone controls a cigar shape (top). With only one bone, the weights of LBS and [Wang and Phillips 2002, Merry et al. 2006] must equal 1 everywhere. When the bone is stretched, these methods must scale the entire shape, resulting in explosion past the bone's endpoints (center). While the bone weights in our method must also be 1 everywhere, the endpoint weights are allowed to vary, such that proper stretching is achievable (bottom).*

example poses do not exist or are difficult to design, making these improvements challenging to apply in practice.

Even with the reliance on example poses aside, the deformation formulations of multi-weight enveloping methods do not sufficiently expand the space of deformations to capture stretching. Consider for example a single bone within a cigar-like shape, as in Figure 6.3. In order to maintain good properties such as reproduction of the identity transformation and translational and rotational invariance, the bone's LBS weights and any of the extra weights of [Wang and Phillips 2002, Merry et al. 2006] must equal 1 everywhere on the shape. This means that if the bone changes its length, the only choice is to scale the whole shape by the same transformation, resulting in "explosion" past either endpoint.

To overcome LBS artifacts, riggers often manually subdivide bones or add special anatomically incorrect bones. Painting weights for these special bones is difficult because their intuitive meaning is less clear. The method of [Mohr and Gleicher 2003] uses example poses to determine such extra bones and their weights automatically. With enough extra bones with proper weights, twisting and stretching can be achieved, but at the sacrifice of an anatomically meaningful skeleton.

Other works such as [Forstmann and Ohya 2006, Yang et al. 2006, Forstmann et al. 2007] use curve or spline skeletons to cope with LBS artifacts. These methods share a similar foundation as our method, but focus on fixing joint collapse and do not explicitly treat stretching. To define correspondences between their curved skeleton "bones" and points on the shape, they rely on inverse Euclidean distance schemes which ignore the geometry of the shape being animated. The new rigging tools and controls needed for curved skeletons are inconsistent with the existing rigging pipeline [Kavan et al. 2008].

Figure 6.4: *A single bone controls a box in 3D. The bone's weights in LBS, DQS, and MWE must equal 1 everywhere. If the bone twists about its axis by* $315°$*, these methods must twist the entire shape uniformly, i.e. simply rigidly rotate it. The bone weights in our method must also be 1 everywhere, but the endpoint weights are allowed to vary, so interesting twisting may be*

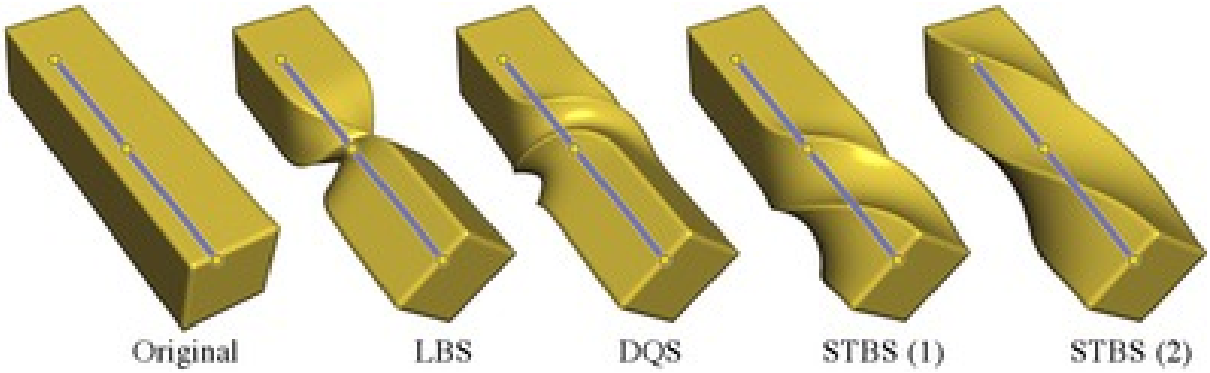


Figure 6.5: *Two bones control a box in 3D. In order to bend properly, the bone weights in an LBS or DQS rig must only overlap close to the joints. As a result these methods must pack interesting twisting near joints. LBS linearly blends rotation matrices, resulting in the candy-wrapper effect. DQS corrects this artifact by blending rotations as quaternions, but twisting is still concentrated near the joint. Our method keeps the same bone weights, but our extra endpoint weights enable twisting to be spread across the length of one bone or both bones.*

Instead of relying on additional weights or alternative rigging metaphors, Kavan et al. [2008] directly solve the joint collapse and candy-wrapper artifacts by blending rigid bone transformations as dual quaternions rather than matrices, leaving the skeleton and bone weights metaphor untouched. However, DQS still relies on a single set of bone weights, so stretching remains unsolved and twisting must still be concentrated near joints (see Figures 6.4 & 6.5).

Recent works in 2D and 3D have successfully demonstrated the flexibility of point handles with associated weight functions (see e.g. [Langer and Seidel 2008] and Chapters 4 & 5). Point weights by construction vary over the shape more than bone weights, and typically, much of the shape is significantly affected by two or more points. This means they are unsuitable for bending limbs rigidly, but properly capture stretching (see Figure 6.6). Many works on shape editing and deformation advocate the point handle metaphor (see e.g. [Igarashi et al. 2005] or the survey in [Botsch and Sorkine 2008]), albeit at much higher computational costs of surface deformation due to the involved global optimizations.

Contributions. Our contribution is to combine the notions of point weights and bone weights, allocating each to the tasks they do well, while maintaining the standard skeleton skinning framework. In our technique, bone weights continue to enforce rigidity and control

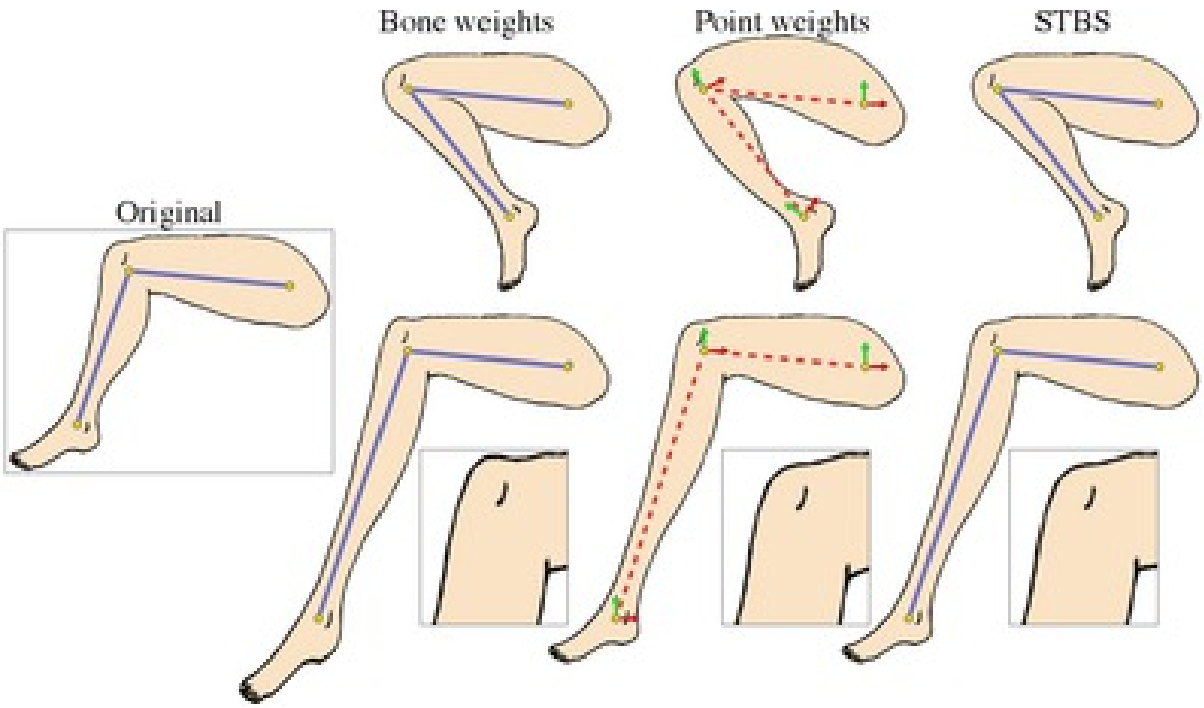


Figure 6.6: *Bones properly capture rigidity necessary to bend a leg (upper left), but stretching with bone weights explodes the foot and knee unintuitively (lower left). Point-handles properly treat stretching as blended translations (lower center), but blending rotations causes limbs to lose their rigidity (upper center). Our stretchable bones solution uses bone weights and point weights, allocating each to the tasks they do well. Our bones bend smoothly at joints (upper right) and stretch intuitively (lower right).*

smooth bending. In addition, we introduce point weights at bone endpoints to enable proper stretching and twisting. Since our goal is orthogonal to fixing the rotation blending artifacts of LBS, our method complements techniques like DQS and works with any underlying skinning method. We show that our method extends the space of deformations available for skinning in a useful way, demonstrate 2D and 3D examples of bending, stretching and twisting, and discuss several options for obtaining the required endpoint weight functions.

6.2 Stretchable, twistable bones

Our goal is to derive a simple skinning equation, capable of deforming 2D and 3D shapes by a skeleton whose bones may stretch and (in 3D) twist. Let $\mathcal{S} \subset \mathbb{R}^d$ denote our target shape in dimension $d = 2, 3$. We denote the set of (possibly disjoint and unordered) bones in the skeleton by the line segments $B_i = \{(1-t)\mathbf{a}_i + t\mathbf{b}_i \mid t \in [0,1]\}, i = 1, ..., m$. We derive our skinning equation by first decomposing the basic linear blend skinning equation. Here, the user defines affine transformations T_i for each bone B_i. The new positions for all points $\mathbf{p} \in \mathcal{S}$ are computed as the weighted combinations:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p})\, T_i \mathbf{p}, \tag{6.1}$$

where $w_i : \mathcal{S} \to \mathbb{R}$ is the scalar *bone weight function* associated with bone B_i. If the bone transformations T_i are rigid, they can be intuitively decomposed into translation and rotation parts, yielding:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p})\ \left\{\mathbf{a}'_i + R_i\left(-\mathbf{a}_i + \mathbf{p}\right)\right\}, \tag{6.2}$$

where R_i is the user-defined rotation that takes the bone B_i's rest vector $(\mathbf{b}_i - \mathbf{a}_i)$ to its pose vector $(\mathbf{b}'_i - \mathbf{a}'_i)$. If the bones are allowed to change length then a scaling term is needed to make sure that the bone endpoints reach their pose positions. The decomposition becomes:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p})\, \left\{\mathbf{a}'_i + R_i\left(S_i\left(-\mathbf{a}_i + \mathbf{p}\right)\right)\right\}, \tag{6.3}$$

where S_i performs anisotropic scaling in the reference frame of B_i, namely, $S_i = X_i^{-1} A_i X_i$, where X_i rotates $(\mathbf{b}_i - \mathbf{a}_i)$ to the x-axis and A_i scales anisotropically along the x-axis by a factor of $\|\mathbf{b}'_i - \mathbf{a}'_i\| / \|\mathbf{b}_i - \mathbf{a}_i\|$.

Notice that for each bone B_i, S_i and R_i are constant over $\mathcal{S}$, so that there is no choice for each bone but to rotate and stretch all points uniformly. For stretching this means that if $\mathbf{p}$ lies *beyond* an endpoint of a bone, it will get overly stretched, as shown in Figure 6.3. This effect is not removed even when multiple bones deform an area (e.g. around a joint), resulting in unwanted bulging (see Figure 6.6). As for twisting, a bone twists all attached points around its axis rigidly (see Figures 6.4 & 6.5), relying on the weighted average to blend twists from different bones. This effectively packs any interesting twisting near the joints where bone weights overlap.

The above problems are due to missing information: a point $\mathbf{p} \in \mathcal{S}$ does not "know" where on each bone B_i it is attached, i.e., it does not know its position relative to either of the bone's

Figure 6.7: *A human model is rigged to a skeleton using bounded biharmonic bone and endpoint weights. Its arm is twisted by* 180° *spreading the twist along the length of the upper and lower arm. In its twisted state, the arm is bent at the elbow. Joint collapse artifacts are corrected by switching to DQS as the underlying skinning formulation. Finally, the neck and arm are stretched.*

endpoints. This causes the excessive stretching (instead of localizing it to the bone area) and prevents gradual twisting along the bone.

We now insert this missing information in the form of *endpoint weight functions* $e_i(\mathbf{p})$ for each bone. These functions vary from 0 to 1 as $\mathbf{p}$'s correspondence shifts from endpoint $\mathbf{a}_i$ to $\mathbf{b}_i$. With these extra weight functions we have enough information to fix Equation (6.3) to handle stretching and twisting correctly. First, we replace the scaling term S_i with a weighted translation along the bone direction:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{ \mathbf{a}_i' + R_i \left(e_i(\mathbf{p})\, \mathbf{s}_i + (-\mathbf{a}_i + \mathbf{p}) \right) \right\}, \tag{6.4}$$

where $\mathbf{s}_i = (\frac{\|\mathbf{b}_i' - \mathbf{a}_i'\|}{\|\mathbf{b}_i - \mathbf{a}_i\|} - 1)(\mathbf{b}_i - \mathbf{a}_i)$, the full stretch vector at $\mathbf{b}_i$.

To allow twisting along bones, we insert an additional rotation term:

$$\begin{aligned} \mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \{ & \mathbf{a}_i' + \\ & R_i\, K_i \left(e_i(\mathbf{p}) \right) \left(e_i(\mathbf{p})\, \mathbf{s}_i + (-\mathbf{a}_i + \mathbf{p}) \right) \}, \end{aligned} \tag{6.5}$$

where $K_i(t)$ is the twisting rotation about the axis $(\mathbf{b}_i - \mathbf{a}_i)$ by angle $(1-t)\theta_{\mathbf{a}_i} + t\theta_{\mathbf{b}_i}$. The angles $\theta_{\mathbf{a}_i}$ and $\theta_{\mathbf{b}_i}$ are the user-defined twists at the endpoints $\mathbf{a}_i$ and $\mathbf{b}_i$, respectively. The new rotation $K_i(e_i(\mathbf{p}))$ is a function of $\mathbf{p}$ and thus is not constant over $\mathcal{S}$, enabling interesting twisting along each bone. Notice that if bone B_i is not stretched or twisted at its endpoints, its contribution is the same as in the original skinning equation (6.1).

6.2.1 Dual-quaternion skinning

Since we are blending rigid transformations, by applying the distributive property, Equation (6.5) may be simplified into a deformation equation that consists of a single rotation and translation per bone:

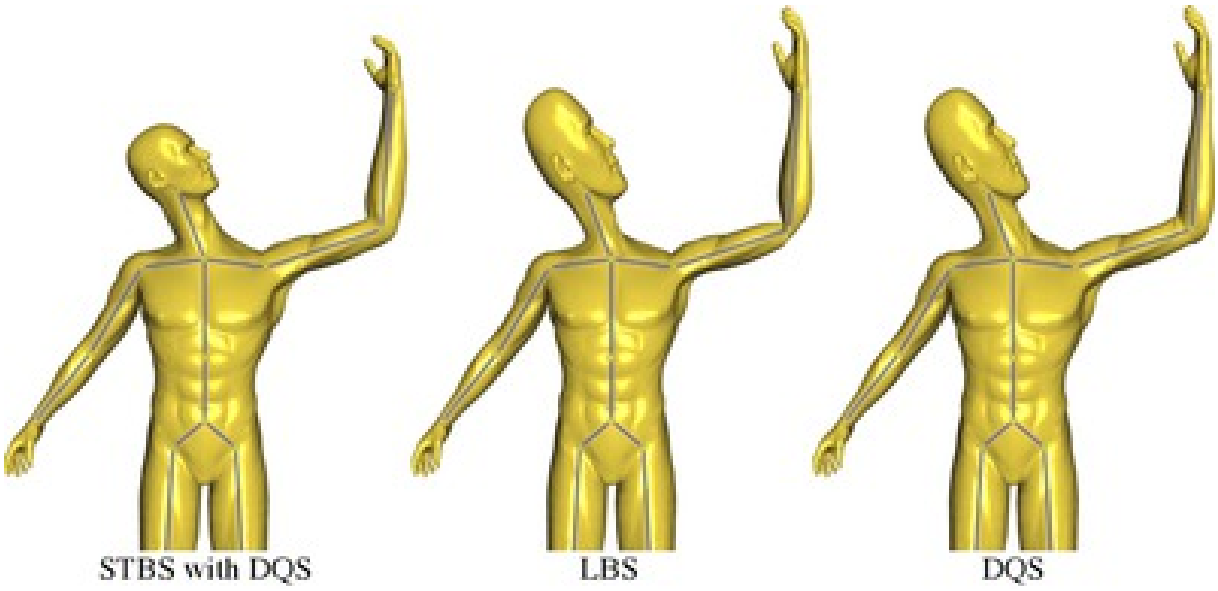


Figure 6.8: *Our final deformation from the sequence in Figure 6.7 prevents explosions in the head and hand. In contrast, using LBS results in joint collapse, isolated twisting and shape explosion. DQS prevents joint collapse, but twisting is still packed near joints and proper stretching is not achieved.*

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{ \mathcal{T}_i \left(e_i(\mathbf{p}) \right) + \mathcal{R}_i \left(e_i(\mathbf{p}) \right) \mathbf{p} \right\}. \tag{6.6}$$

Both the translations $\mathcal{T}_i$ and the rotations $\mathcal{R}_i$ are functions of the endpoint weight functions e_i, evaluated at $\mathbf{p}$, but they are constant w.r.t. the bone weight functions w_i. Thus far our skinning equation, like LBS, treats these translations and rotations as matrix operators and linearly combines them across bones as a sum of each matrix element weighted by the respective bone weights w_i.

Instead, we may blend $\mathcal{T}_i$ and $\mathcal{R}_i$ in their dual quaternion forms [Kavan et al. 2008]. As expected, DQS eliminates collapses near joints when bones are rotated. Combining DQS with our stretchable, twistable bones makes for a powerful and expressive skinning equation, as can be seen in Figures 6.7 & 6.8.

6.2.2 Properties of good endpoint weights

For the stretchable, twistable bones deformation Equation (6.5) to produce visually good deformations, care must be taken in defining both the bone weight functions w_i and the endpoint weight functions e_i. The desirable properties for a bone weight w_i are the same as in standard skinning (see Section 4.3.1 for a detailed discussion): the weight function should be shape-aware (i.e., dependent on the distance measured in the shape, as opposed to the ambient Euclidean space), should equal 1 on the rigid region corresponding to the bone and smoothly tend toward 0, reaching exactly 0 on rigid parts corresponding to other bones; the weights should be bounded between 0 and 1, since negative weights lead to unintuitive "opposite" deformation

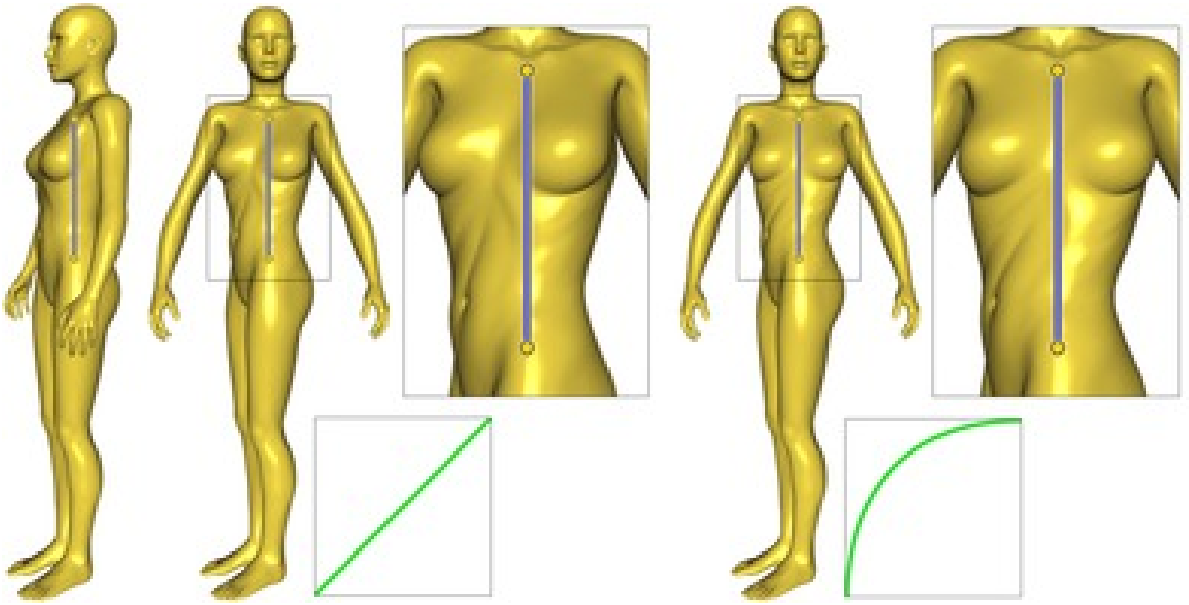


Figure 6.9: *A typical skinning rig must break the spine into many smaller bones in order to capture twisting. Here, a human model is twisted along the spine by applying twists at the top endpoint of a* single *bone. The endpoint weights smoothly blend the twist along the torso. They may be filtered through a spline curve to adjust their effect interactively. Filtered endpoint weights concentrate the twist in the abdomen, keeping the chest more rigid (right). Insets visualize the identity and user-defined spline filters, respectively.*

effects and weights greater than 1 exaggerate the prescribed transformations; the bone weights should partition unity at all points on the shape.

The list of desirable properties for endpoint weights is similar to that of bone weights, namely they should vary smoothly on $\mathcal{S}$, be shape-aware, and bounded between 0 and 1. In addition, the Lagrange (interpolation) property must be fulfilled, to ensure that user-defined positions and twists applied at endpoints are met: $e_i(\mathbf{a}_i) = 0$ and $e_i(\mathbf{b}_i) = 1$. Furthermore, in 2D, since bones lie directly on the shape they control, the endpoint weight functions should provide linear interpolation along their bone segments, i.e., $e_i(\mathbf{p}(t)) = t$ for $\mathbf{p}(t) = (1-t)\mathbf{a}_i + t\mathbf{b}_i, t \in [0,1]$. In 3D, bones are typically inside the volume enclosed by the shape, so this requirement becomes a "convergence" requirement: endpoint weight functions should approach linear interpolation as the shape approaches the bones. Notice that (in 2D) linear interpolation combined with the boundedness property contradicts smoothness exactly at the bone endpoints, where e_i will only be C^0. The importance of linear interpolation over smoothness depends on the application. Note also in Equation (6.5) that e_i is completely independent of the other endpoint and bone weights. Therefore, endpoint weights do not need to partition unity between themselves or with any of the bone weights[2].

[2]Actually, by definition each endpoint weight e_i partitions unity with *another*, unused endpoint weight associated with B_i's *other* endpoint: $\hat{e}_i := 1 - e_i$. This other endpoint weight is naturally factored out of Equation 6.5.

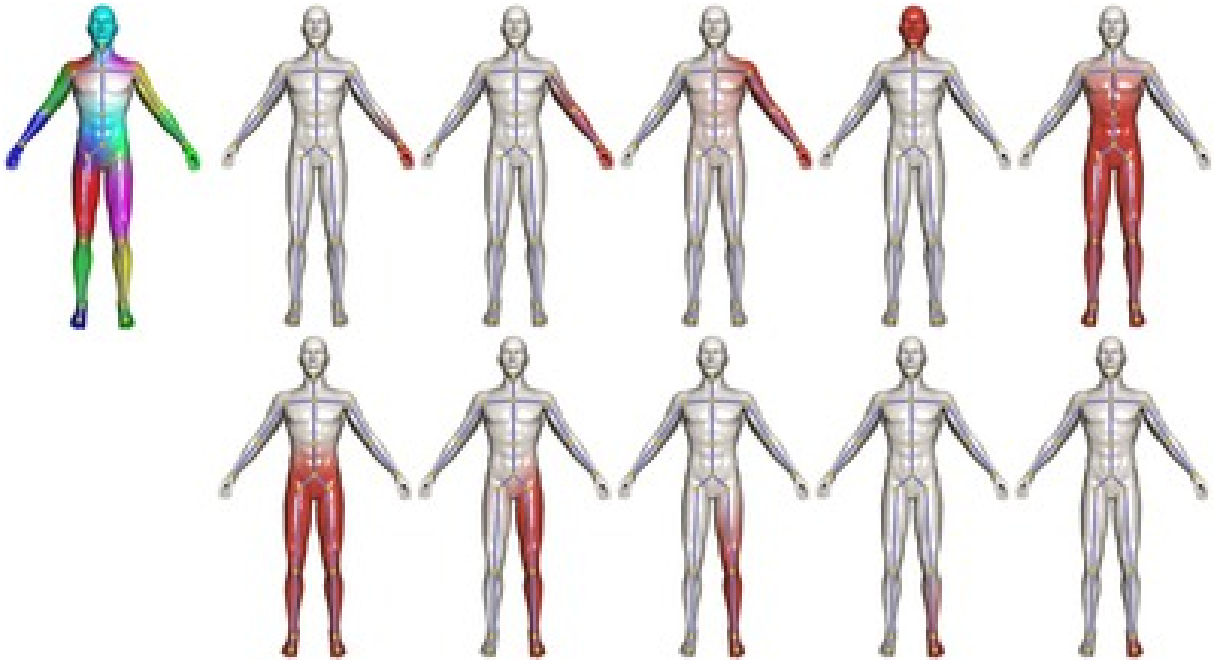


Figure 6.10: *The left image visualizes bone weights for a skeleton in a human model. Each bone is assigned a color. The color at a point on the shape is the weighted average of each bone's color according to the bone weights. The series of images on the right visualizes the endpoint weights of each bone. Red and white correspond to weights of 1 and 0 respectively. Both the bone weights and endpoint here were automatically computed using BBW.*

6.2.3 Defining endpoint weights

Endpoint weights, like bone weights, may be painted manually. This is feasible because, unlike the extra weights of [Wang and Phillips 2002] and [Merry et al. 2006], our extra weights have a clear geometric meaning: for each bone they tell each point in the domain how much it should stretch and twist. The desired weight properties are intuitive and conceivably specifiable by a user assisted with sufficient 2D and 3D weight painting tools (e.g. AUTODESK MAYA). As with LBS, both the bone weights w_i and the point weights e_i may be edited interactively, so a user can make changes to the weights and see the results immediately (see Figure 6.9).

Strictly speaking, the endpoint weights of different bones are unrelated. However, we have found that it is sufficient to supply traditional point-based weight functions for each *joint* in the skeleton. Then the endpoint weights for each bone become a combination of that bone's incident joint weights:

$$e_i = \frac{1}{2}\left((1 - j_{\mathbf{a}_i}) + j_{\mathbf{b}_i}\right), \tag{6.7}$$

where $j_{\mathbf{a}_i}$ and $j_{\mathbf{b}_i}$ are the weight functions at the joints incident on bone B_i at its respective endpoints $\mathbf{a}_i$ and $\mathbf{b}_i$.

Still, painting weights manually can be tedious and time-consuming. In the case that bone weights were assigned automatically, manually painting endpoint weights would disrupt a fully automatic rigging pipeline. Thus, we would like the option to define endpoint weights automatically.

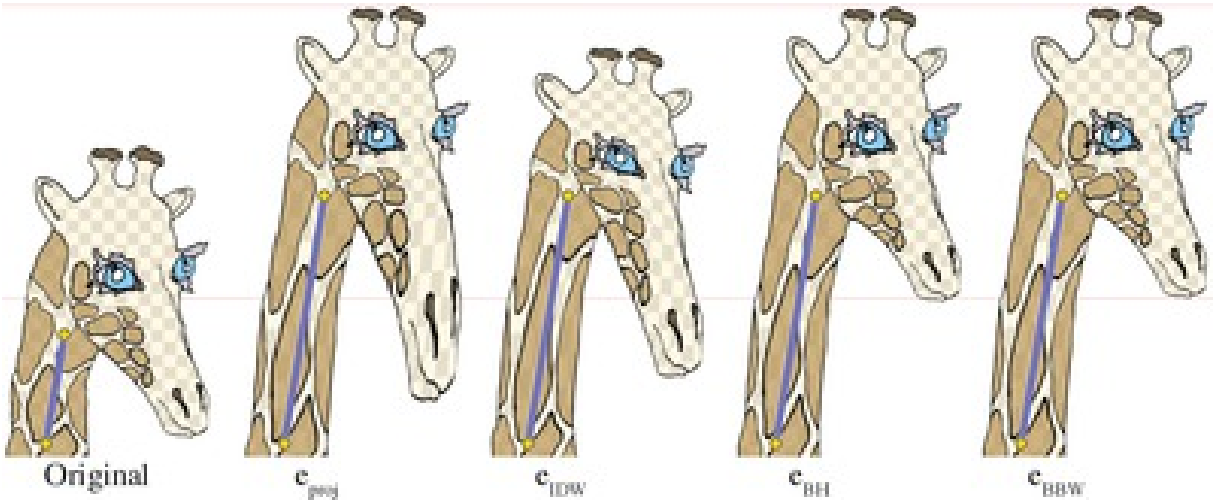


Figure 6.11: *Left to right: A single bone controls the* Giraffe*'s head and neck in 2D. Projecting onto the bone to obtain endpoint weights e_{proj} results in a nonsmooth and shape-*unaware *deformation. Endpoint weights e_{IDW} derived from inverse Euclidean distance joint weights are smooth, but shape-unaware and suffer from the fall-off effect (the top of the head sags too low). In this 2D example, BH endpoint weights e_{BH} and BBW endpoint weights e_{BBW} are virtually indistinguishable, producing appealing deformations.*

There are many existing automatic methods for computing such weights. One simple method is to project each point $\mathbf{p}$ onto the nearest point of each bone, taking the fraction of where it falls between the bone's endpoints as its weight:

$$e_{\text{proj}_i}(\mathbf{p}) = \frac{\|\text{proj}_i(\mathbf{p}) - \mathbf{a}_i\|}{\|\mathbf{b}_i - \mathbf{a}_i\|}, \tag{6.8}$$

where $\text{proj}_i(\mathbf{p})$ is the projection of point $\mathbf{p}$ onto B_i. These weights satisfy the boundedness and linear interpolation properties, but they largely ignore the shape of $\mathcal{S}$ and are only C^0 where they reach the values of 1 and 0. Variants of this type of weight function were used for the curved skeletons in [Forstmann and Ohya 2006, Yang et al. 2006, Forstmann et al. 2007] to associate a frame with each point $\mathbf{p}$ that corresponds to the curve's local frame.

Another immediate method would be to use inverse Euclidean distance weighting to define weights for the set of joint points:

$$j_{\text{IEDW}_i}(\mathbf{p}) = \frac{1}{d_i(\mathbf{p})^\alpha}, \tag{6.9}$$

where $d_i(\mathbf{p})$ is the Euclidean distance from $\mathbf{p}$ to the rest position of joint i. While endpoint weights derived from these joint weights may be smooth (for $\alpha \geq 2$) and bounded, they again ignore the shape of $\mathcal{S}$, and their locality diminishes beyond the endpoints as the ratio of distances to the two endpoints regresses to 0.5.

The above methods are easy to implement and computationally lightweight. They are unsatisfactory to be used directly but serve well as initial guesses for manually painting the endpoint weights.

The automatic bone weights method presented in Section 4 of [Baran and Popović 2007], also known as Bone Heat (BH), may be trivially adapted to define joint weights (by considering a

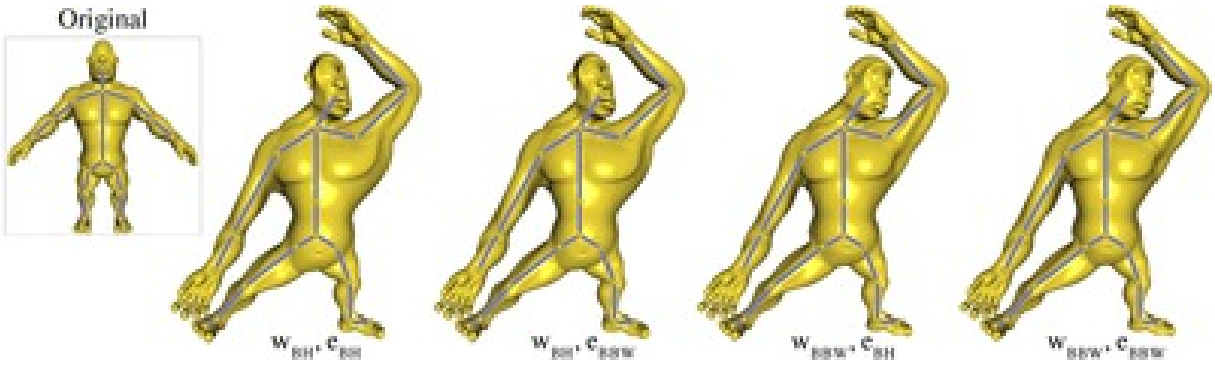


Figure 6.12: *Left to right: The* Ogre *(inset) is deformed with BH bone weights w_{BH} and BH endpoint weights e_{BH}. Switching to BBW endpoint weights e_{BBW} improves the deformation only slightly (e.g. see the shoulder). Changing endpoint weights cannot fix problems arising from insufficient bone weights. Instead, switching to BBW bone weights w_{BBW} noticeably improves the deformation around the head, shoulder and belly. With same bone weights, the endpoint weights of these methods produce visually similar results (comparing the first and final pairs).*

bone shrunk to a point). These weights require visibility computation and solving of large sparse linear systems of equations, but we have found that the quality of these weights in many cases justifies the computation costs. The methods of [Weber et al. 2007] and [Wang et al. 2007] may similarly be adapted to define joint weights. However, they only consider the surface of the shape rather than its volume, so the shape-awareness property is not fully achieved. Furthermore, these methods also require example poses, which often do not exist.

Our bounded biharmonic weights w_{BBW} of Chapter 4 (BBW) minimize the Laplacian energy subject to the constraint $w_{\text{BBW}} \in [0, 1]$ (and additional interpolation constraints, as required). This method may be used to compute joint weights or endpoint weights directly. Earlier we showed that these weights are smooth, localized and shape-aware. The Lagrange and linear interpolation properties are satisfied by imposing appropriate boundary conditions. These weights are the solution to a quadratic programming problem, which means they are somewhat slower to compute than the previous methods. Another limitation of this method is that it requires a volume discretization, which is often difficult to obtain for surfaces in 3D (see Chapter 8). The auxiliary interior vertices of the volume discretization increase the complexity of the precomputation. Nevertheless, we have found that when a volume discretization is available, these weights produce the highest quality results.

When computing both bone weights and endpoint weights from scratch, BBW is preferred since the same implementation produces high quality weights for both bones and endpoints (see Figure 6.10). When bone weights already exist, the additional endpoint weights obtained by BBW and by extending BH are both high quality. In many cases deformations using the endpoint weights obtained with these methods and the same bone weights are virtually identical (see Figure 6.11 and Figure 6.12). However, the extension of BH may suffer from the fall-off effect in regions beyond endpoints (see Figure 6.13).

