TU Berlin, Fakultät IV, Computer Graphics

Interfaces and Algorithms for the Creation, Modification, and Optimization of Surface Meshes

vorgelegt von
Diplom-Ingenieur und Diplom-Informatiker
Andrew Vincent Nealen
aus Washington D.C., USA

Von der Fakultät IV - Elektotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
— Dr.-Ing. —

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr.-Ing. Olaf Hellwich
Berichter: Prof. Dr.-Ing. Marc Alexa
Berichter: Prof. Takeo Igarashi, Ph.D.
Berichter: Prof. John Hughes, Ph.D.

Tag der wissenschaftlichen Aussprache: 12. Oktober 2007

Berlin 2007
D83

“ I like to think I’m the master of my own destiny.”

Randal Graves, *Clerks*, 1994.

For my parents.

Abstract

In this dissertation we present interfaces and algorithms for the creation, modification and optimization of surface meshes. After a short introduction, motivation, and list of contributions, we describe the current state of the art in mesh creation and modeling, and also give an overview of the mathematical tools that are used throughout this dissertation.

For the simple creation of surface meshes, we present an interface for designing freeform surfaces with a collection of 3D curves. The user first creates a rough 3D model by using a sketching interface. Unlike previous sketching systems, the user-drawn strokes stay on the model surface and serve as handles for controlling the geometry. The user can add, remove, and deform these control curves easily, as if working with a 2D line drawing. The curves can have arbitrary topology; they need not be connected to each other. For a given set of curves, the system automatically constructs a smooth surface embedding by applying functional optimization. Our system provides real-time algorithms for both control curve deformation and the subsequent surface optimization. We show that one can create sophisticated models using this system that have not yet been seen in previous sketching or functional optimization systems.

Thereafter, we present methods for the intuitive editing of surface meshes by means of view-dependent sketching. In most existing shape deformation work, editing is carried out by selecting and moving a handle, usually a set of vertices. Our system lets the user easily determine the handle, either by silhouette selection and cropping, or by sketching directly onto the surface. An edit is carried out by sketching a new, view-dependent handle position or by indirectly influencing differential properties along the sketch. Combined, these editing and handle metaphors greatly ease otherwise complex shape modeling tasks.

To further simplify the editing process, we introduce an over-sketching interface for surface mesh editing that automates the processes of determining both the deformation handle, as well as the region to be deformed. The user sketches a stroke that is the suggested position of part of a silhouette of the displayed surface. The system then segments all image-space silhouettes of the projected surface, identifies among all silhouette segments the best matching part, derives vertices in the surface mesh corresponding to the silhouette part, selects a sub-region of the mesh to be modified, and feeds appropriately modified vertex positions together with the sub-mesh into a mesh deformation tool. The overall algorithm has been designed to enable interactive modification of the surface – yielding a surface

editing system that comes close to the experience of over-sketching 2D drawings on paper.

Surface meshes created and edited with our tools – as well as scanned, contoured or simplified models – may contain triangles with bad aspect ratios and/or significant noise. To improve this, we introduce a framework for triangle shape optimization and feature preserving smoothing of triangular meshes that is guided by vertex Laplacians, specifically, the uniformly weighted Laplacian and the discrete mean curvature normal. Vertices are relocated so that they approximate prescribed Laplacians and positions in a weighted least-squares sense; the resulting linear system leads to an efficient, non-iterative solution. We provide different weighting schemes and demonstrate the effectiveness of the framework on a number of detailed and highly irregular meshes; our technique successfully improves the quality of the triangulation while remaining faithful to the original surface geometry, and it is also capable of smoothing the surface while preserving geometric features.

In closing, we discuss our proposed solutions, present open questions related to this dissertation and shape modeling in general, outline possible improvements, and propose areas for further research.

Zusammenfassung

Diese Dissertation beschreibt Benutzerschnittstellen und Algorithmen für die Erzeugung, Modifizierung und Optimierung diskreter Flächen. Nach einer kurzen Einführung, Motivation und Zusammenstellung der neuen Beiträge wird der aktuelle Stand der Technik bezüglich der Erstellung und Modellierung diskreter Flächen beschrieben, sowie ein Überblick der verwendeten mathematischen Werkzeuge gegeben.

Für das einfache Erzeugen diskreter Flächen wird eine Benutzerschnittstelle für das Design von Freiformflächen unter Verwendung dreidimensionaler Kurven präsentiert. Der Benutzer erzeugt ein grobes dreidimensionales Modell unter Zuhilfenahme einer auf Freihand Skizzen basierenden Benutzerschnittstelle. Anders als in bisherigen Systemen bleiben die Skizzen und Striche des Benutzers auf dem Oberflächenmodell bestehen, und dienen als kurvenförmige Griffe (Kontrollkurven) mit denen die Geometrie verändert werden kann. Der Benutzer kann diese Kontrollkurven sehr einfach hinzufügen, entfernen und deformieren, als würde man mit einer zweidimensionalen Linienzeichnung arbeiten. Den Kurven kann eine beliebige Topologie zugrunde liegen; Sie müssen nicht verbunden sein. Für eine gegebene Kurvenmenge wird durch die Optimierung eines Flächenfunktionals automatisch eine interpolierende, glatte Fläche konstruiert. Das System ist mit Echtzeit Algorithmen ausgestattet, sowohl für die Deformation von Kontrollkurven als auch für die darauffolgende Flächenoptimierung. Es werden anspruchsvolle Modelle die mit diesem System erstellt wurden präsentiert, welche mit bisherigen skizzenbasierten Werkzeugen nicht möglich waren.

Daraufhin werden Methoden für das intuitive Editieren diskreter Flächen anhand von blickpunktsabhängigen Skizzen vorgestellt. In den meisten existierenden Arbeiten zur Modelldeformation wird eine Editieroperation durch Auswahl und Verschiebung eines Griffs durchgeführt. Ein Griff wird dabei i.d.R. als Knotenmenge des Graphen repräsentiert. Im neuen System kann der Benutzer diesen Griff entweder durch Auswahl und Zuschnitt einer Silhouette, oder durch direktes Skizzieren auf die Fläche auswählen. Editieroperationen werden danach entweder durch das Skizzieren einer neuen, blickpunktsabhängigen Griffposition, oder durch das Variieren differentieller Eigenschaften entlang der Skizze ausgeführt. In Kombination vereinfachen diese Editier- und Griffmetaphern ansonsten komplexe Flächenmodifikationen.

Um den Prozess des Editierens weiter zu vereinfachen, wird eine skizzenbasierte Benutzerschnittstelle vorgestellt, die das Auswählen von Griff und Deformationsbereich auf der Modelloberfläche automatisiert. Der Benutzer skizziert die neue Position eines Teils einer Silhouette in der aktuellen Ansicht. Daraufhin segmentiert das System alle Bildraumsilhouetten, identifiziert unter allen Silhouettensegmenten das Segment, welches der Benutzerskizze am ähnlichsten ist, leitet daraus die korrespondierenden Knoten der diskreten Fläche ab, wählt eine Region der Fläche zur Deformation aus, und stellt diese Informationen für ein Flächendeformationswerkzeug zur Verfügung. Insgesamt wurde dieser Algorithmus entwickelt, um eine interaktive Modifizierung der Fläche zu ermöglichen – dabei wurde ein Flächendeformationswerkzeug entwickelt, welches dem zweidimensionalen Skizzieren auf Papier sehr nahe kommt.

Sowohl diskrete Flächen die mit den vorgestellten Werkzeugen und Algorithmen erstellt wurden, als auch eingescannte, abgetastete und vereinfachte Modelle können Dreiecke mit schlechtem Seitenverhältnis oder starkes Rauschen beinhalten. Zur Verbesserung wird ein Verfahren zur Dreiecksoptimierung und merkmalserhaltender Glättung vorgestellt, welches von uniform- und Kotangens-diskretisierten Laplace Vektoren gesteuert wird. Knotenpunkte werden verschoben, so dass sie vorgeschriebene Laplace Vektoren und Positionen im Sinne der gewichteten kleinsten Fehlerquadrate approximieren. Das daraus resultierende lineare Gleichungssystem lässt sich effizient lösen. Es werden verschiedene Verfahren zur Gewichtung bereitgestellt. Um die Effizienz des Verfahrens zu demonstrieren werden eine Vielzahl von detaillierten und irregulären Netzen hinsichtlich Dreiecksform und Rauschen optimiert.

Abschließend werden die vorgestellten Lösungen diskutiert, offene Fragen hinsichtlich dieser Dissertation und Flächenmodellierung im Allgemeinen präsentiert, mögliche Verbesserungen skizziert, und Vorschläge für weiterführende Forschung gemacht.

Acknowledgments

Some would argue that writing a dissertation can be an insurmountable challenge. While I do believe this is a possibility, and I have seen it happen, I was fortunate enough to meet and befriend many colleagues along the way, who never failed to give me the right advice at the right time. Add to this the ongoing and unconditional support of my family and friends, the many fun and relaxing moments throughout the past four years, and what seemed complex and long-winded at first, turned out to come naturally in the end. They say *"time flies when you're having fun"*... I couldn't agree more.

First and foremost, my uttermost gratitude goes to my friend, collaborator and mentor Marc Alexa. He realized my potential when I – in my own opinion – was rather clueless, shared his knowledge and experience, and always motivated me to work and play hard. It also didn't hurt too much that we explored what I would call *general nightlife* extensively. Thanks to Marc I was able to spend a significant amount of my time as a Ph.D. student traveling and visiting other computer graphics labs throughout the world, and it was during these travels that I imagined, collaborated on and developed many of the projects that led up to this dissertation.

I am greatly indebted to the brilliant Takeo Igarashi. After simply asking him in April 2005, Takeo invited me to an extensive research visit to his lab at The University of Tokyo. This turned out to be an exhilarating and inspiring experience, thanks to the insightful, inventive and restless nature of Takeo, many helpful and friendly students – Makoto Okabe, Shigeru Owada, Yuki Mori, Takashi Ijiri, Takeshi Nishida, HyoJong Shin and Yoshinori Kawasaki – and the fascination I hold for Tokyo, and Japan in general.

I would also like to thank Daniel Cohen-Or. Danny invited me to Tel Aviv University in the fall of 2004, where we worked on sketch-based interfaces and mesh editing, together with the lovely and frighteningly talented Olga Sorkine, to who I owe so much more than just this research. Thanks also to my friends, Andrei Sharf, Alon Lerner, Ran Gal, Yaron Lipman and Yaar Schnitman at TAU.

And even though the following collaboration is not directly related to this dissertation, I am thankful to Markus Gross and the Computer Graphics Lab of the ETH Zürich for working on my first computer graphics project as a Ph.D. student. In this context, I would like to thank Matthias Müller, Richard Keiser (who also provided the latex template), Bart Adams, Mark Pauly and Mark Carlson. Markus

was also a great help when it came to deciding where to go, and what to do next. It was Markus, Danny and Marc who agreed with, and supported my decision to team up with Takeo. Even though it is already in the past, I still have a hard time believing that this all worked out so well. Also, a big thanks to John Hughes for agreeing to review this thesis and thereby helping me improve it. And had it not been for Johannes Zimmermann, a part of this thesis might not have seen the light of day.

A few people – some of which friends from way back when, some fellow researchers, and some both – turned out to be pillars on which I depend to this day. Rupert, Stefan and Eddy, thanks for being around all the while. It's great to have close friends like you.

Most of my spare time in these past years was spent hanging out, biking, and simply enjoying *the good life* with Martin, Jenny, Steffen, Marc, Madeleine, Anders, Manja, Lennart, and my roommates Matthias and Alex. Good times. I have pictures. And some will be forever censored.

To all my other friends – you know who you are – I send out a collective *thank you* for all the unforgettable moments. It was you all who kept my everyday life in balance, for which I am forever grateful.

Finally, I would like to thank my family. To my parents, who always supported me, my brothers, who always believed in me, and my beloved girlfriend Olga, with whom I wish to spend the rest of my life: thank you all for making this endeavor so very much worth while, supporting me, and motivating me to always move forward.

A few years back I wrote *"it doesn't get much better than this"*. I was wrong. It doesn't happen very often that I enjoy being wrong. This time I'm enthusiastic about it.

Contents

Chapter 1

Introduction

The creation and modification of 3D shapes and surfaces using digital computers is a challenging problem. And while many interesting algorithms and systems have been developed in the past – many of which are used in production environments – most people agree that there is generally no single *correct* solution. To make this difficult problem even more complex, most interfaces and the associated algorithms for the creation and modification of 3D models depend heavily on object representations, modeling metaphors, areas of application, and user skills – to name only a few dependencies. The growing demand for digital assets in film and video games, for example 3D characters and props for pre-visualization, animation and rendering, motivates the evolution of the associated tools for their creation and modification.

In this dissertation, we argue that existing modeling metaphors, while intuitive and simple, can be limiting, especially for artistic and less tech-savvy users. We will show that is indeed possible to leverage such a users creativity, without compromising the underlying mathematical rigor, simply by hiding these subtleties behind easy-to-use interfaces and modeling metaphors.

Some intuitive interfaces for model creation exist, see for example Igarashi et al.'s seminal work Teddy [IMT99], which has been used in PC software packages, various Sony Playstation and Nintendo Gamecube games, and served as an inspiration for the character editor in Maxis upcoming SPORE [Wri06, Max07]. Interestingly, Teddy and other related interfaces and tools for model creation from scratch oftentimes do not support subsequent editing and manipulation of the created surface model.

Many existing professional tools for surface modeling and editing such as Maya [May07] and 3Ds Max [dM07], as well as some seminal research on multiresolution mesh editing [ZSS97] makes use of parametric patches or subdivision surfaces, while another group of research tools casts the problem of surface editing as the solution to a partial differential equation (PDE) on the continuously [MS92] or discretely represented (e.g. triangulated) surface [WW94, KCVS98, BK04a, Sor06]. The interface generally used is the so-called *handle* metaphor: the user

selects (a) the surface region to be edited, also known as the region of interest or ROI, and (b) a subset of this region, which serves as the deformation handle. The user can thereafter apply an affine transformation to the handle vertices, and the resulting surface is defined to minimize certain functions of its differentials. While this methodology is more intuitive than simply moving single vertices at a time, and also mathematically elegant, it may lack the freedom many artists have been become accustomed to from traditional pen and paper design sessions.

Our goal in this dissertation is manifold. We are intrigued by the simplicity of sketch-based interfaces, are generally in favor of discrete, piecewise smooth surfaces resulting from functional optimization, and also want to control the surface deformation via some kind of handle. As a result, the interfaces and algorithms presented in this dissertation all share the *flexible handle* metaphor. Whether the handle is a model silhouette, a ridge or ravine, or even a user placed sketch, we treat flexible curves as continuous entities. And this is not limited to model editing: in our solutions, the typical sketches used in generative systems such as Teddy remain on the surface after model creation, and can be augmented by more user-placed curve handles. Furthermore, whether the user prefers freeform sketching to hint at a desired silhouette deformation, or pulling a vertex of a curve handle embedded in the surface, our handles are always *flexible* – they are not limited to affine transformations.

Interaction is of uttermost importance in a surface creation and modeling tool. Recent developments on linear variational surface deformation [BS07] and sparse linear solvers [Tol03] have made possible the interactive editing of surfaces with tens of thousands of vertices in real-time. We make use of these algorithms, and augment them where necessary in our setting. In some cases though, linear variational surfaces are not sufficient, given the small set of curve constraints we wish to support in our interfaces. Specifically, the resulting surface may concentrate curvature near the curve constraints, due to the absence of normal constraints. Note that this is a deliberate design decision: we do not want the user to supply normals along the curves for reasons of simplicity. Instead, we perform nonlinear surface optimization, which has been popularized in a few recent contributions [BPG06, HSL$^+$06]. Since nonlinear optimization can potentially slow down performance below interactive rates, we introduce some approximations to the exact solution, which speed up our algorithm while still generating surfaces that are visibly piecewise smooth.