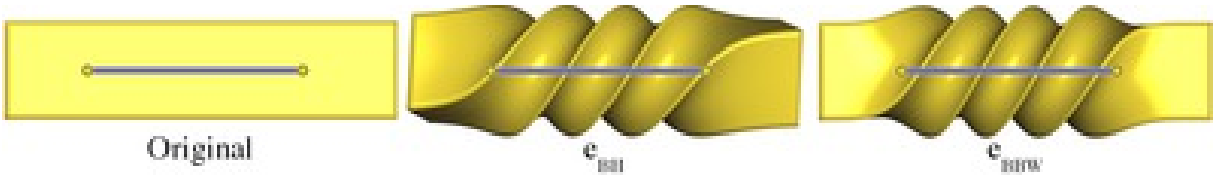


Figure 6.13: *A single bone controls a box in 3D (left). A 360 degree twist is applied to an endpoint using BH endpoint weights e_{BH} (middle). Notice the fall-off effect in the regions beyond each handle: the full twist is not reached. The same twist is applied using BBW endpoint weights e_{BBW} (right). These weights have proper locality, so the full twist is achieved.*

Figure 6.14: *With our method, bone joints may be freely dragged about by the user without worry that changes in bone length will cause explosion artifacts. Here an old photograph of Max Schmeling is brought to life.*

6.3 Implementation and results

We implemented stretchable, twistable bones skinning (STBS) using LBS and DQS as the underlying transformation blending mechanisms. Our implementation supports manually painting bone and endpoint weights or automatically computes them using [Baran and Popović 2007] (BH) or bounded biharmonic weights (BBW). Our timings for computing endpoint weight functions are similar or faster than the bone weight computation times originally reported in those works. When using BH our precomputation time per endpoint weight function is on the order of milliseconds in 2D and seconds in 3D. When using BBW our precomputation time per endpoint weight function is less than a second in 2D and on the order of tens of seconds in 3D, where we use heavily graded tetrahedral meshes for the volume discretization. Endpoint weights can be computed faster than bone weights by taking advantage of the fact that we only care about a bone's endpoint weights in regions of the domain where the bone's bone weight function is greater than zero. In our examples, we limit the optimization to the portion of the domain with corresponding bone weights greater than 1e-7.

A vertex shader implementation of STBS does not differ much from that of LBS or DQS. It requires loading the additional endpoint weights into constant memory once per deformation session, and in 3D transferring the new per-endpoint twist parameters along with the usual bone transformations. Finally, the skinning equation is replaced with either Equation (6.5) or the dual quaternion form of Equation (6.6).

The automatic weights are precomputed before the user begins applying transformations to the bones. In terms of efficiency at deformation time, our method requires twice as many weights as LBS or DQS: it stores the same bone weights and the extra endpoint weights per bone. The original LBS or DQS require respective 8 or 12 shader operations per shape vertex per bone [Kavan et al. 2008]. Because the transformations in Equation (6.6) are not constant over

Figure 6.15: *A dog's yawn is exaggerated with stretchable bones.*

the shape, using STBS with LBS or DQS as the underlying blending method requires extra operations to build the transformation at each shape vertex. In our implementation we count this as an extra 12 operations per shape vertex per bone.

Twisting and stretching bones richly expands the space of possible deformations without forfeiting the rigid skeleton metaphor or modifying existing bones and bone weights. In Figure 6.7, a skeleton rigged to a human properly twists, stretches and bends its arm using STBS. Blending transformations between bones as dual quaternions corrects rotational artifacts when bending at joints. Compare to the resulting deformation produced by LBS or DQS along in Figure 6.8.

In 2D, the ability to stretch bones without worrying about "explosions" near endpoints enables real-time creation and playback of animated of images. Such stretching is necessary for creating foreshortening effects. Figure 6.14 shows stretchable bones deforming a single image into a life-like series of poses.

Endpoint weights allow interesting twisting in 3D to occur over large regions of a shape controlled by a single bone. In Figure 6.9, a human's entire torso is twisted smoothly using a single twistable bone. The endpoint weights in our system may be filtered interactively to alter their effect. The original weights, automatically computed using BBW, twist the human's chest too much. Using a spline filter, the endpoint weights are interactively adjusted so that the twist is concentrated in the abdomen.

Exaggeration is a cornerstone principle in animation. Stretchable bones facilitate stylized and exaggerated actions. In Figure 6.15, stretching the bones in a dog's mouth emphasizes a yawning action, avoiding distracting shape explosion artifacts.

6.4 Conclusion

We have shown that with only slight modifications to existing skinning equations, we are able to expand the space of deformations possible with standard rigid skeleton rigs. Our method does not change the rigid skeleton metaphor, nor does it modify existing skeletons or bone weights.

The additional endpoint weights required by our technique are feasibly painted by the user or computed using automatic point weight methods.

Though our skinning formula and extra weights expand the space of possible deformations, our method is still inherently limited by its simplicity. Like all skinning methods, STBS cannot prevent self-collisions, maintain global properties (e.g. total volume), minimize nonlinear deformation energies, or respond to physically based forces. An obvious extension would be to use STBS as a reduced deformable model for more complicated methods (e.g. as an extension of the next Chapter 7).

In future work, we would like to explore further the role of extra weight functions. For example, the weights used to control stretching and twisting do not have to be the same. It would be interesting to expose these as separate parameters. We would also like to consider the orthogonal problem of specifying stretchable, twistable bone transformations with inverse kinematics or procedural animation. Our endpoint weights have a clear geometric meaning, such that their computation does not require example poses, but fitting to example poses could allow more accurate weights and enable skinning animations with stretchable, twisting bones as in, e.g. [James and Twigg 2005, Kavan et al. 2010]. Finally, it would be interesting to control advanced effects such as muscle bulging by applying simple filters to existing endpoint weights.

7

Fast automatic skinning transformations

Skinning transformations are a popular way to articulate shapes and characters. However, traditional animation interfaces require all of the skinning transformations to be specified explicitly, typically using a control structure (a rig). We propose a system where the user specifies only a subset of the degrees of freedom and the rest are automatically inferred using nonlinear, rigidity energies. By utilizing a low-order model and reformulating our energy functions accordingly, our algorithm runs orders of magnitude faster than previous methods without compromising quality. In addition to the immediate boosts in performance for existing modeling and real-time animation tools, our approach also opens the door to new modes of control: disconnected skeletons combined with shape-aware inverse kinematics. With automatically generated skinning weights, our method can also be used for fast variational shape modeling.

7.1 Introduction

Articulation adds life to geometric shapes in two steps. The rigging stage establishes parameters that provide intuitive control of shape geometry. Once these parameters are specified, the deformation stage computes the actual shape to generate poses. One popular choice of these parameters are joint angles for hierarchies of rigid transformations because they match the skeletal structure of humans, animals, and other characters. However, the best set of articulation parameters depends on the task so free-form deformations, blend shapes, cages, and others are also used, often at the expense of tedious manual tuning.

As shape deformation needs to be computed at every animation frame for every character, this

Figure 7.1: *We achieve high quality deformations by minimizing a nonlinear energy function, while keeping our algorithm extremely fast: skinning transformations for 100 individually animated armadillos (86k triangles each) are computed at 30fps on a single CPU core.*

process must be extremely fast, especially in applications such as computer games and haptics. The most common method is linear blend skinning (LBS), whose articulation parameters are affine transformations, each associated with a *handle*. Originally, handles were just bones, but previous work has explored the use of point handles, region handles, cage vertices, and even abstract handles not associated with any well-defined shape and perhaps not directly controlled by the user.

Handle transformations that drive LBS are typically obtained from motion capture, physical simulation, or manual posing and keyframing. In many cases, however, it is desirable to only specify a subset of their degrees of freedom. For example, a motion capture sequence may not have enough data to fully define the spine configuration, or an artist may want to specify only a subset of controls and have the rest inferred in a reasonable way. In some cases, it is useful to specify only the translational component of a handle transformation.

This chapter presents a method for deforming a shape as naturally as possible when only a subset of the degrees of freedom are specified. Our method computes the unspecified degrees of freedom by minimizing an elastic energy over the shape [Chao et al. 2010]. Our method is extremely efficient, allowing it to be integrated into the skinning pipeline, with computational complexity often dominated by LBS. Using a single CPU thread, our approach can compute missing degrees of freedom for one hundred armadillos each with 17 handles and 86k triangles at 30 frames per second (see Figure 7.1).

To attain our high performance, we speed up the energy minimization in two ways. First, we express the energy only in terms of handle transformations. Second, we use the input skinning weights to determine parts of the shape that are likely to be transformed similarly and simplify the deformation energy accordingly. These two optimizations complement each other: the error incurred by simplifying the energy function tends to be orthogonal to the subspace of a good LBS rig. In fact, we show this reduction often visually improves the deformations by regularizing the energy function. Moreover, our method guarantees deformations as smooth as the input skinning weights.

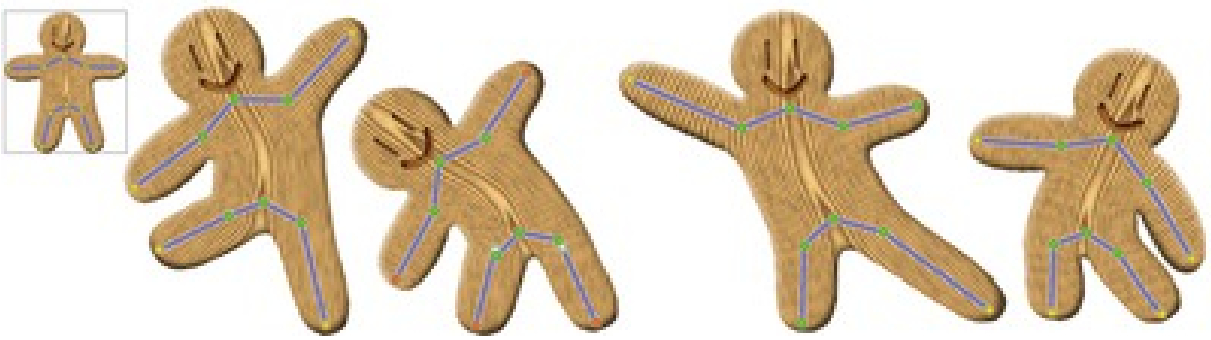


Figure 7.2: *Our method enables shape-aware inverse kinematics, even when skeletons are disconnected.*

The ability to leave some handle transformations unspecified opens up new possibilities for both posing and rigging. For posing, computing missing degrees of freedom results in behavior that is similar to inverse kinematics, but shape-aware. For rigging, the user can deliberately specify extra handles in areas that require additional flexibility without having to worry about their transformations. Additionally, extra handles can offset some undesirable behavior of linear blend skinning and allow our method to be applied to general shape deformation. Given manipulation handles, weights automatically generated by previous works can be augmented with additional extra weights attached to abstract handles. These additional weights are generated with a simple algorithm. With them our method produces results on par with high-quality nonlinear shape deformation methods [Botsch et al. 2007b], but running orders of magnitude faster.

7.2 Related work

Inverse kinematics. IK infers missing degrees of freedom for deformation under some constraints. Traditionally, IK is performed on the skeleton of the character, irrespective of the shape itself, and cannot handle disconnected skeletons or surface handles. MeshIK methods [Sumner et al. 2005, Fröhlich and Botsch 2011] use deformation examples to navigate in the space of plausible poses while interpolating prescribed surface region handles. To speed up the MeshIK optimization, a reduced model similar to ours has been presented [Der et al. 2006]. Unlike our model, it requires examples to infer a reduced model and the energy objective. At runtime, they must factor a dense matrix at every iteration, whereas in our method this can be pre-computed, since the system matrix remains unchanged.

Improving skinning quality. Our weights enrichment strategy is related to previous work aimed at improving skinning quality. Several works used example shapes for this purpose [Lewis et al. 2000, Wang et al. 2007]; similarly to us, Mohr and Gleicher [2003] enriched the space of the original weights, but using shape examples. Nonlinear skinning techniques improve skinning deformations quality without additional data [Kavan et al. 2008], but the nonlinear nature makes such skinning inconvenient as a reduced model. Weights defined for abstract handles can be used to automatically approximate nonlinear skinning methods expressed in closed-form with LBS [Kavan et al. 2009]. Spline skeletons

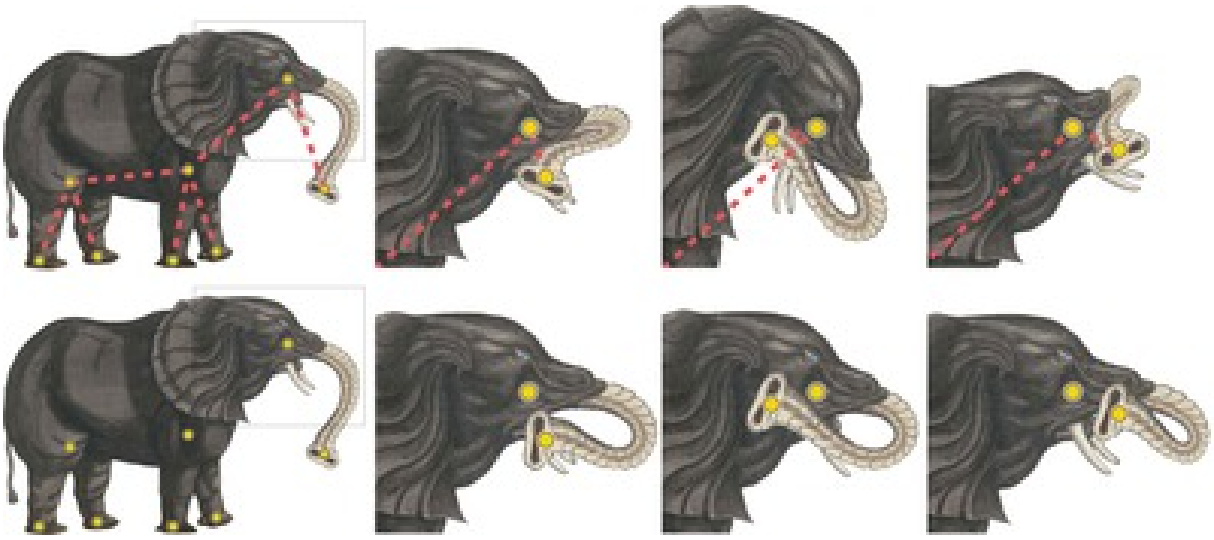


Figure 7.3: *Elephant controlled by 7 points. The user-provided* pseudo-edges *(see Section 4.3.4) fail to produce the expected buckling (bottom, our method).*

[Forstmann and Ohya 2006, Forstmann et al. 2007, Yang et al. 2006] often produce better skinning deformations, but again, they make skinning nonlinear. Others, [Wang and Phillips 2002, Merry et al. 2006, Kavan and Sorkine 2012], generalize the weight functions to matrix-valued or vector-valued forms, modifying the basic LBS formulation (see also Chapter 6). Contrary to these approaches, our method is easier to use and fits into the existing animation pipeline since we only rely on the standard LBS, which is of practical importance due to the number of optimized platform-specific LBS implementations.

Deformation. Recent shape deformation research has concentrated on two main fronts: modeling *physically plausible* deformations (typically by means of variational optimization), and exploring very fast and parallelizable closed-form deformations.

Many works deal with the question of how a shape should deform given arbitrary (user-defined) modeling constraints, typically prescribed locations for some points or regions on the surface. Several energy functionals that the deformed shape should minimize (under the modeling constraints) were proposed; these energies typically measure a form of elastic shape distortion. Quadratic energies that lead to linear optimization problems are discussed in the survey of Botsch and Sorkine [2008]. They are robust and can be optimized at interactive framerates for moderately sized meshes, but suffer from linearization artifacts. Nonlinear energies, e.g. [Botsch et al. 2006a, Au et al. 2006, Sorkine and Alexa 2007, Chao et al. 2010] provide higher-quality deformations but are slower to optimize, and their complexity grows nonlinearly with the size of the mesh. In this work, we take advantage of the special structure of some recently proposed "as-rigid-as-possible" (ARAP) elastic energies [Igarashi et al. 2005, Sorkine and Alexa 2007, Liu et al. 2008, Chao et al. 2010] and modify their formulation so that it can be extremely efficiently minimized in the subspace of skinning deformations.

The second stream of shape deformation research dates back to the FFD framework [Sederberg and Parry 1986] that avoids global variational optimization. Given a set of control objects (handles), such as cages or points on the shape or skeletons, the question

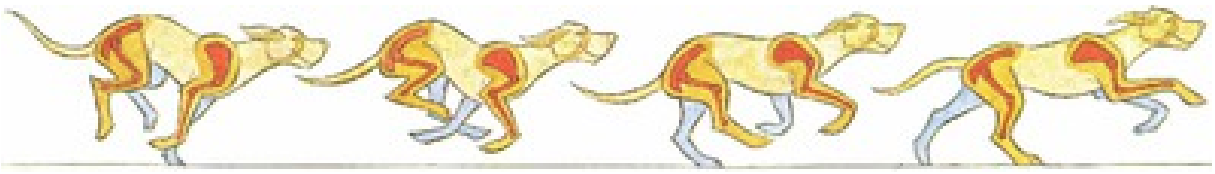


Figure 7.4: *Traditional animators sometimes sketch bones only in limbs, using them to infer the remaining shape [Blair 1994].*

is how to define the *influence* of each handle on each point of the shape. Then, the user-defined transformations at the handles are propagated to each point on the shape by simple local weighted combination, using the influence values. In the case of cages, the influences are defined as generalized barycentric coordinates (see e.g. [Ju et al. 2005, Joshi et al. 2007, Lipman et al. 2008, Weber et al. 2009]) and only translations need to be provided for the cage vertices. Higher-order barycentric coordinates [Langer and Seidel 2008] allow (and require) full affine transformations per cage vertex, offering more flexible deformations: they are equivalent to LBS (see Section 4.6). For skeletal bones several methods exist to define automatic weights [Baran and Popović 2007, Wareham and Lasenby 2008]; full affine transformations need to be provided per bone, and these are linearly combined using the LBS formula. Bounded biharmonic weights can be computed for all common handle types above (see Chapter 4). MLS deformations [Schaefer et al. 2006] use simple inverse-distance based weights but combine the handle influences in a nonlinear manner by local least-squares optimization.

The above approaches enjoy much better performance than variational methods due to their local, parallelizable nature. However, as mentioned, they either require the user to provide full affine transformations per handle, or manipulate cages, which may be difficult for some desired shapes and deformations. Simple heuristic solutions, such as "pseudo-edges" (Section 4.3.4) do not consider their effect on the resulting shape, leading to unintuitive response (see Figure 7.3).

Reduced models. Methods based on reduced models attempt to combine the benefits of variational and closed-form deformation methods. They minimize some shape energy, but instead of doing so on the shape itself, they employ a much smaller number of degrees of freedom and a simpler (faster) deformation subspace, such as cage-based deformations [Huang et al. 2006, Ben-Chen et al. 2009, Weber et al. 2009, Borosán et al. 2010], LBS with skeletons [Shi et al. 2007], LBS with point/region handles [Au et al. 2007, Sumner et al. 2007], or linear subspace of dominant modes of the deformation energy [Hildebrandt et al. 2011]. Such approaches achieve significant improvements in shape articulation, but struggle to reach rates required by time-critical applications while retaining high quality, flexible deformation, and artistic freedom. Many model-reduction schemes do not guarantee interpolation of the user-specified constraints, e.g. because the degrees of freedom are not sufficient to do so. With the exception of [Au et al. 2007], the reduced models are constructed per shape, irrespective of the user constraints (handles), and the deformations generally have a global nature as a result (when manipulating a handle, shape parts far away from it may move even if nearby handles are constrained). This also ignores any semantic information embedded in the handle description [Au et al. 2007] (see also Figure 7.19). Also note that even heavily parallelized GPU implementations such as [Weber et al. 2009] are still an order of magnitude slower than our method running on a single CPU core.

Among the above-mentioned techniques, [Au et al. 2007, Shi et al. 2007] are closest to ours in terms of the technical setup. Au et al. [2007] take user-provided surface-region handles and construct a reduced model by creating additional handles that are isolines of a harmonic propagation field sourced at the input handles. They then minimize a nonlinear Laplacian energy [Au et al. 2006] on the LBS subspace created by those handles. Contrary to our method, their optimization still requires updates of the entire unreduced mesh in each iteration, making it slow in comparison. MeshMuppetry [Shi et al. 2007] uses a skeletal LBS reduced model to optimize a similar nonlinear Laplacian energy and additional constraints such as the balance of the character. They optimize both the skinning weights and the skeleton transformations simultaneously, which again results in time complexity dependent on the number of primitives. In contrast, we assume fixed skinning weights provided by the user and design deformations that *respect* the artistic intention encapsulated in the weights. This allows us to formulate an algorithm with time complexity independent of the number of primitives of the input model. We also explore new modes of control, such as combination of skeletal and point controls, as well as disconnected skeletons (see Figure 7.2). Disconnected skeletons are especially interesting as traditional animators often employ them as a starting point for creating lifelike characters (see Figure 7.4).

Decimating the input mesh, Manson and Schaefer [2011] optimize an ARAP energy [Sorkine and Alexa 2007] at a coarse resolution. They then reintroduce fine details by constrained local optimization. While faster than full-scale optimization, the nonlinear upsampling step is still orders of magnitude slower than our skinning model.

Our use of skinning weights to cluster vertices and further simplify optimization is related to previous works that assume rigid patches during example-driven deformation. Pekelny and Gotsman [2008] find clusters to track the motion of piecewise rigid shapes. Huang et al. [2008] optimize the assignment of overlapping rigid clusters to define a locally as-rigid-as-possible deformation for shape registration. Most similar to our method, Der et al. [2006] cluster surface vertices to simplify a nonlinear optimization over a reduced, skinning model. Their method uses example poses to define skinning weights and then clusters vertices according to their k-greatest skinning weights. Simple clustering was also used for physically based, shape-matching deformations in the opposite sense of our method: to increase degrees of freedom [Müller et al. 2005].

Reduced models are also common in physical, elastic simulation, typically employing dense subspace models [Barbič and James 2005, An et al. 2008], where they are used to efficiently integrate the equations of motion. As discussed above, skinning provides the advantage of local control and is starting to be used in physical simulation [Gilles et al. 2011, Faure et al. 2011].

7.3 Method

Denote by $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^d$ ($d = 2$ or 3) the rest-pose vertex positions of the input mesh $\mathcal{M}$. The user specifies a set of control handles $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_m\}$, which could be bones of a skeleton, points or regions on the shape. To deform the shape, the user must to specify affine transformations $\mathbf{T}_j \in \mathbb{R}^{d\times(d+1)}$ for each handle $\mathbf{h}_j$. We denote the deformed vertex positions by $\mathbf{v}'_1, \ldots, \mathbf{v}'_n$. The handle transformations lead to modeling constraints $\mathbf{h}'_j = \mathbf{T}_j\mathbf{h}_j$.

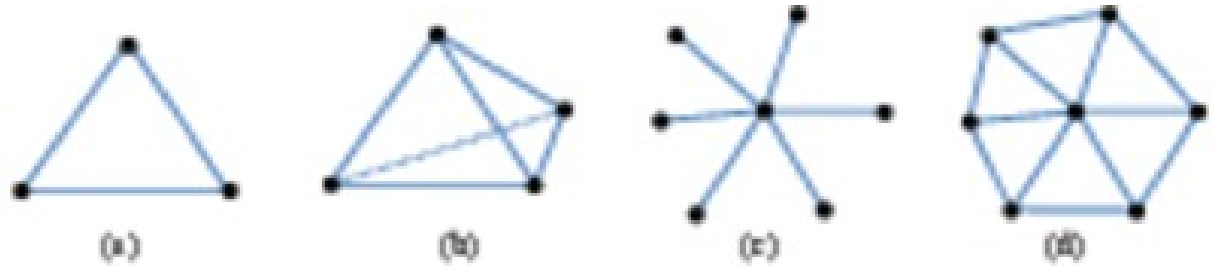


Figure 7.5: *Typical edge sets: triangle (a), tetrahedron (b), spokes (c), spokes and rims (d).*

There are many methods to compute a deformed shape based on the given modeling constraints. The fastest method is linear blend skinning (LBS), which in addition to the inputs above also requires skinning weight functions $w_j : \mathcal{M} \to \mathbb{R}$. For each point $\mathbf{p}$ on the shape, $w_j(\mathbf{p})$ specifies the amount of influence of $\mathbf{T}_j$ on $\mathbf{p}$. The skinning weights are often painted manually by specialized rigging artists, but may also be computed automatically using recent methods (see Chapters 4 & 5). The LBS deformation of $\mathcal{M}$'s vertices is then given by

$$\mathbf{v}_i' = \sum_{j=1}^{m} w_j(\mathbf{v}_i)\ \mathbf{T}_j \begin{pmatrix} \mathbf{v}_i \\ 1 \end{pmatrix}. \tag{7.1}$$

This formula can be equivalently expressed in matrix form:

$$\mathbf{V}' = \mathbf{M}\mathbf{T}, \tag{7.2}$$

where $\mathbf{V}' \in \mathbb{R}^{n \times d}$ is the matrix whose rows are the deformed vertex positions, $\mathbf{M} \in \mathbb{R}^{n \times ((d+1)m)}$ is the matrix combining rest-pose vertex positions $\mathbf{v}_i$ with vertex weights $w_j(\mathbf{v}_i)$, and $\mathbf{T} \in \mathbb{R}^{((d+1)m) \times d}$ stacks transposed transformation matrices $\mathbf{T}_j$ (see e.g. [Kavan et al. 2010] for details).

7.3.1 Automatic degrees of freedom

Specifying all the degrees of freedom for all affine transformations $\mathbf{T}_1, \ldots, \mathbf{T}_m$ could be burdensome. It is often easier and more intuitive for the user to specify translations only, dragging the control handles around. However, if the $\mathbf{T}_j$'s consist of translations alone, Equation (7.1) leads to unnatural, sheared deformations. Many other linear deformation formulations suffer from the same problem of "translation-insensitivity" [Botsch and Sorkine 2008]: when specifying translations at handles, no local rotations are generated by the deformation, and the shape and its details are distorted as a result. We can also leave some transformations $\mathbf{T}_j$ entirely unconstrained, which corresponds to shape-aware inverse kinematics.

We propose *optimizing* the remaining degrees of freedom of the handle transformations $\mathbf{T}_j$ that the user did not specify explicitly. We wish to find such $\mathbf{T}_j$'s that the deformed shape minimizes a deformation energy E under the imposed modeling constraints and the LBS setting, Equation (7.2). We can write the user-specified modeling constraints as:

$$\underbrace{\begin{bmatrix} \mathbf{I}_{\text{full}} \\ \mathbf{M}_{\text{pos}} \end{bmatrix}}_{\mathbf{M}_{\text{eq}}} \mathbf{T} = \underbrace{\begin{bmatrix} \mathbf{T}_{\text{full}} \\ \mathbf{P}_{\text{pos}} \end{bmatrix}}_{\mathbf{P}_{\text{eq}}}, \tag{7.3}$$

where $\mathbf{I}_{\text{full}}$ are identity blocks corresponding to the fully constrained transformations and $\mathbf{T}_{\text{full}}$ are their user-provided values (as in traditional skinning) and $\mathbf{M}_{\text{pos}}$ are rows of $\mathbf{M}$ corresponding to positional constraints on $\mathcal{M}$'s vertices and $\mathbf{P}_{\text{pos}}$ their values. All constraints together can be concisely written as $\mathbf{M}_{\text{eq}}\mathbf{T} = \mathbf{P}_{\text{eq}}$, where $\mathbf{M}_{\text{eq}} \in \mathbb{R}^{c\times((d+1)m)}$ and $\mathbf{P}_{\text{eq}} \in \mathbb{R}^{c\times d}$ and our construction guarantees they are feasible and full rank.

To obtain the remaining, unconstrained degrees of freedom of $\mathbf{T}_j$'s (i.e., linear components of positionally constrained handles and all components of unconstrained handles), we propose minimizing a shape deformation energy $E : \mathbb{R}^{n\times d} \to \mathbb{R}^+$. $E(\mathbf{V}')$ measures the deformation between the rest-pose shape $\mathcal{M}$ and the deformed shape $\mathcal{M}'$. We focus on energies E measuring local deviation from rigidity, advantageous for good detail preservation and intuitive elastic behavior [Sorkine and Alexa 2007, Liu et al. 2008, Chao et al. 2010]. This family of energy functions can be expressed as:

$$E(\mathbf{V}', \mathbf{R}) = \frac{1}{2}\sum_{k=1}^{r}\sum_{(i,j)\in\mathcal{E}_k} c_{ijk}\|(\mathbf{v}'_i - \mathbf{v}'_j) - \mathbf{R}_k(\mathbf{v}_i - \mathbf{v}_j)\|^2, \tag{7.4}$$

where $\mathbf{R}_1, \ldots, \mathbf{R}_r \in SO(d)$ are local rotations, $\mathcal{E}_1, \ldots, \mathcal{E}_r$ are their corresponding sets of edges (see Figure 7.5), and $c_{ijk} \in \mathbb{R}$ are weighting coefficients, typically the familiar cotangent weights [Chao et al. 2010]. It is convenient to rewrite Equation (7.4) in matrix notation. We start by separating terms quadratic and linear in $\mathbf{V}'$:

$$E(\mathbf{V}', \mathbf{R}) = \frac{1}{2}E_2(\mathbf{V}') - E_1(\mathbf{V}', \mathbf{R}) + \text{const}, \tag{7.5}$$

$$E_2(\mathbf{V}') = \sum_{k=1}^{r}\sum_{(i,j)\in\mathcal{E}_k} c_{ijk}(\mathbf{v}'_i - \mathbf{v}'_j)^\mathsf{T}(\mathbf{v}'_i - \mathbf{v}'_j), \tag{7.6}$$

$$E_1(\mathbf{V}', \mathbf{R}) = \sum_{k=1}^{r}\sum_{(i,j)\in\mathcal{E}_k} c_{ijk}(\mathbf{v}'_i - \mathbf{v}'_j)^\mathsf{T}\mathbf{R}_k(\mathbf{v}_i - \mathbf{v}_j), \tag{7.7}$$

The quadratic term can be written in matrix form as:

$$E_2(\mathbf{V}') = \sum_{k=1}^{r}\text{tr}(\mathbf{C}_k\mathbf{A}_k^\mathsf{T}\mathbf{V}'\mathbf{V}'^\mathsf{T}\mathbf{A}_k), \tag{7.8}$$

where $\mathbf{A}_k \in \mathbb{R}^{n\times|\mathcal{E}_k|}$ is directed incidence matrix corresponding to (arbitrarily oriented) edges $\mathcal{E}_k$ and $\mathbf{C}_k \in \mathbb{R}^{|\mathcal{E}_k|\times|\mathcal{E}_k|}$ is a diagonal matrix with weights c_{ijk}. Due to the properties of the trace, we can rewrite this as:

$$E_2(\mathbf{V}') = \text{tr}\left(\mathbf{V}'^\mathsf{T}\left(\sum_{k=1}^{r}\mathbf{A}_k\mathbf{C}_k\mathbf{A}_k^\mathsf{T}\right)\mathbf{V}'\right) \tag{7.9}$$

$$= \text{tr}(\mathbf{V}'^\mathsf{T}\mathbf{L}\mathbf{V}'), \tag{7.10}$$

where we denoted the middle sum as $\mathbf{L} \in \mathbb{R}^{n\times n}$, which for all energies discussed in the literature [Sorkine and Alexa 2007, Liu et al. 2008, Chao et al. 2010] is the standard symmetric, cotangent Laplacian matrix (up to a constant scale factor). Similarly, the linear term can be

written as:

$$E_1(\mathbf{V}', \mathbf{R}) = \sum_{k=1}^{r} \mathrm{tr}(\mathbf{C}_k \mathbf{A}_k^\mathsf{T} \mathbf{V}' \mathbf{R}_k \mathbf{V}^\mathsf{T} \mathbf{A}_k) \tag{7.11}$$

$$= \mathrm{tr}\left(\left(\sum_{k=1}^{r} \mathbf{R}_k \mathbf{V}^\mathsf{T} \mathbf{A}_k \mathbf{C}_k \mathbf{A}_k^\mathsf{T} \right) \mathbf{V}' \right) \tag{7.12}$$

$$= \mathrm{tr}(\mathbf{R}\mathbf{K}\mathbf{V}'), \tag{7.13}$$

where $\mathbf{R} = (\mathbf{R}_1, \ldots, \mathbf{R}_r)$ and $\mathbf{K} \in R^{dr \times n}$ stacks differential rest pose coordinates $\mathbf{V}^\mathsf{T} \mathbf{A}_k \mathbf{C}_k \mathbf{A}_k^\mathsf{T}$. We therefore obtain:

$$E(\mathbf{V}', \mathbf{R}) = \frac{1}{2}\mathrm{tr}(\mathbf{V}'^\mathsf{T} \mathbf{L} \mathbf{V}') - \mathrm{tr}(\mathbf{R}\mathbf{K}\mathbf{V}') + \mathrm{const.} \tag{7.14}$$

In this form, it is easy to formulate our reduced-order optimization by plugging in the linear blend skinning formula $\mathbf{V}' = \mathbf{M}\mathbf{T}$:

$$\begin{aligned} \underset{\mathbf{T},\mathbf{R}}{\mathrm{argmin}} \quad & \frac{1}{2}\mathrm{tr}(\mathbf{T}^\mathsf{T} \tilde{\mathbf{L}} \mathbf{T}) - \mathrm{tr}(\mathbf{R}\tilde{\mathbf{K}}\mathbf{T}) \\ \text{subject to} \quad & \mathbf{M}_{\mathrm{eq}}\mathbf{T} = \mathbf{P}_{\mathrm{eq}}, \;\; \mathbf{R} \in SO(d)^r, \end{aligned} \tag{7.15}$$

where $\tilde{\mathbf{L}} = \mathbf{M}^\mathsf{T}\mathbf{L}\mathbf{M}$ and $\tilde{\mathbf{K}} = \mathbf{K}\mathbf{M}$.

To solve this optimization problem, we follow the local-global approach of Sorkine and Alexa [2007]. First we fix $\mathbf{T}$ and solve for $\mathbf{R}$ (local step). Then we fix $\mathbf{R}$ and solve for $\mathbf{T}$ (global step).

Local step. For fixed $\mathbf{T}$, we are left maximizing $\mathrm{tr}(\mathbf{R}\mathbf{S})$, where $\mathbf{S} = \tilde{\mathbf{K}}\mathbf{T}$ is constant. This amounts to maximizing each $d \times d$ block $\mathrm{tr}(\mathbf{R}_i \mathbf{S}_i)$ individually. It is well known [Sorkine and Alexa 2007] that the rotation maximizing the trace is $\mathbf{R}_i = \mathbf{Q}_i^\mathsf{T} \mathbf{U}_i^\mathsf{T}$, where $\mathbf{S}_i = \mathbf{U}_i \mathbf{\Sigma}_i \mathbf{Q}_i$ is the singular value decomposition. For $d = 3$ we employ the optimized SVD routines by McAdams and colleagues [2011] that avoid reflections, i.e., guarantee $\det(\mathbf{R}_i) > 0$.