Finally, should the surfaces still have badly shaped triangles, or excessive noise from undesired detail editing, we can leverage our mathematical framework to globally optimize the mesh, either by reshaping the triangles, smoothing out small detail, or both. Our mesh optimization algorithm has also been successfully tested on *real world* input from range scanners, 3D photography, and meshes with badly shaped triangles resulting from mesh simplification [GH97].

1.1 Contributions

In this dissertation we present methods for the creation, modification and optimization of surface meshes, based on a unified modeling metaphor – the flexible handle – and underlying mathematical framework – extended Laplacian surface editing and general least-squares methods. Our main contributions are:

- **An interface that enables the design of 3D models with 3D control curves.** We present an interface for designing freeform surfaces with a collection of 3D curves. The user first creates a rough 3D model by using a sketching interface. Unlike previous sketching systems, the user-drawn strokes stay on the model surface and serve as handles for controlling the geometry. The curves can have arbitrary connectivity.

- **A fair surface definition based on curve constraints.** For a given set of curves, our system automatically constructs a smooth surface embedding that interpolates the curves by applying functional optimization. To ensure an interactive, responsive environment, we have developed a real-time nonlinear optimization algorithm to compute piecewise smooth surfaces, represented as triangle meshes.

- **A detail preserving, real-time 3D curve editing and peeling interface.** Our curve deformation algorithm is based on discrete co-rotational methods. The user can add, remove, and deform control curves easily, as if working with a 2D line drawing. A deformation is carried out simply by dragging single curve vertices.

- **A sketch-based modeling interface that uses silhouettes and user-sketches as input.** We present methods for the intuitive editing of surface meshes by means of view-dependent sketching. In most existing shape deformation work, editing is carried out by selecting and moving a *handle*, usually a set of vertices. Our system lets the user easily determine the handle, either by silhouette selection and cropping, or by sketching directly onto the surface. An edit is carried out by sketching a new, view-dependent handle position or indirectly by influencing differential properties along the sketch.

- **An automation mechanism for sketch-based deformation.** We introduce an over-sketching interface for surface mesh editing that automates the processes of determining both the deformation handle, as well as the region to be deformed. The user sketches a stroke that is the suggested position for part of a silhouette of the displayed surface. The system then segments all image-space silhouettes of the projected surface, identifies among all silhouette segments the best matching part, derives vertices in the surface mesh corresponding to the silhouette part, selects a sub-region of the mesh

to be modified, and feeds appropriately modified vertex positions together with the sub-mesh into a mesh deformation tool.

- **A global mesh optimization framework based on discrete Laplacians and least-squares methods.** We develop a framework for triangle shape optimization and feature preserving smoothing of triangular meshes which is guided by vertex Laplacians, specifically, the uniformly weighted Laplacian and the discrete mean curvature normal. Vertices are relocated so that they approximate prescribed Laplacians and positions in a weighted least-squares sense; the resulting linear system leads to an efficient, non-iterative solution.

1.2 Outline

This dissertation is organized as follows:

- **Chapter 2** introduces the fundamental concepts and notations. Section 2.1 presents the current state of the art in interactive shape modeling, where closely related to our own work. We outline the basics of least-squares methods, including intuitive, simple derivations, in Section 2.2. An introduction to discrete differential geometry (DDG) is given in Section 2.3, and the chapter concludes with applications of least-squares methods and DDG to shape modeling, specifically extended Laplacian surface editing, in Section 2.4.

- **Chapter 3** presents our mesh creation and modification tool *FiberMesh*. Section 3.2 describes the small set of operations we have incorporated into our user interface and their intended use, while Section 3.3 deals with the algorithmic details of detail-preserving curve deformation and subsequent real-time, nonlinear surface optimization. The chapter concludes with a discussion, an informal user study, and results in Sections 3.4 and 3.5.

- **Chapter 4** presents our sketch-based interface for detail-preserving feature editing. Section 4.3 describes the oversketching and editing of object-space silhouettes by placing positional constraints. The system is extended to the indirect modeling of ridges, ravines and suggestive contours [DFRS03] by embedding the user-sketch in the mesh, and subsequently modifying the discrete differential properties of the surface along this sketch.

- **Chapter 5** describes some extensions to Chapter 4. Specifically, we greatly reduce the required user operations (Section 5.3) by fully automating both handle and region-of-interest (ROI) selection. Section 5.4 give a detailed description on image-space silhouette extraction and segmentation, handle estimation by partial matching, and selection of the mesh sub-region to be

modified. The resulting vertex sets and modified handle positions are used in a mesh deformation tool.

- **Chapter 6** presents a global mesh optimization framework for triangle shape optimization and mesh smoothing. The method builds on our local mesh optimization, presented in Section 4.4.1, but extends it to entire meshes. After introducing the general idea in Section 6.4, the various parameters and weights for global shape-preserving triangle optimization are described in Section 6.5. Section 6.6 shows how the same underlying framework can be leveraged to perform feature preserving mesh smoothing.
- **Chapter 7** concludes this dissertation with a summary of the presented work, a discussion of advantages and drawbacks of the proposed methods, and an outlook on future work.

1.3 Publications and collaborations

The work presented in this dissertation is the result of international collaborations and projects that have been published as follows:

- The overview of least-squares methods in Section 2.2 was previously published in parts as a technical report under the title *An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation* [Nea04].
- Section 2.4 on Laplacian surface editing and its extensions is taken from presentations on the topic, inspired by the original publication [SLCO+04]. The presentations were held in various formats at ACM SIGGRAPH 2005 (Los Angeles, USA), the 2005 Summer School on Interactive Shape Modeling (Darmstadt, Germany), the MPI Graphics Seminar September 2006 (Saarbrücken, Germany), and ACM GRAPHITE 2006 (Kuala Lumpur, Malaysia).
- Our surface creation and modification tool presented in Chapter 3 was published and presented as *FiberMesh: Designing Freeform Surfaces with 3D Curves* [NISA07] at ACM SIGGRAPH 2007 in San Diego, USA, in collaboration with Takeo Igarashi from The University of Tokyo, Japan, as well as Olga Sorkine and Marc Alexa from TU Berlin, Germany.
- The feature editing technique presented in Chapter 4 was published and presented as *A Sketch-Based Interface for Detail-Preserving Mesh Editing* [NSACO05] at ACM SIGGRAPH 2005 in San Diego, USA, in collaboration with Olga Sorkine and Daniel Cohen-Or from Tel Aviv University, Israel, and Marc Alexa from TU Darmstadt, Germany.

- Automated sketch-based editing as described in Chapter 5 was published and presented as *SilSketch: Automated Sketch-Based Editing of Surface Meshes* [ZNA07] at the Eurographics Workshop on Sketch-Based Interfaces and Modeling in Riverside, USA, in collaboration with Johannes Zimmermann and Marc Alexa from TU Berlin, Germany.

- The mesh optimization framework presented in Chapter 6 was published and presented as *Laplacian Mesh Optimization* [NISA06] at ACM GRAPHITE 2006 in Kuala Lumpur, Malaysia, in collaboration with Takeo Igarashi from The University of Tokyo, Japan, as well as Olga Sorkine and Marc Alexa from TU Berlin, Germany.

- The mathematical observations in Appendix A were published as a technical report under the title *A Note on Boundary Constraints for Linear Variational Surface Design* [NS07], in collaboration with Olga Sorkine from TU Berlin, Germany.

Chapter 2

Fundamentals

In this chapter we elaborate on the current state of the art in mesh creation and editing. Furthermore, we introduce the basics of discrete differential geometry, Laplacian surface editing, and the associated Least-Squares methods that are relevant for this dissertation.

2.1 State of the Art

To provide sufficient context, this section attempts to give an overview of existing and ongoing work in various fields of interactive shape modeling. Specifically, we describe work on sketch-based modeling and surface-based modeling techniques. Note that we only summarize research efforts that are closely related to our own work. For other shape editing paradigms, such as parametric patches [Far90, GH95, SZBN03], space deformations (also known as free form deformations) [Bar84, SP86, Coq90, HHK92, MJ96, SF98, BK05, JSW05, SMW06, ACWK06, vFTS06, JMD+07], physically-based deformations [TPBF87, GM97, NMK+06], volume sculpting [Wil90, Nay90, GH91, WK95, PF01, FCG02, Bær02, JBS06], and other related work, we refer to the original publications, as well as some surveys throughout this dissertation.

2.1.1 Sketch-based modeling

Sketch-based modeling can be traced back to Ivan Sutherland's pioneering work *Sketchpad* [Sut63]. While Sketchpad was mainly targeted at engineering drawings, such as electrical circuit diagrams, trusses and bridges, it was also used for artistic drawings in 2D. Specifically, the user could sketch (or trace) freeform shapes, and explore variants by modifying and editing the sketches (see Figs. 9.9 and 9.10 in [Sut63]). More recently, this general principle was popularized in the Computer Graphics community in the SKETCH paper [ZHH96]. SKETCH presents the first interface, with which it is possible to create, place, edit, group,

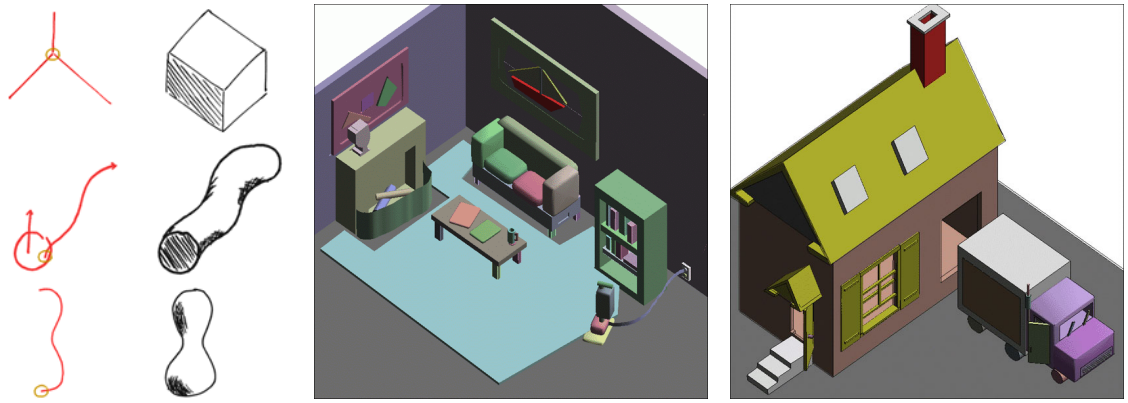


Figure 2.1: some gestures (left) and two scenes created with SKETCH [ZHH96].

and copy 3D geometry using simple scribbles and gestures (Figure 2.1). Most of the scenes are inherently rectilinear, but SKETCH already incorporates the possibility to model surfaces of revolution, extrusions, and curved cylinders. On the downside, its gesture-based nature makes it difficult to extend the interface further, which is why the authors regard it as a proof-of-concept application [ZHH96].

A representative of a *direct* and mostly gesture-free sketch-based modeling system was presented by Igarashi and co-workers in form of the highly popular *Teddy* [IMT99]. With Teddy, simple free form shapes can be sketched onto an empty canvas, simply by providing the silhouette of the desired shape (Figure 2.2, left). The system then automatically extrudes (inflates) the 2D silhouette, resulting in a rotund 3D shape. Since the mesh creation and extrusion procedure is inherently heuristic, the mesh turns out to be quite irregular and can contain badly shaped triangles (Figure 2.2, right). Teddy supports numerous modeling operations, such as create, cut, and extrude, which all have the property that they are applied immediately after the user places a stroke. Note that this differs significantly from work on optimization-based reconstruction [LS96, MKL05], where

Figure 2.2: A Teddy modeling session (left), some results (middle), and a wireframe mesh (right) [IMT99].

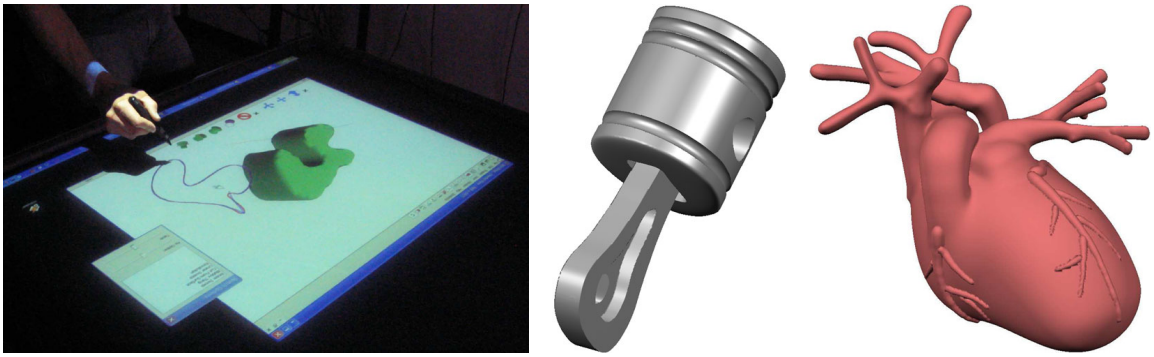


Figure 2.3: A ShapeShop modeling session (left) and some results [SWSJ05].

the user draws the entire 2D (wireframe) shape, and the system then solves a global optimization problem to infer 3D vertex positions. The interfaces and algorithms presented throughout this dissertation share the *immediate* nature of the Teddy modeling operations.

To alleviate some of the problems associated with meshing, Karpenko and coworkers proposed the use of variational implicits [TO99] instead of directly creating and manipulating meshes [KHR02]. Their system also includes an oversketching tool [Bau94, FRSS04]. Concurrently, Igarashi and Hughes presented an algorithm that beautifies *Teddy* meshes using an extension of *skin* [MCCH99] based on implicit quadratic surface fitting.

A similar multi-view modeling tool was presented by Bourguignon and coworkers [BCCD04]. In the *Relief* system the users strokes in a single, fixed view are interpreted as both the 2D shape outline as well as the displacement map, from which depths of new and existing vertices are inferred.

To enable more complex modeling tasks than Teddy [IMT99] or sketching variational implicits [KHR02], Schmidt and co-workers utilize hierarchical implicit volume models, also known as blob-trees [BW99], in their *ShapeShop* tool [SWSJ05]. Since complex shapes are constructed using blending and CSG operators, and the entire construction hierarchy is maintained throughout con-

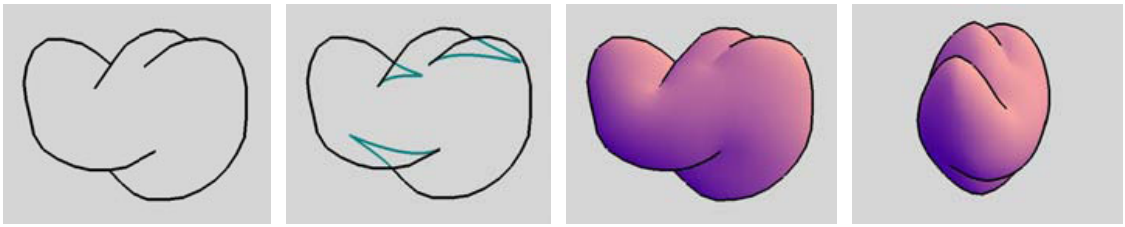


Figure 2.4: The process of inferring a smooth embedding from a complex sketch [KH06]. The image on the far right shows the model from a differen viewpoint.

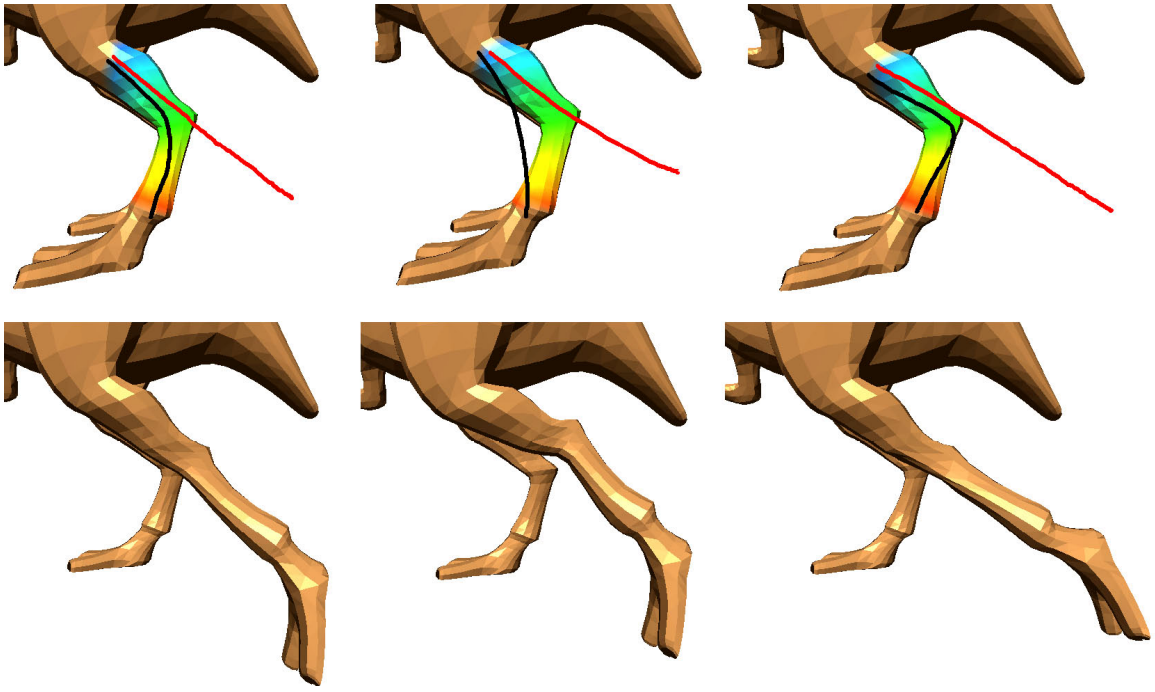


Figure 2.5: Three different reference/target sketch configurations demonstrated by Kho and Garland [KG05].

struction, any individual operation can be edited or removed independently (Figure 2.3). See also the early work of Bloomenthal and Wyvill on interactive techniques for implicit modeling [BW90].

Karpenko and Hughes developed the *SmoothSketch* method [KH06] with which a smooth surface can be inferred from a silhouette curve that includes cusps and T-junctions (where the contour is occluded). The proposed algorithm infers the hidden contours, and computes a smooth embedding based on these contours (Figure 2.4). In the work of Kara and Shimada [KS07], a smooth surface is computed between a network of previously sketched boundary curves. Unlike SmoothSketch, these curves can be sketched (and modified) from multiple viewpoints. Note that both [KH06] and [KS07] require the user to sketch multiple, potentially disjoint curves, and explicitly notify the application that a modification and/or creation operation is to be applied.

While most of the described sketch-based modeling applications above are targeted primarily towards model *creation* from scratch (to various extents), Kho and Garland [KG05] provide a mesh *editing* and deformation tool. Specifically, the user sketches a reference stroke onto the model, from which a subset of vertices to be displaced is inferred, and then provides a target stroke, resulting in a mesh deformation that is computed from the turning angles of both reference and target sketch (Figure 2.5).

2.1.2 Surface-based modeling techniques

Early finite element and functional optimization methods

In the early 90's, the first physically motivated, surface based modeling tools were established in the Computer Graphics research community. These tools were an attempt to introduce a higher degree of flexibility to the surface modeling process – for example, compared to modifying control points for a set of parametric patches.

Deformable curve and surface finite elements were introduced by Celniker and Gossard [CG91], motivated by Terzopoulos and co-workers pioneering work on physically-based modeling [TPBF87]. In their work, curves and meshes deform to minimize specific energy functionals, subject to user provided constraints and loads (Figure 2.6). This basic paradigm, finding a suitable representation/discretization for a curve/surface and deforming it such that it minimizes a certain energy functional subject to user-provided constraints, is the key ingredient to all surface-based modeling methods.

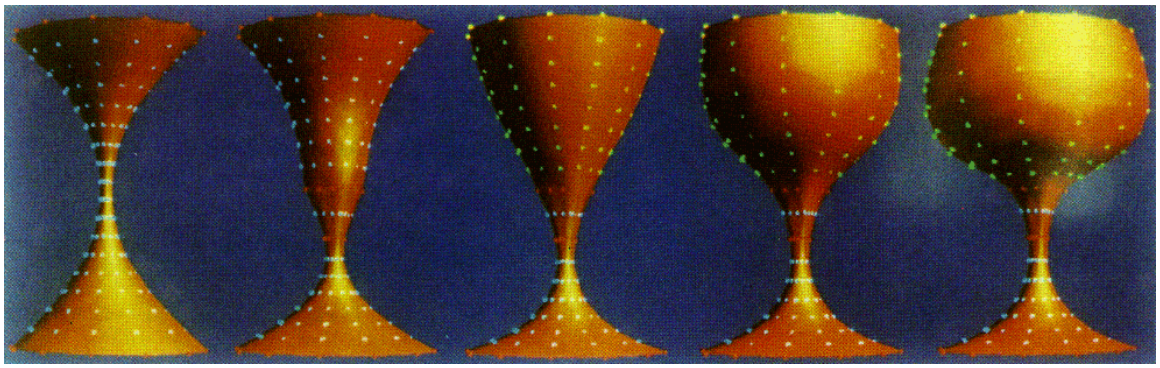


Figure 2.6: Deformable finite elements for shape design [CG91].

Moreton and Sequin extend this approach to Bézier patches [MS92]. From a given 3D network of connected line segments and continuity constraints, the algorithm computes a network with minimal variation of curvature (MVC). This network is thereafter used to initialize Bézier patch boundaries for the resulting minimal variation of curvature surface (MVS) with tangent continuity. Both curve network and surfaces are computed using nonlinear optimization techniques. Rhoades presents a bending operator, with which parametric patches can be edited by deforming (or *bending*) the normal field, and optimizing control points, such that the surface normals are close to the desired normals [Rho93].

A similar approach is described by Welch and Witkin [WW94], the main difference being that their shape is represented as a triangle mesh. The system allows the construction of surfaces of arbitrary topology, and furthermore performs inner

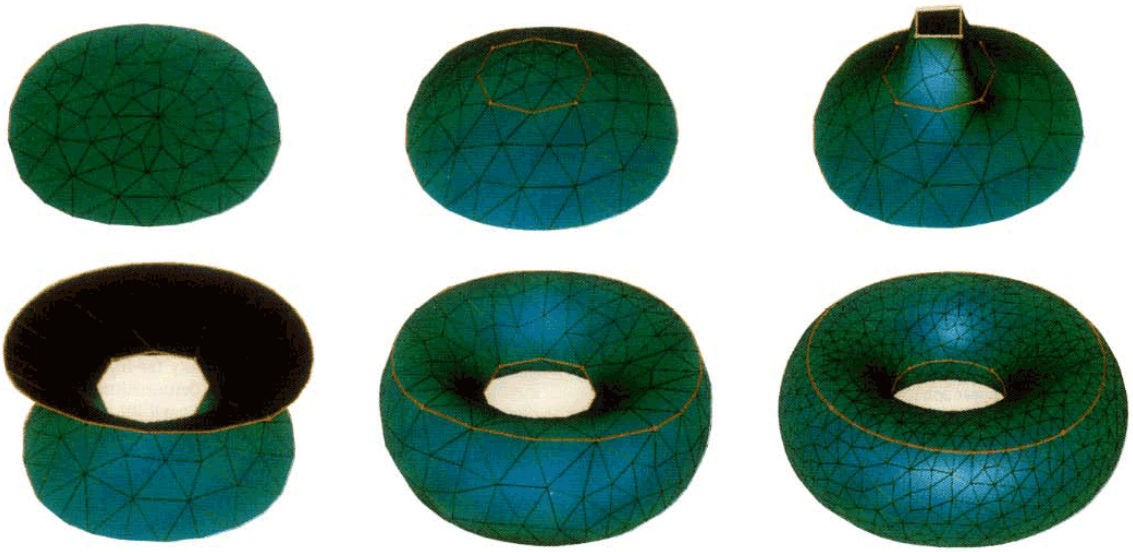


Figure 2.7: Creating a torus from a sheet of triangle mesh and user-provided (positional) constraints (= yellow lines) [WW94]. The resulting variational shape approximation is designed to minimize squared principal curvatures.

fairing and resampling of the surface on the fly (Figure 2.7). Outer fairing is performed by functional optimization, specifically Laplacian smoothing [Fie88]. To maintain interactivity for reasonably sized meshes, the optimization is carried out per vertex, instead of solving a global error minimization problem.

Multi-resolution methods

Multi-resolution mesh editing is a surface editing paradigm that has become widely popular since its introduction in 1997 [ZSS97]. The idea has been adapted and applied in various settings, such as multiresolution signal processing on meshes [GSS99]. The basic principle is to represent the mesh in a multi-resolution hierarchy, perform an editing operation (displacement of faces/vertices) on a coarse version of the mesh, and propagate this editing operation throughout the hierarchy, up to the finest mesh. Operations of this nature tend to modify the global shape, while maintaining local characteristics. Kobbelt and co-workers have extended this paradigm by designing the edit of the base-mesh to minimize curvature [KCVS98] (Figure 2.8). This is achieved through variational principles and discrete fairing [Kob00, Tau95].

In Figure 2.8 the black line defines the region of the mesh which is to be edited, and the white line is the user handle. The mesh is first smoothed in this region (center left), and the difference (the *detail*) between the two left meshes is stored with respect to local frames. After the user changes the position of the white handle curve, the boundary constraints for curvature minimization change, and

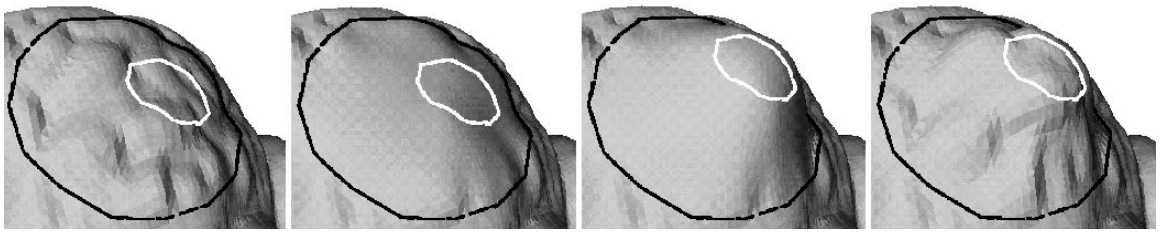


Figure 2.8: The principle of multi-resolution modeling [Kob00]. See text for details.

therefore the base surface undergoes a smooth deformation (center right). Adding the stored detail information yields the result on the far right.

Schneider and Kobbelt describe a geometric fairing method for irregularly sampled meshes, which they use in the context of free-form surface design [SK01]. This algorithm is described in more detail in Section 3, since it inspired our own variational approach for surface fairing. Essentially, they use a tessellation independent discretization of mean curvature, also known as the cotan weights [PP93a], and iteratively minimize a nonlinear fourth order partial differential equation, resulting in a smooth surface that satisfies up to G^1 boundary conditions.

Linear variational methods

Inspired by the use of differential coordinates – also known as discrete Laplacians – for local mesh morphing and editing [Ale03], Sorkine and co-workers developed a general framework for detail-preserving surface editing [SLCO+04]. After the user transforms a handle on the mesh, the system deforms the region of interest (ROI) such that the deformed Laplacian vectors are as-close-as-possible to the original Laplacians in the least-squares sense (Figure 2.9). This formulation leads to a single sparse linear system that can be solved efficiently. Note that Laplacian surface editing is fundamentally different from the multi-resolution methods

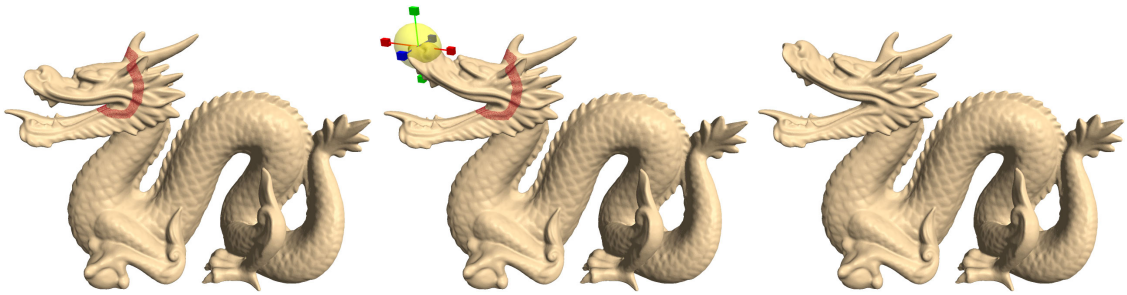


Figure 2.9: Laplacian surface editing [SLCO+04].

described above, since *detail* is incorporated into the optimization process, instead of being previously filtered out. The problem which arises, is that discrete Laplacians are not invariant w.r.t. global rotations of the mesh. To overcome this issue, Sorkine and co-workers implicitly compute linearized rotations for each Laplacian, an assumption which only holds for small rotations, yet is necessary to maintain the linear system property. Some recent work has been dedicated to computation of per-vertex similarity transformations [FAT07]. Also, Laplacian surface editing paradigms have been successfully applied to volumetric graphs [ZHS+05]. For more details on Laplacian surface editing, see Section 2.4. In a similar vein, Botsch and Kobbelt accelerate their multi-resolution modeling system by using a quadratic energy functional, leading to a linear system of equations for the minimizers, while also incorporating boundary continuity interpolation between C^0 and C^2. [BK04a].

Concurrently with Laplacian surface editing, and motivated by developments in Poisson image editing [PGB03], Yu and co-workers presented a mesh editing tool based on gradient field manipulation [YZX+04]. After the user transforms a given handle on the mesh, the transformation is smoothly propagated away from the handle. These individual per-face transformations create a disjoint mesh, which comprises the guidance vector field (= the resulting per-face gradient vectors). Solving the Poisson equation for each of the three coordinate functions conceptually *stitches* the mesh back together. In the original publication, the transformation propagation schemes are all based on geodesic distance, but more elaborate methods have been explored [ZRKS05].

Lipman and co-workers introduced the notion of linear rotation-invariant coordinates for meshes [LSLCO05], based on discrete forms defined on the mesh. Editing operations are carried out by manipulating vertex positions as well as local frame orientations, which is why this technique is also known as *moving frame manipulation* [LCOGL07]. The surface reconstruction algorithm described in [LSLCO05] consists of solving two sparse linear systems: one for the relationship between local frames of adjacent vertices, and one for computing vertex positions from the local frames. Shi and co-workers have accelerated this algorithm significantly using multigrid methods [SYBF06].

An alternative formulation in 2D, minimizing triangle shape distortion, is the key ingredient to the shape manipulation system presented by Igarashi and co-workers [IMH05a]. In search of a single quadratic error functional, which would lead to a single linear system, they prove that such a functional does not exist, and therefore split the problem into a rotation and scale part, each with its own quadratic error functional. The solution is then obtained by solving two linear systems sequentially. Using this method, interesting 2D animations can be created with only a few constraints (Figure 2.10).

For more details on linear variational surface deformation methods, see the reviews by Sorkine and Botsch [Sor06, BS07].

Figure 2.10: Animating a frog with three constraints by moving a single vertex [IMH05a].

Recent nonlinear optimization methods

There is a recent trend in the Computer Graphics community to revert to nonlinear optimization for surface-based modeling algorithms, instead of linearizing rotations, using quadratic error functionals exclusively, or other viable approximations. This development can be attributed to recent developments on multi-resolution/multi-grid methods, sparse linear solvers, and dimensional model reduction.