Global step. For fixed $\mathbf{R}$, Equation (7.15) turns into a quadratic minimization problem with linear equality constraints. The constraints can be handled by introducing a matrix $\mathbf{\Lambda} \in \mathbb{R}^{c \times d}$ of Lagrange multipliers (see Section 2.2.2), arriving at the Lagrangian:

$$\mathscr{L} = \frac{1}{2}\mathrm{tr}(\mathbf{T}^\mathsf{T} \tilde{\mathbf{L}} \mathbf{T}) - \mathrm{tr}(\mathbf{R}\tilde{\mathbf{K}}\mathbf{T}) + \mathrm{tr}(\mathbf{\Lambda}^\mathsf{T}(\mathbf{M}_{\mathrm{eq}}\mathbf{T} - \mathbf{P}_{\mathrm{eq}})). \tag{7.16}$$

Recalling standard matrix calculus identities, we differentiate:

$$\frac{\partial \mathscr{L}}{\partial \mathbf{T}} = \tilde{\mathbf{L}}\mathbf{T} - \tilde{\mathbf{K}}^\mathsf{T}\mathbf{R}^\mathsf{T} + \mathbf{M}_{\mathrm{eq}}^\mathsf{T}\mathbf{\Lambda} \tag{7.17}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{\Lambda}} = \mathbf{M}_{\mathrm{eq}}\mathbf{T} - \mathbf{P}_{\mathrm{eq}}, \tag{7.18}$$

where we exploit the symmetry of $\tilde{\mathbf{L}}$. Setting these derivatives to zero, we obtain:

$$\begin{bmatrix} \tilde{\mathbf{L}} & \mathbf{M}_{\mathrm{eq}}^T \\ \mathbf{M}_{\mathrm{eq}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{T} \\ \mathbf{\Lambda} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{K}}^\mathsf{T}\mathbf{R}^\mathsf{T} \\ \mathbf{P}_{\mathrm{eq}} \end{bmatrix}. \tag{7.19}$$

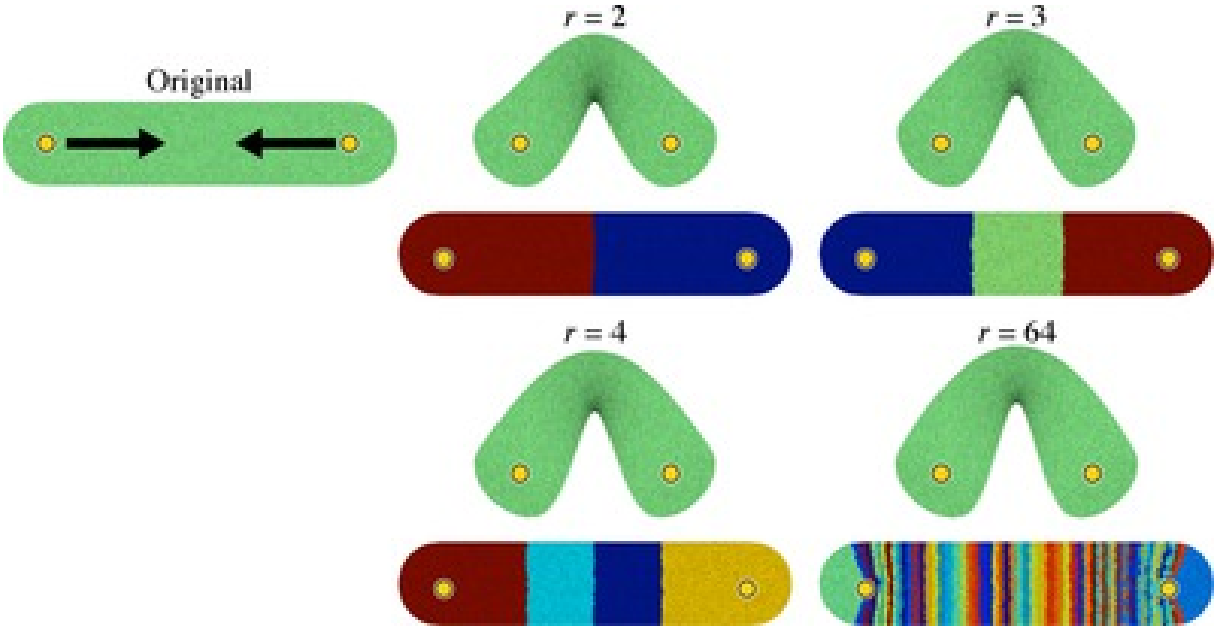


Figure 7.6: *2D deformation with various numbers of rotation clusters, r. Notice that even if only two rotation clusters are used the deformation is smooth and finds a reasonable shape. Increasing the number of rotation clusters improves this shape with diminishing returns.*

With enough constraints to determine all translational degrees of freedom of $\tilde{\mathbf{L}}$, the system matrix is non-singular. We can thus precompute its inverse $\mathbf{\Pi}$ and express the solution as:

$$\mathbf{T} = [\mathbf{\Pi}_1 \; \mathbf{\Pi}_2] \begin{bmatrix} \tilde{\mathbf{K}}^\mathsf{T}\mathbf{R}^\mathsf{T} \\ \mathbf{P}_{\text{eq}} \end{bmatrix} = \mathbf{\Gamma}_{\text{solve}}\mathbf{R}^\mathsf{T} + \mathbf{\Phi}_{\text{solve}},$$

where $\mathbf{\Gamma}_{\text{solve}} \in \mathbb{R}^{(d+1)m \times dr}$ and $\mathbf{\Phi}_{\text{solve}} \in \mathbb{R}^{(d+1)m \times d}$ can be precomputed[1].

Final algorithm. One iteration of the reduced alternating optimization can be summarized as follows:

1. $\mathbf{S} = \tilde{\mathbf{K}}\mathbf{T}$,
2. Turn $\mathbf{S}$ into $\mathbf{R}$ using SVDs,
3. $\mathbf{T} = \mathbf{\Gamma}_{\text{solve}}\mathbf{R}^\mathsf{T} + \mathbf{\Phi}_{\text{solve}}$.

The only nonlinear step is 2, consisting of r SVDs of $d \times d$ matrices. While the linear steps 3 and 1 could be combined together, it is typically more efficient to use two $dr \times (d+1)m$ and $(d+1)m \times dr$ matrices rather than one $dr \times dr$ matrix. The complexity is independent of the number of vertices n, however, with r on the order of number of primitives, we still cannot guarantee real-time framerates. In the following section we discuss how to select representative rotations so that we need only $r = O(m)$ while obtaining results similar to much higher r.

[1] In Chapter 2 we repeated wrote that we *never* compute system inverses, however in this case we really do. This allows the subsequent, further precomputation. We observe robust inversions using both LU and SV decomposition using the EIGEN [Guennebaud et al. 2010] or MATLAB [MATLAB 2012] libraries.

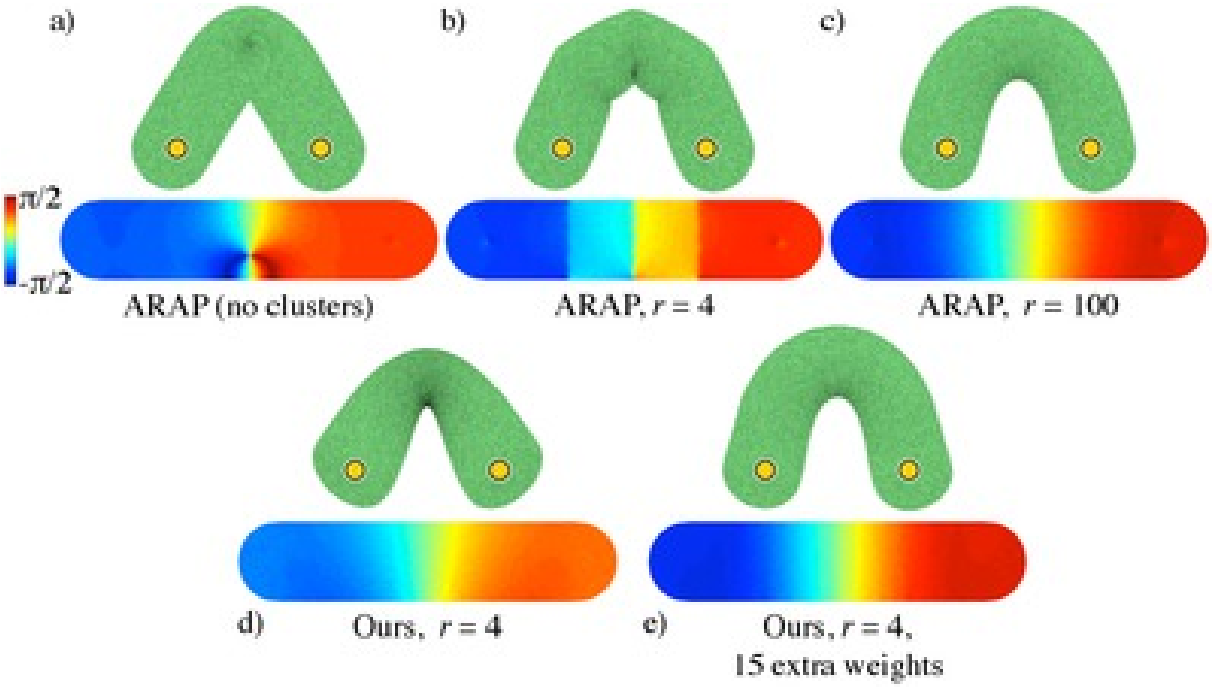


Figure 7.7: *Top: unreduced per-triangle ARAP introduces a singularity in the rotation field (visualized below). Per-triangle ARAP energy with $r = 4$ prevents this singularity; the non-smooth transitions are no longer noticeable with $r = 100$. Bottom: our method (using smooth weights) is always smooth, even with $r = 4$. After enriching our deformation subspace with 15 additional weights we obtain the desired result.*

7.3.2 Rotation clusters

Linear blend skinning constrains the resulting deformations to a small linear subspace, where the motions of neighboring vertices are typically highly correlated. Therefore, it seems unnecessary to estimate local rotations at each edge set. Instead, we propose clustering vertices undergoing similar deformations. Instead, we cluster the vertices into $\mathcal{V}_1, \ldots, \mathcal{V}_r \subseteq \{1, \ldots, n\}$ so that their best-fit rotations can stand in for rotation of each edge set.

Linear blend skinning deformations are governed by vertex weights w_i and vertices with similar weights undergo similar rotations (regardless of spatial proximity). Therefore, we cluster vertices into r clusters based on their Euclidean distance in *weight space*, where for each vertex $\mathbf{v}_i$ we assign a vector of weights $(w_1(\mathbf{v}_i), \ldots, w_m(\mathbf{v}_i))$. In this representation rigid components comprised of vertices with identical weights collapse to a single point and performing k-means clustering achieves the desired grouping.

The number of clusters r can be used to trade off quality for speed. However, we observed that typically $r = O(m)$ is sufficient (we often simply set $r = 2m$), and higher numbers do not result in further visual improvements (see Figure 7.6). Interestingly, rotation clusters often improve deformation quality even with a unreduced formulation where each vertex position is an unknown. The clusters act as a regularization term that helps avoid singularities, though with small r the unreduced optimization suffers from lack of smoothness (see Figure 7.7b). Let us consider smoothness of discrete weight functions or deformation fields in an informal

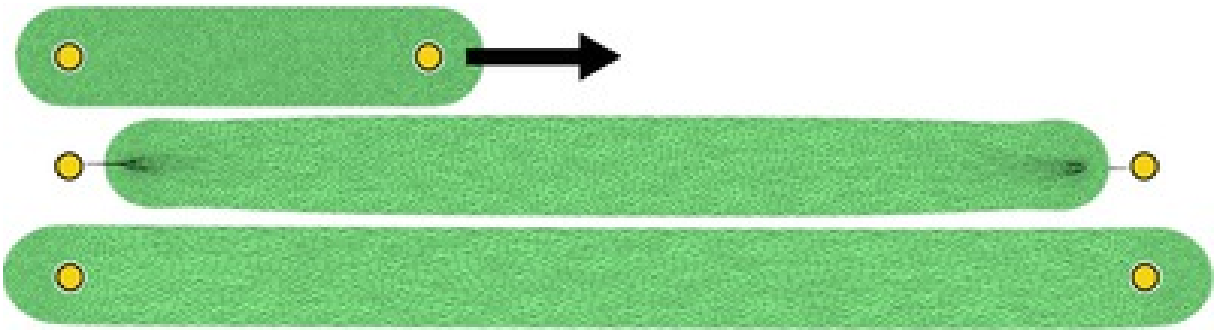


Figure 7.8: *The cigar shape (top) is stretched using unreduced per-triangle ARAP which introduces a C^1 discontinuity at each point handle (middle). Our subspace is restricted to smooth deformations, so our result is also smooth (bottom).*

perceptual sense (as is often the case for manually painted weights) or in terms of convergence to a C^1 function (see Chapter 3). Then, because our final deformation is dictated by linear blend skinning, it is always as smooth as the input skinning weights w_i. This smoothness is thus guaranteed at cluster boundaries and also near handle boundaries, where unreduced variational methods typically produce sharp discontinuities (see Figures 7.7 and 7.8)[2]

7.3.3 Additional weight functions

Linearly blending handle transformations can be insufficient to create natural deformations. To enrich the space of attainable deformations, we can automatically create a set of abstract handles whose transformations are entirely determined by our optimization. For example, bending a cigar using two handles at its ends will tend to introduce a pinch (see Figure 7.7d). By generating additional weights functions along the cigar, we allow our optimization to find a more pleasing deformation (see Figure 7.7e).

In theory, an arbitrary weight function, appended as additional four columns of matrix $\mathbf{M}$, cannot increase the deformation energy because the original deformation subspace is contained in the new one. Still, constructing additional weights that keep the deformation fair is not straightforward. Simply adding additional bounded biharmonic weights (see Chapter 4) tends to introduce interpolation artifacts similar to Shepard interpolation (see the illustration in [Lewis et al. 2000]). Using low-frequency Laplacian eigenvectors is tempting, but their global support can hurt local control and LBS performance. Suitable additional weights should be smooth, localized, and preserve the nature of the original weights (especially if manually painted).

Partition of unity is usually required of skinning weights to ensure translation invariance. Due to our translation invariant energies, our method (somewhat surprisingly) requires only the original

[2]These discontinues show up in two ways. First, as is evident by the local-global optimization method, the ARAP solutions are finally solutions to a Poisson equation which will be at best C^0 at fixed values (see Chapter 4). This happens with any type of handle: bone, region, point. The fact that isolated fixed point values are result in discontinuous indicator functions for continuous Laplace equations, exacerbates this problem. Second, the discrete ARAP energies assign *finite* energy to what amounts to a rotational singularity (see Figure 7.7a), which should receive infinite energy and thus be avoided. Thanks to Peter Schröder for clarifying this.

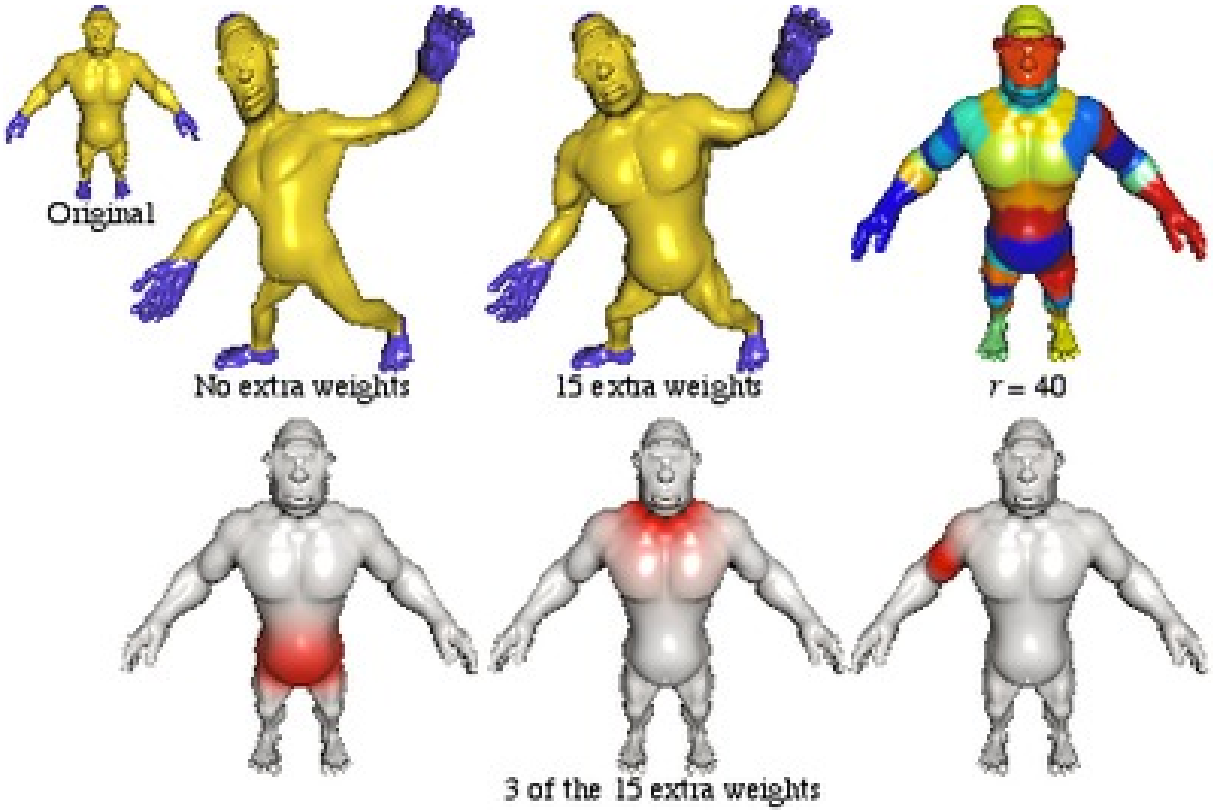


Figure 7.9: *With five fully constrained region handles, our method reduces to LBS with its volume loss issues. Enriching our subspace with 15 additional weight functions (shown in red) allows our method to find more natural deformations.*

weights to partition unity, the additional weights can be arbitrary. This is because a global translation by vector $\mathbf{t} \in \mathbb{R}^d$ is in our skinning subspace (shifting all of the original transformations by $\mathbf{t}$ and keeping the additional constraints unchanged shifts all vertices by $\mathbf{t}$), and therefore if the user translates all constraints by $\mathbf{t}$, the resulting optimized transformations exactly reproduce this translation. This is a consequence of the fact the value of our energy functions (Equation 7.4) remains unchanged when translating all vertices by $\mathbf{t}$.

Also, the scale of the additional weights does not matter because the optimization considers all linear combinations. However, we do require them to be zero at handles, i.e., where one of the original weights has value one. This way we ensure that our constraints are trivially feasible.

To preserve the deformation properties of the original weights, we propose embedding our mesh $\mathcal{M}$ in weight space, the same as when computing rotation clusters. To this end we create smooth isotropic cubic B-spline basis functions in weight space, centered at some seed locations $\mathbf{s}_1, \ldots, \mathbf{s}_p \in \mathbb{R}^m$. Let $\mathbf{e}_1, \ldots, \mathbf{e}_m \in \mathbb{R}^m$ be unit m-simplex vertices, $e_{i,j} = \delta_{i,j}$. To obtain a good distribution of the seed locations, we employ a discrete multi-dimensional version of optimized farthest points [Schlömer et al. 2011]. In each iteration, we choose a new location for seed $\mathbf{s}_i$:

$$\mathbf{s}_i = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{M}} d_i(\mathbf{x}), \tag{7.20}$$

$$d_i(\mathbf{x}) = \min(\min_{j \neq i} \|\mathbf{x} - \mathbf{s}_j\|, \ \tfrac{1}{2} \min_k \|\mathbf{x} - \mathbf{e}_k\|), \tag{7.21}$$

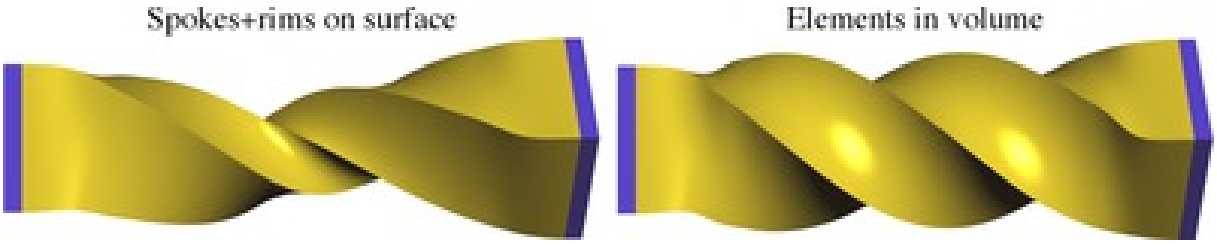


Figure 7.10: *Our reduced model preserves the nature of different energy functions: spokes and rims (left) behave like thin shells, while tetrahedra (right) correspond to volumetric elasticity.*

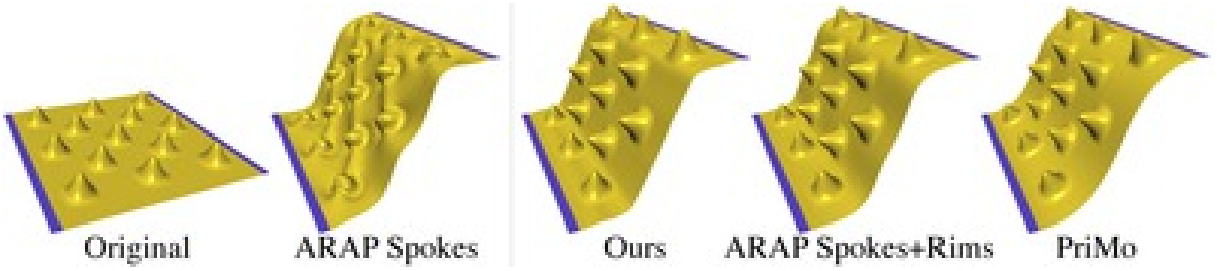


Figure 7.11: *Left: unreduced spokes-only ARAP energy [Sorkine and Alexa 2007] produces artifacts due to indefinite terms in the energy function. Right: our method utilizing the spokes and rims energy [Chao et al. 2010] fixes these issues and surprisingly accurately models behavior of bumpy rubber, similar to the unreduced solution and to behavior observed in silicone ice cube trays. Meanwhile, PriMo [Botsch et al. 2006a] produces a fair wave corresponding to a thin metal sheet, but seems to ignore the bumps.*

i.e., moving $\mathbf{s}_i$ as far as possible from the other seeds and corners. We push $\mathbf{s}_i$ from the corners twice as far to facilitate the requirement of vanishing additional weights at all $\mathbf{e}_i$. Because Delaunay triangulation in higher dimensions is prohibitively expensive, we estimate the argmax in Equation (7.20) by random sampling on $\mathcal{M}$ in weight space. The algorithm is guaranteed to converge because the distance between two closest points cannot decrease. For the final $\mathbf{s}_i$, we define the additional weight function simply as $B(\|\mathbf{x} - \mathbf{s}_i\|/(2d_i(\mathbf{s}_i)))$, where $B(t) : [0,1] \to [0,1]$ is a cubic B-spline basis function.

This way of generating additional weights is only one of many possible, but we enjoy its simplicity and speed. Variational methods are much slower (see Chapter 4). In our experiments, subspaces enriched with these additional weights support natural deformations with respect to our energy functions. Especially in situations with a small number of original weights we observe significant improvement of the deformation quality (see Figure 7.9).

7.4 Results

We experimented with a number of previously discussed ARAP energies. For 2D images, we use the per-triangle energy [Liu et al. 2008]. For 3D shapes, the user's choice of energy controls the desired behavior (see Figure 7.10). The surface-based spokes and rims energy [Chao et al. 2010] treats models as thin shells while eliminating the artifacts of the spokes energy [Sorkine and Alexa 2007] (see Figure 7.11). Alternatively, the per-tetrahedron energy

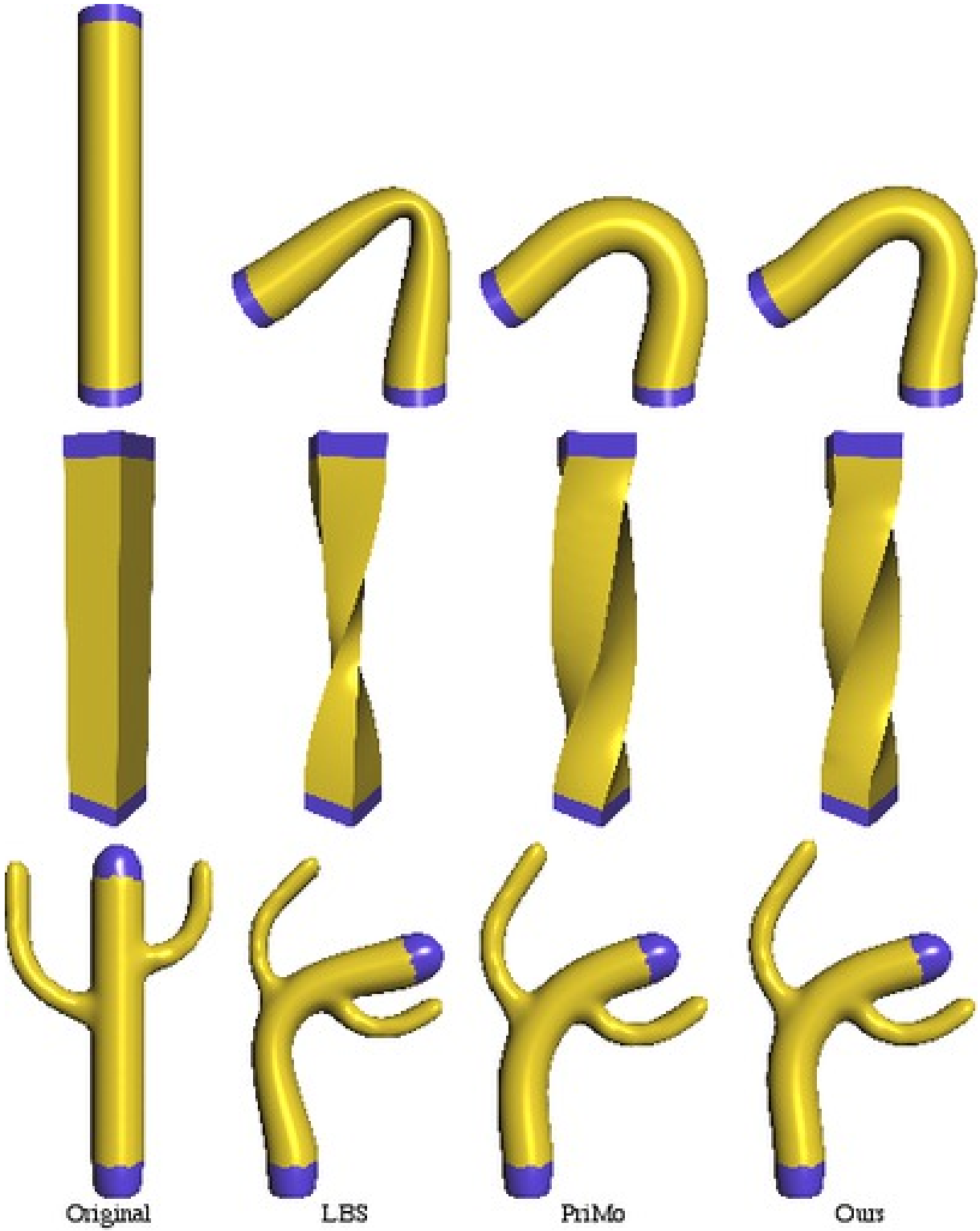


Figure 7.12: *Our method achieves visually equivalent quality as full nonlinear optimization (PriMo) [Botsch et al. 2006a] and runs much faster.*

	Input model				Model reduction			Runtime	Precomputation		
Model	d	n	Triangles	Type	m_{orig}	m_{extra}	r	1 Iter.	Full	Switch	Add. Weights
Gingerbread man	2	2899	5543	Tri	9	0	18	11μs	0.227s	14ms	0s
Bubble man	2	6383	11804	Tri	5	0	10	7μs	0.266s	3ms	0s
Female	3	45659	91314	Sp+rim	17	0	34	19μs	3.359s	16ms	0s
Armadillo	3	47162	86482	Tet	17	0	34	20μs	8.365s	15ms	0s
Octopus	3	149666	299328	Tet	9	0	18	10μs	9.957s	9ms	0s
Cylinder	3	4802	9600	Sp+rim	2	30	64	40μs	0.471s	29ms	4.4s
Cactus	3	5261	10518	Sp+rim	2	30	64	40μs	0.586s	35ms	4.882s
Bar	3	6084	12106	Sp+rim	2	30	64	42μs	0.582s	32ms	4.266s
Ogre	3	28837	52306	Tet	5	15	40	22μs	2.717s	6ms	12.166s
Cross	3	29090	58176	Sp+rim	2	30	64	41μs	3.738s	49ms	10.09s
Bumpy plane	3	40401	80000	Sp+rim	2	30	64	41μs	4.328s	33ms	7.812s
Wiener dog	3	48187	85672	Tet	18	15	66	43μs	12.708s	77ms	11.123s

Table 7.1: *Model statistics and performance. d denotes the dimension, n the number of vertices,* Elements *refers to the type of elements used in the energy formulation (triangles* (Tri, Sp+rim) *or tetrahedra* Tet*);* Sp+rim *refers to spokes and rims energy. m_{orig} is the number of original weights and m_{extra} is the number of additional weights, r is the number of rotation clusters. We report the time for one optimization iteration (*1 Iter.*), the full precomputation time (*Full*), and the precomputation time when switching constraint types at handles (*Switch*).*

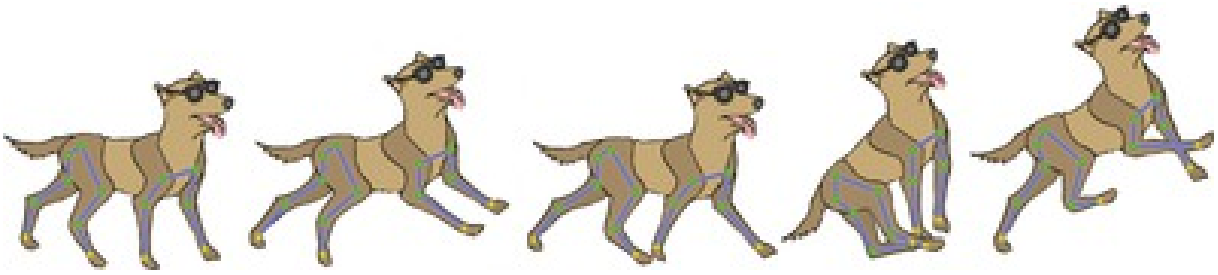


Figure 7.13: *Left to right: the* Blinden Hund *is rigged to a skeleton composed of two disconnected components. The yellow joints provide manipulators for a mesh-aware inverse kinematics deformation.*

[Chao et al. 2010] treats shapes as enclosed volumes. Note that thanks to our reduced-order formulation, the tetrahedra participate in preprocessing only, where internal complexity is folded into our reduced system. At runtime our method considers only the surface.

To assess deformation quality, in Figure 7.12 we compare our results on the survey benchmark to those of Figure 10 in [Botsch and Sorkine 2008], with our skinning weights automatically generated using bounded biharmonic weights and enriched with additional weights per Section 7.3.3. As expected, the mesh quality of our reduced nonlinear method surpasses the quality of linear models. Meanwhile, the deformations are comparable to high quality nonlinear methods [Botsch et al. 2006a] that are orders of magnitude slower to compute.

The summary of our results can be found in Table 7.1. Our timings were obtained on a single-core CPU implementation on a laptop with a 2.5GHz Core i7-2860QM. We typically perform 15 iterations of the local-global optimization each at about 10μs to 40μs, resulting in total solve times of about 0.15 to 0.6 milliseconds. With our models, this performance is close to that of an LBS shader (with 4 weights per vertex, running in parallel on a GeForce 560M GPU). In addition to full precomputation times, we also report the time necessary to switch constraints or change their type (i.e. invert the system matrix in Eq. 7.19). In our figures and the

Figure 7.14: *Shape-aware IK: a female model animated solely by positional constraints at the endpoints of the head, hands and feet.*

video accompanying [Jacobson et al. 2012a] we use yellow dots for positional constraints (both at point handles or bone endpoints), green dots for unconstrained points, and red dots when specifying full transformations. Transformations at region handles (blue) are fully specified unless otherwise noted.

In the realm of real-time animation, our approach provides shape-aware inverse kinematics supporting both classical (Figure 7.14) and disconnected (Figures 7.13 & 7.15) skeletons. To facilitate user control, our system also supports traditional hierarchical transformation chains (forward kinematics) even when only a few points are ultimately used as constraints in our optimization. Our optimization may be used to assuage the common LBS candy-wrapper effect by spreading twist across multiple bones (see Figure 7.16). The blue line segments drawn between joints denote this hierarchy and are not intended to visualize the affine transformations at joints, which may be partially or entirely optimized by our system. While most variational methods require constraints on each connected mesh component, our method can easily handle multiple connected components, implicitly "glued together" by manually painted skinning weights (see Figure 7.17).

Reduced variational models are typically associated with a higher energy value and a loss of quality due to the reduction of degrees of freedom. While our model reduction also increases the original energy value, projection into our deformation subspace has a nice regularization side-effect, forcing the solution to more visually pleasing deformations than the non-reduced method. For example, with $\mathcal{C}^1$ weight functions, we can guarantee $\mathcal{C}^1$ deformations, unlike non-reduced ARAP energy minimization (see Figure 7.8). Similarly, the reduction of the energy term using rotation clusters (Section 7.3.2) helps to avoid deformation field singularities occasionally produced by the non-reduced method (see Figure 7.7).

The utility of generating additional weights at abstract handles is illustrated in Figure 7.9. If all region handles are fully constrained, there is no room for optimization and our method reduces to LBS. However, after adding 15 additional weights, our technique results in more natural,

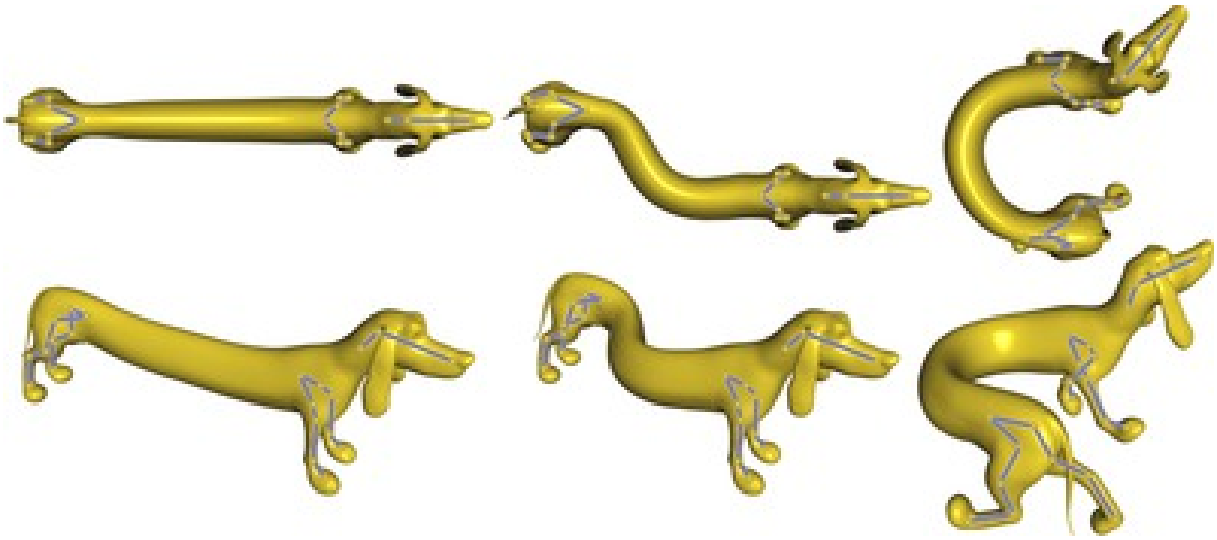


Figure 7.15: *Three disconnected skeletons control the* Wiener dog. *Additional abstract handle weights are generated along the belly allowing it to deform elastically.*

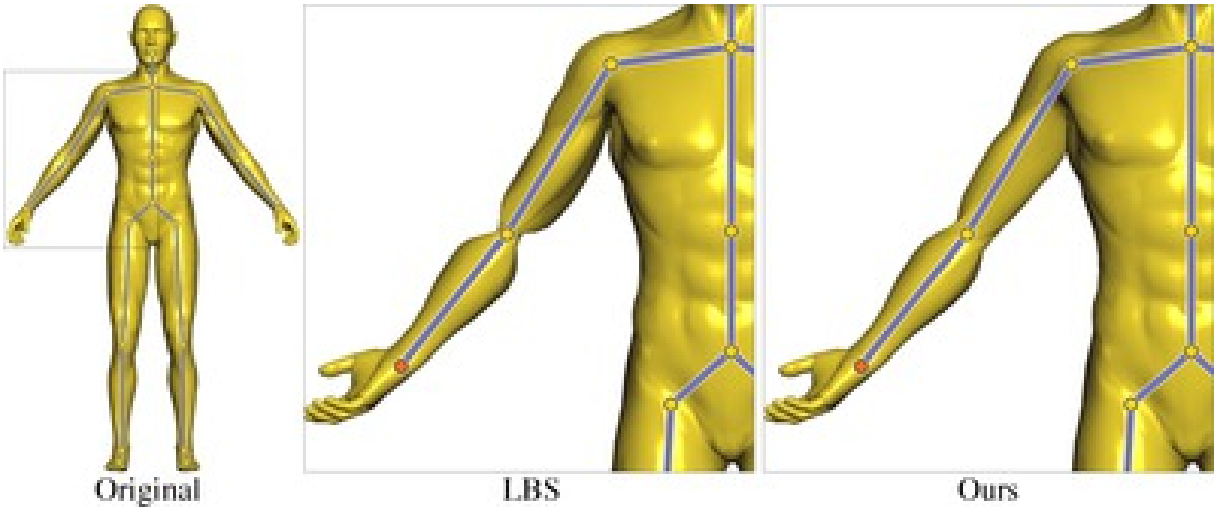


Figure 7.16: *Our method naturally spreads elbow twist across the skeleton, avoiding the LBS artifacts.*

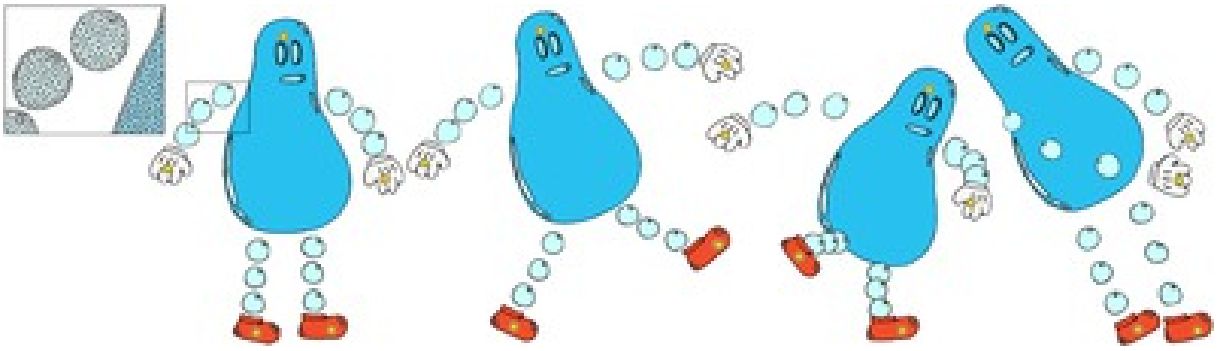


Figure 7.17: Bubble Man *with 17 connected components and only 5 positional handles. Our deformation subspace (LBS) is designed to prevent the components from falling apart.*

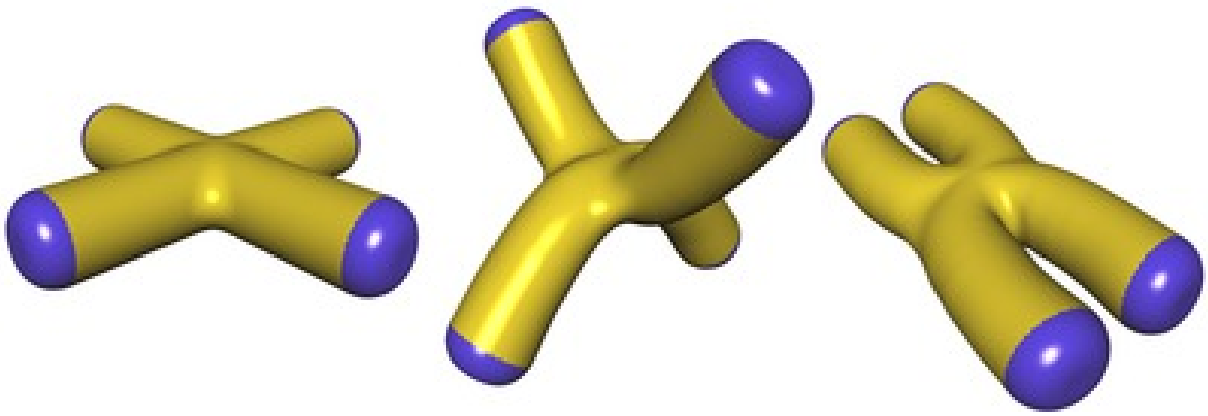


Figure 7.18: *Deforming a cross with region controls produces a smooth, high quality shape in real time.*

shape-preserving deformations. Additional weight functions were also used in the benchmark examples (see Figure 7.12) and a cross-shape model (see Figure 7.18), designed to stress-test fairness of a branching structure's deformation.

7.5 Limitations and future work

To guarantee real-time performance, we use a fixed number of iterations in the local-global optimization and therefore, we cannot guarantee that our solution has converged completely. However, by initializing the optimization by the previous frame's transformations, we do not observe any disturbing artifacts even when using only 15 iterations (note that a reduced model typically requires much fewer iterations than an unreduced one). Inverse-kinematics methods often incorporate many other tunable parameters, such as balance and joint limit constraints [Shi et al. 2007]. Incorporating such terms is an obvious direction for improving the scope of our results.

The additional weights employed at abstract handles to bolster our optimization space are simple to construct, but may not be optimal for any given input mesh, linear blend skinning setup, and number of desired additional weights. With many overlapping original weights, our embedding in weight space causes our additional weights to overlap more. This in turn results in the practical disadvantage of having to support a large number of influencing handles in a shader implementation of linear blend skinning. Specifying the maximum number of influencing weights at any given point is an interesting constraint on our additional weight construction that we do not yet consider, even though previous weight reduction methods can be used as an immediate alternative [Landreneau and Schaefer 2010].

Another interesting line of future work, inspired by Figure 7.11, is to study different types of bending terms to support a user-tunable range of elastic effects for surfaces.

We justify clustering vertices in weight space because vertices with similar weights will have similar deformation gradients. This assumption holds for typical weight functions with small gradients, allowing us to use a small number of rotation groups. Robust and efficient handling of abruptly changing weights would be an interesting future direction.

7.6 Conclusion

Linear deformation methods are still sometimes preferred over nonlinear ones because of their speed and simplicity, resulting in compromises in quality [Botsch and Sorkine 2008]. By reducing both the space of possible deformations and simplifying the elastic energy function, our method avoids this compromise and delivers a method for deformations that are both fast and of high quality. We introduce new modes of control: shape-aware inverse kinematics combined with disconnected skeletons. Our method is simple to implement, with the time-critical inner loop only consisting of dense matrix multiplications and low-dimensional ($d \times d$) singular value decompositions. We believe our method will help to promote nonlinear deformation methods in applications where run-time efficiency is a primary concern.

7.7 Appendix: Physically based dynamics

Adding physically based dynamics to our optimization is straightforward. By treating the gradient of our ARAP energies as internal forces we can introduce a conservation of momentum term. This barely changes the system matrix and right-hand side, but drastically changes the behavior. Now the automatic transformations are smooth functions of time (see Figure 7.19).

To accommodate standard notation used in physically based animation, we will use slightly different notation than the previous section. In this section: $\mathbf{x} \in \mathbb{R}^{n \times 3}$ (née $\mathbf{v}'$) are the unknown mesh vertex positions, $\mathbf{M} \in \mathbb{R}^{n \times n}$ is the lumped mass matrix (see Section 2.1.3), and $\mathbf{W}$ (née $\mathbf{M}$) is the matrix composed of skinning weights and rest positions.

7.7.1 ARAP with dynamics

First we describe how dynamics are added to the unreduced ARAP. This is similar to Section 3 of [Chao et al. 2010]. A starting point of a physical simulation is Newton's second law:

$$\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{int}} = \mathbf{M}\,\mathbf{a}, \tag{7.22}$$

where $\mathbf{f}_i^{\text{ext}} \in \mathbb{R}^{n \times d}$, $\mathbf{f}^{\text{int}} \in \mathbb{R}^{n \times d}$, $\mathbf{M} \in \mathbb{R}^{n \times n}$, and $\mathbf{a} \in \mathbb{R}^{n \times d}$ are the external forces, internal forces, mass matrix and acceleration.

Examining the ARAP energy $E_{\text{ARAP}}(\mathbf{x})$ in Equation (7.4) we can think of $-\nabla E_{\text{ARAP}}(\mathbf{x})$ as a force which pushes $\mathbf{x}$ towards an optimal configuration (critical point of $E_{\text{ARAP}}(\mathbf{x})$). When we do not have dynamics, we say we are finding a *static* solution because for each frame we follow this gradient until convergence, until equilibrium. For dynamics, we can substitute the internal forces with $-\nabla E_{\text{ARAP}}(\mathbf{x})$ in Equation (7.22):

$$\mathbf{f}_{\text{ext}} - \nabla E_{\text{ARAP}}(\mathbf{x}) = \mathbf{M}\,\mathbf{a}. \tag{7.23}$$

During our simulation we will let $\mathbf{f}_{\text{ext}}$ be some arbitrary forces that are held constant for any given time step t. We also hold the mass matrix $\mathbf{M}$ constant and we will in fact use the diagonal, lumped version (see Section 2.1.3). Treating the acceleration term $\mathbf{a}$ as *unknown*, we may use

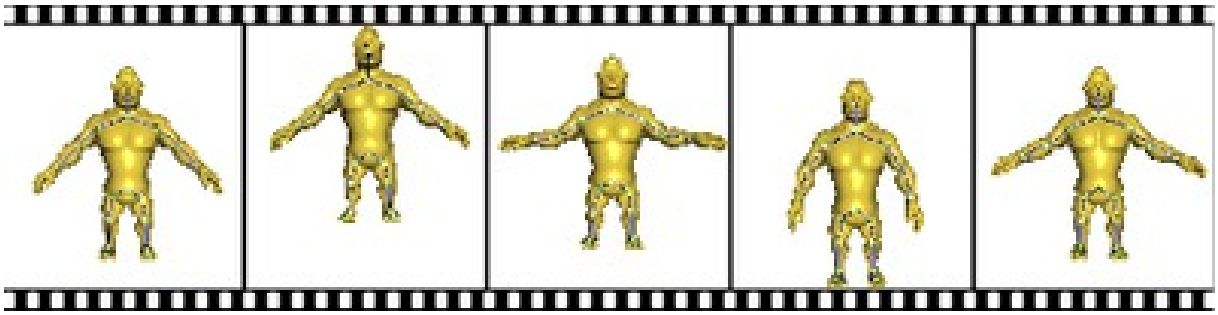


Figure 7.19: *Only the head of the* Ogre *is moved up and down all other transformations are free. A static animation would be a global rigid transformation. Instead an animation with dynamics shows interesting elastic behavior. Notice that the arms bend at the elbow, this semantic information is derived implicitly from the skinning subspace.*

finite-differencing in time to expand it as a linear function of unknown velocities $\mathbf{v}$ at the current time step and the velocities of the previous time step $\mathbf{v}_0$:

$$\mathbf{a} = \frac{\mathbf{v} - \mathbf{v}_0}{h}, \tag{7.24}$$

where h is the constant scalar time step size. The unknown velocities may in turn be expanded into linear functions of vertex positions:

$$\mathbf{v} = \frac{\mathbf{x} - \mathbf{x}_0}{h}. \tag{7.25}$$

And finally this means the unknown accelerations $\mathbf{a}$ are just a linear function of current positions, previous positions and previous velocities, of which $\mathbf{x}_0$ and $\mathbf{v}_0$ are known:

$$\mathbf{a} = \frac{\frac{\mathbf{x}-\mathbf{x}_0}{h} - \mathbf{v}_0}{h} \tag{7.26}$$

$$= \frac{1}{h^2}\mathbf{x} - \frac{1}{h^2}\mathbf{x}_0 - \frac{1}{h}\mathbf{v}_0. \tag{7.27}$$

Substituting this into our Equation (7.23) produces:

$$\mathbf{f}_{\text{ext}} - \nabla E_{\text{ARAP}}(\mathbf{x}) = \mathbf{M}\left(\frac{1}{h^2}\mathbf{x} - \frac{1}{h^2}\mathbf{x}_0 - \frac{1}{h}\mathbf{v}_0\right) \tag{7.28}$$

$$= \frac{1}{h^2}\mathbf{M}\mathbf{x} - \frac{1}{h^2}\mathbf{M}\mathbf{x}_0 - \frac{1}{h}\mathbf{M}\mathbf{v}_0. \tag{7.29}$$

If we take the anti-derivative with respect to $\mathbf{x}^\mathsf{T}$ we have the quantity (energy):

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{a}^\mathsf{T}\mathbf{M}\mathbf{a} - \mathbf{x}^\mathsf{T}\mathbf{f}_{\text{ext}} + E_{\text{ARAP}}(\mathbf{x}). \tag{7.30}$$

Expanding $\mathbf{a}$ as above gives:

$$E(\mathbf{x}) = \frac{1}{2}\left(\frac{1}{h^2}\mathbf{x} - \frac{1}{h^2}\mathbf{x}_0 - \frac{1}{h}\mathbf{v}_0\right)^\mathsf{T}\mathbf{M}\left(\frac{1}{h^2}\mathbf{x} - \frac{1}{h^2}\mathbf{x}_0 - \frac{1}{h}\mathbf{v}_0\right) - \mathbf{x}^\mathsf{T}\mathbf{f}_{\text{ext}} + E_{\text{ARAP}}(\mathbf{x}).$$

Treating $\mathbf{x}_0$ and $\mathbf{v}_0$ as constant we can remove constant scalar terms and gather like terms with respect to $\mathbf{x}$:

$$E(\mathbf{x}) = \mathbf{x}^\mathsf{T}\frac{1}{2h^2}\mathbf{M}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{M}\left(-\frac{1}{h^2}\mathbf{x}_0 - \frac{1}{h}\mathbf{v}_0\right) + \mathbf{x}^\mathsf{T}\mathbf{f}_{\text{ext}} + E_{\text{ARAP}}(\mathbf{x}). \tag{7.31}$$

Notice how this affects our "local/global" optimization. For the global step, we need to add $\frac{1}{h^2}\mathbf{M}$ to the quadratic coefficient matrix in E_{ARAP}. This matrix is constant so adding it will only affect precomputation. We also have new linear terms for the global step. These terms depend on $\mathbf{x}_0$ and $\mathbf{v}_0$ which will be changing per time step. They do not depend on the rotations $\mathbf{R}$ found during the local step. This means some portion of the global step's right-hand side will depend on $\mathbf{R}$ and another part will depend on $\mathbf{x}_0$ and $\mathbf{v}_0$. Because none of the new terms in Equation (7.31) depend on $\mathbf{R}$, the local step (SVDs) does not change at all.

7.7.2 Reduction

We have already written ARAP with dynamics as a familiar energy minimization in Equation (7.31). The substitution proceeds as in Section 7.3.1. In this notation our LBS subspace is represented by:

$$\mathbf{x} = \mathbf{W}\mathbf{T}, \tag{7.32}$$

where $\mathbf{W} \in \mathbb{R}^{n\times((d+1)m)}$ is the matrix combining rest-pose vertex positions $\mathbf{x}^i$ with vertex weights $w_j(\mathbf{x}^i)$, and $\mathbf{T} \in \mathbb{R}^{((d+1)m)\times d}$ stacks transposed transformation matrices $\mathbf{T}^j$.

We may also parameterize the velocities and previous positions in this subspace:

$$\begin{aligned}\mathbf{v} &= \frac{1}{h}\left(\mathbf{W}\mathbf{T} - \mathbf{W}\mathbf{T}_0\right) && (7.33)\\ &= \mathbf{W}\mathbf{S}, && (7.34)\\ \mathbf{x}_0 &= \mathbf{W}\mathbf{T}_0, && (7.35)\end{aligned}$$

where $\mathbf{S} = \frac{1}{h}\left(\mathbf{T} - \mathbf{T}_0\right)$ is the change in transformations between time steps.

Making these substitutions we have an energy optimization similar to Equation (7.15):

$$\begin{aligned}\underset{\mathbf{T},\mathbf{R}}{\text{argmin}} \quad & \frac{1}{2}\text{tr}\left(\mathbf{T}^\mathsf{T}\left(\tilde{\mathbf{L}} + \tilde{\mathbf{M}}\right)\mathbf{T}\right) - \text{tr}\left(\left(\mathbf{R}\tilde{\mathbf{K}} + \left(\mathbf{T}_0 + h\mathbf{S}_0\right)^\mathsf{T}\tilde{\mathbf{M}}\right)\mathbf{T}\right)\\ \text{subject to} \quad & \mathbf{W}_{\text{eq}}\mathbf{T} = \mathbf{P}_{\text{eq}}, \quad \mathbf{R} \in SO(d)^r,\end{aligned} \tag{7.36}$$

where $\tilde{\mathbf{L}} = \mathbf{W}^\mathsf{T}\mathbf{L}\mathbf{W}$, $\tilde{\mathbf{M}} = \frac{1}{h^2}\mathbf{W}^\mathsf{T}\mathbf{M}\mathbf{W}$, and $\tilde{\mathbf{K}} = \mathbf{K}\mathbf{W}$. Similarly all computation in the inner loop is collapsed to complexities on the order of $\mathbf{T}$ rather than $\mathbf{x}$. Even external forces such as gravity or buoyancy can benefit from precomputation using $\tilde{\mathbf{M}}$. An open research question for future work is how to incorporate collision response without affecting performance.

8

Robust inside-outside segmentation via generalized winding numbers

Solid shapes in computer graphics are often represented with boundary descriptions, e.g. triangle meshes, but animation, physically-based simulation, and geometry processing are more realistic and accurate when explicit volume representations are available. Tetrahedral meshes which exactly contain (interpolate) the input boundary description are desirable but difficult to construct for a large class of input meshes. Character meshes and CAD models are often composed of many connected components with numerous self-intersections, non-manifold pieces, and open boundaries, precluding existing meshing algorithms. We propose an automatic algorithm handling all of these issues, resulting in a compact discretization of the input's inner volume. We only require reasonably consistent orientation of the input triangle mesh. By generalizing the winding number for arbitrary triangle meshes, we define a function that is a perfect segmentation for watertight input and is well-behaved otherwise. This function guides a graphcut segmentation of a constrained Delaunay tessellation (CDT), providing a minimal description that meets the boundary exactly and may be fed as input to existing tools to achieve element quality. We highlight our robustness on a number of examples and show applications of solving PDEs, volumetric texturing and elastic simulation.

8.1 Introduction

A large class of surface meshes used in computer graphics represent solid 3D objects. Accordingly, many applications need to treat such models as volumetric objects: for example, the animation or physically-based simulation of a hippopotamus would look quite different (and unrealistic) if handled as a thin shell, rather than a solid (see Figure 8.19). Since many oper-

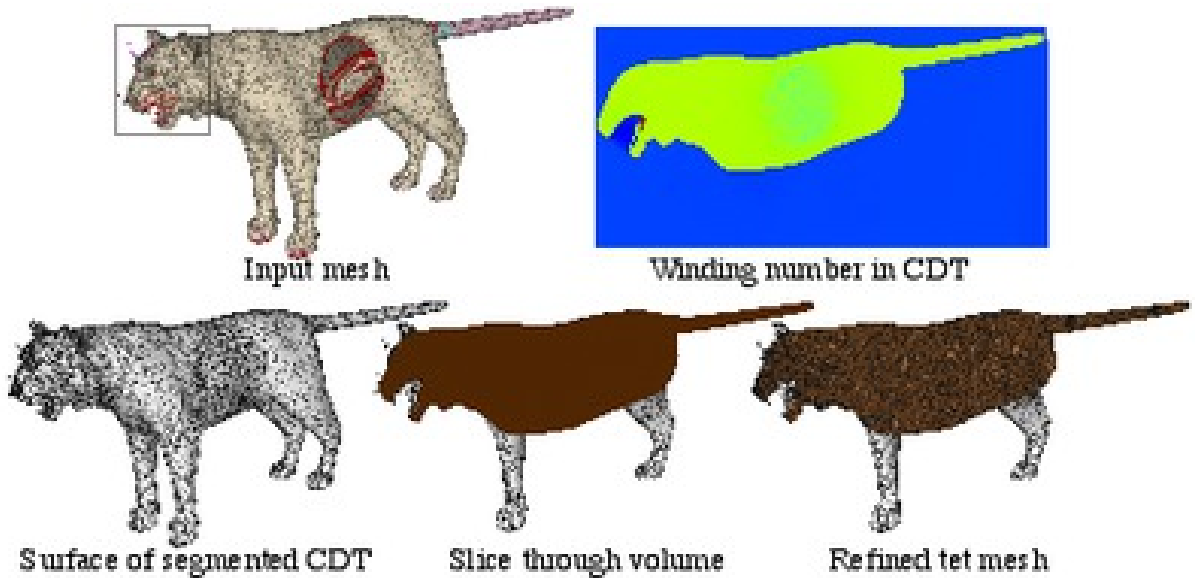


Figure 8.1: *The* Big SigCat *input mesh has 3442 pairs of intersecting triangles (bright red), 1020 edges on open boundaries (dark red), 344 non-manifold edges (purple) and 67 connected components (randomly colored). On top of those problems, a SIGGRAPH logo shaped hole is carved from her side.*

ations in animation, simulation and geometry processing require an explicit representation of an object's volume, for example for finite element analysis and solving PDEs, a conforming[1] tetrahedral meshing of the surface is highly desired, as it enables volumetric computation with direct access to and assignment of boundary surface values. However, a wide range of "real-life" models, although they *appear* to describe the boundary of a solid object, are in fact *unmeshable* with current tools, due to the presence of geometric and topological artifacts such as self-intersections, open boundaries and non-manifold edges. As a consequence, processing is often limited to the surface, bounding volumetric grids [McAdams et al. 2011] or approximations with volume-like scaffolding [Zhou et al. 2005, Baran and Popović 2007, Zhang et al. 2010].

The aforementioned artifacts are common in man-made meshes, as these are the direct output of human creativity expressed through modeling tools, which very easily allow such artifacts to appear. Sometimes they are even purposefully introduced by the designer: for example, character meshes will typically contain many overlapping components representing clothing, accessories or small features, many of which have open boundaries (see Figure 8.2). Modelers choose very specific boundary mesh layout and vertex density, necessary for articulation or faithful representation of important features while staying on a tight vertex budget. It is therefore highly desirable to avoid remeshing and subsequent interpolation, and at the same time obtain a valid and precise representation of the object's inner volume.

In this chapter, we propose an automatic algorithm for producing volumetric meshes that fully contain the geometry of the input surface model. Our method robustly handles artifacts common in man-made meshes while still supporting the full set of quality assurances, as do existing conforming meshing tools.

[1] Contrary to some authors' use of "conforming" to mean that every mesh edge is locally Delaunay, we use it simply to mean that the volume mesh interpolates to the boundary description.

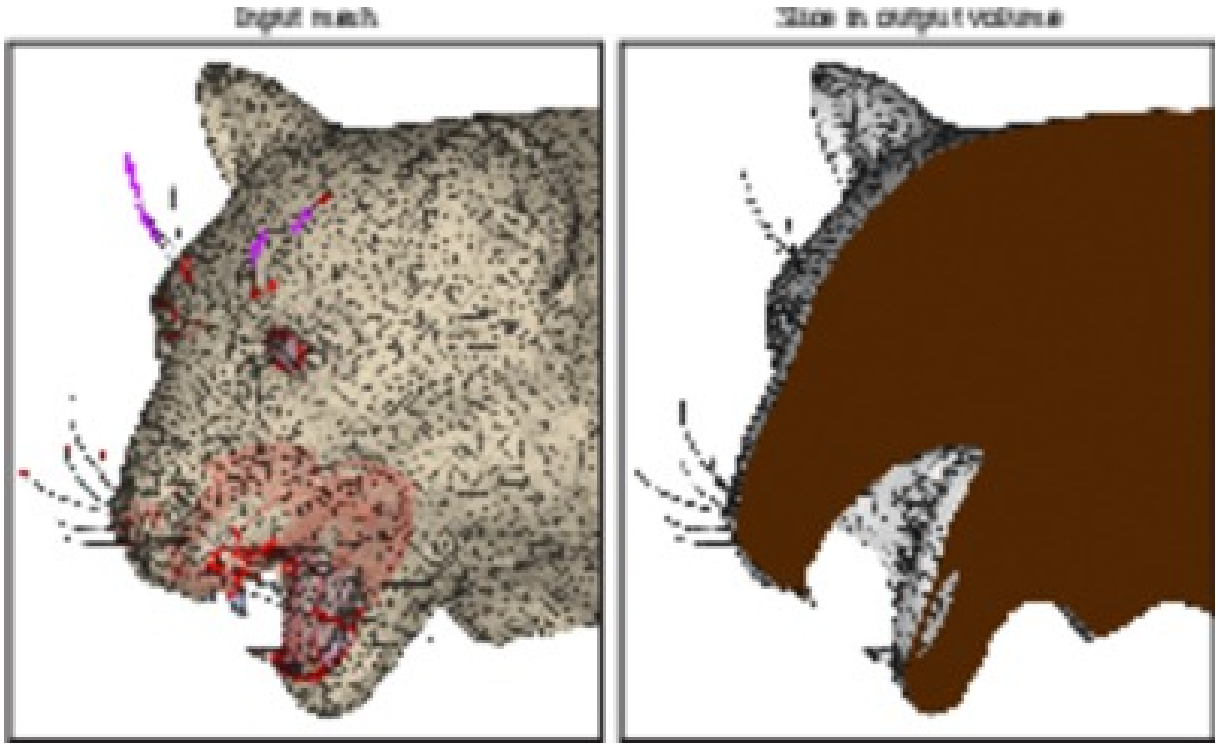


Figure 8.2: *Each whisker, tooth and eye of the* Big SigCat *is a separate component, intersecting the body. On top of this, many have open boundaries (dark red) and non-manifold edges (purple).*

Past years have seen many advances in algorithms for generating high quality simplicial (triangles in $\mathbb{R}^2$, tetrahedra in $\mathbb{R}^3$) volume meshes. The popular tools TRIANGLE [Shewchuk 1996] and TETGEN [Si 2003] are examples of methods that exactly conform to a given piecewise-linear boundary description. Such tools support a wide range of features, in particular concerning element quality, but they fail when the input boundary descriptions contain geometric ambiguities or flaws which make the inner volume even remotely ambiguous. The number of these issues in a common man-made model may range in the hundreds or thousands (see Figure 8.1), so manual clean up is time consuming and deadeningly tedious. An alternative could be to treat this as a surface repair problem, but this precludes exactly maintaining the original boundary vertices and facets during local fixup operations [Attene 2010]. Surface reconstruction techniques are not quite suitable in our setting either, because they focus on recovering surfaces from scans of real solids, where the artifacts should only arise from scanning errors, and hence partial or complete remeshing and loss of input features may occur.

Our method generally follows the steps of reconstruction based on constrained Delaunay tessellation (CDT): we compute the (tetrahedral) CDT of the convex hull of the input vertices and facets. We rely on state-of-the-art CDT tools, which currently require certain pre-processing of the input, such as subdivision of self-intersecting facets and discarding degeneracies. The goal is then to segment the CDT volume into "inside" and "outside" elements, such that the set of inside elements comprises a valid conforming tet mesh. Our main contribution is the introduction of a new inside-outside confidence function by generalizing the *winding number*. Though similar at a high enough level, signed distance functions do not encode segmentation confidence. They smoothly pass through zero at the surface, whereas our function has a sharp jump there. Away from the surface our function is smooth (in fact, harmonic!). It defines a

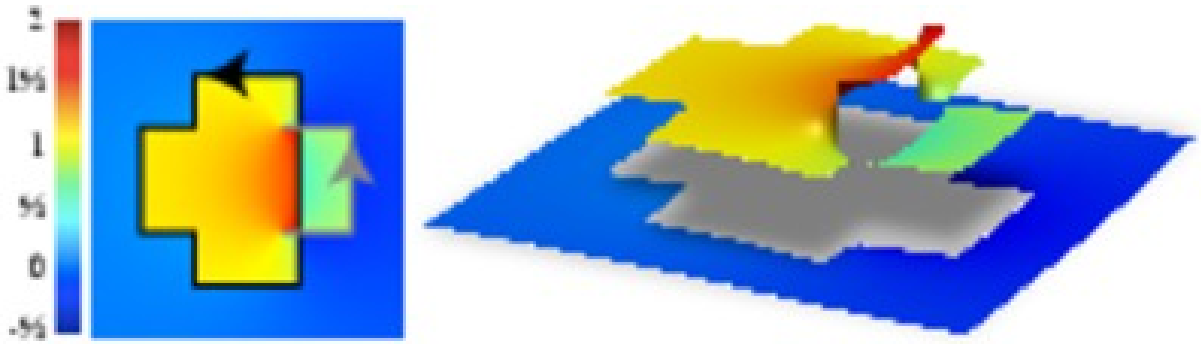


Figure 8.3: *The winding number intuitively captures self-intersections, maintaining boundary exactly (cf. Figure (5) in [Shen et al. 2004]).*

perfect, piecewise-constant segmentation of space when the input is perfect (i.e. a watertight surface). When the input contains ambiguities and missing information, the well-behaved nature of our function makes it suitable for guiding an energy-minimizing segmentation of the CDT, which can be efficiently computed using graphcut. We can constrain the segmentation to exactly contain all input vertices and facets, as well as ensure surface manifoldness. The final output is a minimal tetrahedral mesh carved from the CDT which may be post-processed using existing tools to achieve high-quality elements or heterogeneous sizing.

We evaluate our algorithm on a wide range of inputs, which are otherwise unmeshable with existing tools. We demonstrate the usefulness of the method via applications such as physically-based elasticity simulation, skinning weight computation for real-time animation, geometric modeling and volumetric texturing. Our algorithm offers a step towards a new level of robustness for unstructured volumetric meshing, which will potentially have a large impact on the standard computer graphics pipeline, especially as geometry processing turns toward treating solids as solids rather than operating (often just out of convenience or obligation) on merely the surface.

8.2 Related work

Surface repair. Artifacts of surface meshes, such as violations of the connected 2-manifoldness, consistent orientation or watertightness properties, not only disturb conforming volumetric meshing but also surface-based processing, because the majority of geometric algorithms assumes clean input. Although the problem of mesh repair has been extensively studied, it remains elusive in practice [Ju 2009, Campen et al. 2012]. Most methods for meshing of polygon soups into surfaces do not robustly deal with self-intersecting input facets [Hoppe et al. 1993, Rossignac and Cardoze 1999, Guéziec et al. 2001, Podolak and Rusinkiewicz 2005] insofar as promising a volume-meshable, *watertight* surface. Some methods do offer guarantees; they work by globally remeshing the output [Ju 2004, Bischoff et al. 2005] or by making local modifications at the cost of not maintaining the original mesh (geometry and/or connectivity) in troublesome areas [Yamakawa and Shimada 2009, Attene 2010]. Bischoff and Kobbelt [2005] repair CAD models while trying to preserve the original meshing, but they assume that the input is divided into manifold patches that do not self-

intersect, and their method requires a spatially-varying threshold for gap filling. [Attene 2010] provide volume meshing as an application of their watertight output. However, because their algorithm iterates between removing troublesome patches and hole filling, large portions of the original mesh may be deleted (see Figure 8.17). Holes are filled by locally modifying the mesh and become hard boundary constraints for volume meshing. Conversely, our winding number function incorporates global information to intelligently resolve missing information ambiguities. A volumetric tool for general surface repair exists [Nooruddin and Turk 2003], but its voxel-based nature does not scale well for large, detailed models and complicates interpolation of the input mesh. Unlike our method, the seminal work of [Murali and Funkhouser 1997] is not restricted to consistently oriented input. However, their voting-based approach is prone to mis-assignment in regions of overlap and loss of small details [Attene 2010].

Surface reconstruction can be seen as an alternative way to obtain a clean, watertight surface mesh. However, most reconstruction algorithms are tailored to perform well on noisy point cloud inputs and hence do not strive to preserve the input mesh structure. Algorithms like the Zipper of [Turk and Levoy 1994] stitch range images by generally only modifying the mesh along the overlap, but this approach is only suitable for well-aligned range images. A host of reconstruction methods, starting with [Hoppe et al. 1992], fit an implicit function to the input surface geometry and extract a level set, which is guaranteed to be watertight for well-behaved functions; recent methods are quite robust to noisy data [Kazhdan et al. 2006, Mullen et al. 2010] and even unoriented data [Alliez et al. 2007]. However, the original input mesh is generally lost during contouring. Shen et al. [2004] design level-sets using moving least squares to perfectly interpolate input facets, but contouring loses any premeditated discretization distribution. Further, due to the oscillatory nature of their function, the exact interpolation constraint may need to be relaxed when components overlap (see our Figure 8.3 and their Figure 5).

Surface reconstruction of point clouds has been achieved with graphcut segmentation on voxel-grids [Hornung and Kobbelt 2006] and on Delaunay meshes [Wan et al. 2011]. Later, [Wan et al. 2012] tackled open surfaces via graphcut on a level-set of an intersection of approximating "crusts". Our method, as many previous methods, segments a volume from a constrained Delaunay tessellation of the input convex hull. The peeling procedure of [Dey and Goswami 2003] fills surface holes ensuring a watertight result, though possibly non-manifold. Further, it requires a fine enough initial discretization to prevent a degenerate solution. The spectral method of [Kolluri et al. 2004] improves upon this. They provide similar post-processing heuristics to ours for ensuring manifoldness. However, extending their spectral analysis to interpolate input facets is not obvious.