Based on a set of angles and lengths relating a vertex to its immediate neighbors, Scheffer and Kraevoy introduce pyramid coordinates [SK04], which are nonlinear in the vertex positions, but invariant under rigid transformations. These coordinates encode the local shape around each vertex, and are preserved throughout editing operations.

Huang and co-workers formulate an energy functional with a linear projection constraint, and nonlinear Laplacian, skeleton, and volume constraints [HSL+06]. To overcome the slow convergence and numerical instability of the iterative nonlinear solver, they project the deformation energy and constraints onto a control mesh (a subspace) using mean value coordinate interpolation [JSW05]. Combined with multi-resolution techniques, this speeds up the algorithm significantly, enabling interactive response times for meshes up to 170K vertices.

In their PriMo system, Botsch and co-workers model physically accurate deformation behavior inspired by thin shells, by embedding the surface in a layer of coupled volumetric prisms [BPG06]. The resulting nonlinear elastic energy is minimized very efficiently using a combination of local and global shape matching techniques on a multigrid hierarchy. This results in a system that runs at interactive rates (= 1 fps) for meshes with 180K triangles.

Our surface reconstruction algorithm presented in Chapter 3 falls into the category of nonlinear optimization methods.

2.2 Least-squares methods

Least-squares and weighted least-squares methods are commonplace tools used in a wide variety of applications. For a beautiful and illustrative introduction, see Strang's lecture notes [Str98]. Since the methods and algorithms described throughout this dissertation make heavy use of these tools, we describe them in sufficient detail. First, we use calculus to derive the linear system of equations for the global least-squares approximation of function values from scattered data using multivariate polynomials of some degree. *Scattered data* refers to an arbitrary set of points in $\mathbb{R}^d$ that carry scalar quantities (i.e. a scalar field in d dimensional parameter space). Thereafter, the linear system of equations, also known as the normal equations, is derived using linear algebra, specifically by orthogonal projection onto a subspace. Later, in Section 2.4, we describe how this formulation can be utilized in our setting: computing geometry from a set of linear constraints that are to be satisfied in the least-squares, or a weighted least-squares sense.

2.2.1 Least-squares approximation

Problem Formulation. Given N points located at positions $\mathbf{p}_i$ in $\mathbb{R}^d$ where $i \in [1 \dots N]$ (for $\mathbb{R}^3$, $\mathbf{p} = [x,y,z]^T$). We wish to obtain a globally defined function $b(\mathbf{p})$ that approximates given scalar values b_i at points $\mathbf{p}_i$ in a least-squares sense with the error functional $J_{LS} = \sum_i \|b(\mathbf{p}_i) - b_i\|^2$. Thus, we pose the following minimization problem

$$\min_{b \in \prod_m^d} \sum_i \|b(\mathbf{p}_i) - b_i\|^2, \tag{2.1}$$

where b is taken from $\prod_m^d$, the space of polynomials of total degree m in d spatial dimensions, and can be written as

$$b(\mathbf{p}) = \mathbf{a}(\mathbf{p})^T \mathbf{x} = \mathbf{a}(\mathbf{p}) \cdot \mathbf{x}, \tag{2.2}$$

where $\mathbf{a}(\mathbf{p}) = [a_1(\mathbf{p}), \dots, a_k(\mathbf{p})]^T$ is the polynomial basis vector and $\mathbf{x} = [x_1, \dots, x_k]^T$ is the vector of unknown coefficients, which we wish to obtain as a result of minimizing (2.1). Here are some examples for polynomial bases:

(a) for $m = 2$ and $d = 2$, $\mathbf{a}(\mathbf{p}) = [1, x, y, x^2, xy, y^2]^T$ (bivariate, quadratic)

(b) for a linear fit in $\mathbb{R}^3$ ($m = 1$, $d = 3$), $\mathbf{a}(\mathbf{p}) = [1, x, y, z]^T$ (trivariate, linear)

(c) for fitting a constant for arbitrary d, $\mathbf{a}(\mathbf{p}) = [1]$ (multivariate, constant)

In general, the number k of elements in $\mathbf{a}(\mathbf{p})$ (and therefore in $\mathbf{x}$) is given by $k = \frac{(d+m)!}{m!d!}$, see [Lev98, FM03].

2.2.2 Solution by calculus

We can minimize Equation 2.1 by setting the partial derivatives of the error functional J_{LS} w.r.t. the unknown coefficients to zero, that is $\nabla J_{LS} = 0$ where $\nabla = [\partial/\partial x_1, \ldots, \partial/\partial x_k]^T$, which is a necessary condition for a minimum. By taking partial derivatives with respect to the unknown coefficients $x_1, \ldots, x_k$, we obtain a linear system of equations (LSE) from which we can compute $\mathbf{x}$

$$\begin{aligned}
\partial J_{LS}/\partial x_1 = 0: &\quad \sum_i 2a_1(\mathbf{p}_i)[\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} - b_i] = 0 \\
\partial J_{LS}/\partial x_2 = 0: &\quad \sum_i 2a_2(\mathbf{p}_i)[\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} - b_i] = 0 \\
&\quad \vdots \\
\partial J_{LS}/\partial x_k = 0: &\quad \sum_i 2a_k(\mathbf{p}_i)[\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} - b_i] = 0.
\end{aligned}$$

In matrix-vector notation, this can be written as

$$\begin{aligned}
\sum_i 2\mathbf{a}(\mathbf{p}_i)[\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} - b_i] &= \\
2\sum_i [\mathbf{a}(\mathbf{p}_i)\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} - \mathbf{a}(\mathbf{p}_i)b_i] &= \mathbf{0}.
\end{aligned}$$

Dividing by the constant and rearranging yields the following LSE

$$\sum_i \mathbf{a}(\mathbf{p}_i)\mathbf{a}(\mathbf{p}_i)^T\mathbf{x} = \sum_i \mathbf{a}(\mathbf{p}_i)b_i, \tag{2.3}$$

which is solved as

$$\mathbf{x} = [\sum_i \mathbf{a}(\mathbf{p}_i)\mathbf{a}(\mathbf{p}_i)^T]^{-1} \sum_i \mathbf{a}(\mathbf{p}_i)b_i. \tag{2.4}$$

If the square matrix $\mathbf{A}_{LS} = \sum_i \mathbf{a}(\mathbf{p}_i)\mathbf{a}(\mathbf{p}_i)^T$ is nonsingular (i.e. $det(\mathbf{A}_{LS}) \neq 0$), substituting Equation 2.4 into Equation 2.2 provides the fit function $b(\mathbf{p})$. For small k ($k < 5$), the matrix inversion in Equation 2.4 can be carried out explicitly, otherwise numerical methods are the preferred tool, see [PTVF92] [1]. In our applications, we often use TAUCS [Tol03] [2] and Newmat [3].

Example. Say our data points live in $\mathbb{R}^2$ and we wish to fit a quadratic, bivariate polynomial ($d = 2$, $m = 2$) and therefore $\mathbf{a}(\mathbf{p}) = [1, x, y, x^2, xy, y^2]^T$ (see above),

[1] at the time of writing this dissertation, [PTVF92] was available online in pdf format through `http://www.nr.com/`

[2] `http://www.tau.ac.il/~stoledo/taucs/`

[3] `http://www.robertnz.net/nm_intro.htm`

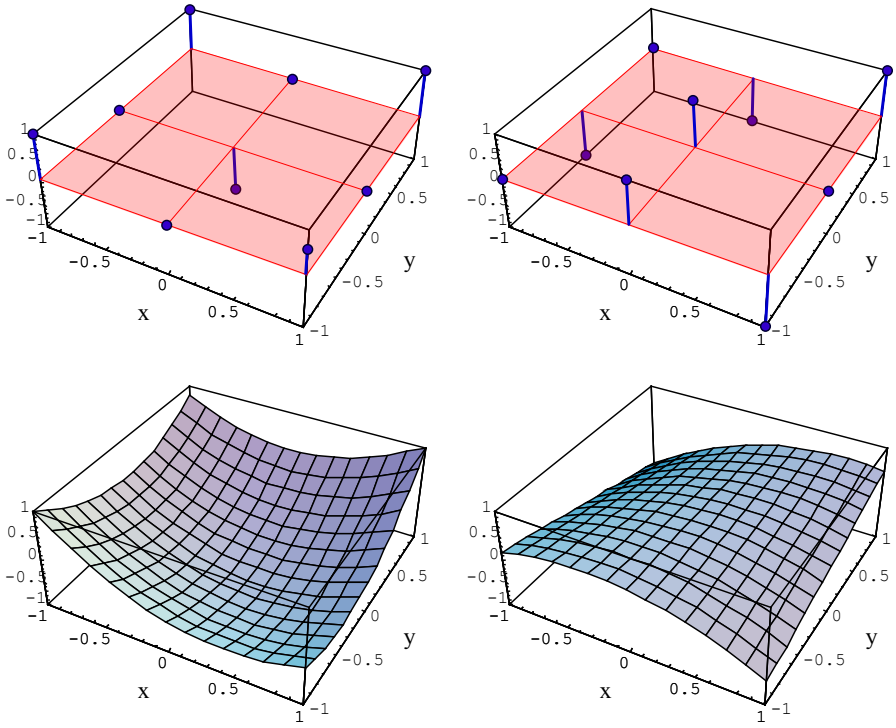


Figure 2.11: Fitting bivariate, quadratic polynomials to 2D scalar fields: the top row shows the two sets of nine data points (see text), the bottom row shows the least squares fit function. The coefficient vectors $[x_1,\ldots,x_6]^T$ are $[-0.834,-0.25,0.75,0.25,0.375,0.75]^T$ (left column) and $[0.334,\ 0.167,\ 0.0,-0.5,\ 0.5,\ 0.0]^T$.

then the resulting LSE looks like this

$$\sum_i \begin{bmatrix} 1 & x_i & y_i & x_i^2 & x_iy_i & y_i^2 \\ x_i & x_i^2 & x_iy_i & x_i^3 & x_i^2y_i & x_iy_i^2 \\ y_i & x_iy_i & y_i^2 & x_i^2y_i & x_iy_i^2 & y_i^3 \\ x_i^2 & x_i^3 & x_i^2y_i & x_i^4 & x_i^3y_i & x_i^2y_i^2 \\ x_iy_i & x_i^2y_i & x_iy_i^2 & x_i^3y_i & x_i^2y_i^2 & x_iy_i^3 \\ y_i^2 & x_iy_i^2 & y_i^3 & x_i^2y_i^2 & x_iy_i^3 & y_i^4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \sum_i \begin{bmatrix} 1 \\ x_i \\ y_i \\ x_i^2 \\ x_iy_i \\ y_i^2 \end{bmatrix} b_i.$$

Consider the set of nine 2D points P_i ={(1,1), (1,-1), (-1,1), (-1,-1), (0,0), (1,0), (-1,0), (0,1), (0,-1)} with two sets of associated function values b_i^1 ={1.0, -0.5, 1.0, 1.0, -1.0, 0.0, 0.0, 0.0, 0.0} and b_i^2 ={1.0, -1.0, 0.0, 0.0, 1.0, 0.0, -1.0, -1.0, 1.0}. Figure 2.11 shows the fit functions for the scalar fields b_i^1 and b_i^2.

2.2.3 Solution by projection onto a subspace

For a different but also very common notation, note that the solution for $\mathbf{x}$ in Equation 2.3 solves the following (generally over-constrained) LSE, where each row represents a single, linear constraint

$$\begin{bmatrix} \mathbf{a}^T(\mathbf{p}_1) \\ \vdots \\ \mathbf{a}^T(\mathbf{p}_N) \end{bmatrix} \mathbf{x} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix},$$

$$\text{or simply} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.5}$$

in the least-squares sense, using the normal equations

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\hat{\mathbf{x}} &= \mathbf{A}^T\mathbf{b} \\ \hat{\mathbf{x}} &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \end{aligned} \tag{2.6}$$

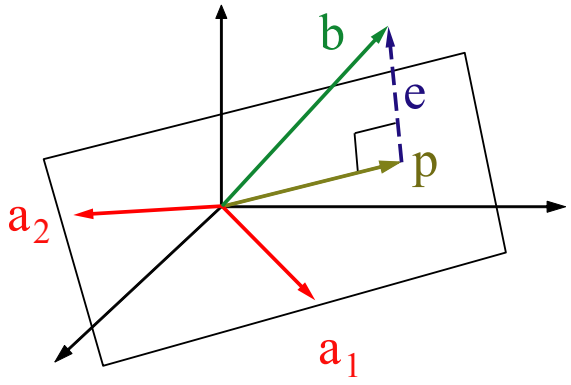


Figure 2.12: Solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ in the least-squares sense by orthogonal projection of $\mathbf{b}$ into the column space $\mathbf{C}(\mathbf{A})$ of $\mathbf{A}$. In this example, $\mathbf{C}(\mathbf{A})$ is represented as the plane spanned by the column vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ of the matrix $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2]$.

The derivation is rather elegant, see Figure 2.12. Given a matrix $\mathbf{A}$ with columns $\mathbf{a}_1, \ldots, \mathbf{a}_n$, any point $\mathbf{p}$ in the column space $\mathbf{C}(\mathbf{A})$ of a matrix $\mathbf{A}$ can be represented as a linear combination of the columns with coefficients $x_1, \ldots, x_n$ as

$$\mathbf{p} = \mathbf{a}_1\, x_1 + \mathbf{a}_2\, x_2 + \ldots + \mathbf{a}_n\, x_n = \mathbf{A}\mathbf{x}. \tag{2.7}$$

For general, overconstrained systems, the solution vector $\mathbf{b}$ from Equation 2.5 is not in $\mathbf{C}(\mathbf{A})$, as shown in Figure 2.12, and $\mathbf{A}\mathbf{x} = \mathbf{b}$ has no solution. In other words, since we generally have more linear constraints than unknowns, the matrix $\mathbf{A}$ has more rows than columns, and unless these constraints are linearly dependent, $\mathbf{b}$ is not in the column space of $\mathbf{A}$.

On the other hand, if we replace $\mathbf{b}$ with $\mathbf{p}$ in Equation 2.5, where $\mathbf{p}$ is the orthogonal projection of $\mathbf{b}$ onto $\mathbf{C}(\mathbf{A})$, we obtain the LSE $\mathbf{Ax} = \mathbf{p}$, which has a solution $\mathbf{x} = \hat{\mathbf{x}}$, and $\mathbf{p} = \mathbf{A}\hat{\mathbf{x}}$. This can be illustrated with Figure 2.12: it is obvious that every vector $\mathbf{b}$ has a part in the column space, and a perpendicular part in the nullspace of $\mathbf{A}$, denoted as $\mathbf{p}$ and $\mathbf{e}$ respectively. Since $\mathbf{Ax} = \mathbf{b} = \mathbf{p} + \mathbf{e}$ has no solution, we simply remove $\mathbf{e}$ and solve $\mathbf{Ax} = \mathbf{p}$ instead.

Furthermore, note that for any $\mathbf{x}$, the error between $\mathbf{Ax}$ and the solution $\mathbf{b}$ is

$$\|\mathbf{b} - \mathbf{Ax}\|^2 = \|\mathbf{p} - A\mathbf{x}\|^2 + \|\mathbf{e}\|^2 \tag{2.8}$$

by Pythagorean theorem. Since $\mathbf{p} = \mathbf{A}\hat{\mathbf{x}}$, by choosing $\mathbf{x}$ to be $\hat{\mathbf{x}}$ we reduce the term $\mathbf{p} - A\mathbf{x}$ to zero, and thereby minimize the squared length of $\mathbf{b} - \mathbf{Ax} = \mathbf{e}$.