Unstructured tetrahedral mesh generation. Efficient creation of Delaunay tessellations is well studied; refer to [Shewchuk 2012] for an excellent survey. The methods can be subdivided into those that exactly interpolate input vertices and faces [George et al. 1990, Shewchuk 1996, Joshi and Ourselin 2003, Si 2003, Geuzaine and Remacle 2009] and those that approximate watertight input surfaces [Shimada and Gossard 1995, Alliez et al. 2005, Bridson et al. 2005, Labelle and Shewchuk 2007]. We heavily rely on the former category to mesh the convex hull of our input. Additionally, as our method outputs a minimal

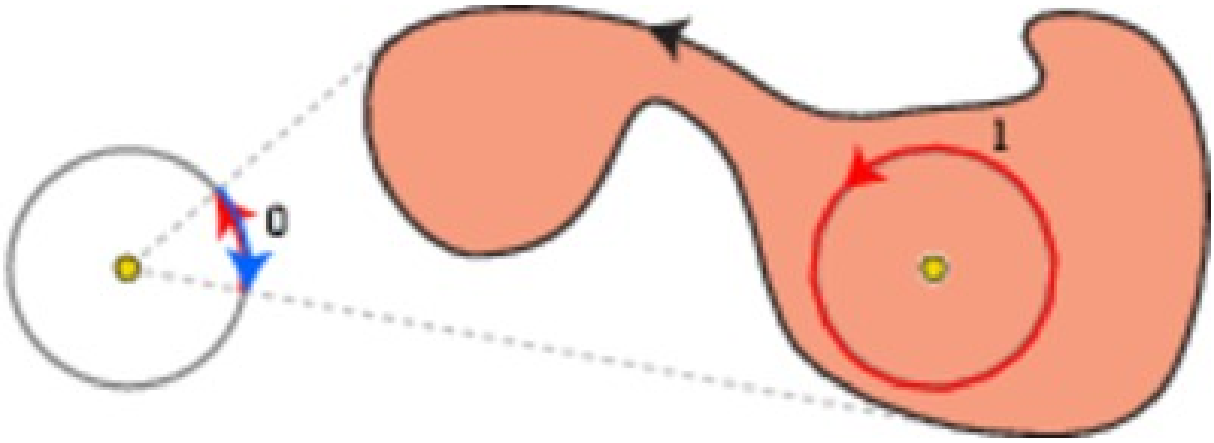


Figure 8.4: *Winding number is the signed length of the projection of a curve onto a circle at a given a point divided by 2π. Outside the curve, the projection cancels itself out. Inside, it measures one.*

tetrahedral mesh, we may post-process with mesh refinement tools [Schöberl 1997, Si 2003, Klingner and Shewchuk 2007, Geuzaine and Remacle 2009] to achieve element quality.

Winding number, inside-outside tests. The winding number of closed curves is an old concept [Meister 1769/70]. To the best of our knowledge, no previous work has generalized "winding numbers" computed as integrals on open curves or surfaces. However, many related functions exist. Mean value coordinates (MVC) use a similar projection integral [Floater 2003, Ju et al. 2005], but lack the jump discontinuity across the boundary that gives the winding number its unique segmentation property. MVC are also notably not harmonic, and thus may oscillate and not satisfy the maximum principle. In the terminology presented by [Zhou et al. 2008], our winding number adheres to an "object-based" definition of inside-outside. Thus we are a complement to their "view-based" definition. Their method uses ray-shooting combined with graphcut to achieve a different set of applications, more suitable to computer vision.

8.3 Method

Our goal is a tetrahedral mesh conforming to an input shape. We achieve this by computing a constrained Delaunay tessellation (CDT) containing the input vertices and facets, then by evaluating a generalization of the winding number for each element we segment inside and outside elements of the CDT resulting in the final tet mesh.

Let the input shape in $\mathbb{R}^d$ be described by a list of n vertices $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$, $\mathbf{v}_i \in \mathbb{R}^d$ and a list of m simplicial facets $\mathcal{F} = \{f_1, \ldots, f_m\}$ where $f_i \in \{1, 2, \ldots, n\}^d$ (we only consider $d = 2$ and $d = 3$)[2]. The goal is then to find a set of elements $\mathcal{E} \subset \{1, \ldots, k\}^{d+1}$ defined over a set of vertices $\mathbf{v}_\mathcal{E}$ which represent the area (if $d = 2$) or volume (if $d = 3$) of $(\mathcal{V}, \mathcal{F})$. In the

[2] The integral definition of the winding number (thus, also our generalization) is well-defined for any dimension. Though our proofs for harmonicity are not, we hypothesize this to be the case.

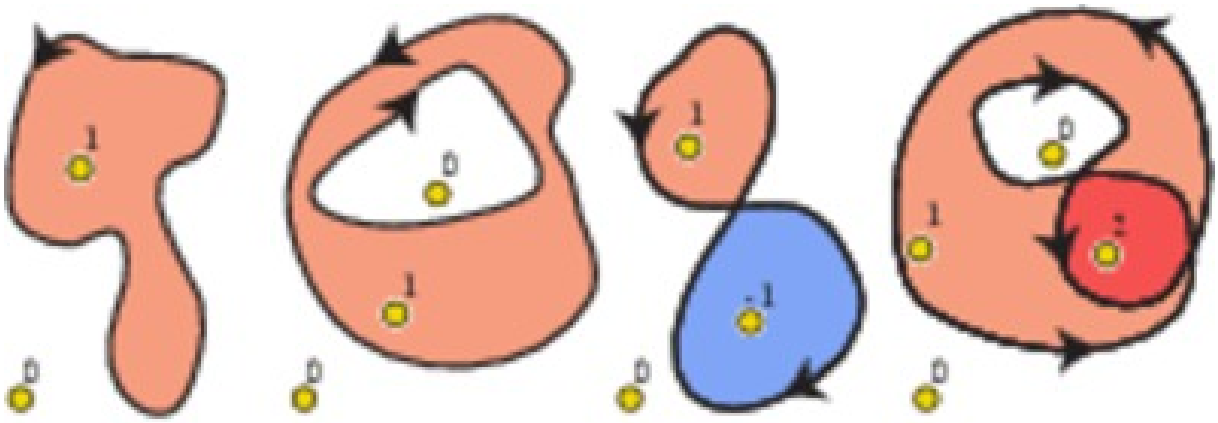


Figure 8.5: *Winding number exactly segments inside and outside for concave, high-genus, inverted and overlapping curves. Multiple components are also naturally handled: consider this entire figure and the winding numbers remain the same.*

ideal case, we achieve *exact interpolation*: $\mathbf{v}_\mathcal{E} = \mathcal{V}$ and all facets in $\mathcal{F}$ appear as subfacets of elements in $\mathcal{E}$. Note, facets and elements correspond to triangles and tetrahedra in $\mathbb{R}^3$ and edges and triangles in $\mathbb{R}^2$.

Although $\mathcal{F}$ forms a graph or *mesh* over $\mathcal{V}$, the input is not assumed to be $(d-1)$-manifold, orientable or closed. We do assume the mesh *intuitively represents* or loosely approximates the surface of some solid and has reasonably consistent orientation. This is motivated by the observation that most practical input meshes were created in such a way that they *appear* to be the surface of some solid when rendered with single-sided lighting.

We first construct an inside-outside confidence function which generalizes the winding number. We then evaluate the integral average of this function at each element in a CDT containing $(\mathcal{V}, \mathcal{F})$. Finally, we select a subset $\mathcal{E}$ of the CDT elements via graphcut energy optimization with optional constraints to enforce strict facet interpolation and manifoldness.

8.4 Winding number

The traditional *winding number* $w(\mathbf{p})$ is a signed, integer-valued property of a point $\mathbf{p}$ with respect to a closed Lipschitz curve $\mathcal{C}$ in $\mathbb{R}^2$. Intuitively, if we imagine there is an observer located at $\mathbf{p}$ tracking a moving point along $\mathcal{C}$, the winding number tells us the number of full revolutions the observer took. Full counter-clockwise revolutions increase the count by one, while clockwise turns subtract one. In other words, $w(\mathbf{p})$ is the number of times $\mathcal{C}$ wraps around $\mathbf{p}$ in the counter-clockwise direction. Without loss of generality let $\mathbf{p} = \mathbf{0}$, parameterize $\mathcal{C}$ using polar coordinates and define

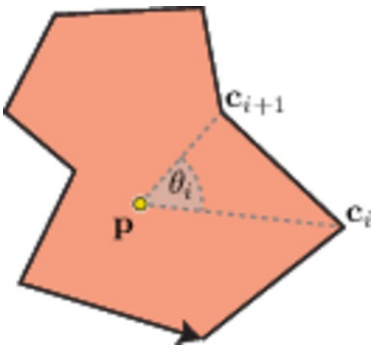
$$w(\mathbf{p}) = \frac{1}{2\pi} \oint_\mathcal{C} d\theta. \tag{8.1}$$

It is the signed length of the projection of $\mathcal{C}$ onto the unit circle around $\mathbf{p}$ divided by 2π (see Figure 8.4). A value of 0 or 1 means $\mathbf{p}$ lies outside or inside $\mathcal{C}$, respectively. The winding

number distinguishes outside and inside for curves enclosing regions of arbitrary genus, and also identifies regions of overlap (see Figure 8.5).

The integral in Equation (8.1) provides an immediate and exact discretization if $\mathcal{C}$ is piecewise linear:

$$w(\mathbf{p}) = \frac{1}{2\pi} \sum_{i=1}^{n} \theta_i, \tag{8.2}$$



where θ_i is the **signed** angle between vectors from two consecutive vertices $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ on $\mathcal{C}$ to $\mathbf{p}$. Let $\mathbf{a} = \mathbf{c}_i - \mathbf{p}$ and $\mathbf{b} = \mathbf{c}_{i+1} - \mathbf{p}$, then:

$$\tan\left(\theta_i(\mathbf{p})\right) = \frac{\det([\mathbf{a}\,\mathbf{b}])}{\mathbf{a} \cdot \mathbf{b}} = \frac{a_x b_y - a_y b_x}{a_x b_x + a_y b_y}. \tag{8.3}$$

8.4.1 Generalization to $\mathbb{R}^3$

The winding number immediately generalizes to $\mathbb{R}^3$ by replacing angle with *solid angle*. The solid angle Ω of a Lipschitz surface $\mathcal{S}$ with respect to a point $\mathbf{p} \in \mathbb{R}^3$ (w.l.o.g. let $\mathbf{p} = \mathbf{0}$) is defined using spherical coordinates to be:

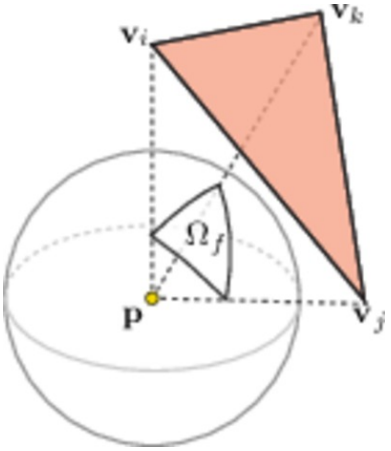
$$\Omega(\mathbf{p}) = \iint_{\mathcal{S}} \sin(\phi)\, d\theta d\phi. \tag{8.4}$$

It is the signed surface area of the projection of $\mathcal{S}$ onto the unit sphere centered at $\mathbf{p}$.

Let the winding number of a closed surface $\mathcal{S}$ at point $\mathbf{p}$ be defined as $w(\mathbf{p}) := \Omega(\mathbf{p})/4\pi$. The same classification properties apply as in $\mathbb{R}^2$. The notion of "winding", now counts the (signed) total number of times the surface wraps around a point.

And again, if we have a triangulated, piecewise-linear surface, there is an immediate and exact discretization of Equation (8.4):

$$w(\mathbf{p}) = \sum_{f=1}^{m} \frac{1}{4\pi} \Omega_f(\mathbf{p}), \tag{8.5}$$



where Ω_f is the solid angle of the **oriented** triangle $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ with respect to $\mathbf{p}$. Let $\mathbf{a} = \mathbf{v}_i - \mathbf{p}$, $\mathbf{b} = \mathbf{v}_j - \mathbf{p}$, $\mathbf{c} = \mathbf{v}_k - \mathbf{p}$ and $a = \|\mathbf{a}\|$, $b = \|\mathbf{b}\|$, $c = \|\mathbf{c}\|$; then following [van Oosterom and Strackee 1983]:

$$\tan\left(\frac{\Omega(\mathbf{p})}{2}\right) = \frac{\det([\mathbf{a}\,\mathbf{b}\,\mathbf{c}])}{abc + (\mathbf{a} \cdot \mathbf{b})c + (\mathbf{b} \cdot \mathbf{c})a + (\mathbf{c} \cdot \mathbf{a})b}. \tag{8.6}$$

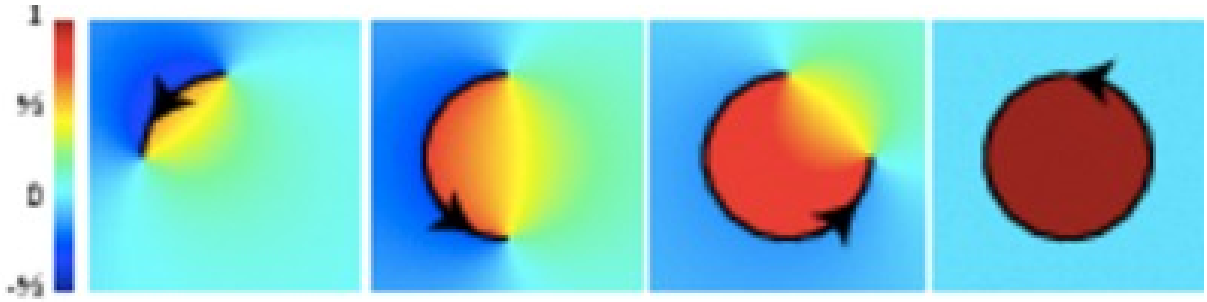


Figure 8.6: *Left to right: winding number field with respect to an open, partial circle converging to a closed circle. Note the ± 1 jump discontinuity across the curve. Otherwise the function is harmonic: smooth with minimal oscillation.*

8.4.2 Open, non-manifold and beyond

The simplicity of the discrete formulae in Equations (8.2) and (8.5) begs the question, what will happen if the input is open? Or non-manifold? Or otherwise ambiguous?

We first consider open curves in $\mathbb{R}^2$. Instead of an indicator, step function, Equation (8.2) is now an otherwise smooth function that jumps by ± 1 across the curve (see Figure 8.6). In fact, the smoothness and fairness of this *generalized winding number* may be well understood. Except on the curve, it is harmonic! This implies $\mathcal{C}^\infty$ smoothness and minimal oscillations — highly desirable properties.

The sum of harmonic functions is harmonic, so it suffices to show that all θ_i and Ω_i are harmonic. This is easy to do using symbolic differentiation and simplification using Maple [Char et al. 1983] (see Figure 8.7). In $\mathbb{R}^3$ treating all triangle vertices $\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k$ as symbolic variables makes Maple run out of memory, therefore we take advantage of invariance to translation and fix $\mathbf{v}_i = (0, 0, 0)$.

The winding number is not *simply* the unique harmonic function determined by setting one side of the boundary to 0 and the other to 1, as if by a diffusion curve of [Orzan et al. 2008] (also cf. [Davis et al. 2002]). This is true if and only if the input is watertight. Rather, the winding number is the sum of harmonic functions corresponding to each input facet, setting one side to $-1/2$ and the other to $1/2$ (see Figure 8.9). We do not explicitly control the boundary conditions — they are implicitly defined by the boundary winding number itself. This allows graceful shift from a perfect segmentation function to a smooth confidence measure as artifacts appear in the boundary. Unlike [Orzan et al. 2008] who solve a variational problem, we have a closed-form expression to evaluate the winding number.

Equation (8.5) may be interpreted as an instance of the boundary element method (BEM) for evaluating the solution to the Laplace equation. If we define Dirichlet boundary conditions on each side of our facets using the winding number, the solution of the Laplace equation on the entire space is exactly equivalent to $w(\mathbf{p})$ for $\mathbf{p} \in \mathbb{R}^d$. This follows from the uniqueness property of harmonic functions.

An alternative understanding of the winding number is to shoot rays in every direction from $\mathbf{p}$. For each ray sum ± 1 for each signed intersection with the input. The traditional *and our generalized* winding number is the average of these values. This understanding is useful

```
# define Laplacian operator in 2d
Laplacian2 := (f,x,y) -> diff(f,x,x) + diff(f,y,y);
# arbitrary position for vi, a := vi - p
a_x := vi_x-px; a_y := vi_y-py;
# arbitrary position for vj, b := vj - p
b_x := vj_x-px; b_y := vj_y-py;
# determinant of (a,b)
detab := a_x*b_y - b_x*a_y;
# a dot b
adotb := a_x*b_x + a_y*b_y;
quotient := detab / adotb;
sub := 2*arctan(simplify(quotient));
simplify(Laplacian2(sub,px,py),symbolic);
# result is 0
```

```
# define Laplacian operator in 3d
Laplacian3 := (f,x,y,z) -> diff(f,x,x) + diff(f,y,y) + diff(f,z,z);
# vi := (0,0,0), a := vi - p
a_x :=    0-px; a_y :=    0-py; a_z :=    0-pz;
# arbitrary position for vj, b := vj - p
b_x := vj_x-px; b_y := vj_y-py; b_z := vj_z-pz;
# arbitrary position for vk, c := vk - p
c_x := vk_x-px; c_y := vk_y-py; c_z := vk_z-pz;
# determinant of (a,b,c)
detabc := a_x*b_y*c_z + b_x*c_y*a_z + c_x*a_y*b_z -
  a_x*c_y*b_z - b_x*a_y*c_z - c_x*b_y*a_z;
a := sqrt(a_x*a_x+a_y*a_y+a_z*a_z);
b := sqrt(b_x*b_x+b_y*b_y+b_z*b_z);
c := sqrt(c_x*c_x+c_y*c_y+c_z*c_z);
# divisor in atan
divisor := a*b*c + (a_x*b_x+a_y*b_y+a_z*b_z)*c +
  (b_x*c_x+b_y*c_y+b_z*c_z)*a + (c_x*a_x+c_y*a_y+c_z*a_z)*b;
wabc := 2*arctan(detabc / divisor);
simplify(Laplacian3(wabc,px,py,pz),symbolic);
# result is 0
```

Figure 8.7: MAPLE *code proving that signed angle in* $\mathbb{R}^2$*, solid angle* $\mathbb{R}^3$*, and, by extension, the winding number are harmonic.*

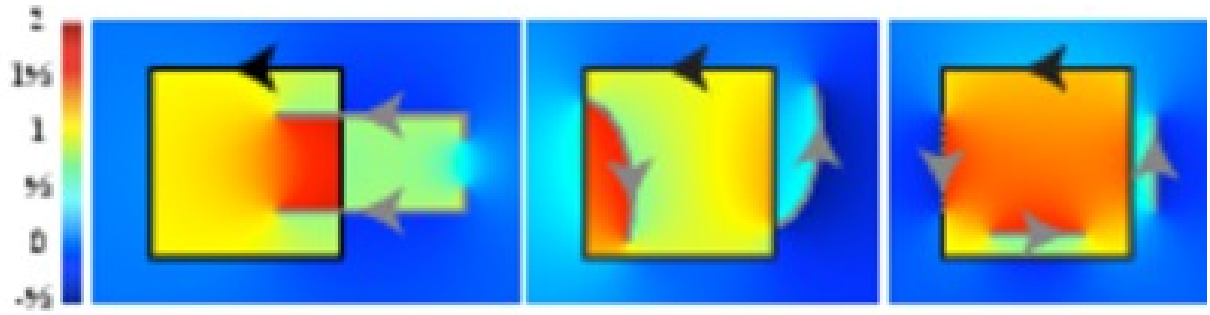


Figure 8.8: *Winding number gracefully handles holes (in grey curve, left), non-manifold attachments (middle), and exactly or nearly duplicate facets (right).*

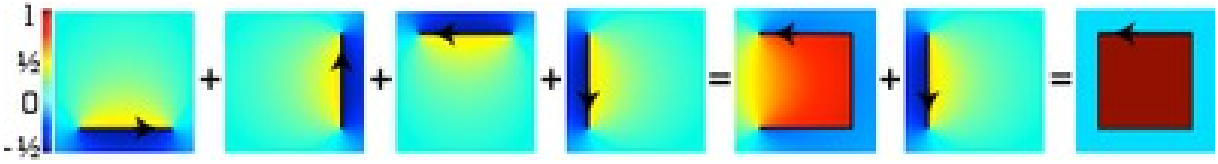


Figure 8.9: *The winding number is the sum of harmonic functions defined for each facet.*

conceptually, but difficult to realize as an algorithm. While casting a few rays is possible [Nooruddin and Turk 2003, Houston et al. 2003], this approximation will be noisy in the presence of open boundaries and non-manifold edges. By considering the input's projection on the unit ball around $\mathbf{p}$ instead, our algorithm is tantamount to shooting all possible rays.

The jump discontinuity across the input facets provides the winding number a unique advantage as a confidence measure in contrast to other methods (e.g. signed distance fields). Such measures continuously approach a zero level-set, where the difference between the measure at a clearly inside point (just to the inside of a facet) and a clearly outside point (just to the outside) diminishes. In contrast, the winding number instead becomes ever more confident and the measure approaches the discontinuous boundary conditions at that facet, regardless of whether the facet is part of a watertight component (see Figure 8.3).

Non-manifold edges appear often in 3D character meshes to describe thin clothing or accessories. It is convenient to conceptually treat each manifold patch of $(\mathcal{V}, \mathcal{F})$ as an appropriately open or closed surface. Each patch then contributes independently to the total winding number. Thus non-manifold edges affect the winding number in a similarly predictable manner to open boundaries (see Figure 8.8 middle).

In character meshes and CAD models, there may be entirely duplicated or nearly duplicated patches of the input mesh. These shift the winding number range locally (see Figure 8.8 right). This disqualifies simply thresholding the winding number for final segmentation, hence our use of a carefully crafted graphcut energy.

8.4.3 Hierarchical evaluation

The discrete formulae in Equations 8.2 and 8.5 give a direct route to a naive implementation to compute $w(\mathbf{p})$: simply sum the contribution of θ_i or Ω_i for each input facet. This is embar-

Algorithm 1: construct_hierarchy($T, \mathcal{V}, \mathcal{F}$)

Inputs:
- T root of subtree in hierarchy
- $\mathcal{V}$ mesh vertex positions
- $\mathcal{F}$ list of facets in bbox(T)

begin
- $E \leftarrow$ exterior_edges($\mathcal{F}$) *// Compute list of exterior edges of $\mathcal{F}$*
- $T.\bar{\mathcal{S}} \leftarrow$ closure(E) *// Compute closure of $\mathcal{F}$*
- **if** $|\mathcal{F}| < 100$ *or* $|T.\bar{\mathcal{S}}| \geq |\mathcal{F}|$ **then**
  - $T.\mathcal{F} \leftarrow \mathcal{F}$ *// mark as leaf and save $\mathcal{F}$*
  - **return**
- **end**
- $\mathcal{F}_{\text{left}} \leftarrow$ restrict($\mathcal{V}, \mathcal{F}$,bbox(T.left)) *// Restriction of $\mathcal{F}$, left*
- $\mathcal{F}_{\text{right}} \leftarrow$ restrict($\mathcal{V}, \mathcal{F}$,bbox(T.right)) *// Restriction of $\mathcal{F}$, right*
- construct_hierarchy(T.left, $\mathcal{V}, \mathcal{F}_{\text{left}}$) *// recurse*
- construct_hierarchy(T.right, $\mathcal{V}, \mathcal{F}_{\text{right}}$) *// recurse*

end

Algorithm 2: hier_winding(p, T, V) $\rightarrow w$

Inputs:
- $\mathbf{p}$ evaluation point
- T root of subtree in hierarchy
- $\mathcal{V}$ mesh vertex positions

Outputs:
- w exact generalized winding number at p

begin
- **if** T *is a leaf* **then**
  - $w \leftarrow$ naive_winding($\mathbf{p}, T.\mathcal{F}, \mathcal{V}$) *// use all faces $T.\mathcal{F}$*
- **else if** $\mathbf{p}$ *is outside bbox(T)* **then**
  - $w \leftarrow -$naive_winding($\mathbf{p}, T.\bar{\mathcal{S}}, \mathcal{V}$) *// use closure $T.\bar{\mathcal{S}}$*
- **else**
  - $w_{\text{left}} \leftarrow$ hier_winding($\mathbf{p}, T$.left, $\mathcal{V}$) *// recurse left*
  - $w_{\text{right}} \leftarrow$ hier_winding($\mathbf{p}, T$.right, $\mathcal{V}$) *// recurse right*
  - $w \leftarrow w_{\text{left}} + w_{\text{right}}$ *// sum*
- **end**
- **return** w

end

rassingly parallel and the geometric definition invites the possibility of a shader-style parallel implementation. However, the asymptotic runtime would still grow linearly with the number of input facets. A facet's effect on $w(\mathbf{p})$ diminishes with respect to its distance to $\mathbf{p}$. We could asymptotically speed up our evaluation with an adaptation of the Fast Multipole Method, however this would only be an approximation. Instead, we achieve exact evaluation and asymptotic performance gains by noticing that the winding number obeys the following simple property.

Consider a possibly open surface $\mathcal{S}$ and an arbitrary *closing surface* $\bar{\mathcal{S}}$ such that $\partial\bar{\mathcal{S}} = \partial\mathcal{S}$ and $\bar{\mathcal{S}} \cup \mathcal{S}$ is some closed, oriented surface $\mathcal{T}$. Then if $\mathbf{p}$ is outside the convex hull of $\mathcal{T}$, we know that $w_{\mathcal{S}}(\mathbf{p}) + w_{\bar{\mathcal{S}}}(\mathbf{p}) = w_{\mathcal{T}}(\mathbf{p}) = 0$. Interestingly this means $w_{\mathcal{S}}(\mathbf{p}) = -w_{\bar{\mathcal{S}}}(\mathbf{p})$, regardless of how $\bar{\mathcal{S}}$ is constructed. Notice this result is trivial if $\mathcal{S}$ is closed, as $w_{\mathcal{S}}(\mathbf{p}) = 0$.

We can conceptually express our mesh as a union of manifold patches. We define *exterior edges* as boundary edges of such a segmentation. In $\mathbb{R}^3$, if $\mathbf{p}$ lies outside the convex hull of $(\mathcal{V}, \mathcal{F})$, then we collect all exterior edges and trivially triangulate each with an arbitrary vertex. Though ugly from a surface repair point of view, these triangles indeed represent a valid closing of $(\mathcal{V}, \mathcal{F})$ and will only be used for winding number evaluation. Note that the segmentation into manifold patches is never explicitly computed. Rather we traverse around each facet in order, and for each directed edge i, j we increment $\texttt{count}(i, j)$ if $i < j$ and decrement $\texttt{count}(j, i)$ if $j < i$. In this way we keep track of how many *extra* times each edge is seen in the forward or backward direction. Finally all edges with $\texttt{count}(i, j) \neq 0$ are declared exterior and triangulated with some arbitrary vertex k with orientation $\{i, j, k\}$ if $\texttt{count}(i, j) = c > 0$ and $\{j, i, k\}$ if $\texttt{count}(i, j) = -c < 0$. These triangles are repeated $|c|$ times to account for possible multiple coverage of the same exterior edge. In $\mathbb{R}^2$, we analogously find *exterior vertices* and connect them to an arbitrary vertex using appropriately oriented line segments.

For reasonably tessellated meshes, the number of exterior edges and thus the number of closing triangles will be $\mathcal{O}(\sqrt{m})$. We exploit this by evaluating the winding number using a bounding volume hierarchy partitioning $\mathcal{F}$. Though there is an art to optimizing bounding volume hierarchies, we opt for a simple axis-aligned-bounding-box hierarchy. We initialize the root with the bounding box of $\mathcal{V}$. We precompute the exterior edges and closure of $\mathcal{F}$, then we simply bisect the box, splitting its longest side. Each facet of $\mathcal{F}$ is distributed to the child whose box contains the facet's barycenter. We recurse on each child. Splitting stops when the number of a box's exterior edges approximately equals the number of its facets or when the number of its facets slips below a threshold ($\approx$100). This stopping criterion ensures that worst case performance stays the same. See Algorithm 1. To evaluate the winding number, we traverse this hierarchy recursively. When we reach a box of which the evaluation point is outside, we evaluate using the closure. See Algorithm 2. In general we see large speed ups (see Figures 8.10 & 8.11).

8.5 Segmentation

We segment according to the winding number by selecting a subset of the elements in a constrained Delaunay tessellation of the convex hull of $(\mathcal{V}, \mathcal{F})$. We may then refine this mesh to meet quality criterion using [Si 2003] or [Klingner and Shewchuk 2007].

Theoretically the only problems when computing a CDT on our input mesh $(\mathcal{V}, \mathcal{F})$ are self-

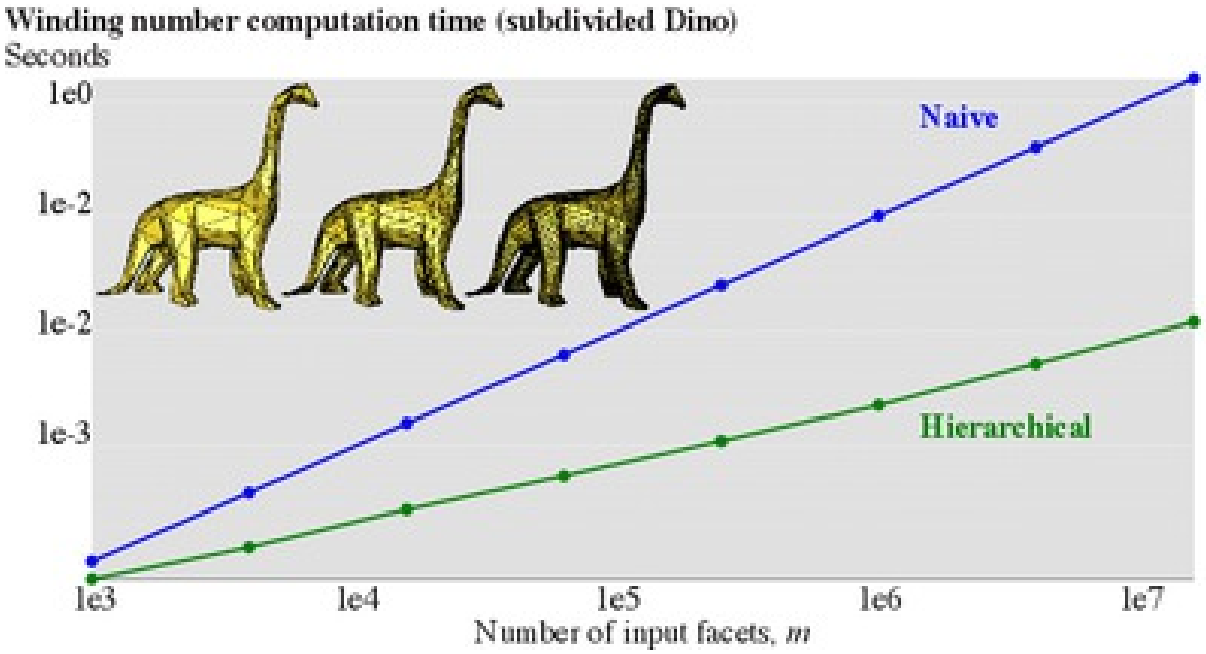


Figure 8.10: *Hierarchical evaluation performs asymptotically better than the naive implementation on the subdivided* Dino. *Naive (blue) fits neatly to $m^{0.94}$, hierarchical (green) to $m^{0.43}$.*

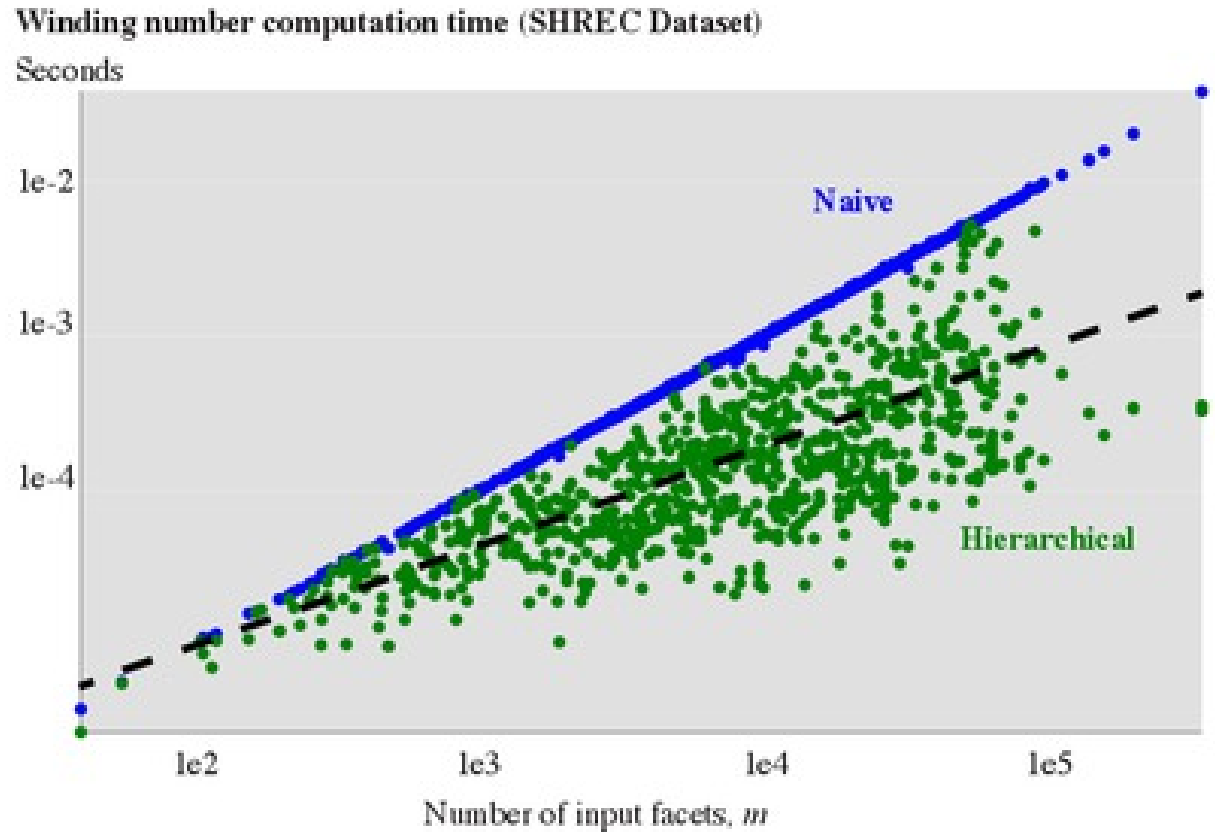


Figure 8.11: *Hierarchical evaluation performs asymptotically better than the naive implementation on a large set of different meshes. Naive (blue) fits neatly to $m^{1.00}$, hierarchical (green) fits in least squares sense to $m^{0.55}$ (black line).*

intersections. In $\mathbb{R}^2$, the TRIANGLE program [Shewchuk 1996] automatically adds Steiner points at line segment intersections. To our knowledge there is no equivalent in $\mathbb{R}^3$. So, we first remove any duplicate or degenerate facets. Then we compute all triangle-triangle intersections using the exact construction kernel in [CGAL]. This kernel is exact even for difficult cases like coplanar, overlapping triangles. It specifies the locations for Steiner points and constraint segments on each offending triangle. We solve a separate 2D CDT problem to meet each set of constraints. Alternatively, employing [Campen and Kobbelt 2010] promises performance gains.

Unfortunately, efficient CDT algorithms are prone to numerical issues and fail when input constraints are too close together. Thus additional clean up is occasionally required. Rather than remesh the entire input, we notice that in practice a CDT is possible when no facets are constrained. Thus we enforce as many facets as permitted by our choice of CDT meshing software [Si 2003]. Troublesome facets are removed or subdivided according to a small area and small angle threshold. Subdivision helps ensure minimal disturbance of the facet interpolation.