As shown, the error vector $\mathbf{e} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ is orthogonal to the subspace. to compute $\hat{\mathbf{x}}$, we utilize this fact, and set the inner products of all column space with the error vector to zero

$$\begin{aligned} \mathbf{a}_1^T(\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}) &= 0 \\ &\vdots \\ \mathbf{a}_n^T(\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}) &= 0, \end{aligned} \tag{2.9}$$

which can also be written as

$$\begin{bmatrix} - \mathbf{a}_1^T - \\ \vdots \\ - \mathbf{a}_n^T - \end{bmatrix} \begin{bmatrix} \\ \mathbf{b} - \mathbf{A}\hat{\mathbf{x}} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{0} \\ \\ \end{bmatrix} = \mathbf{A}^T(\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}) = \mathbf{A}^T\mathbf{b} - \mathbf{A}^T\mathbf{A}\hat{\mathbf{x}}. \tag{2.10}$$

Rearranging yields the normal equations 2.6. Note though, that $\mathbf{A}^T\mathbf{A}$ is only invertible if and only if $\mathbf{A}$ has full column rank (= linearly independent columns).

2.2.4 Weighted least squares

Any row in Equation 2.5 represents a single, linear constraint, and these constraints can be individually weighted. For this we construct the diagonal matrix of weights $\mathbf{W}$, where w_{ii} is the scalar weight for constraint i, and plug it into the least-squares system as

$$\begin{aligned} \mathbf{WAx} &= \mathbf{Wb} \\ \mathbf{A}^T\mathbf{W}^2\mathbf{Ax} &= \mathbf{A}^T\mathbf{W}^2\mathbf{b} \\ \mathbf{x} &= (\mathbf{A}^T\mathbf{W}^2\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^2\mathbf{b}. \end{aligned} \tag{2.11}$$

Equation 2.11 is known as *weighted least-squares*. As described above, $\mathbf{A}^T\mathbf{W}^2\mathbf{A}$ is invertible if and only if $\mathbf{WA}$ has full column rank. For any $\mathbf{W}$ where this is the case, Equation 2.11 yields a unique solution.

2.3 Discrete differential geometry

Discrete differential geometry deals with the discrete analogon to the differential geometry of curves and surfaces [do 76]. For an excellent and applied introduction, see the course notes of Grinspun et al. [GDP+06].

Looking at the surface fairing techniques described in Section 2.1.2, most of which use quadratic, or higher order energy functionals of the general form

$$E_k(S) = \int F_k(S_{u\ldots u}, S_{u\ldots uv}, \ldots, S_{v\ldots v}), \tag{2.12}$$

where the expressions S_* represent partial derivatives of order k with respect to a surface parametrization $S : \Omega \to \mathbb{R}^3$, which is locally nearly isometric [BK04a]. To generate a smooth discrete or continuous surface, one derives the following differential equation

$$\mathbf{L}^k(\mathbf{x}) = 0 \qquad \mathbf{x} \in \Omega \setminus \delta\Omega \tag{2.13}$$

$$\mathbf{L}^k(\mathbf{x}) = b_j(\mathbf{x}) \qquad \mathbf{x} \in \delta\Omega \tag{2.14}$$

by applying variational calculus to Equation 2.12, and solves the resulting linear or higher order differential equation, with respect to given boundary conditions $b_j(\mathbf{x})$. In Equations 2.13 and 2.14, $\mathbf{L}$ is the Laplace operator of order k. In some settings, such as ours, is is necessary to discretize the Laplace operator, since the underlying surface representation is triangle mesh. Therefore, in this section we will elaborate on discrete Laplacian operators, both for curves as well as surfaces, and point out key observations that are used throughout this dissertation.

2.3.1 Piecewise linear curves

For the optimization and fairing of piecewise linear (PWL) curves, we will employ both first order

$$L_0 = \mathbf{v}_i - \mathbf{v}_{i-1} \tag{2.15}$$

and second order differentials

$$L_1 = \mathbf{v}_i - \frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{v}_j. \tag{2.16}$$

First order differentials are simply the edge vectors in the PWL curve and encode arc length. Second order differentials are vectors pointing from $\mathbf{v}_i$ to the centroid of the two adjacent vertices (in a curve, N_i is always ≤ 2) and encode the notion of *detail*, which can be interpreted as a curvature-like value (Figure 2.13). The vector in Equation 2.16 is also known as the uniformly weighted, discrete Laplacian vector, or graph Laplacian vector. As we will show in Section 3.3.1, it can be

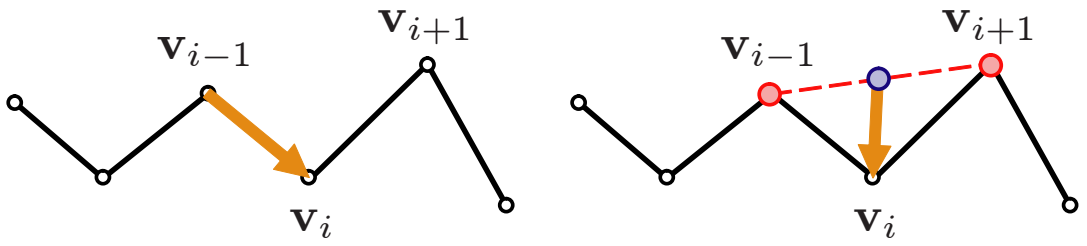


Figure 2.13: Discrete 1st and 2nd order differentials of a piecewise linear curve.

beneficial to minimize the difference between neighboring first order differentials, since for smooth, nearly straight curves, second order differentials vanish.

If the goal is to obtain a true estimate of discrete curvature at a PWL curve vertex, it is obvious that Equation 2.16 does not suffice. This is illustrated in Figure 2.14: if the curve $\{\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}\}$ is scaled anisotropically in the direction of $\overline{\mathbf{v}_{i-1}\ \mathbf{v}_{i+1}}$, for any scaling factor, the curve $\{\mathbf{v}'_{i-1}, \mathbf{v}_i, \mathbf{v}'_{i+1}\}$ generates to the same uniform discrete Laplacian vector. Clearly, these different configurations have different discrete curvatures.

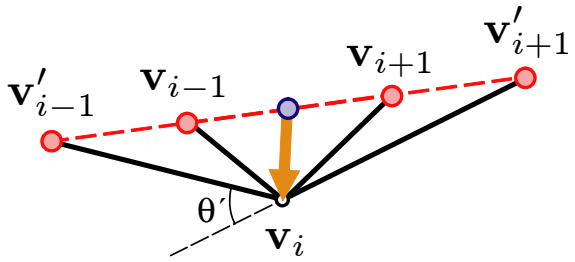


Figure 2.14: Both configurations $\mathbf{v}$ and $\mathbf{v}'$ generate the same uniform discrete Laplacian L_1.

Grinspun at al. [GDP+06] define the mean curvature normal for a PWL curve as the gradient of length L

$$\kappa\hat{\mathbf{n}} = \nabla L = 2sin\frac{\theta}{2}\hat{\mathbf{n}}, \tag{2.17}$$

where θ is the turning angle between two adjacent curve segments, $\hat{\mathbf{n}}$ is the unit normal, and κ is the curvature $1/R$. The notion that all curvature lives in the vertices is derived from the discrete Gauß map, where edges map to points, while vertices map to arcs. This notion of discrete curvature has the nice property that the total signed curvature (= the sum of turning angles) obeys the turning number theorem, which states that for a closed curve, the total signed curvature is an integer multiple of 2π.

Furthermore, for irregularly sampled PWL curves, the orientation of the uniform Laplacian vector does not provide the curve normal (Figure 2.15, left).

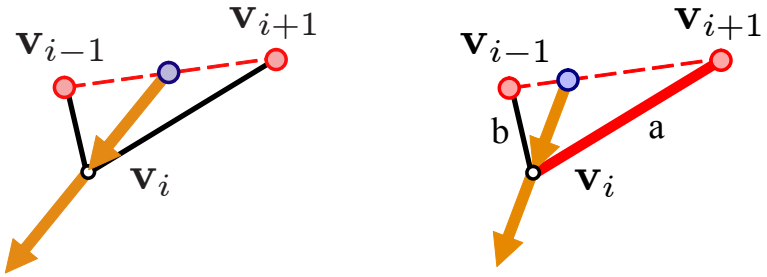


Figure 2.15: Obtaining the normal orientation for a piecewise linear curve. See text for details.

In such cases, it is necessary to modify the uniform weights in Equation 2.16 as

$$L_{1,w} = \mathbf{v}_i - \sum_{j \in N_i} w_{ij} \mathbf{v}_j, \tag{2.18}$$

where $\sum_j w_{ij} = 1$. The weights are computed from the opposite edge length: for Figure 2.15 (right), denote the lengths of segments $\overline{\mathbf{v}_i\,\mathbf{v}_{i+1}}$ and $\overline{\mathbf{v}_i\,\mathbf{v}_{i-1}}$ as a and b respectively, then the weights of vertices are computed as

$$w_{i,i-1} = \frac{a}{a+b} \qquad\qquad w_{i,i+1} = \frac{b}{a+b}.$$

Note that this is identical to the summation of arc-length weighted edge normals.

2.3.2 Piecewise linear surfaces

Laplacian operators can easily be defined for discrete surfaces (in our case, triangle meshes) as well. These 3-vectors have various names in the literature, such as *detail-* or *differential-coordinates*, as well as *detail vectors*.

The equivalent of the uniform Laplacian operator on a mesh is defined as

$$L_{1,u} = \mathbf{v}_i - \frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{v}_j. \tag{2.19}$$

Similar to the curve setting, this vector does not point in normal direction for irregularly sampled surfaces (Figure 2.16, left), yet has the property, that it only depends on the mesh connectivity. In other words, the weights $1/|N_i|$ are identical for any two geometries, since they only depend on the number of adjacent vertices.

There have been numerous efforts to quantify the notions of Gaussian and mean curvature for discrete surfaces, and many of these are based on the work of Pinkall and Polthier [PP93a].

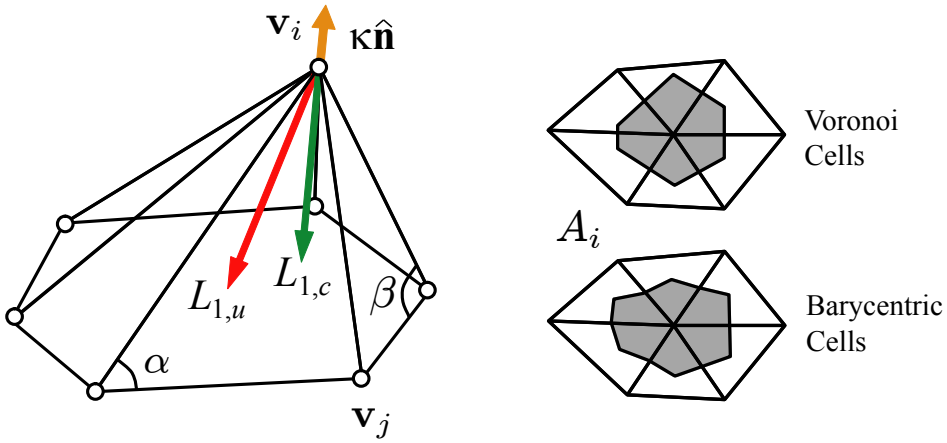


Figure 2.16: Left: the 1-ring around a vertex and the discrete quantities associated with it. Right: area estimates for the center vertex.

To obtain a Laplacian vector that points in normal direction, we can use the cotangent weights [PP93a, DMSB99]

$$L_{1,c} = \mathbf{v}_i - \sum_{j \in N_i} w_{ij} \mathbf{v}_j \tag{2.20}$$

where the weights

$$w_{ij} = cot\alpha + cot\beta \tag{2.21}$$

for each adjacent vertex are computed from the angles shown in Figure 2.16 (left). Unlike Equation 2.19, since we need to compute angles, these weights depend on the mesh geometry. This vector is generally known as the cotangent (or *cotan*) Laplacian.

Meyer at al. [MDSB03] present a robust method for computing discrete mean curvature normals for each vertex. By computing an area estimate for each vertex (Figure 2.16, right) one can scale the cotangent Laplacian as

$$L_{1,c} \frac{1}{2A_i} = \kappa\hat{\mathbf{n}}, \tag{2.22}$$

resulting in the mean curvature normal $\kappa\hat{\mathbf{n}}$. Note that $\kappa\hat{\mathbf{n}}$ is shown flipped in Figure 2.16 for illustration purposes.

We will make use of the following three observations throughout this dissertation

1. The uniform Laplacian has a tangential component, while the cotangent Laplacian does not (Figure 2.17, left). We use this in Chapters 4 and 6 for tangential smoothing (improvement of inner fairness) of triangle meshes by solving a global least-squares problem.

2. For nearly equal edge lengths, the uniform Laplacian $L_{1,u}$ is a viable approximation of the cotangent Laplacian $L_{1,c}$ (Figure 2.17, right). We will use this approximation to speed up our computations in Chapter 3

3. The cotangent Laplacian $L_{1,c}$ is equal to the mean curvature normal multiplied by a vertex area estimate. $L_{1,c}$ is also known as the *integrated* mean curvature [WBH+07]. In other words, given a scalar mean curvature, a normal orientation, and an area estimate around a vertex, one can reconstruct the cotangent Laplacian (this is used in Chapter 3).

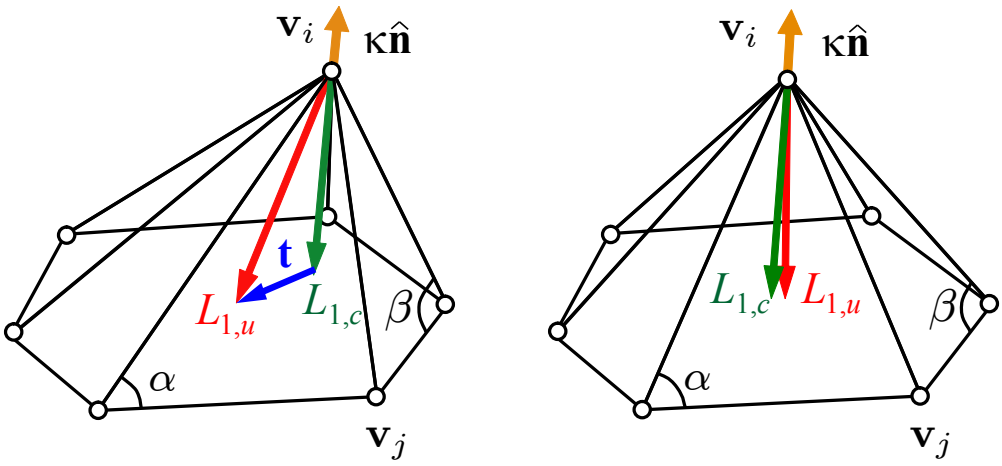


Figure 2.17: For an irregularly sampled surface, the uniform ($\mathbf{L}_{1,u}$) and cotangent ($\mathbf{L}_{1,c}$) Laplacian vectors generally differ (left), since $\mathbf{L}_{1,u}$ has a tangential component $\mathbf{t}$, while $\mathbf{L}_{1,c}$ points in normal direction. For nearly equal edge lengths, $\mathbf{L}_{1,u} \approx \mathbf{L}_{1,c}$.

2.4 Laplacian surface editing

The feature editing tools presented in Chapters 4 and 5 do not rely on a specific mesh deformation technique. Our methods generate three sets of vertices: the region of interest (ROI) R, the static anchors $S \subset R$, and the handle vertices $H \subset R$, including a displacement vector for each handle vertex in H. There are many mesh deformation tools that rely on these input parameters. In our implementations, we use Laplacian surface editing [SLCO+04, LSCOL04] for the following reasons:

- Laplacian surface editing uses linear constraints and generates a surface that satisfies these linear constraints in the least-squares sense. We add various linear constraints in our formulations, and using Laplacian surface editing makes it simple to incorporate them into the existing system matrix.
- Both the local (Chapter 4) as well as the global (Chapter 6) tangential smoothing approaches use the uniform Laplacian operator to solve for new vertex positions, and we can reuse the factorization of the Laplacian matrix when experimenting with the various parameters for these procedures. This significantly speeds up computations.
- In Chapter 6 we present a general framework for triangle shape optimization and mesh smoothing, and show that both Laplacian surface editing and least-squares meshes [SCO04, SCOIT05] are special cases of this framework.