By using an imperfect CDT, we are relaxing our strict interpolation constraint. However, surface repair methods like [Attene et al. 2007] are much more aggressive (see Figure 8.17). Further, our preprocessing is solely to facilitate construction of the CDT, which is orthogonal to our volume segmentation problem. All original facets are still used to compute the winding number. When improved CDT methods appear, our method will immediately see benefits.

8.5.1 Energy minimization with graphcut

We now have a standard segmentation problem. If the input is perfectly free of ambiguities then the winding number already acts as an exact segmentation. If the input is not perfectly clean then we need a more sophisticated segmentation. An obvious first approach is to apply a simple threshold:

$$\text{is\_outside}(e_i) = \begin{cases} \text{true} & \text{if } w(e_i) < 0.5 \\ \text{false} & \text{otherwise} \end{cases}, \tag{8.7}$$

where by abuse of notation, let $w(e_i) = \frac{1}{V}\int_{e_i} w(\mathbf{p})dV$ be the integral average of w in element e_i. However, this does not incorporate coherency between neighboring elements (see Figure 8.12).

Instead we propose an energy functional, consisting of a data term and smoothness term, whose minimum respects the winding number, but behaves better due to enforced smoothness. The energy is written:

$$E = \sum_{i=1}^{m} \left[u(x_i) + \gamma \frac{1}{2} \sum_{j \in N(i)} v(x_i, x_j) \right], \tag{8.8}$$

where x_i is the unknown binary segmentation function at element e_i, $N(i)$ is the set of elements sharing a facet with e_i and γ is a parameter balancing the data and smoothness terms. We define the data term as:

$$u(x_i) = \begin{cases} \max(w(e_i) - 0, 0) & \text{if } x_i = \texttt{outside} \\ \max(1 - w(e_i), 0) & \text{otherwise} \end{cases}, \tag{8.9}$$

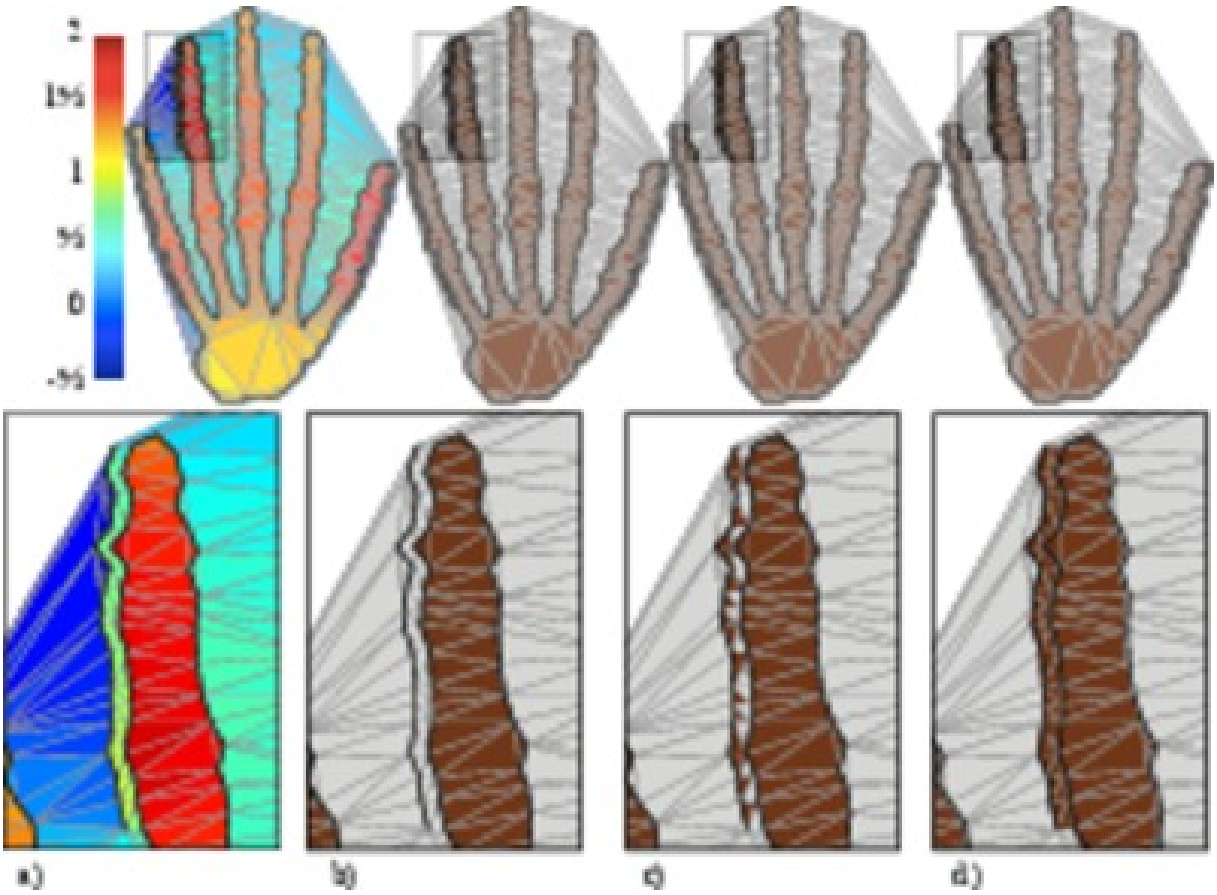


Figure 8.12: *The winding number inside a hand with thin accessories (a). Without constraints accessories may be lost (b). Adding the incident element with highest winding number, recovers them (c). Local improvement of the graphcut energy encourages smoothness (d).*

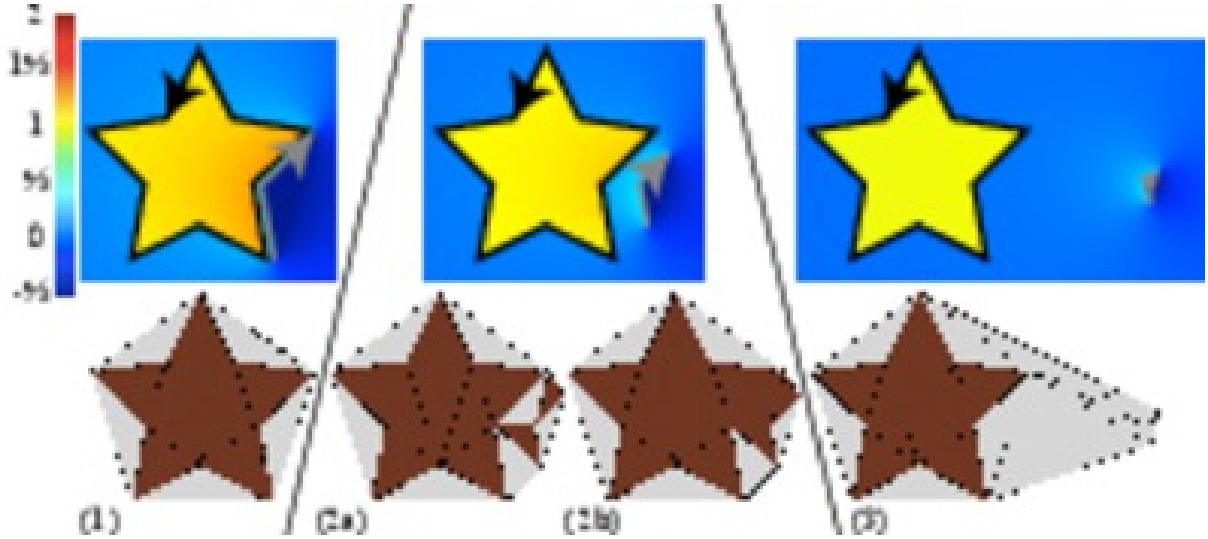


Figure 8.13: *Thresholding winding number finds unambiguous attachments (1). Harder cases require facet constraints. Splinters (2a) are avoided by local improvement with a smoothness energy fixes this (2b). Finally, the winding number can detect outliers (3).*

These terms will become edge weights in a graphcut optimization and thus must be non-negative. If $\gamma = 0$ then the optimal solution coincides with constant thresholding [Chen et al. 2011].

We use an exponential function to achieve a discontinuity-aware smoothness term [Boykov and Funka-Lea 2006]:

$$v(x_i, x_j) = \begin{cases} 0 & \text{if } x_i = x_j \\ \frac{a_{ij} \exp\left(|w(e_i) - w(e_j)|^2\right)}{2\sigma^2} & \text{otherwise} \end{cases}, \tag{8.10}$$

where a_{ij} is the length/area (for $d = 2/3$) of the facet shared between e_i and e_j, and σ is a "noise"-estimation parameter. A graph with appropriate edge-weights is constructed according to [Kolmogorov and Zabin 2004], and the optimal segmentation is found by running a max-flow algorithm.

One last question remains: how to evaluate the integral average of the winding number per element? A simple solution is to evaluate w at the barycenter of each element. This works well for inputs without major issues and when the CDT contains reasonably well-shaped elements. For extremely difficult cases we can increase the accuracy of this integral by using more quadrature points. We use a simple symmetric scheme of [Zhang et al. 2009] and see diminishing returns on the number of points.

8.5.2 Optional hard constraints

Our generalized winding number combined with graphcut can be seen as an outlier detector if some of the input facets $\mathcal{F}$ do not appear as subfacets of the segmented elements $\mathcal{E}$, as this only happens when the input is ambiguous (see Figure 8.13). Unfortunately, we cannot efficiently and *optimally* enforce facet interpolation as hard constraints. Enforcing these constraints as infinite penalty terms in Equation (8.8) results in a *nonregular* function in the parlance of [Kolmogorov and Zabin 2004]. They prove that such functions, and thus our constraints, can not be optimized using graphcut.

For completeness we implement a simple heuristic approach to ensuring facet constraints are met. We march over unsatisfied constraints and satisfy them by adding the incident element with largest winding number. After each update we greedily optimize the energy in Equation (8.8) by recursively testing whether to flip the assignment of elements neighboring any just-flipped elements. During improvement we do not allow flips that violate any already satisfied constraints. We converge to a local and feasible minimum w.r.t. the energy and the facet constraints.

We may similarly enforce a surface manifoldness constraint by marching over edges and vertices in the CDT. When a non-manifold issue is found we simply sort incident elements in descending order according to their winding number and label them inside until local manifoldness is achieved. Again we greedily improve after each step to a local minimum. The method of [Attene et al. 2007] converts sets of tetrahedra (e.g. our output) into manifold volumetric meshes, and alternatively could post-process our output without manifoldness constraints.

Input model							Computation time				Output
Input model	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\partial\mathcal{F}\rvert$	#self-int.	#CC	#nme.	pre.	CDT	w	cut	$\lvert\mathcal{E}\rvert$
Tree	2599	4067	1097	386	32	0	1.99	0.48	1.06	0.06	11643
Holy Cow	2632	5080	206	83	1	0	0.74	0.02	0.05	0.05	9232
Bikini Woman	2827	5204	477	472	11	24	2.29	0.22	0.60	0.06	13057
Ant	2859	5258	152	1578	62	1	7.14	0.48	0.59	0.06	18466
SWAT Man	5277	9820	551	2806	51	24	12.12	1.14	2.47	0.08	31317
Frog	6614	13216	0	316	3	0	1.75	0.39	1.57	0.06	21909
Dog	7953	15848	56	0	1	0	0.51	0.67	2.45	0.07	27707
Rhino	8071	16031	23	2150	26	0	10.29	0.73	4.37	0.10	74446
Alien Space-object	8762	17692	0	1686	32	0	13.41	0.74	7.86	0.10	57293
Skeleton	11963	21551	0	4095	206	0	25.17	4.19	31.87	0.28	217517
Flying Bug	12603	23932	1200	1731	25	0	9.12	1.77	8.69	0.10	62285
Crocodile	17332	34404	0	5236	65	0	22.33	0.20	6.88	0.13	98719
Bear	24936	23530	320	5572	37	0	24.62	0.15	5.38	0.15	56605
Beast	32311	64618	0	969	1	0	7.84	2.98	40.10	0.36	192613
Ballet Woman's Head	39068	76618	1146	8660	44	0	33.19	4.72	92.39	0.10	201991
Big SigCat	40224	60502	1020	3442	65	344	18.88	1.67	9.22	0.12	95896
Phone	42003	83998	0	1597	11	3	15.50	2.76	17.76	0.20	150159
Elephant Head	52740	105056	416	613	5	0	11.26	2.94	19.47	0.27	186025
Ballet Woman	70488	139324	1714	9734	44	0	45.95	7.07	153.23	0.83	615313

Table 8.1: *Statistics for the various examples. $\lvert\mathcal{V}\rvert$ and $\lvert\mathcal{F}\rvert$ are the number of vertices and facets in the input 3D model. $\lvert\partial\mathcal{F}\rvert$ is the number of boundary edges, #self-int. the number of intersecting pairs of facets, #CC the number of connected components, and #nme. the number of non-manifold edges. We report timings for each stage of our algorithm in seconds: (pre.) pre-processing (dominated by self-intersection meshing), constructing a CDT with* TETGEN *(CDT), hierarchically evaluating the winding number w, and final graphcut segmentation (cut). The number of elements in the output tet mesh is $\lvert\mathcal{E}\rvert$.*

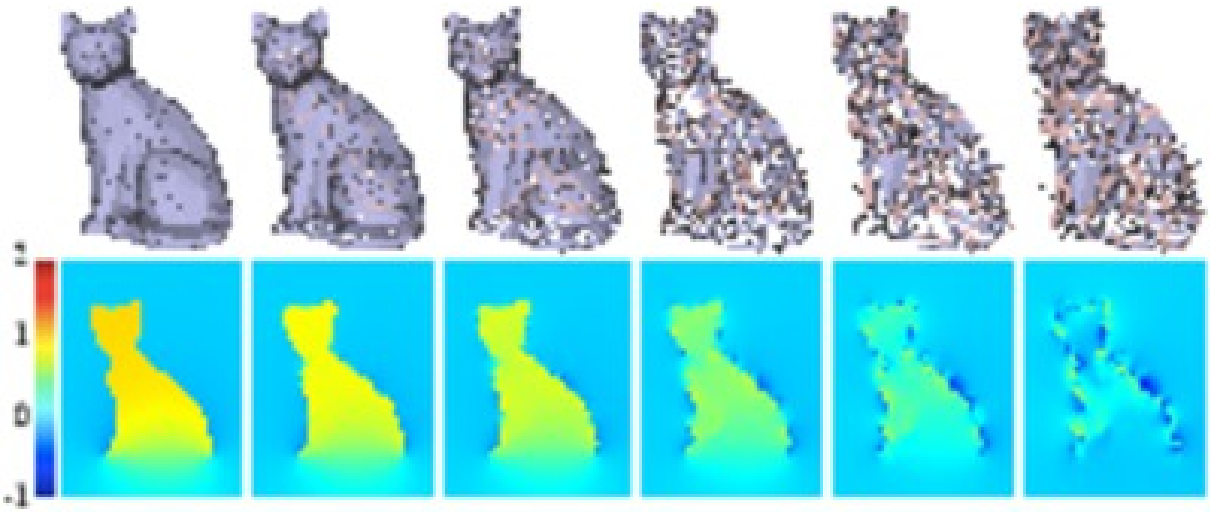


Figure 8.14: *Each triangle of the* Cat *(originally with open bottom) is ripped off and slowly rotated in a random direction. The winding number gracefully degrades.*

8.6 Experiments and results

We evaluated our algorithm on a large number of input shapes (see Figures 8.22-8.25). In this table we show the input mesh, highlighting artifacts; a slice through the bounding box, showing the winding number computed for each element of the CDT; and our resulting tet mesh with cut-away slices. We show success on a variety of man-made meshes: CAD models (e.g. *Phone*, *Alien Space-object*) and character meshes (e.g. *Skeleton*, *SWAT Man*, *Ballet Woman*, *Crocodile*). Our input and output meshes are publicly available as supplemental material to [Jacobson et al. 2013a].

Meshes like the *Skeleton* contain many slightly overlapping connected components. These could be meshed independently and combined using boolean operations, but this complicates implementation and will not work for inputs like *SWAT Man*, whose overlapping components have open boundaries and non-manifold edges. For *SWAT Man*, we activate our optional constraints ensuring that all input facets are contained in the final tet mesh. This is necessary for such applications as physically-based simulation requiring safe contact detection.

The *Ant* has minimal triangulation for the thin legs and antennae, which our method preserves. This not only allows direct access to and assignment of boundary values, but enables efficient storage as the input mesh and our output tet mesh share the same vertex set.

The *Ballet Woman* contains a very detailed mouth (see also *Ballet Woman's Head* in the supplemental video of [Jacobson et al. 2013a]). Our meshing preserves these features while still correctly segmenting *out* the mouth cavity.

We report statistics in Table 8.1. Our timings were obtained on an iMac Intel Core i7 3.4GHz computer with 16GB memory. Our implementation is serial except for computing the winding number, which uses an OPENMP parallel for loop over the evaluation points. We tested the performance of our hierarchical evaluation versus a naive one with two experiments. First, we measured average computation time of a single evaluation in the bounding box of the *Dino* mesh under increasing subdivision levels (see Figure 8.10). Next we considered 700 (target) models of the SHREC dataset [Bronstein et al. 2010] (see Figure 8.11). For both experiments we average the computation time of 1000 random samples in the test shape's bounding box. Both experiments show that in general our hierarchical evaluation performs asymptotically better.

We stress tested our generalization of the winding number by considering how the function responds to degenerating input. The *Cat* in Figure 8.14 has an open base, and its winding number is a smooth (harmonic) field in $\approx[0,1]$. We separate each triangle of the mesh and slowly rotate it in an arbitrary direction, evaluating the effect on the winding number. The winding number field maintains the image of cat until the triangles have rotated by π, when the mesh as a whole clearly breaks our consistent orientation assumption.

We compare our method to first repairing the input as a surface using [Attene 2010] and meshing the result (see Figure 8.15). The *Elephant*'s ear flips inside-outside making volume determination badly ill-posed there: our method deletes the region creating a topological handle. Attene's MESHFIX deletes the region and then fills the hole with a different topology, but other parts of the mesh suffer: the tusks and eyes are also deleted. In Figure 8.17, [Attene 2010] fills the holes in the *Holey Cow* with the same topology as our method, but deletes the entire tail, which self-intersects its udder. Because our method avoids such drastic surface changes, we may com-

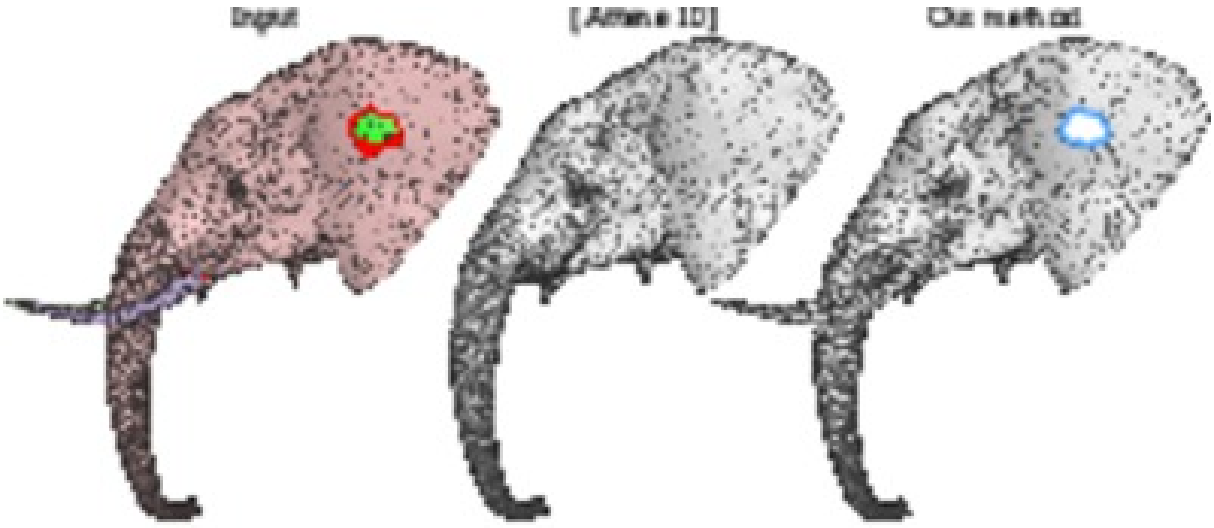


Figure 8.15: *The ears of the* Elephant Head *overlap and flip inside-out (bright green) creating a negative volume. The result of [Attene 2010] creates a watertight surface, but the tusks and eyes are conspicuously missing. Our winding number identifies this region ($w < 0$), but our segmentation removes the region creating a hole (actually topological handle, blue).*

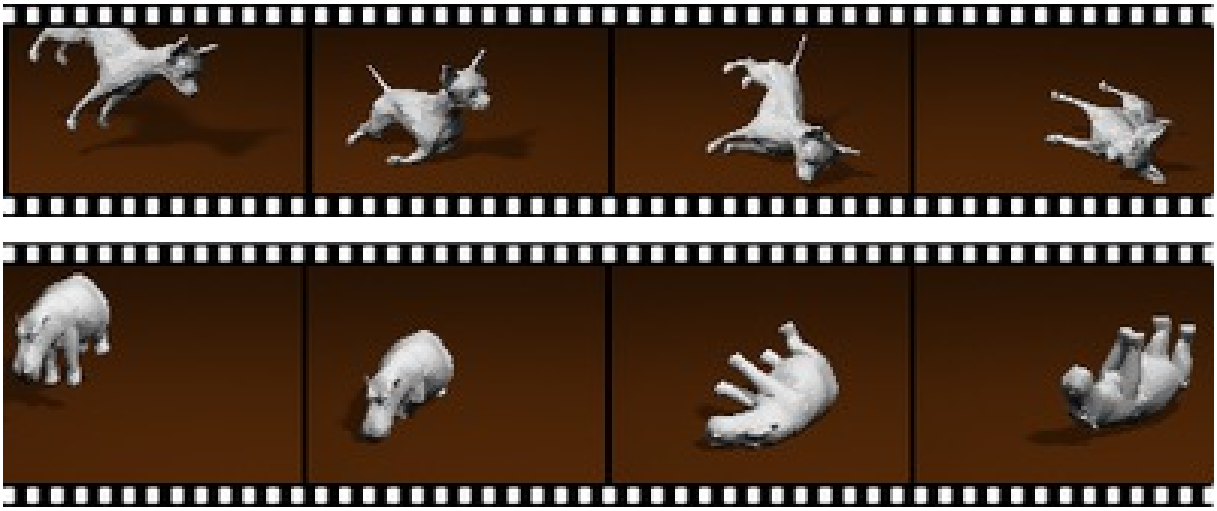


Figure 8.16: *Physically based, elasticity simulation of refined volumes computed with our method.*

pute a volumetric texturing using [Takayama et al. 2008] that meets the original surface (see Figure 8.18). One may then simply render the original surface and only show the inner texture when the *Tree* is cut.

In lieu of computing a volume discretization, many geometry processing tasks may be instead conducted on the surface. For example, the self-intersections in the *Beast* might have previously discouraged the use of a volumetric deformation technique due to the manual cleanup involved in preparing the model for tet meshing. Bending with surface-based technique reveals shell-like collapses when compared to a volumetric technique using a our volume discretization (see Figure 8.19). Some techniques—such as computing skinning weights automatically with methods like the one described in Chapter 4—are designed specifically for volumes (see Figure 8.20). Without our method, this algorithm has a limited set of inputs or requires tedious

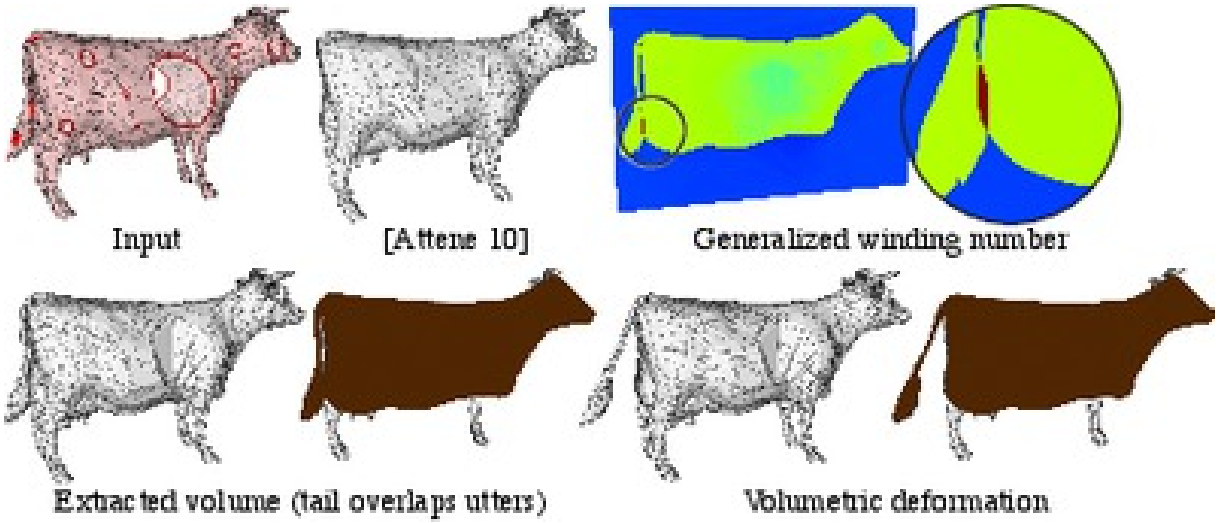


Figure 8.17: *Left to right:* Holey Cow *with its tail intersecting its udder. [Attene 2010] fills the holes, but deletes the tail. A slice through the winding number shows correct assignment of 0 outside, 1 inside the main part, and 2 inside the overlapping tail (red), inset. This may be meshed as usual gluing the tail to the body. Or we may duplicate this doubly covered region and* glue *it to either side. This allows the tail and its volume to be pulled out.*

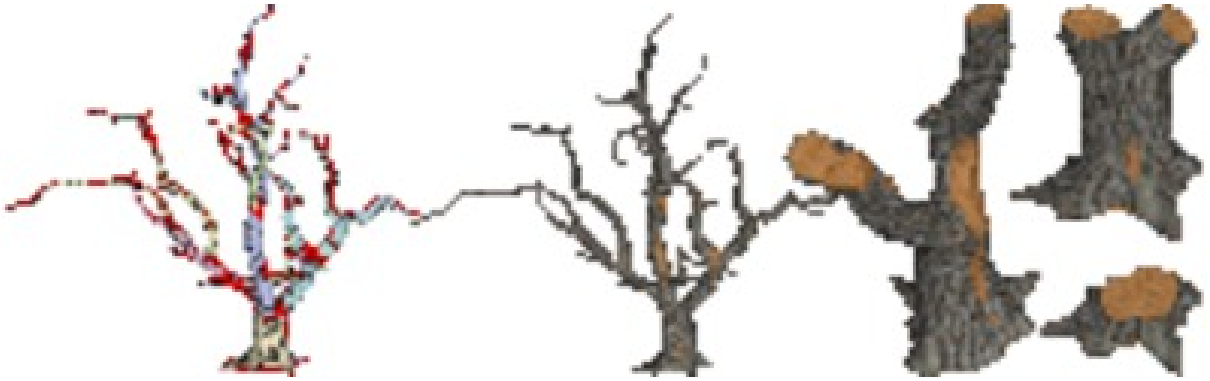


Figure 8.18: *The* Tree *contains many intersections and open boundaries (left). Our method is robust to these, producing a compatible mesh for applying volumetric texturing (right).*

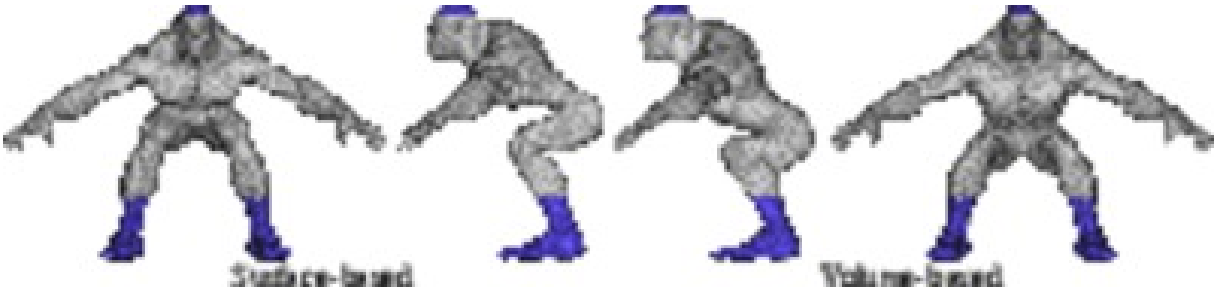


Figure 8.19: *Self-intersections in the otherwise clean* Beast *prevent volume-meshing with previous methods. Surface-based deformation is one option, but bending causes shell-like collapses not present in a volume deformation enabled by our method.*

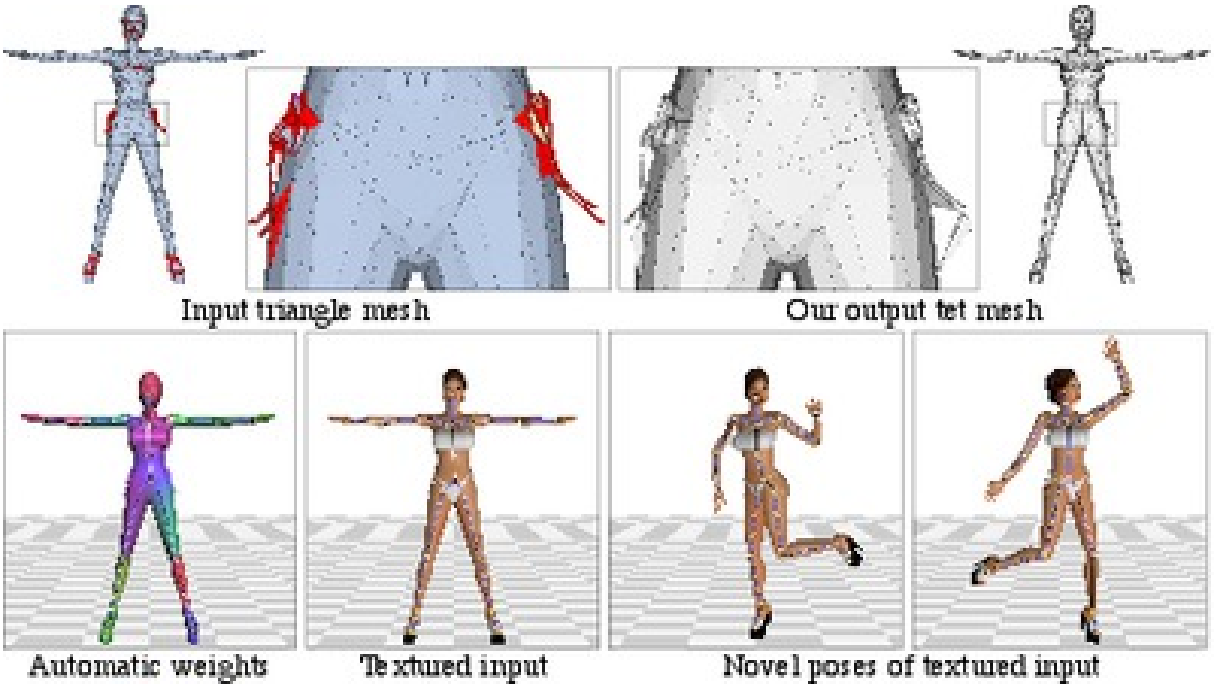


Figure 8.20: *Left to right: the* Bikini Woman *has many artifacts, as well as thin sheet-like accessories. Their volumes are ambiguous, but our facet constraints ensure some trivial connection. This enables automatic, volumetric skinning weight computation (see Chapters 4 & 5) on a refinement of our output. Only the weights on the original vertices are needed to deform the original textured input mesh.*

user preparation of input (defeating its automation gains). State of the art physically-based elasticity simulation techniques also require tetrahedral volume meshes. Our method accordingly expands the domain of inputs for these methods (see Figures 8.16).

8.7 Limitations and future work

The winding number and our generalization rely heavily on the orientation of input facets. Triangle soups with unknown or erroneous orientations would need further preprocessing (e.g.

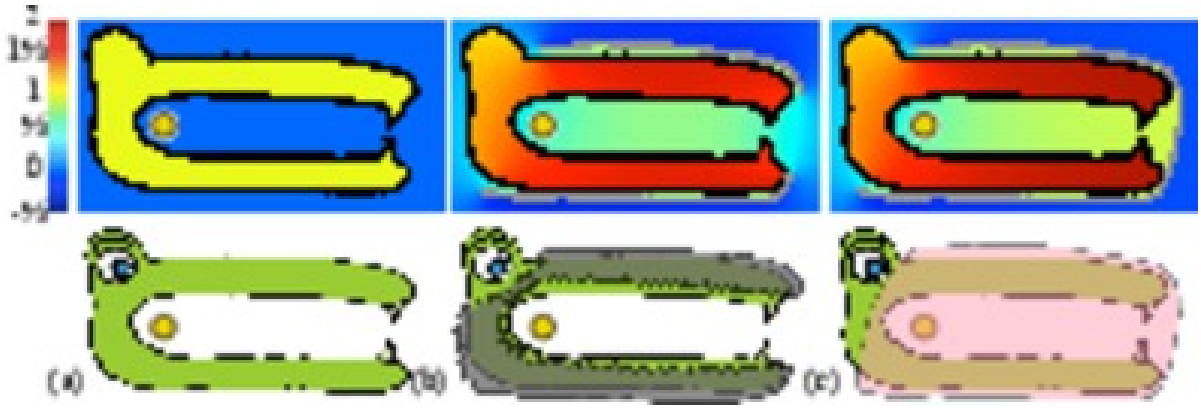


Figure 8.21: *Inside-outside of the* Snake *(a) becomes ambiguous when thin sheets are used to represent accessories such as a ski-mask (b) or a muzzle (c). In (b) and (c) the winding number at the yellow points are similar, but the semantic inside-outside classification is opposite.*

with [Borodin et al. 2004]). Since a single facet has a drastically different effect on the total winding number when its orientation agrees with its neighbors, it would be interesting to use the notion of our generalized winding number to verify or correct triangle orientations.

The number of connected components in our output is not controlled even when manifoldness is constrained. It would be interesting to extend the work of [Chen et al. 2011] to 3D, enabling such topological constraints in our graphcut segmentation.

Many meshes contain sheet-like features not part of the main solid *body*, such as leaves on a tree or cape on an action hero. Such features are typically *two-sided* and would require special treatment to consider the thin solids they represent. Conversely the accessories or nearly duplicated regions we do handle may also cause ambiguities. When duplicated surfaces nearly enclose a concavity, the winding number increases and may cause the region to be marked as inside (see Figure 8.21). The difference between inside and outside in these cases is a matter of semantics. To alleviate this, such accessories could be tagged as non-participatory for the winding number computation, but still constrained during our segmentation. Achieving such tagging automatically is an interesting direction for future work in the accelerating field of retrieving semantics from 3D shapes.

Our generalized winding number correctly identifies regions of overlap even in the presence of surface artifacts such as holes (see Figure 8.17). This suggests the ability to construct volume discretization that *respect* self-intersections of the original surface (rather than "correct" them). We show a proof-of-concept of this idea, by duplicating the meshing inside the *Holey Cow*'s overlapping tail (where $w \approx 2$) and gluing separately to the tail and body. The tail and its volume may then be deformed in and out of the body. More complicated overlaps are far from trivial to untangle and we continue to investigate this problem in our future work.

8.8 Conclusion

Generalizing the winding number to arbitrary triangle meshes proves to be a powerful and mathematically beautiful tool. The core of our method is simple to implement and our hierarchical

acceleration structure enables efficient evaluation on large models. The winding number's harmonic nature and implicitly defined, discontinuous boundary conditions make it ideal for guiding our graphcut segmentation when input meshes contain self-intersections, open boundaries, and non-manifold pieces. We hope that our algorithm's success on previously *unmeshable* models will encourage volumetric processing of solid shapes throughout computer graphics.

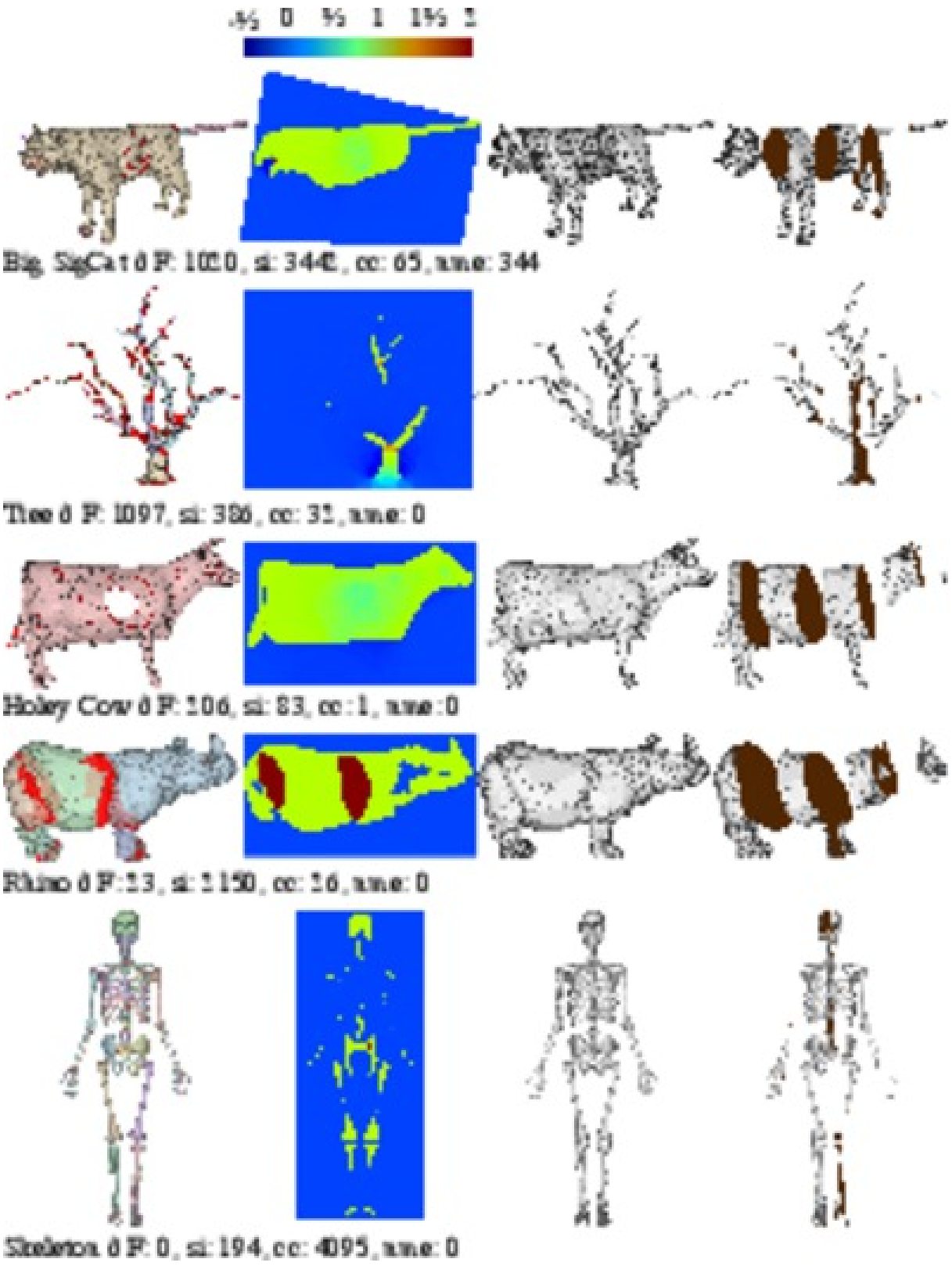


Figure 8.22: *Each row shows left to right: input model with connected components randomly colored, self-intersections facets marked in red, open boundaries in dark red and non-manifold edges in purple; slice through CDT visualizing winding number; surface of output mesh;* hot-dog *slice view of output mesh.*

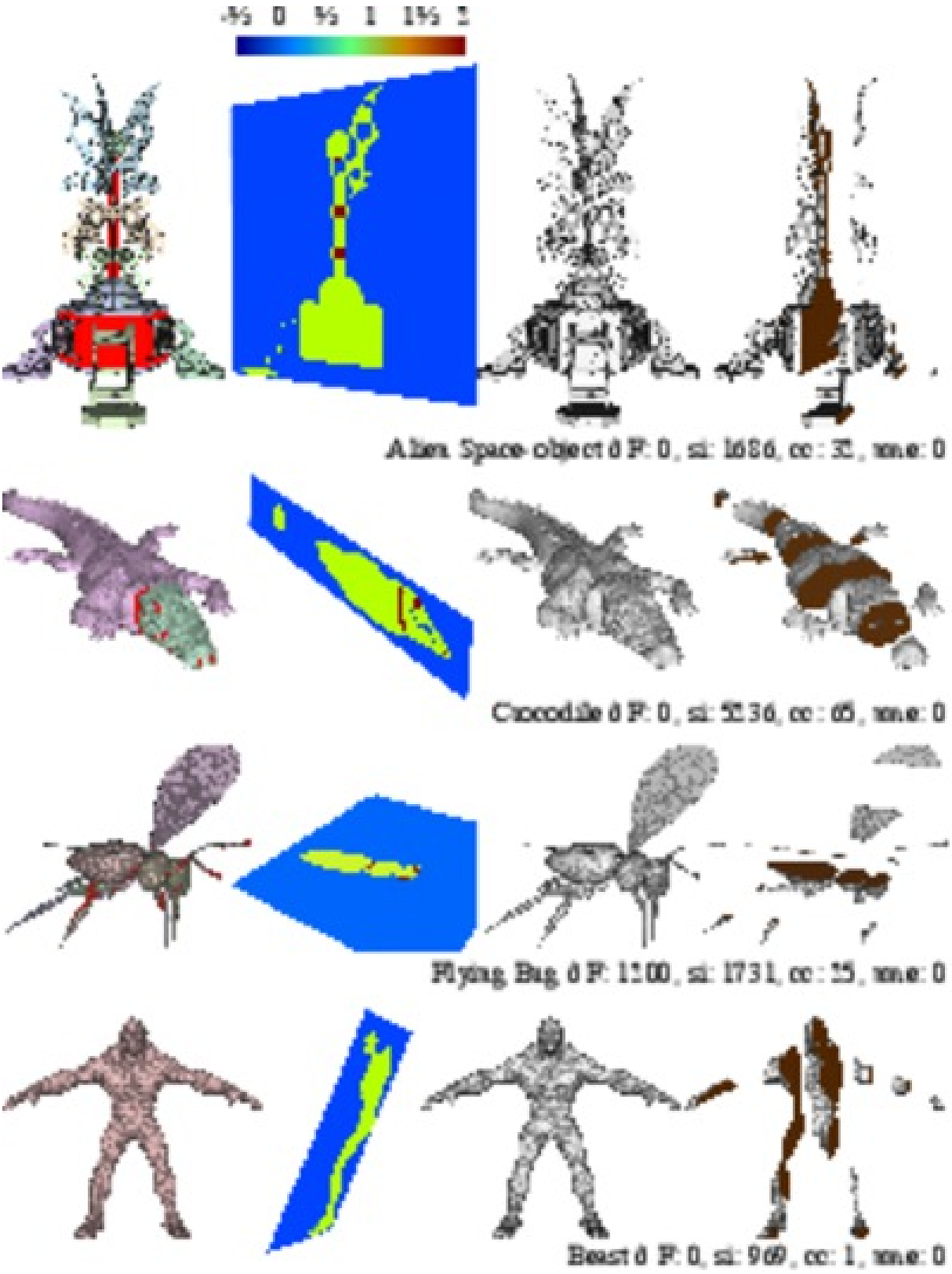


Figure 8.23: *Results continued.*

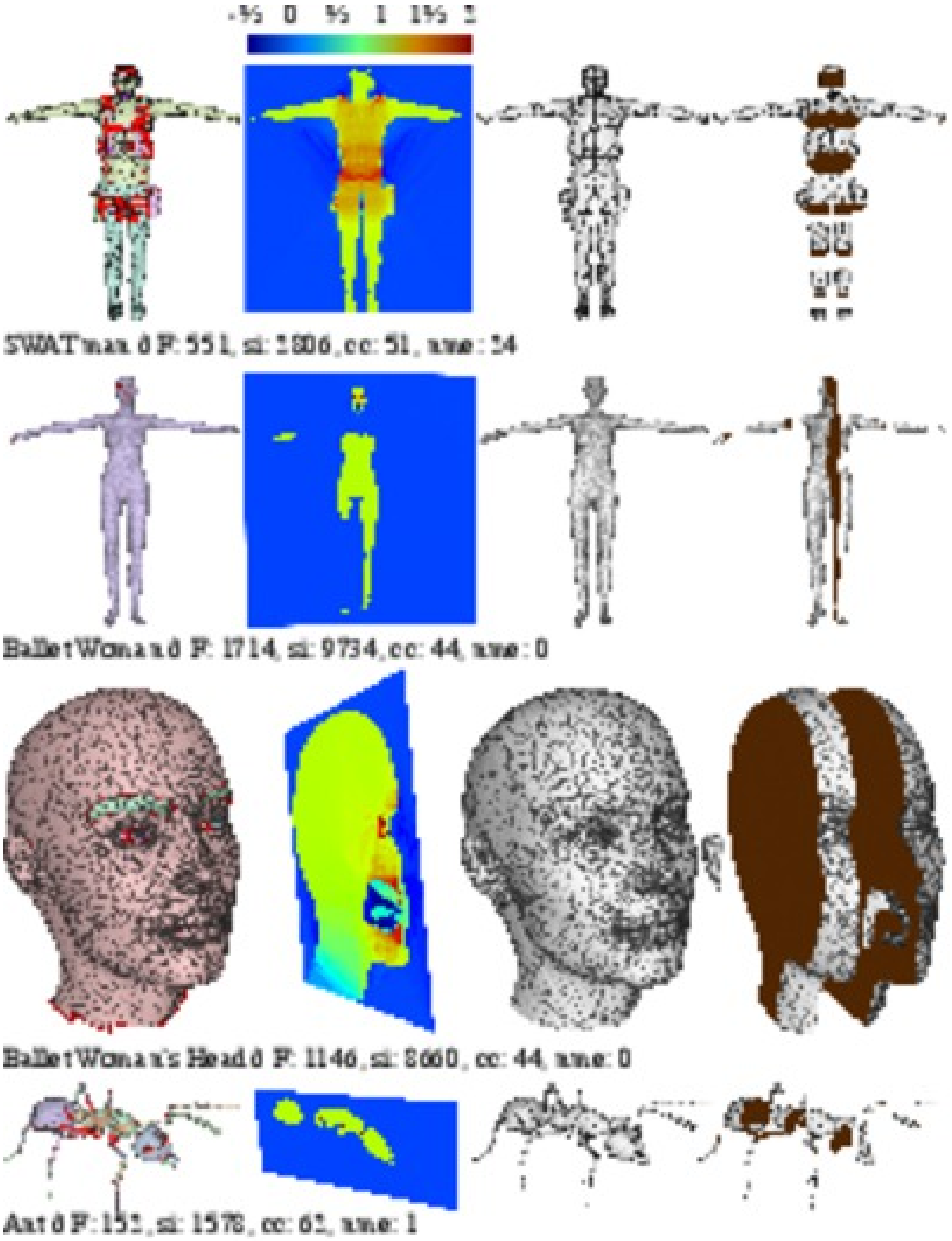


Figure 8.24: *Results continued.*

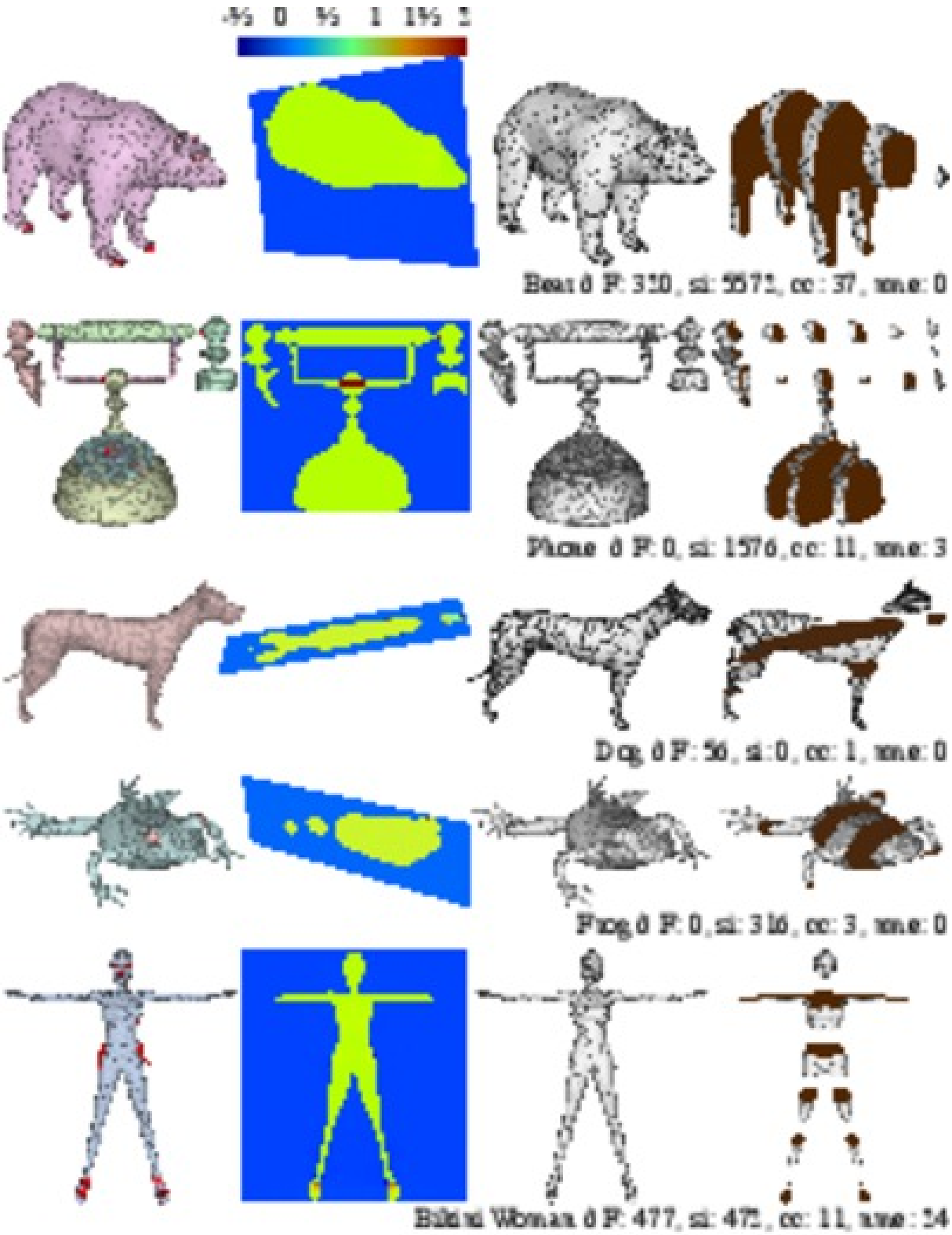


Figure 8.25: *Results continued.*

9

Conclusion

Our sequence of algorithms for real-time shape deformation climaxes in a system that can deform shapes using nonlinear elastic energy optimization at unprecedented rates (see Chapter 7).

Although our algorithms require a bit more machinery than, say, Phong shading, we argue those Phong principles of Chapter 1 still apply. All of our deformation works are deemed successful only because of their ability to create perceptually believable and visually pleasing deformations. Although these claims are subjective (one could and *should* conduct a perceptual user study to verify them), we derive our techniques from common place assumptions, e.g. bones should deform rigidly and if a user moves a control to the left, the shape should not move to the right.

As for simplicity, our required mathematics are not much more than that which is described in Chapter 2. The most advanced subroutines necessary for reproduction are sparse Cholesky decomposition and 3D constrained Delaunay tessellation. Both, thankfully, exist already in robust, efficient libraries (e.g. [MATLAB 2012]).

By founding our work around linear blend skinning, we invariantly maintain real-time frame rates through out our shape deformation techniques. Similarly, with each work we strive to provide practical and expressive interface to our algorithms' parameters. In fact, it is with these interfaces in mind that we design our algorithms in the first place. Hence, human control and expressiveness remains easy.

9.1 Recapitulation of core contributions

We began by establishing a continuous foundation for useful intrinsic, shape energies in Chapter 3. This gave us reassurances when employing discretizations (e.g. of the Laplacian Energy) in later chapters in terms of mesh independence and convergence. This is not only practically important for applications, but also convenient for conducting the research in subsequent chapters: we can prototype on low-resolution discretization with confidence that our solutions approximate continuous solutions and hence will agree with high-resolution solutions.

With this foundation, we examined how such energies can be employed to craft linear blend skinning weight functions for real-time deformation. As we found, minimizers of these energies as employed previously for point and region handles (e.g. [Botsch and Kobbelt 2004, Sorkine et al. 2004]) were not sufficient for two core reasons.

First, some tasks are better completed with alternative control structures such as rigid skeletons and cages. Supporting all handle types and all combinations is important for providing the user with the right tool for her task. Second, while these energies produce fair deformations or surfaces, they do not produce quality weight functions, insofar as they do not simultaneously satisfy a list of redeeming properties: smoothness, boundedness, monotonicity, locality.

We designed automatic weight functions that support all handle types (via appropriate boundary conditions) and meet these properties by constrained optimization. This began with finding biharmonic functions subject to box constraints which directly enforce boundedness and facilitate locality (Chapter 4), and culminated in a system that approximates solutions to a nonlinear, nonconvex optimization promising monotonic weight functions (Chapter 5).

Our weight functions do not ensure that the resulting surface or displacements are biharmonic functions, but rather they tell us that the *blending* is biharmonic (see Section 4.7). Comfortable with this concept, we explored different blending schemes, more powerful then simple linear blend skinning. In Chapter 6, we derived a new, nonlinear skinning formula which enables stretching and twisting of skeleton bones. We achieved this by utilizing a second set of our previously defined weight functions. Our automatic weight functions prove to be an important tool in our research. We can quickly prototype such ideas for skinning, which would otherwise require trained artists.

We further expanded the space of deformations possible with skinning, this time not by changing the underlying LBS formula but by enabling a simpler interface (Chapter 7). By defining nonlinear, elastic energy optimization in the subspace of a given skinning rig, we may optimize for all but a subset of the skinning transformations. In this way we met users' high level constraints, while ensuring a quality deformation of the shape itself. This reduced the effort required to find new poses and define animations. With extra, procedurally generated weight functions, we bolstered the LBS subspace into a space capable of achieving variational modeling results on par with full-resolution, nonlinear techniques, but orders of magnitude faster.

Finally, all this work in 3D remains of only academic interest without the ability to provide our algorithms with suitable input. In 3D, we operate on volume discretizations, which are difficult to obtain for typical shapes represented by triangle meshes with holes, non-manifold edges, multiple components and self-intersections.

In Chapter 8, we designed a new confidence function by generalizing the winding number for non-watertight models. Treating this function as a data-term, we used graphcut energy optimization to segment a constrained Delaunay tesslation of the domain. Using a CDT ensures exact interpolation of the input mesh. This makes integration our previous methods easier and less prone to interpolation errors. Our generalized winding number function enjoys a number of nice properties, stemming from the fact that it is harmonic. We were only able to prove this fact by first noticing it empirically. Our meticulous study of polyharmonic functions made it obvious that our generalized winding number *just had to be* harmonic: it was only a matter of proving it.

9.2 Publications

The work contained in this thesis has lead to a number of scientific publications, appearing in top-tier graphics journals (*ACM Transactions on Graphics* and *Computer Graphics Forum*) and presented at top-tier graphics conferences (SIGGRAPH, SIGGRAPH Asia, and ACM/Eurographics Symposium on Geometry Processing).

9.2.1 Journal publications

JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, to appear.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE-HORNUNG, O. 2013. Bounded biharmonic weights for real-time deformation. *Communications of ACM*, to appear.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4.

JACOBSON, A., WEINKAUF, T., AND SORKINE, O. 2012. Smooth shape-aware functions with controlled extrema. In *Proc. SGP*.

JACOBSON, A., AND SORKINE, O. 2011. Stretchable and twistable bones for skeletal shape deformation. *ACM Trans. Graph. 30*, 6, 165:1–165:8.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4.

JACOBSON, A., TOSUN, E., SORKINE, O., AND ZORIN, D. 2010. Mixed finite elements for variational surface modeling. In *Proc. SGP*.

9.2.2 Technical reports

JACOBSON, A. 2012. Bijective mappings with generalized barycentric coordinates: a counterexample. Tech. Rep. 780, ETH Zurich.

JACOBSON, A., AND SORKINE, O. 2012. A cotangent Laplacian for images as surfaces. Tech. Rep. 757, ETH Zurich.

9.3 Reflections

This thesis represents over three years of research on real-time shape deformation. This has resulted in a wealth of code and a healthy repertoire of techniques. First hand implementations of and experimentation with dozens of previous techniques has been incalculably valuable to progressing the research and culminating in the contributions of this thesis. These experiments have cultivated a perspective which can hopefully assist others who wish to reimplement this work or begin similar explorations into the exciting field of real-time shape deformation.

Constrained optimization is a powerful hammer. Learning to use modern solvers and understanding—at some reasonable level—how they work has been essential to this thesis. Applying the variational principle to our problems results in energy optimization, so it is prudent to know what is currently possible and what is not. For example, Chapters 4 and 5 depend heavily on the availability of efficient, large sparse quadratic programming solvers. Knowing that such a solver exists guides our decision in Chapter 5 to convert the topological constraints of the difficult, ideal problem to the linear inequality constraints of our more practical problem. It is also important to embrace the variational principle as a means to verify results. Two important questions to ask are: "Is the energy lower?" and "Has the optimization converged?" If the energy is lower, but the solution looks worse, then perhaps the energy is wrong. This may not always mean that a better energy exists in plain sight. In the end, we are always only approximating that ill-posed energy which measures "user surprise". Evaluating whether convergence has been obtained is critical, not just for ensuring the best possible results but also for verifying that the solver is doing its job. We were long confused by the lack of smoothness and locality when prototyping our weight functions in Chapter 4. Eventually we realized that convergence was not being obtained with the solver we were using at the time [Byrd et al. 1995]. Noticing this led to switching solvers (to MOSEK or the active set technique in Section 2.2.3) and only then could we properly experiment to verify the properties of our new weight functions.

All of the methods in this thesis were decidedly implemented *without* an advanced (complicated) mesh data-structure (e.g. half-edge data structure). Instead a mesh is nearly always represented simply as a list of vertex positions $\mathbf{V} \in \mathbb{R}^{n \times 3}$ and a list of triangle or tetrahedra indices $\mathbf{F}$ or $\mathbf{T}$. This not only matches our linear algebra manipulations (e.g. Chapter 7) and facilitates working with the standard computer graphics pipeline, but also places focus on globally defined, intrinsic quantities (e.g. Laplacian energy) rather than local ones (e.g. sharp features). This not coincidentally mirrors the focus of this thesis. In choosing a mesh data structure for research prototyping, we have found that one should justify any inconvenience or complication. For example, to compute the cotangent Laplacian on an irregular mesh, a per-triangle definition suffices with a $(\mathbf{V}, \mathbf{F})$ storage of a mesh, whereas a loop over the neighborhood rings nested inside a loop over each vertex in a half-edge data structure is unnecessarily complicated and more prone to border-case errors (not to mention it strays farther from our FEM roots in Section 2.1). Further, it has been much easier to *level up* an implementation when necessary—from $(\mathbf{V}, \mathbf{F})$ to CGAL or OPENMESH—than to *level down*. We have embraced this concept and built our open source prototyping library, LIBIGL, in MATLAB and C++ [Jacobson et al. 2013b].

Throughout our projects, we strive to maintain dimension invariance (at least up to $\mathbb{R}^2$ and $\mathbb{R}^3$). This not only increases the applicability and scope of each of our contributions, but also provides a straightforward path for research. In $\mathbb{R}^2$, shapes may be quickly designed to test a

theory at hand (see Section 9.4.2). Making the leap to $\mathbb{R}^3$ is not always trivial: rotations are no longer commutative, curvatures are more complicated, volume-meshing is more involved, etc. However, most of our core concepts are directly analogous and we see immediate benefits by first experimenting in $\mathbb{R}^2$.

9.3.1 Lingering, unsolved problems

Our linear blend skinning subspace in Chapter 7 regularizes the as-rigid-as-possible energy optimization into producing smooth deformations. Otherwise the full-resolution solution would show derivative discontinuities at fixed constraints (see Figure 7.8) and internal singularities in the rotation field (see Figure 7.7). Our subspace inherits its smoothness from our weight functions, which are defined according to energies corresponding to sufficiently high-order PDEs. Directly combining high-order smoothness with nonlinear elastic energy optimization at the full, unreduced level remains a challenging problem. Even standing state-of-the-art full resolution solutions (e.g. [Botsch et al. 2006a, Botsch et al. 2007b]) show derivative discontinuities, kinks, near constrained handles. One promising direction is to bridge the gap between sparse subspace deformation and intrinsic full resolution deformation using smooth intrinsic bases [Hildebrandt et al. 2011].

Isolated point constraints are perhaps the most primitive and most intuitive interface for modeling, but also the least rigorously defined. To study point constraints rigorously would be to reexamine their fundamental motivation. Is a point constraint really just a small region constraint? How small? If it is infinitesimal then how can we measure its influence properly?

In our experiments dealing with bone handles of an internal skeletons, we have noticed that the precise placement of joints (where bones meet) has a drastic impact on the resulting deformation [Kavan and Sorkine 2012]. It not only affects our automatic weight computation, but also determines the relative center of rotation during posing. This means it affects not just the deformation quality, but the effectiveness of the interface itself. This sensitivity could be alleviated by assisting the user's placement or repositioning of handles in their rest state.

In our work, we ignore collisions (a.k.a. interference). Ensuring local and global injectivity while meeting user constraints is an important and largely still unsolved problem in modeling [Harmon et al. 2011]. Treating collisions would imply new interpretations of intrinsic and semantic information. However, the discontinuous nature of collision response makes it difficult to deal with, not to mention while maintaining real-time performance.

In order to achieve real-time performance in our current work, we push as much computation as we can into *precomputation* before the deformation session starts. Much of our precomputation efforts are spent reducing the full resolution deformation problem to a smaller subspace. For example, our smooth weight functions remove unnecessary high-frequency deformations resulting in a more appropriate low-frequency subspace of deformations from a sparse set of handles. The very problem statement of handle-based deformation presumes such a low-frequency solution, but our precomputation ignores this. Much effort is spent computing weights via energies defined on a fine resolution discretization of the domain, but we know *a priori* that the solution will be considerably lower frequency. For example, the minute body hairs on a character will not significantly effect the weight function attached to the forearm bone. This implies that we

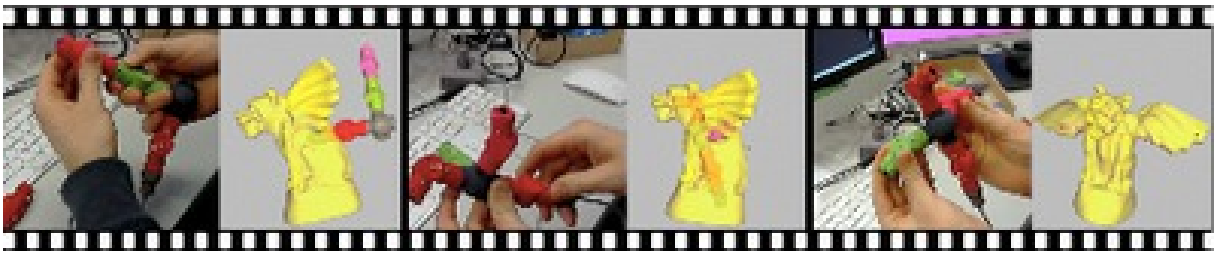


Figure 9.1: *Our physical interface prototype can be constructed on the fly and used to control skeletons of arbitrary topology.*

could get away with much coarser representations of our input geometry for precomputation tasks. Doing so would require care to ensure intrinsic information like a shape's topology and geodesic distances are respected. We would need to balance approximation power with savings from a coarser representation.

The more we can improve precomputation performance, the more we can push these subroutines into the interactive experience. This opens new avenues for interactive weight definition, where the user adjusts handle positions and sees changes to deformations in real-time; or for on-the-fly subspace adjustment for collision response in simulation (Section 7.7).

9.4 Future work

The story of real-time shape deformation does not end with this thesis, but rather begins a new chapter. We can continue to push the quality and control achievable in real-time. We are now interested in measuring *quality* in terms of a deformation's incorporation of semantics and in exploring novel interfaces for controlling our real-time deformation systems.

9.4.1 Physical interfaces

We are deforming 3D shapes at near haptic rates. Yet we mostly control these deformations using a 2D mouse and our feedback is via a 2D monoscopic display. Even in a reduced skeleton-based deformation system the number of degrees of freedom is on the order of 100. Chapter 7 addresses this for intuitive automatic assignment, but if we want full control then we need a new interface with an equitable amount of degrees of freedom.

We are currently working on designing a device with such degrees of freedom. Leveraging recent advances in 3D printing and electronic sensing, we have designed a set of mechanical *joints* which continuously report their configuration (see Figure 9.1). Each joint's 3 rotational degrees of freedom may then be mapped (at > 120 Hz) to the rotational degrees of freedom in a virtual skinning skeleton controlling a character on screen. The joints may be rearranged into any tree topology: the same parts can build a human, spider, or hand.

9.4.2 Semantics

Our weight functions incorporate semantics implied by the user's placement of control handles (e.g. skeletons tell us which parts should be rigid), but this is still indirect. In future work, we would like to investigate what are the best interfaces for the user to directly indicated semantics with minimal effort. We are inspired by the success of Daniel Sýkora and colleagues' work on designing interfaces for manipulating 2D cartoons [Sýkora et al. 2005, Sýkora et al. 2009].

Another interesting question is whether semantics useful for real-time shape deformation can be derived automatically. Even in the four years writing this thesis, we have witnessed an explosion in the amount of 3D data. Cutting-edge works by Guibas and colleagues (e.g. [Ovsjanikov et al. 2011, Huang et al. 2012] and Cohen-Or, Zhang, and colleagues (e.g. [Sidi et al. 2011, Wang et al. 2012]) have studied how co-analysis of large sets of shapes may be used effectively for tasks like classification, exploration, correspondence and segmentation. We would like to point these high-powered techniques at improving real-time shape deformation quality. Perhaps we can leverage the robust volumes of Chapter 8.

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Trans. Graph. 24*, 3.

ALLIEZ, P., COHEN-STEINER, D., TONG, Y., AND DESBRUN, M. 2007. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. SGP*.

AMARA, M., AND DABAGHI, F. 2001. An optimal C^0 finite element algorithm for the 2D biharmonic problem: theoretical analysis and numerical results. *Numer. Math. 90*, 1, 19–46.

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph. 27*, 165:1–165:10.

ANDERSEN, E. D., AND ANDERSEN, K. D. 2000. The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High Performance Optimization*. Kluwer Academic Publishers, 197–232.

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24*, 3.

ATTENE, M., FERRI, M., AND GIORGI, D. 2007. Combinatorial 3-Manifolds from Sets of Tetrahedra. In *Proc. CW*.

ATTENE, M. 2010. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer 26*, 11, 1393–1406.

AU, O. K.-C., TAI, C.-L., LIU, L., AND FU, H. 2006. Dual Laplacian editing for meshes. *IEEE TVCG 12*, 3, 386–395.

AU, O. K.-C., FU, H., TAI, C.-L., AND COHEN-OR, D. 2007. Handle-aware isolines for scalable shape editing. *ACM Trans. Graph. 26*, 3, 83.

BADLER, N. I., AND SMOLIAR, S. W. 1979. Digital representations of human movement. *ACM Comput. Surv. 11*, 1, 19–38.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3, 72:1–72:8.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph. 24*, 3, 982–990.

BARTH, T. J. 1992. Aspects of unstructured grids and finite-volume solvers for the euler and navier-stokes equations. In *AGARD, Special Course on Unstructured Grid Methods for Advection Dominated Flows*, vol. 1.

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Variational harmonic maps for space deformation. *ACM Trans. Graph. 28*, 3, 34:1–34:11.

BISCHOFF, S., AND KOBBELT, L. 2005. Structure preserving CAD model repair. *Comput. Graph. Forum 24*, 3, 527–536.

BISCHOFF, S., PAVIC, D., AND KOBBELT, L. 2005. Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4.

BLAIR, P. 1994. *Cartoon Animation*. Walter Foster Publishing, Inc., Irvine, CA, USA.

BLOOR, M. I. G., AND WILSON, M. J. 1990. Using partial differential equations to generate free-form surfaces. *Computer Aided Design 22*, 4, 202–212.

BORODIN, P., ZACHMANN, G., AND KLEIN, R. 2004. Consistent Normal Orientation for Polygonal Meshes. In *Proc. CGI*.

BOROSÁN, P., HOWARD, R., ZHANG, S., AND NEALEN, A. 2010. Hybrid Mesh Editing. Eurographics Association, Norrköping, Sweden, H. P. A. Lensch and S. Seipel, Eds., 41–44.

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3, 630–634.

BOTSCH, M., AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. *Comput. Graph. Forum 24*, 3, 611–621.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 213–230.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. PriMo: Coupled prisms for intuitive surface modeling. In *Proc. SGP*, 11–20.

BOTSCH, M., PAULY, M., RÖSSL, C., BISCHOFF, S., AND KOBBELT, L. 2006. Geometric modeling based on triangle meshes. In *Eurographics 2006 Course Notes*.

BOTSCH, M., PAULY, M., KOBBELT, L., ALLIEZ, P., LÉVY, B., BISCHOFF, S., AND RÖSSL, C. 2007. Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH courses*.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. *Comput. Graph. Forum 26*, 3, 339–347.

BOTSCH, M., PAULY, M., KOBBELT, L., ALLIEZ, P., AND LEVY, B. 2008. Geometric modeling based on polygonal meshes. In *Eurographics 2008 Courses*.

BOTSCH, M., KOBBELT, L., PAULY, M., ALLIEZ, P., AND LÉVY, B. 2010. *Polygon Mesh Processing*. AK Peters.

BOTSCH, M. 2005. *High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes*. Shaker Verlag Aachen. PhD Thesis, RWTH Aachen.