2.4.1 Basic formulation

Given n vertices and their 1-rings (Figure 2.18, left) one can construct either uniform or cotan Laplacian vectors for each coordinate function x, y and z in $\mathbb{R}^3$ using an $n \times n$ matrix that encodes the appropriate Laplacian weights (see Figure 2.18, right, and Section 2.3.2).

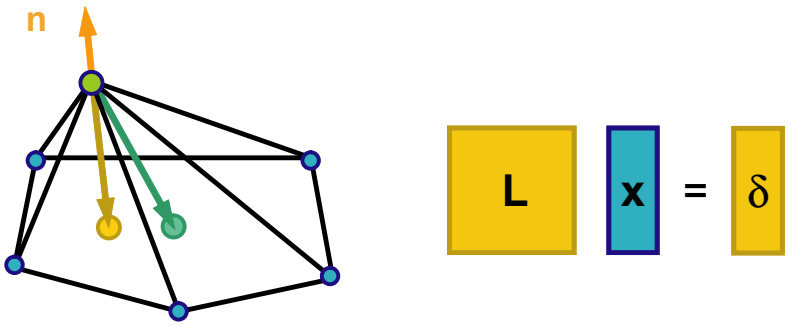


Figure 2.18: Left: the 1-ring around a vertex and the cotan (yellow) and uniform (green) Laplacian vectors. Right: computing the cotan Laplacians using matrix-vector multiplication for a single, scalar coordinate function.

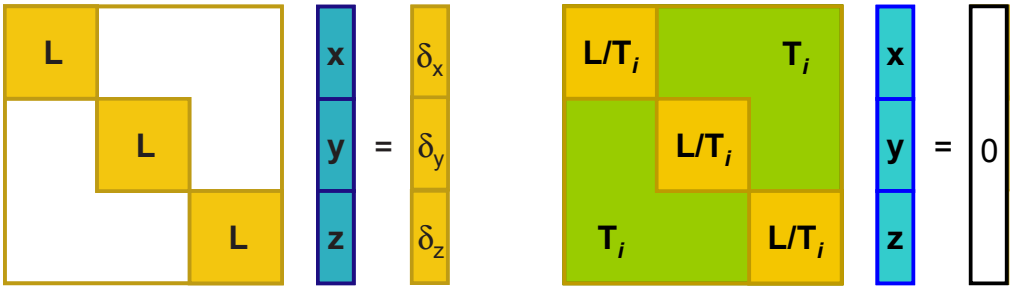


Figure 2.19: Left: the $3n \times 3n$ linear system for all coordinate functions in $\mathbb{R}^3$. Right: incorporating linearized rotations potentially adds T_i coefficients in all matrix blocks.

To construct Laplacian vectors for all three coordinate functions simultaneously, we only need to copy the L matrix twice (Figure 2.19, left).

The inverse operation, constructing the mesh from its Laplacian vectors, is not (yet) possible: the matrix L in Figure 2.18 has rank $n-1$ and is not invertible, since any global translation of the mesh results in the same Laplacian vectors. By fixing a single vertex though, the matrix becomes invertible, and we can uniquely reconstruct the mesh from its Laplacian coordinates.

Unfortunately, while Laplacian coordinates are invariant w.r.t. global translation, they are not invariant under general affine transformations of the mesh: simply rotating the 1-ring results in a different orientation of the Laplacian vector. Sorkine and co-workers linearize small rotations and uniform scaling using a skew-symmetric matrix, and incorporate the matrix coefficients into the linear system. The process is rather involved, and details can be found in [SLCO+04]. In any case, the system can now only be solved for all coordinate functions simultaneously, since the coefficients are no longer only in the block diagonal matrices (Figure 2.19, right).

Once the system of linear constraints is set up for the Laplacian vectors, adding positional constraints simply amounts to adding three rows (one for each coor-

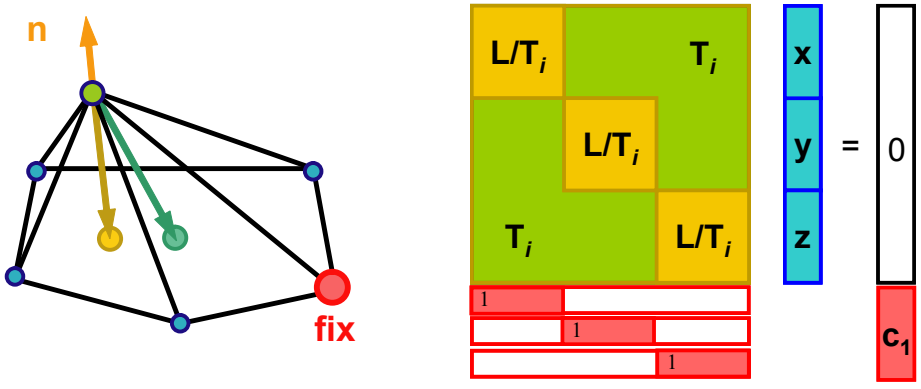


Figure 2.20: Adding positional constraints to the system (left) and the corresponding linear constraints (right).

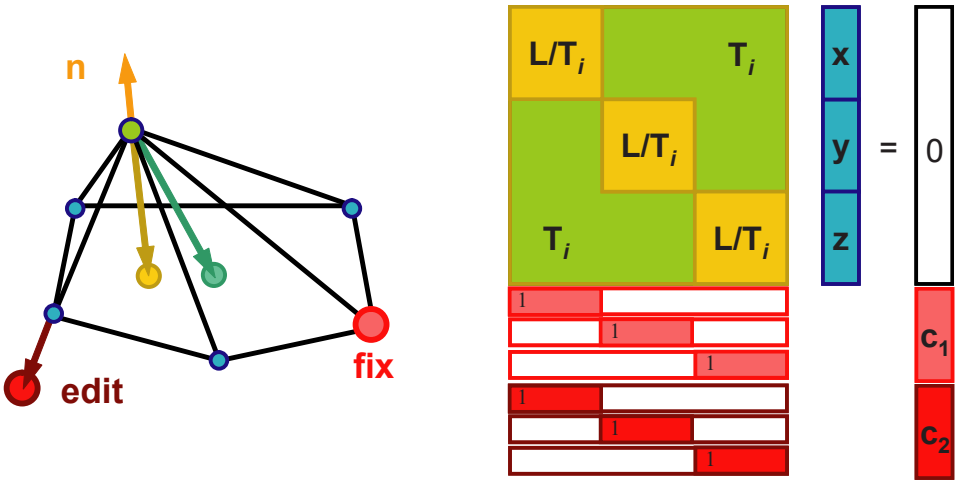


Figure 2.21: Adding editing constraints to the system (left) and the corresponding linear constraints (right).

dinate function) to the system, which we denote as $\mathbf{Ax} = \mathbf{b}$. This is shown in Figure 2.20, where the 3-vector $\mathbf{c}_1$ encodes the position of the fixed vertex in $\mathbb{R}^3$. The overall system $\mathbf{Ax} = \mathbf{b}$ becomes invertible, and we can compute vertex positions as $\mathbf{x} = \mathbf{A}^+\mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, where $\mathbf{A}^+$ is known as the moore-penrose pseudoinverse [Pen55].

To carry out a detail-preserving edit, the user might want to alter the position of a vertex, or a set of vertices. To implement this edit, we again add three rows to the system matrix. The difference is, that instead of using the original vertex position on the right hand side vector $\mathbf{b}$, we instead use the vertices displaced position. Figure 2.21 illustrates the vertex sets mentioned above: in this case, the set R consists of all vertices, while S and H contain the fixed and edited vertex respectively. For general edits, this matrix is no longer invertible, and there is no exact solution to $\mathbf{Ax} = \mathbf{b}$, since the (sub)space of meshes does not contain a solution that satisfies both the Laplacian and positional constraints. The idea is to solve the system in the least-squares sense by projecting $\mathbf{b}$ onto the subspace of meshes (Section 2.2), thereby obtaining a mesh $\hat{\mathbf{x}}$ with small error. In practice, we construct the normal equations $\mathbf{A}^T\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^T\mathbf{b}$ and solve for unknown vertex positions $\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$ (Equation 2.6). Note that since user interaction only modifies the right hand side $\mathbf{b}$ (the edited vertex positions), we can factorize the matrix $\mathbf{A}^T\mathbf{A}$ once, and only need to perform fast back-substitution while the user drags the handle.

2.4.2 Extensions

For our feature editing tools (Chapters 4 and 5), we have found that it is beneficial to not only constrain vertex positions, but rather *any* position on the mesh. Since

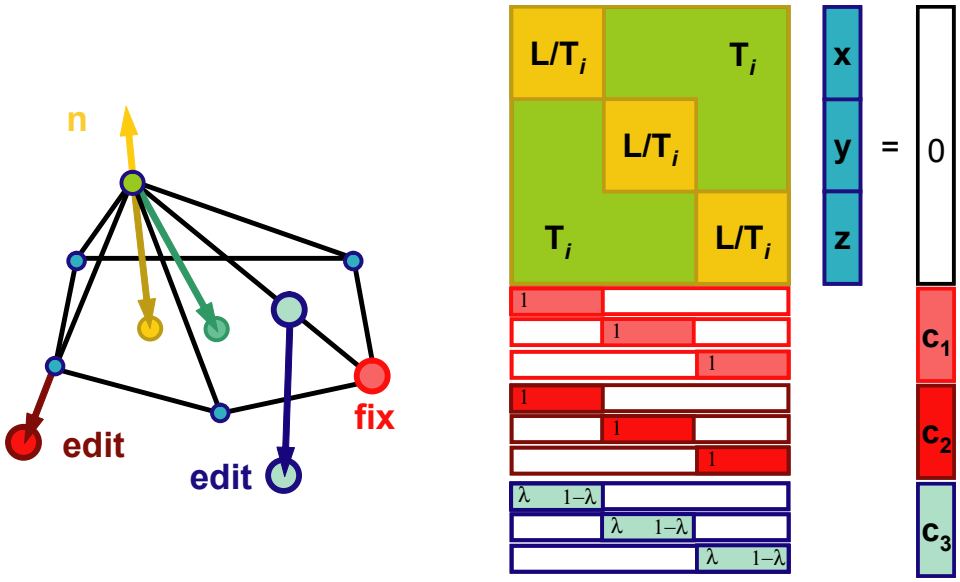


Figure 2.22: Adding other linear editing constraints to the system.

on-edge or on-face positions can be represented as a linear combination of the adjacent vertices, adding these linear constraints to the system is, yet again, adding lines to the matrix **A** and the right hand side **b**, and solving in the least-squares sense (Figure 2.22).

Thanks to the power of least-squares techniques, our extended Laplacian surface editing framework is not limited to vertex constraints. We have implemented a tool, by which the user can scale the magnitude of the Laplacian vectors. This editing operation creates sharp ridges and ravines, as we show in Chapter 4. For the linear system, this means removing the coefficients related to (linearized) rotations and scales, and instead prescribing the magnitude of the Laplacian vector on the right hand side (Figure 2.23).

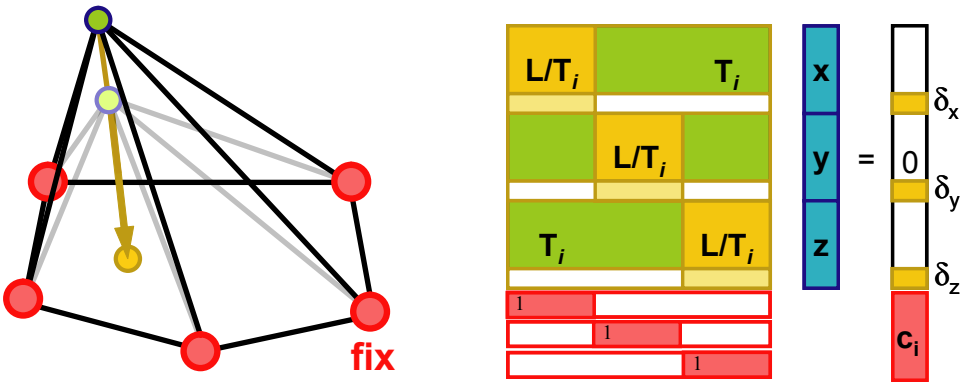


Figure 2.23: Scaling Laplacian vectors.

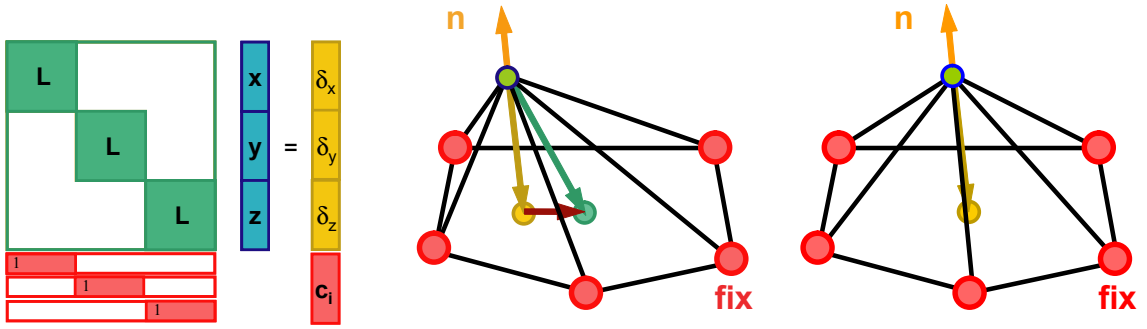


Figure 2.24: Tangential smoothing. See text for details.

We leverage global least-squares for one-step tangential smoothing in Chapter 6. As described in Section 2.3, the uniform Laplacian has a tangential component, while the cotan Laplacian does not. This component is depicted as the dark red difference vector between uniform and cotan Laplacian in Figure 2.24 (middle). If we set up the linear system, such that $\mathbf{L}_u\mathbf{x} = \delta_{cotan}$, subject to boundary constraints, we are forcing the uniform Laplacian to be equal to the cotan Laplacian (Figure 2.24, left). This can be interpreted as a form of tangential (or Laplacian) smoothing, and improves inner fairness (= triangle shapes) of the mesh, while retaining local features (Figure 2.24, right).

Finally, all the above systems can be modified by weighting the linear constraints non-uniformly. Using a diagonal matrix of weights $\mathbf{W}$, where w_{ii} is the scalar weight for constraint i, and plugging it into the least-squares system yields $\mathbf{x} = (\mathbf{A}^T\mathbf{W}^2\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^2\mathbf{b}$, as described in Section 2.2. This principle is used widely in our work – specifically, for approximate sketching (Chapters 4 and 5) and feature preserving mesh smoothing (Chapter 6).

Chapter 3

Designing Freeform Surfaces with 3D Curves

3.1 Introduction

Current tools for free-form design, and the resulting design process, can be roughly categorized into two groups. The group of professional modeling packages makes use of parametric patches or subdivision surfaces [May07, dM07], where the user has to lay out the coarsest level patches in an initial modeling stage, and then modify control points to generate details. Because it is difficult for inexperienced users to generate the control structure for an intended shape from scratch, a group of research tools [IMT99, IH03, SWSJ05, KH06, KS07] as well as in-game character editors [Max07, Gin07] are built around intuitive modeling metaphors such as sketching, trying to hide the mathematical subtleties of surface description from the user. However, some of these tools lack a high-level control structure, making it difficult to iteratively refine the design, or re-use existing designs.

We try to bridge the gap by using curves, a universally accepted modeling metaphor, as an interface for designing a surface. Notice that curves appear in both tools mentioned above: they appear as parameter lines, or seams where locally parameterized patches meet; they are sketched to generate or modify shape, or they are extracted from the current shape and used as handles. Also note that traditional design is mostly based on drawing characteristic curves.