BOTSCH, M. 2011. Smoothing & fairing. In *Graduate School at SGP*.

BOUAZIZ, S., DEUSS, M., SCHWARTZBURG, Y., WEISE, T., AND PAULY, M. 2012. Shape-up: Shaping discrete geometry with projections. *Comput. Graph. Forum 31*, 5, 1657–1667.

BOYKOV, Y., AND FUNKA-LEA, G. 2006. Graph cuts and efficient nd image segmentation. *IJCV 70*, 2.

BRAESS, D. 2002. *Finite Elements: Theory, fast solvers and applications in solid mechanics, 2nd edn*, vol. 13.

BRAMBLE, J., AND FALK, R. 1985. A mixed-Lagrange multiplier finite element method for the polyharmonic equation. *Mathematical Modelling and Numerical Analysis 19*, 4.

BRAND, M., AND CHEN, D. 2011. Parallel quadratic programming for image processing. In

Proc. ICIP.

BREMER, P.-T., EDELSBRUNNER, H., HAMANN, B., AND PASCUCCI, V. 2004. A topological hierarchy for functions on triangulated surfaces. *IEEE TVCG 10*, 4, 385 – 396.

BREZZI, F., AND FORTIN, M. 1991. Mixed and hybrid finite element methods, volume 15 of Springer Series in Computational Mathematics. *Springer-Verlag, New York 2*, 5, 2.

BRIDSON, R., TERAN, J., MOLINO, N., AND FEDKIW, R. 2005. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers 21*, 1, 2–18.

BRONSTEIN, A. M., BRONSTEIN, M. M., CASTELLANI, U., FALCIDIENO, B., FUSIELLO, A., GODIL, A., GUIBAS, L. J., KOKKINOS, I., LIAN, Z., OVSJANIKOV, M., PATANÉ, G., SPAGNUOLO, M., AND TOLDO, R. 2010. SHREC 2010: robust large-scale shape retrieval benchmark. In *3DOR*, 71–78.

BYRD, R. H., LU, P., NOCEDAL, J., AND ZHU, C. 1995. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput. 16*, 5, 1190–1208.

BYRD, R. H., NOCEDAL, J., AND WALTZ, R. A. 2006. Knitro: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization*, Springer Verlag, 35–59.

CAMPEN, M., AND KOBBELT, L. 2010. Exact and Robust (Self-)Intersections for Polygonal Meshes. *Comput. Graph. Forum 29*, 2.

CAMPEN, M., ATTENE, M., AND KOBBELT, L. 2012. A Practical Guide to Polygon Mesh Repairing. *Eurographics Tutorial*.

CARR, J., BEATSON, R., CHERRIE, J., MITCHELL, T., FRIGHT, W., MCCALLUM, B., AND EVANS, T. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proc. ACM SIGGRAPH*, 67–76.

CARR, J. C., BEATSON, R. K., MCCALLUM, B. C., FRIGHT, W. R., MCLENNAN, T. J., AND MITCHELL, T. J. 2003. Smooth surface reconstruction from noisy range data. In *Proc. ACM GRAPHITE*, 119–127.

CARR, H., SNOEYINK, J., AND VAN DE PANNE, M. 2004. Simplifying flexible isosurfaces using local geometric measures. In *Proc. Visualization*, 497–504.

CELNIKER, G., AND GOSSARD, D. 1991. Deformable curve and surface finite-elements for free-form shape design. In *Proc. SIGGRAPH*, 257–266.

CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (July), 38:1–38:6.

CHAR, B., GEDDES, K., AND GONNET, G. 1983. The Maple symbolic computation system. *SIGSAM 17*, 3–4, 31–42.

CHEN, C., FREEDMAN, D., AND LAMPERT, C. 2011. Enforcing topological constraints in random field image segmentation. In *Proc. CVPR*.

CIARLET, P. G., AND RAVIART, P.-A. 1974. A mixed finite element method for the biharmonic equation. In *Mathematical aspects of finite elements in partial differential equations*. 125–

145. Publication No. 33.

CIARLET, P. 1978. *The finite element method for elliptic problems*. North-Holland.

CLARENZ, U., DIEWALD, U., DZIUK, G., RUMPF, M., AND RUSU, R. 2004. A finite element method for surface restoration with smooth boundary conditions. *Comput. Aided Geom. Design 21*, 5, 427–445.

CRANE, K., WEISCHEDEL, C., AND WARDETZKY, M. 2012. Geodesics in heat. *ACM Trans. Graph.*, Conditionally accepted.

CRANE, K. 2012. private communication.

DAVIS, J., MARSCHNER, S. R., GARR, M., AND LEVOY, M. 2002. Filling holes in complex surfaces using volumetric diffusion. In *3DPVT*, 428–438.

DAVIS, T. A. 2004. UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30*, 2, 196–199.

DECKELNICK, K., AND DZIUK, G. 2006. Error analysis of a finite element method for the Willmore flow of graphs. *Interfaces and free boundaries 8*, 1, 21.

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Trans. Graph. 25*, 3, 1174–1179.

DEROSE, A. D. 1985. *Geometric Continuity: A Parameterization Independent Measure of*. PhD thesis, University of California at Berkeley.

DEY, T. K., AND GOSWAMI, S. 2003. Tight cocone: a water-tight surface reconstructor. In *Proc. SM*.

DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall.

DUFF, I. S. 2004. Ma57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw. 30*, 2.

DZIUK, G. 1988. Finite elements for the Beltrami operator on arbitrary surfaces. *Partial differential equations and calculus of variations*, 142–155.

DZIUK, G. 1990. An algorithm for evolutionary surfaces. *Numerische Mathematik 58*, 1, 603–611.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002. Topological persistence and simplification. *DCG 28*, 4, 511 – 533.

FALK, R. 1978. Approximation of the biharmonic equation by a mixed finite element method. *SIAM Journal on Numerical Analysis 15*, 3, 556–567.

FAURE, F., GILLES, B., BOUSQUET, G., AND PAI, D. K. 2011. Sparse meshless models of complex deformable solids. *ACM Trans. Graph. 30* (August), 73:1–73:10.

FELIPPA, C. A. 2004. *Introduction to Finite Element Methods*. College of Engineering, University of Colorado.

FINCH, M., SNYDER, J., AND HOPPE, H. 2011. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph. 30*, 6, 166:1–166:10.

FLEISHMANN, P., KOSIK, R., HAINDL, B., AND SELBERHERR, S. 1999. Simple mesh examples to illustrate specific finite element mesh requirements. In *8th Int. Meshing Roundtable*.

FLOATER, M. S. 2003. Mean value coordinates. *Computer-Aided Geometric Design 20*, 1, 19–27.

FORMAN, R. 1998. Morse theory for cell-complexes. *Adv. in Math. 134*, 1, 90–145.

FORSTMANN, S., AND OHYA, J. 2006. Fast skeletal animation by skinned arc-spline based deformation. In *Proc. Eurographics, short papers volume*.

FORSTMANN, S., OHYA, J., KROHN-GRIMBERGHE, A., AND MCDOUGALL, R. 2007. Deformation styles for spline-based skeletal animation. In *Proc. SCA*, 141–150.

FRÖHLICH, S., AND BOTSCH, M. 2011. Example-driven deformations based on discrete shells. *Comput. Graph. Forum 30*, 8, 2246–2257.

GEORGE, P. L., HECHT, F., AND SALTEL, E. 1990. Automatic 3d mesh generation with prescribed meshed boundaries. *IEEE Transactions on Magnetics 26*, 2, 771–774.

GEORGIEV, T. 2004. Photoshop healing brush: a tool for seamless cloning. In *Proc. ECCV ACV Workshop*, 1–8.

GEUZAINE, C., AND REMACLE, J. F. 2009. GMSH: A 3-D finite element mesh generator with built-in pre- and post-processing . *Numerical Methods in Engineering*.

GILLES, B., BOUSQUET, G., FAURE, F., AND PAI, D. 2011. Frame-based elastic models. *ACM Trans. Graph. 30*, 2.

GINGOLD, Y. I., AND ZORIN, D. 2006. Controlled-topology filtering. In *Proc. SPM*, 53–61.

GINGOLD, Y. 2008. private communication.

GRINSPUN, E., GINGOLD, Y., REISMAN, J., AND ZORIN, D. 2006. Computing discrete shape operators on general meshes. In *Computer Graphics Forum*, vol. 25, 547–556.

GUENNEBAUD, G., JACOB, B., ET AL., 2010. Eigen v3. http://eigen.tuxfamily.org.

GUÉZIEC, A., TAUBIN, G., LAZARUS, F., AND HOM, B. 2001. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE TVCG 7*, 2.

HARMON, D., PANOZZO, D., SORKINE, O., AND ZORIN, D. 2011. Interference aware geometric modeling. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA) 30*, 6, 137:1–137:10.

HELENBROOK, B. T. 2003. Mesh deformation using the biharmonic operator. *Int.. Journal for Numerical Methods in Engineering 56*, 7, 1007–1021.

HERON. 60. *Metrica*. Alexandria, Roman Egypt.

HILDEBRANDT, K., SCHULZ, C., TYCOWICZ, C. V., AND POLTHIER, K. 2011. Interactive surface modeling using modal analysis. *ACM Trans. Graph. 30*, 5, 119:1–119:11.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph. 26*, 2.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. *ACM Trans. Graph.*.

HORMANN, K., AND SUKUMAR, N. 2008. Maximum entropy coordinates for arbitrary polytopes. *Comput. Graph. Forum 27*, 5, 1513–1520.

HORNUNG, A., AND KOBBELT, L. 2006. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Proc. SGP*.

HOUSTON, B., BOND, C., AND WIEBE, M. 2003. A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications*.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3, 1126–1134.

HUANG, Q.-X., ADAMS, B., WICKE, M., AND GUIBAS, L. J. 2008. Non-rigid registration under isometric deformations. In *Proc. SGP*, 1449–1457.

HUANG, Q.-X., ZHANG, G.-X., GAO, L., HU, S.-M., BUTSCHER, A., AND GUIBAS, L. 2012. An optimization approach for extracting and encoding consistent maps in a shape collection. *ACM Trans. Graph. 31*, 6.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4.

JACOBSON, A., WEINKAUF, T., AND SORKINE, O. 2012. Smooth shape-aware functions with controlled extrema. In *Proc. SGP*.

JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, to appear.

JACOBSON, A., PANOZZO, D., AND DIAMANTI, O., 2013. libigl: A simple C++ geometry processing library. http://igl.ethz.ch/projects/libigl/.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

JOSHI, P., AND CARR, N. A. 2008. Repoussé: automatic inflation of 2d artwork. In *Proc. SBIM*, 49–55.

JOSHI, B., AND OURSELIN, S. 2003. BSP-Assisted Constrained Tetrahedralization. *IMR*, 251–260.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3, 71.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3.

JU, T. 2004. Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3.

JU, T. 2009. Fixing geometric errors on polygonal models: a survey. *Journal of Computer Science and Technology 24*, 1.

KARNI, Z., FREEDMAN, D., AND GOTSMAN, C. 2009. Energy-based image deformation. In *Proc. SGP*, 1257–1268.

KAVAN, L., AND SORKINE, O. 2012. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph. 31*, 6.

KAVAN, L., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4, 105:1–105:23.

KAVAN, L., COLLINS, S., AND O'SULLIVAN, C. 2009. Automatic linearization of nonlinear skinning. In *Proc. I3D*, 49–56.

KAVAN, L., SLOAN, P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. *Comput. Graph. Forum 29*, 2, 327–336.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP*.

KLINGNER, B. M., AND SHEWCHUK, J. R. 2007. Agressive tetrahedral mesh improvement. In *Proc. IMR*.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH*, 105–114.

KOLLURI, R., SHEWCHUK, J., AND O'BRIEN, J. 2004. Spectral surface reconstruction from noisy point clouds. *Proc. SGP*.

KOLMOGOROV, V., AND ZABIN, R. 2004. What energy functions can be minimized via graph cuts? *IEEE PAMI 26*, 2.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3.

LACARELLE, A., FAUSTMANN, T., GREENBLATT, D., PASCHEREIT, C., LEHMANN, O., LUCHTENBURG, D., AND NOACK, B. 2009. Spatiotemporal Characterization of a Conical Swirler Flow Field Under Strong Forcing. *Journal of Engineering for Gas Turbines and Power 131*, 031504.

LANDRENEAU, E., AND SCHAEFER, S. 2010. Poisson-based weight reduction of animated meshes. *Comput. Graph. Forum 29*, 6, 1945–1954.

LANGER, T., AND SEIDEL, H.-P. 2008. Higher order barycentric coordinates. *Comput. Graph. Forum 27*, 2, 459–466.

LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH*.

LEE, J. 1997. The Law of Cosines in a Tetrahedron. *J. Korea Soc. Math. Ed. Ser. Pure Appl. Math. 4*, B.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Trans. Graph. 23*, 689–694.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proc. SIGGRAPH*, 165–172.

LI, D., SUN, X., REN, Z., LIN, S., TONG, Y., GUO, B., AND ZHOU, K. 2012. Transcut: Interactive rendering of translucent cutouts. *IEEE TVCG 19*, 3.

LIAO, S.-H., FENG TONG, R., XIANG DONG, J., AND DONG ZHU, F. 2009. Gradient field based inhomogeneous volumetric mesh deformation for maxillofacial surgery simulation. *Computers & Graphics 33*, 3.

LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proc. SMI*, 181–190.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

LIPMAN, Y., KOPF, J., COHEN-OR, D., AND LEVIN, D. 2007. GPU-assisted positive mean value coordinates for mesh deformations. In *Proc. SGP*, 117–124.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph. 27*, 3.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM Trans. Graph. 25*, 3, 646–653.

LIU, L., ZHANG, L., XU, Y., GOTSMAN, C., AND GORTLER, S. J. 2008. A local/global approach to mesh parameterization. *Comput. Graph. Forum 27*, 5, 1495–1504.

MACNEAL, R., OF TECHNOLOGY. DIVISION OF ENGINEERING, C. I., AND SCIENCE, A. 1949. *The Solution of Partial Differential Equations by Means of Electrical Networks*. PhD thesis, California Institute of Technology.

MAGNENAT-THALMANN, N., AND THALMANN, D. 1987. The direction of synthetic actors in the film rendez-vous a montreal. *Computer Graphics and Applications, IEEE 7*, 12.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface*, 26–33.

MANSON, J., AND SCHAEFER, S. 2010. Moving least squares coordinates. In *Proc. SGP*, 1517–1524.

MANSON, J., AND SCHAEFER, S. 2011. Hierarchical deformation of locally rigid meshes. *Comput. Graph. Forum 30*, 8, 2387–2396.

MATLAB. 2012. *7.13.0.564 (R2011b)*. The MathWorks Inc., Natick, Massachusetts.

MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30* (Aug.), 37:1–37:12.

MEISTER, A. 1769/70. Generalia de genesi figurarum planarum et inde pendentibus earum

ajfectionibus. *Novi Comm. Soc. Reg. Scient. Gotting.*, 144–180+9 plates.

MERRY, B., MARAIS, P., AND GAIN, J. 2006. Animation space: A truly linear framework for character animation. *ACM Trans. Graph. 25*, 4, 1400–1423.

MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics 3*, 35–57.

MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*. Springer-Verlag, 35–57.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (July), 562–568.

MONK, P. 1987. A mixed finite element method for the biharmonic equation. *SIAM Journal on Numerical Analysis 24*, 4, 737–749.

MORETON, H. P., AND SÉQUIN, C. H. 1992. Functional optimization for fair surface design. In *Proc. SIGGRAPH*, 167–176.

MULLEN, P., DE GOES, F., DESBRUN, M., COHEN-STEINER, D., AND ALLIEZ, P. 2010. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Comput. Graph. Forum 29*, 5.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3.

MURALI, T. M., AND FUNKHOUSER, T. A. 1997. Consistent solid and boundary representations from arbitrary polygonal data. In *Proc. I3D*.

NAUMOV, M. 2011. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the gpu. *NVIDIA Technical Report*.

NAUMOV, M. 2012. Parallel incomplete-lu and Cholesky factorization in the preconditioned iterative methods on the gpu. *NVIDIA Technical Report*.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh: designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3, 41.

NI, X., GARLAND, M., AND HART, J. C. 2004. Fair morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph. 23*, 3, 613–622.

NOCEDAL, J., AND WRIGHT, S. 2006. *Numerical Optimization, Second Edition*.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG 9*, 2.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3, 92:1–92:8.

OVSJANIKOV, M., LI, W., GUIBAS, L., AND MITRA, N. J. 2011. Exploration of continuous variability in collections of 3d shapes. *ACM Trans. Graph. 30*, 4.

PEKELNY, Y., AND GOTSMAN, C. 2008. Articulated object reconstruction and markerless motion capture from depth video. *Comput. Graph. Forum 27*, 2, 399–408.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. 22*, 3, 313–318.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

PINKALL, U., AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experiment. Math. 2*, 1, 15–36.

PODOLAK, J., AND RUSINKIEWICZ, S. 2005. Atomic volumes for mesh completion. In *Proc. SGP*.

REUTER, M., BIASOTTI, S., GIORGI, D., PATANČ, G., AND SPAGNUOLO, M. 2009. Discrete Laplace-Beltrami operators for shape analysis and segmentation. *Computers & Graphics 33*, 3, 381–390.

ROSSIGNAC, J., AND CARDOZE, D. 1999. Matchmaker: manifold BReps for non-manifold r-sets. In *Proc. SMA*, 31–41.

RUSTAMOV, R. M. 2011. Multiscale biharmonic kernels. In *Proc. SGP*, 1521–1531.

SAYAS, F. 2008. *A gentle introduction to the Finite Element Method*.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. *ACM Trans. Graph. 25*, 3, 533–540.

SCHLÖMER, T., HECK, D., AND DEUSSEN, O. 2011. Farthest-point optimized point sets with maximized minimum distance. In *Proc. of the ACM SIGGRAPH Symp. on High Performance Graphics*, 135–142.

SCHNEIDER, R., AND KOBBELT, L. 2000. Generating fair meshes with G^1 boundary conditions. In *Geometric Modeling and Processing Conference Proceedings*.

SCHNEIDER, R., AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 4 (May), 359–379.

SCHÖBERL, J. 1997. NETGEN: An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*.

SCHOLZ, R. 1978. A mixed method for 4th order problems using linear finite elements. *RAIRO Anal. Numér. 12*, 1, 85–90, iii.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proc. SIGGRAPH*, 151–160.

SHARF, A., LEWINER, T., SHKLARSKI, G., TOLEDO, S., AND COHEN-OR, D. 2007. Interactive topology-aware surface reconstruction. *ACM Trans. Graph. 26*, 3.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph. 23*, 3, 896–904.

SHEPARD, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In

Proceedings of the 1968 23rd ACM national conference, ACM, 517–524.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. vol. 1148 of *Lecture Notes in Computer Science*.

SHEWCHUK, J. 2012. Unstructured Mesh Generation. *Combinatorial Scientific Computing 12*, 257.

SHI, X., ZHOU, K., TONG, Y., DESBRUN, M., BAO, H., AND GUO, B. 2007. Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graph. 26*, 3, 81:1–81:10.

SHIMADA, K., AND GOSSARD, D. C. 1995. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing.

SI, H., 2003. TETGEN: A 3D delaunay tetrahedral mesh generator. `http://tetgen.berlios.de`.

SIBSON, R. 1981. *Interpolating multivariate data*. John Wiley & Sons, ch. A brief description of natural neighbor interpolation, 21–36.

SIDI, O., VAN KAICK, O., KLEIMAN, Y., ZHANG, H., AND COHEN-OR, D. 2011. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Trans. Graph. 30*, 6.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. SGP*, 109–116.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proc. SGP*, 179–188.

SORKINE, O., COHEN-OR, D., IRONY, D., AND TOLEDO, S. 2005. Geometry-aware bases for shape approximation. *IEEE Transactions on Visualization and Computer Graphics 11*, 2, 171–180.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3, 80.

SÝKORA, D., BURIÁNEK, J., AND ŽÁRA, J. 2005. Sketching cartoons by example. In *Proc. SBIM*, 27–34.

SÝKORA, D., DINGLIANA, J., AND COLLINS, S. 2009. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Comput. Graph. Forum 28*, 2, 599–608.

TAKAYAMA, K., OKABE, M., IJIRI, T., AND IGARASHI, T. 2008. Lapped solid textures: Filling a model with anisotropic textures. *ACM Trans. Graph. 27*, 3.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. SIGGRAPH*, 351–358.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE TVCG*

11, 3, 317–328.

THOMAS, F., AND JOHNSTON, O. 1987. *Disney Animation: The Illusion of Life*. Abbeville Press.

TONG, Y., LOMBEYDA, S., HIRANI, A. N., AND DESBRUN, M. 2003. Discrete multiscale vector field decomposition. *ACM Trans. Graph. 22*, 3.

TOSUN, E. 2008. *Geometric modeling using high-order derivatives*. PhD thesis. AAI3330186.

TURK, G., AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *SIGGRAPH*.

VAN OOSTEROM, A., AND STRACKEE, J. 1983. The Solid Angle of a Plane Triangle. *Biomedical Engineering, IEEE Transactions on*, 2.

WAN, M., WANG, Y., AND WANG, D. 2011. Variational surface reconstruction based on Delaunay triangulation and graph cut. *Int. Journal of Numerical Engineering*.

WAN, M., WANG, Y., BAE, E., TAI, X., AND WANG, D. 2012. Reconstructing Open Surfaces via Graph-Cuts. *IEEE TVCG 19*, 2.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proc. SCA*, 129–138.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 73.

WANG, O., LANG, M., FREI, M., HORNUNG, A., SMOLIC, A., AND GROSS, M. 2011. Stereobrush: interactive 2d to 3d conversion using discontinuous warps. In *Proc. SBIM*, 47–54.

WANG, Y., ASAFI, S., VAN KAICK, O., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2012. Active co-analysis of a set of shapes. *ACM Trans. Graph. 31*, 6.

WARDETZKY, M., BERGOU, M., HARMON, D., ZORIN, D., AND GRINSPUN, E. 2007. Discrete quadratic curvature energies. *Computer Aided Geometric Design 24*, 8-9, 499–518.

WARDETZKY, M., MATHUR, S., KÄLBERER, F., AND GRINSPUN, E. 2007. Discrete Laplace operators: no free lunch. In *Proc. SGP*, 33–37.

WAREHAM, R., AND LASENBY, J. 2008. Bone Glow: An improved method for the assignment of weights for mesh deformation. *Articulated Motion and Deformable Objects*, 63–71.

WEBER, O., AND GOTSMAN, C. 2010. Controllable conformal maps for shape deformation and interpolation. *ACM Trans. Graph. 29*, 4, 78:1–78:11.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Comput. Graph. Forum 26*, 3, 265–273.

WEBER, O., BEN-CHEN, M., AND GOTSMAN, C. 2009. Complex barycentric coordinates with applications to planar shape deformation. *Comput. Graph. Forum 28*, 2, 587–597.

WEBER, O., BEN-CHEN, M., GOTSMAN, C., AND HORMANN, K. 2011. A complex view of barycentric mappings. *Comput. Graph. Forum 30*, 5.

WEBER, O., PORANNE, R., AND GOTSMAN, C. 2012. Biharmonic coordinates. *Comput. Graph. Forum 31*, 8.

WEBER, J. 2000. Run-time skin deformation. In *Proc. Game Developers Conference*.

WEINKAUF, T., GINGOLD, Y., AND SORKINE, O. 2010. Topology-based smoothing of 2D scalar fields with C^1-continuity. *Comput. Graph. Forum (proc. EuroVis) 29*, 3, 1221–1230.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH*, 247–256.

XU, J., AND ZIKATANOV, L. 1999. A monotone finite element scheme for convection-diffusion equations. *Math. Comput. 68*, 228.

XU, G., PAN, Q., AND BAJAJ, C. L. 2006. Discrete surface modelling using partial differential equations. *Comput. Aided Geom. Design 23*, 2, 125–145.

YAMAKAWA, S., AND SHIMADA, K. 2009. Removing self intersections of a triangular mesh by edge swapping, edge hammering, and face lifting. *Proc. IMR*.

YANG, X., SOMASEKHARAN, A., AND ZHANG, J. J. 2006. Curve skeleton skinning for human and creature characters. *Comput. Animat. Virtual Worlds 17*, 3-4, 281–292.

YÜCER, K., JACOBSON, A., HORNUNG, A., AND SORKINE, O. 2012. Transfusive image manipulation. *ACM Trans. Graph. 31*.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. In *Computer Graphics Forum (Proceedings of Eurographics)*, 601–609.

ZHANG, L., CUI, T., AND LIU, H. 2009. A set of symmetric quadrature rules on triangles and tetrahedra. *J. Comput. Math 27*, 1, 89–96.

ZHANG, S., NEALEN, A., AND METAXAS, D. 2010. Skeleton Based As-Rigid-As-Possible Volume Modeling. In *Proc. Eurographics, short papers volume*, 21–24.

ZHENG, Y., AND TAI, C.-L. 2010. Mesh decomposition with cross-boundary brushes. *Comput. Graph. Forum 29*, 2, 527–535.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph Laplacian. *ACM Trans. Graph, 24*, 3, 496–503.

ZHOU, K., ZHANG, E., BITTNER, J., AND WONKA, P. 2008. Visibility-driven mesh analysis and visualization through graph cuts. *IEEE TVCG 14*, 6.

ZHU, Y., AND GORTLER, S. J. 2007. 3D deformation using moving least squares. Tech. Rep. Tr-10-07, Harvard University, Cambridge, MA.

Appendix: 2D dataset

This dataset contains all 2D vector graphics cartoons used throughout this thesis. They are my own creations and free to use for academic, non-military research.

Appendix: *Curriculum Vitæ*

Alec Jacobson

Office Address:
CAB G 82.2
Universitaetstrasse 6
8092 Zurich
Switzerland
people.inf.ethz.ch/ jalec
jacobson@inf.ethz.ch

Home Address:
Schaffhauserstrasse 123
8057 Zürich
Switzerland
alecjacobson.com
alecjacobson@gmail.com

Education

Ph.D. Eidgenössische Technische Hochschule Zürich, Computer Science, 2013: 5.5/6.0 GPA

M.A. New York University, Computer Science, May 2011: 4.0/4.0 GPA

B.A. New York University, Mathematics & Computer Science Joint Major with high departmental honors, 2009, Magna Cum Laude: 3.8/4.0 GPA

Honors Diploma Graduate, Mayo High School, Rochester, MN, 2005: 5.2/4.0 weighted GPA

Graduate, University of Minnesota Talented Youth Math Program (UMTYMP Algebra I and II, Geometry, Math Analysis, Calculus I, II and III), Sept. 1999 - May 2004: 4.0/4.0 GPA

Awards, Scholarships, Fellowships Received

Intel PhD Fellowship, 2012

Back Cover Image, Proceedings of ACM SIGGRAPH, 2011

New York University Henry M. MacCracken Fellowship, 2009-2011

Grand Prize, Games For Learning Institute Game Design Challenge 2009, Microsoft Research

New York University Founder's Day Award, 2009

New York University Trustee's Scholarship, 2005-2009

Mayo Foundation Scholarship, 2005-2009

New York University Dean's list, 2006, 2008, 2009

Mayo Clinic Summer Mentored Undergraduate Research Fellowship, 2007

Rochester Film Festival, Best Animated Short Film, 2005

Journal Publications

Alec Jacobson, Ladislav Kavan, Olga Sorkine. "Robust Inside-Outside Segmentation using Generalized Winding Numbers." ACM Transactions on Graphics, Vol. 32(4), 2013 (ACM SIGGRAPH Proceedings).

Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brain Whited, Maryann Simmons and Olga Sorkine-Hornung. "Ink-And-Ray: Global Illumination for Hand-Drawn Animation." ACM Transactions on Graphics, *to appear*, 2013.

Kaan Yücer, Alec Jacobson, Alexander Hornung, Olga Sorkine. "Transfusive Image Manipulation." ACM Transactions on Graphics, Vol. 31(6), 2012 (ACM SIGGRAPH Asia Proceedings).

Alec Jacobson, Tino Weinkauf, Olga Sorkine. "Smooth Shape-Aware Functions with Controlled Extrema." Computer Graphics Forum, Vol. 31(4), 2012 (EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing Proceedings issue).

Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, Olga Sorkine. "Fast Automatic Skinning Transformations." ACM Transactions on Graphics, Vol. 31(4), 2012 (ACM SIGGRAPH Proceedings).

Alec Jacobson, Olga Sorkine. "Stretchable and Twistable Bones for Skeletal Shape Deformation." ACM Transactions on Graphics, Vol. 30(6), 2011 (ACM SIGGRAPH Asia Proceedings).

Alec Jacobson, Ilya Baran, Jovan Popović, Olga Sorkine. "Bounded Biharmonic Weights for Real-Time Deformation." ACM Transactions on Graphics, Vol. 30(4), 2011 (ACM SIGGRAPH Proceedings).

Alec Jacobson, Elif Tosun, Olga Sorkine, Denis Zorin. "Mixed Finite Elements for Variational Surface Modeling." Computer Graphics Forum, Vol. 29(5), pp. 1565–1574, 2010 (EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing Proceedings).

Technical Reports

Alec Jacobson. "Bijective Mappings with Generalized Barycentric Coordinates: A Counterexample" ETH Zurich, 2012.

Alec Jacobson, Olga Sorkine. "A Cotangent Laplacian Images as Surfaces." ETH Zurich, 2012.

Research Experience

PhD Research, Advisor: Olga Sorkine, Interactive Geometry Lab, ETH Zurich, Zurich, Switzerland, 2011-*present*

PhD Research, Advisor: Olga Sorkine, Media Research Lab, New York University, New York, NY, 2009-2011

Summer Research Intern, Advisor: Jovan Popović, Advanced Technology Labs, Adobe Systems, Seattle, WA, Summer 2010

Undergraduate Research, Advisor: Denis Zorin, Media Research Lab, New York University, New York, NY, 2008-2009

Summer Mentored Undergraduate Research Fellowship, Advisor: Željko Bajzer, Mayo Clinic, Rochester, MN, 2007

Work Experience

America Counts math intervention tutor and student teacher, Clinton Public Middle School for Artists and Writers, New York, NY, 2008-2009

Calculus and Basic Algorithms Homework Grader, Math Department, New York University, New York, NY, 2006-2009

Intern at the IBM Executive Briefing Center, Rochester, MN, Summer 2008

Computer Skills

MATLAB	C, C++	Java, C#	Mathematica	Maple	LaTeX, TeX
Qt	Ruby	Python	PHP	HTML	Javascript
OpenGL	UNIX tools	CGAL	CUDA	OpenMP	XNA
Development:	Vim	Xcode	Visual Studio	Eclipse	NetBeans

Teaching Experience

Computer Graphics, Teaching Assistant, ETH Zurich, Zurich, Switzerland, 2011-2012

America Counts math intervention tutor and student teacher, Clinton Public Middle School for Artists and Writers, New York, NY, 2008-2009

Calculus Grader, Math Department, New York University, New York, NY, 2006-2008

Volunteer computer programming teacher at Courant Splash! (cSplash), NYU, one-day festival of classes in the mathematical and computer sciences for high school students, Spring 2008

Volunteer tutor at Rochester English for Speakers of Other Languages Summer School, 2007

Volunteer tutor of three elementary school boys from an African refugee family in the South Bronx, 2005-2006

Conference Talks

ACM SIGGRAPH, *Fast Automatic Skinning Transformations*, August 8, 2012

EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing, *Smooth Shape-Aware Functions with Controlled Extrema*, July 16, 2012

ACM SIGGRAPH Asia, *Stretchable, Twistable Bones for Skeletal Shape Deformation*, December 14, 2011

ACM SIGGRAPH, *Bounded Biharmonic Weights for Real-Time Deformation*, August 10, 2011

EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing, *Mixed Finite Elements for Variational Surface Modeling*, July 6, 2010

Invited Talks

Max-Planck-Institut für Informatik, Saarbrücken, *Achieving High-Quality Shape Deformation in Real Time*, invited by Tino Weinkauf, February 26, 2013

Workshop on Computer Graphics and Emerging Technology, Shenzhen Institutes of Advanced Technology, *Achieving High-Quality Shape Deformation in Real Time*, invited by Baoquan Chen, November 26, 2012

New York University, *Fast Automatic Skinning Transformations*, invited by Denis Zorin, July 31, 2012

NSF Workshop on Barycentric Coordinates in Geometry Processing and Finite/Boundary Element Methods, *High-quality weight functions via constrained optimization*, invited by Kai Hormann, July 25, 2012

Freie Universität, Berlin, *High Quality Weight Functions via Constrained Optimization*, invited by Konrad Polthier, June 22, 2012

LiberoVision, Zurich, *Real-time Shape Deformation: Bounded Biharmonic Weights and Stretchable, Twistable Bones*, invited by Remo Ziegler, February 2, 2012

DISI University of Genoa, *Real-time Deformation: Bounded Biharmonic Weights and Stretchable, Twistable Bones*, invited by Enrico Puppo, June 27, 2011

ETH Zurich-Disney Research Zurich Tech Talk, *Mixed Finite Elements for Variational Surface Modeling*, invited by Alexander Hornung, October 20, 2010

Languages

English (native)
Spanish (proficient)
German (beginner)

Academic Service

Program Committee Member:
Symposium on Geometry Processing, 2013
EUROGRAPHICS Short Papers, 2012-2013

Reviewer:
ACM SIGGRAPH
ACM SIGGRAPH Asia
Computer Graphics Forum
CVPR Conference on Computer Vision and Pattern Recognition
ECCV European Conference on Computer Vision
EUGRAPHICS
EUGRAPHICS Short Papers
Graphics Interface
IEEE Computer Graphics and Applications

IEEE Transactions on Visualization and Computer Graphics
Pacific Graphics
SIBGRAPI Conference on Graphics, Patterns and Images

Solo Art Exhibitions

Jewcy Main Office, Brooklyn, NY, 2008

"Humans I See Every Day," Bronfman Center Gallery, New York, NY, 2008

Jewcy's Online Featured Artist, Brooklyn, NY, 2008

Group Art Exhibitions

"Einstein on Witherspoon Street: Expressions for Social Justice," Bronfman Center Gallery, New York, NY, 2009

"Becoming: Visions of Childhood," Bronfman Center Gallery, New York, NY, 2008

"Muslim-Jewish Art Collaborative Project," Bronfman Center Gallery, New York, NY, 2008

"Clear is not a Color," L. Iago Gallery, New York, NY, 2006

References

Olga Sorkine (PhD advisor, 2009-present)
Assistant Professor
Department of Computer Science
ETH Zurich
CNB G 106
Universtitaetstrasse 6
8092 Zurich, Switzerland
+ 41 44 632 83 57
sorkine@inf.ethz.ch

Jovan Popović (Internship advisor, summer 2010)
Principal Scientist
Advanced Technology Labs
Adobe Systems
801 N. 34th Street
Seattle, WA 98103
206-675-7000
jovan@adobe.com

Denis Zorin (Undergraduate research advisor, 2008-2009)
Professor of Computer Science and Mathematics
Computer Science Department
Courant Institute of Mathematical Sciences
New York University
719 Broadway, New York, NY 10003
212-998-3405
dzorin@courant.nyu.edu

Željko Bajzer (Mentor for research fellowship, 2007)
Professor of Biophysics
Department of Biochemistry and Molecular Biology
Mayo Clinic
200 First St SW
Rochester, MN 55905
507-284-8584
bajzer@mayo.edu