Yet, design is a process. We cannot expect a user to draw the control (or characteristic) curves of a shape into free space. Our first fundamental idea is to let the user **define control curves by drawing them onto the shape** in its current design stage. These curves can be used as handles for deformation right after their definition, as in other tools, or at any other time in the design process. Of course, the effect of control curves can be modified (i.e. smooth vs. sharp edge), they can be removed from the current design, and there are no restrictions on their

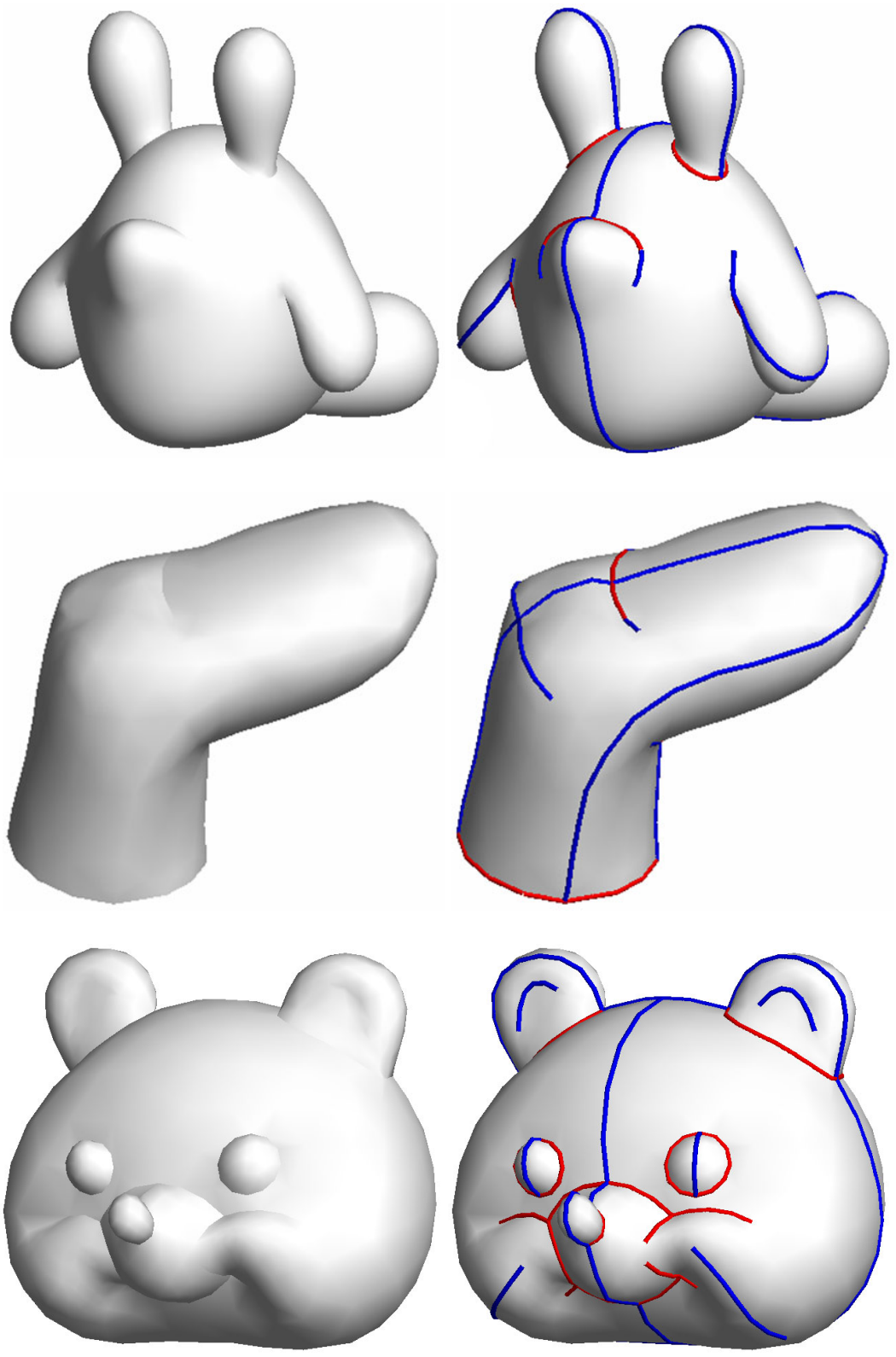


Figure 3.1: Modeling results using FIBERMESH. The user interactively defines the control curves, combining sketching and direct manipulation, and the system continuously presents fair interpolative surfaces defined by these curves (blue = smooth curve, red = sharp curve).

placement and topological structure. Specifically, they may be connected to or intersect other curves, or not; this is more general than recent developments for parameterized surfaces [SZBN03, SWZ05]).

The second fundamental principle is that **the shape is defined by the control curves** at any stage of the design process. While we found it important to serve the process of construction, and this is also what defines the topology of the surface, the result should be independent of when a control curve was modified. We achieve this by defining the surface to minimize certain functions of its differentials [MS92, WW94], while constraining it by the control curves.

It is crucial that both the modification of curves as well as the computation of surface geometry allow for an interactive and smoothly responding system. For this we build on the recent advances in discrete Laplacian [SLCO+04, YZX+04, BK04a] and other higher order or non-linear functionals for surface processing [HSL+06, BPG06, WBH+07].

Uniquely combining interface metaphors (Section 3.2) with geometry processing techniques (Section 3.3), our contributions are

- A fair surface definition based on curve constraints, and an accompanying functional optimization algorithm that runs at interactive rates.
- A detail preserving, real-time 3D curve editing and peeling interface, and a curve deformation algorithm based on discrete co-rotational methods.
- The generation and smooth embedding of initial surface components by sketching a planar control curve on a canvas.
- An interface that enables the design of 3D models with 3D control curves. The user's 2D sketching operations turn into 3D curves, and they serve as handles for subsequent editing.

Welch and Witkin [WW94] propose an interactive modeling system where the user can cut up surfaces and paste them together, while the system continuously generates a fair interpolative surface. They demonstrate the capability of the approach by showing topologically non-trivial shapes such as branching surfaces and a Klein bottle. Our work extends this approach, allowing the user to design more practical models such as 3D characters (Figure 3.1), by introducing a high-level user interface for curve control.

Discrete differential surface representations are an active research area [Sor06]. One of the key driving forces is the use of highly efficient sparse linear solvers [Tol03, Dav04]. They can efficiently solve matrix systems of tens of thousands of entries, which makes it possible to process interesting 3D meshes in real-time. Most efforts have been dedicated to the editing of *existing* 3D models. The work described in this chapter is an attempt to apply these techniques and tools to surface *creation* from scratch.

3.2 User interface

From the user's point of view, our system can be seen as an extension to a freeform modeling system based on silhouette sketching, such as Teddy [IMT99]. The user interactively draws the silhouette of the desired geometry and the system automatically constructs a (rotund) surface via functional optimization, such that its silhouette matches the user's sketch. However, unlike previous systems, the user's original stroke *stays* on the model surface and serves as a handle for further geometry control. The user can manipulate these curves interactively and the surface geometry changes accordingly. In addition, the user can freely add and remove control curves on the surface. These extensions enable the design of far more elaborate shapes than those possible with sketching alone. Figure 3.2 shows an overview of the process.

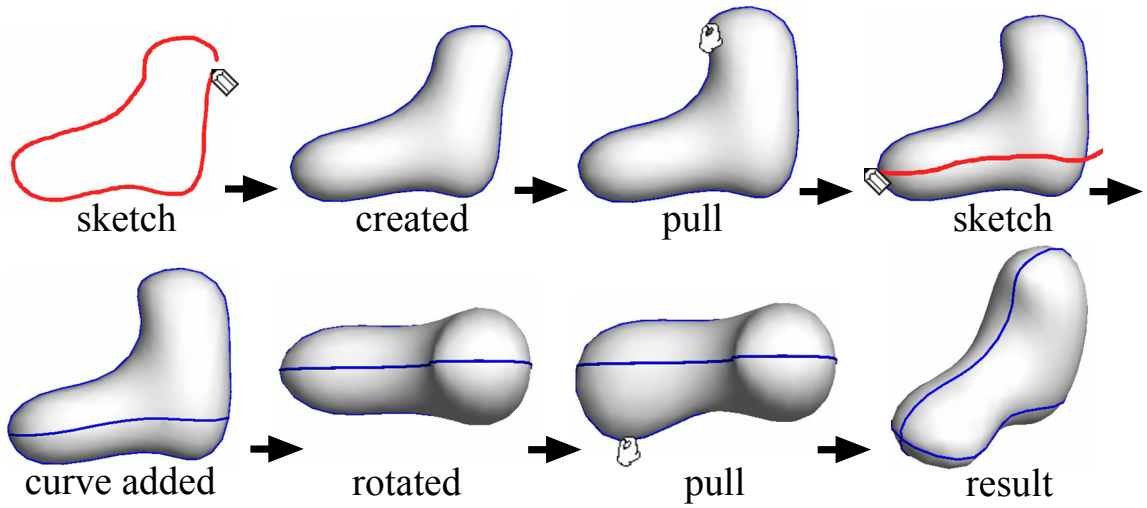


Figure 3.2: An example modeling sequence.

In a sense, our modeling process is similar to traditional modeling methods, such as parametric patches and subdivision surfaces: the user also defines nets of curves and the system automatically generates a smooth surface based on these. An advantage of our interface, is that the user does not need to worry about the topology of the curves. Traditional methods require the user to cover the entire surface with triangle or quad regions. Our method is much more flexible: curves need not be connected to other curves and much fewer curves can represent simple geometry. It is also important that, instead of providing individual points as an interface, our interface treats curves as continuous entities. We believe this can help smooth the "skill transfer" from 2D drawing to 3D modeling.

Various interactive modeling methods have been proposed in research contexts, including direct 3D editing [PF01], spatial deformation operations [SF98, ACWK06, vFTS06] and surface based deformation tools [ZSS97, KCVS98].

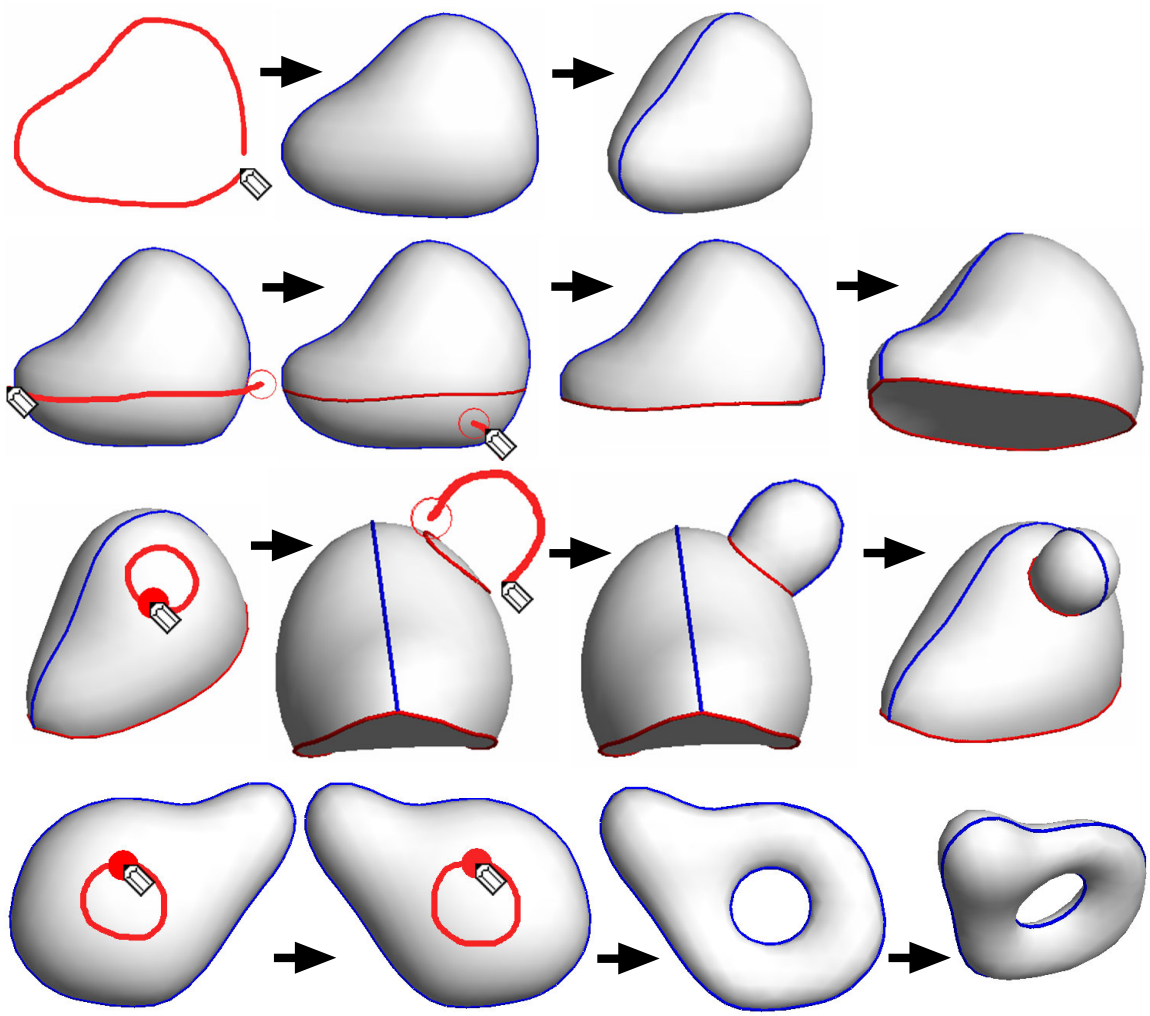


Figure 3.3: Sketching operations (from top to bottom): creation, cut, extrusion and tunnel.

These modeling methods provide reduced degrees-of-freedom handles (curves and control meshes) for surface control. The user can focus on the high-level control and the system automatically maintains aesthetic consistency. Our approach is unique in that we use curves also as the *definition* of the surface, not only as temporal handles for deformation.

Our current modeling interface consists of five tools (modes): sketching tool, deformation tool, rubbing tool, erasing tool, and type change tool. The user switches between these tools via menu selection or a keyboard shortcut.

3.2.1 Sketching tool

Our system provides five kinds of sketching operations: creation, cut, extrusion, tunnel (Figure 3.3), and add-control-curve (Figure 3.4). When the user draws a

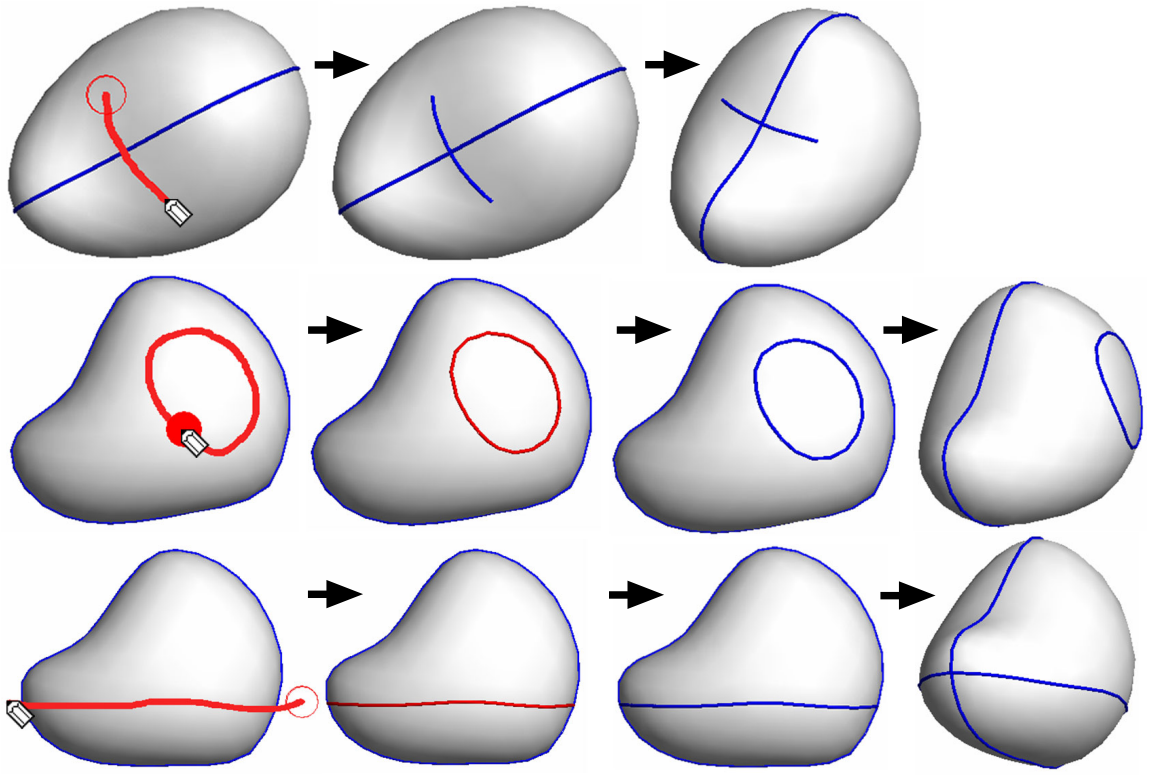


Figure 3.4: Adding control curves: open stroke (top), closed stroke (middle) and cutting stroke (bottom). The user needs to click after drawing a stroke to make it a control curve in the cases of closed stroke and cutting stroke.

closed stroke on a blank canvas, the system automatically inflates the closed area and presents an initial 3D model. The user draws a stroke crossing the model to cut it. Drawing a closed stroke on the object surface followed by a silhouette stroke creates an extrusion. If the user draws another closed loop on the opposite side of the surface, the system generates a tunnel. These operations are borrowed from the original Teddy system, but the difference is that the user's original strokes stay on the model surface as control curves. These control curves literally define the surface shape (as positional constraints in the surface optimization), and the user can modify the shape by deforming these control curves. New control curves can be added by drawing an open stroke on the object surface, drawing a closed stroke followed by clicking, and by drawing a cutting stroke followed by clicking (Figure 3.4). The last method is very useful during the early stages of model creation, since it allows the user to quickly generate a convenient handle to adjust the amount of inflation (or *fatness*).

The control curves are divided into two types: smooth curves (blue) and sharp curves (red). A smooth curve constrains the surface to be smooth across it, while a sharp curve only places positional constraints with C^0 continuity. These types

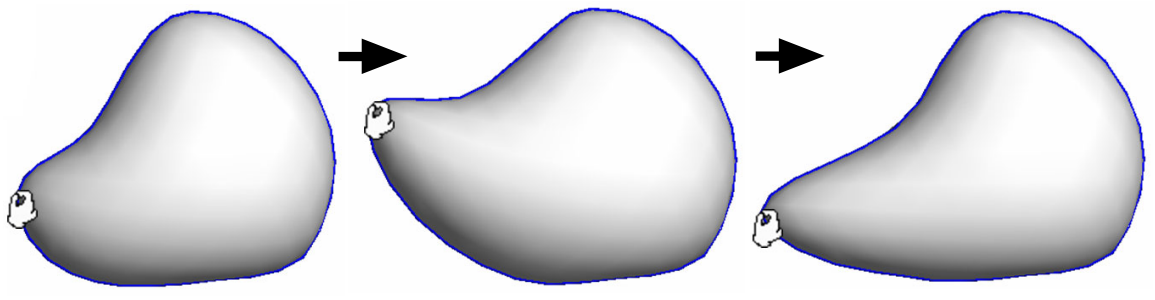


Figure 3.5: Pulling a curve. The deformed curve segment is determined by how much of it the user peels off.

are automatically assigned to the newly added curves according to the sketching operation the user applied. The creation operation generates a smooth curve that corresponds to the silhouette. A cutting operation generates a sharp curve. An extrusion generates one sharp curve along the base, and one smooth curve along the silhouette. When the user paints a new curve on the surface, it is initially defined as a smooth curve. The type of these curves can be freely changed afterwards using the type change tool.

3.2.2 Deformation tool

The deformation tool lets the user grab a curve at any point and pull it to the desired location. The curve deforms accordingly, preserving local details as much as possible (see Figure 3.5 and Section 3.3.1). Editing operations are always applied to the control curves, not directly to the surface. If the user wants more control, new control curves must be added on the surface. Explicit addition of control curves exposes the surface structure in a clear way, and the curves serve as a convenient handle for further editing.

We use a peeling interface for the determination of the deformed curve segment (region of interest, ROI) [IMH05a]. The size of the curve segment to be deformed is proportional to the amount of pulling. The more the user pulls, the larger the deformed curve segment becomes. This frees the user from manually specifying the ROI before starting deformation and enables dynamical adjustment of the ROI during deformation. This peeling effect propagates to the other curves connected to the deformed curve, which allows the user to deform a larger area of the surface.

3.2.3 Rubbing tool

The rubbing tool is used for smoothing a curve. As the user drags the mouse back and forth (rubs) near the target curve, the curve gradually becomes smooth. The more the user rubs, the smoother the curve becomes (Figure 3.6). This tool

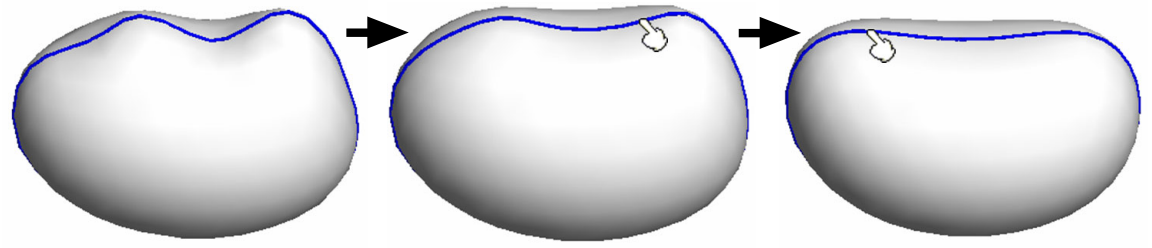


Figure 3.6: Rubbing (smoothing) a curve.

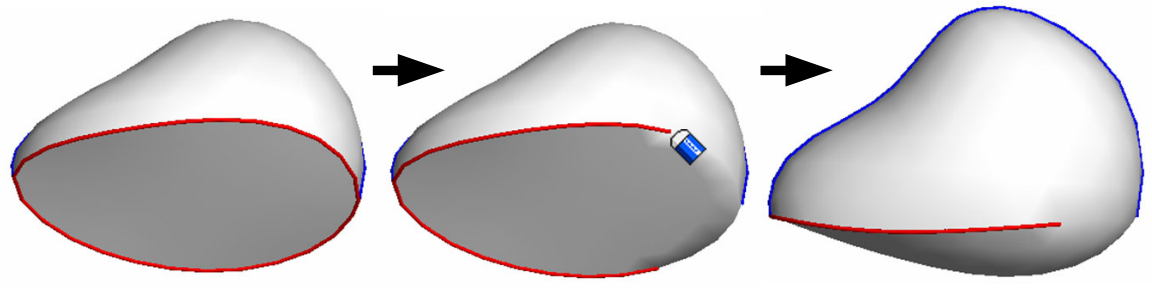


Figure 3.7: Erasing a control curve (left: before erasing, middle: immediately after erasing, right: after surface optimization).

is very important because the curves resulting from sketching can contain noise, and localized deformation can also introduce jaggy parts. It might be possible to automatically apply denoising after each user interaction, but it is not clear to which extent smoothing should be applied. Our rubbing tool provides an intuitive and convenient interface for specifying the target area to apply smoothing to, as well as the amount of smoothing. A similar, gesture-based smoothing operator for implicit surfaces is presented by Galyean and Hughes [GH91]. Our current implementation moves each vertex being rubbed one by one so that it locally improves inner and outer fairness.

3.2.4 Erasing tool and type change tool

The erasing tool works as a standard curve segment eraser: the user drags the cursor along a control curve to erase it. This is equivalent to removing constraints that define the surface. The system optimizes the surface when the user finishes an erasing operation (releases the mouse button, Figure 3.7). The type change tool is for changing the type of a control curve. Like the erasing tool, the user drags the cursor along a curve to change the property. If the curve is a sharp curve, it converts it to a smooth curve (or curve segment), and vice versa. As with the erasing tool, the system updates the surface geometry according to the property change and presents the result after the user finishes the operation (releases the mouse button, Figure 3.8).

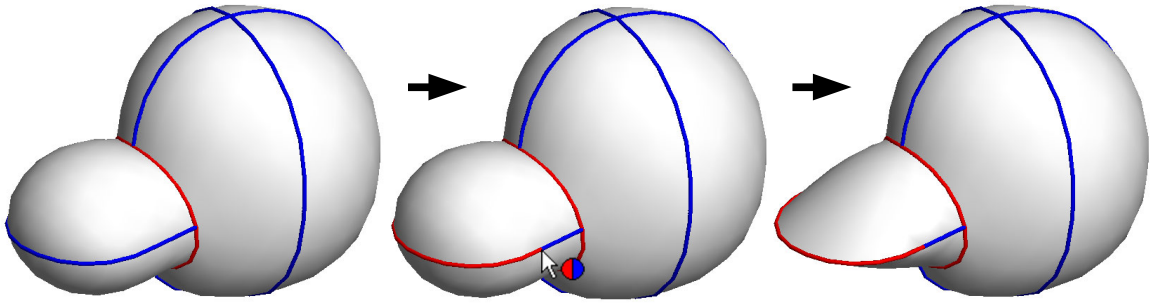


Figure 3.8: Changing the curve type (left: before the change, middle: immediately after the change, right: after surface optimization).

3.3 Algorithm

To implement the described interface we propose an algorithm that consists of two main steps: curve deformation and surface optimization. The additional steps, mesh construction and remeshing (Section 3.3.3), only occur at the end of the modeling operations creation, extrusion, cut, and deformation.

Instead of solving for both curve positions and fair surface simultaneously, we have found that decoupling the curve deformation from the surface optimization step is fast, intuitive, produces aesthetically pleasing results, and supports our fundamental principle of defining shape by control curves. The user first deforms (pulls) the curve(s) using the deformation tool (Section 3.3.1), after which the new curve positions are fed to the surface optimization step as positional constraints (Section 3.3.2). During curve pulling, these two operations are performed sequentially to achieve interactive updates of both the curves and the surface they define.

3.3.1 Curve deformation

The user interface for curve deformation is a usual direct manipulation method: the user grabs and drags a point on a curve, and the curve deforms smoothly within the peeled ROI. The current implementation always moves the grabbed point parallel to the screen.

The algorithm we use is a variant of detail-preserving deformation methods using differential coordinates [Sor06], combined with co-rotational methods [Fel07]. Geometry is represented using differential coordinates, and the final result is obtained by solving a sequence of linear least-squares problems, subject to boundary (positional) constraints. The main challenge in this framework is the computation of appropriate rotations for the differential coordinates. One approach is to explicitly compute rotations beforehand, typically by smoothly interpolating the prescribed orientation constraints defined by the user [YZX+04, LSLCO05, ZHS+05, ZRKS05]. These methods are not appli-

cable in our setting because the user should only need to drag a vertex without specifying rotations. Another approach is to implicitly compute rotations as a linear combination of target vertex positions [SLCO+04, FAT07]. Our technique is similar to these methods, but we explicitly represent rotation matrices as separate free variables. This is due to the fact that neighboring vertices along a curve are nearly collinear and inappropriate for deriving rotations from them.

Conceptually, what we want to solve is the following error minimization problem

$$\arg\min_{\mathbf{v},\mathbf{R}} \Bigg\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \mathbf{R}_i\delta_i\|^2 + \sum_{i\in C_1} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{i,j\in E} \|\mathbf{R}_i - \mathbf{R}_j\|_F^2 + \sum_{i\in C_2} \|\mathbf{R}_i - \mathbf{R}_i'\|_F^2 \Bigg\}, \tag{3.1}$$

where $\mathbf{L}(\cdot)$ is the differential operator, $\mathbf{v}_i$ represents the vertex coordinates, $\mathbf{R}_i$ represents rotations associated with these vertices in the deformed curve, $\|\cdot\|_F$ is the Frobenius norm, E is the set of curve edges, C_1 and C_2 are the sets of constrained vertices, and primed values are given constraints. The first term minimizes the difference between the resulting differential coordinates and the rotated original differential coordinates δ_i. The second term represents positional constraints (we use three constrains: two at the boundary of the ROI and one at the handle). The third term ensures that the rotations are smoothly varying along the curve [ACP03, SP04, FAT07], and the last term represents rotational constraints (we use two constraints at the boundary of the ROI). These four terms also need to be appropriately weighted to obtain visually pleasing results. We have omitted these weights in the above equation for simplicity.

A problem with this approach is that $\mathbf{R}$ is not linear. Unconstrained transformation includes shearing, stretching, and scaling, which is undesirable for our application. Similar to [SLCO+04], we therefore use a linearized rotation matrix to represent small rotations. In order to accommodate large rotations, we iteratively compute the gross rotation by concatenating small delta rotations obtained by solving a linear system at each step.

In summary, what we solve in each step is the following minimization problem

$$\arg\min_{\mathbf{v},\mathbf{r}} \Bigg\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \mathbf{r}_i\mathbf{R}_i\delta_i\|^2 + \sum_{i\in C_1} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{i,j\in E} \|\mathbf{r}_i\mathbf{R}_i - \mathbf{r}_j\mathbf{R}_j\|_F^2 + \sum_{i\in C_2} \|\mathbf{r}_i\mathbf{R}_i - \mathbf{R}_i'\|_F^2 \Bigg\}, \tag{3.2}$$

where $\mathbf{R}_i$ is the gross rotation obtained from the previous iteration step and fixed in each minimization step. $\mathbf{r}_i$ is a linearized incremental rotation represented as a skew symmetric matrix with three unknowns

$$\mathbf{r}_i = \begin{bmatrix} 1 & -r_{iz} & r_{iy} \\ r_{iz} & 1 & -r_{ix} \\ -r_{iy} & r_{ix} & 1 \end{bmatrix}.$$

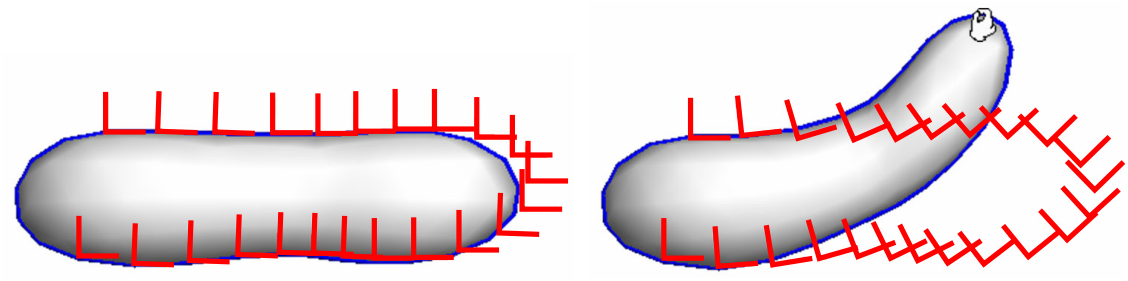


Figure 3.9: Rotated local coordinate frames (red) after curve deformation by pulling a single vertex.

As a whole, this minimization problem amounts to the solution of a sparse linear system and it returns optimal vertex positions and delta rotations $\mathbf{r}_i$. We update target gross rotations as $\mathbf{R}_i \leftarrow \mathbf{r}_i\mathbf{R}_i$, and also orthonormalize them using polar decomposition [FAT07]. Figure 3.9 shows the resulting gross rotations obtained using three iterations of this algorithm.

One remaining issue is the choice of differential coordinates L. We have tested two options: first order differentials (L_0) and second order differentials (L_1)

$$L_0 = \mathbf{v}_i - \mathbf{v}_{i-1}, \qquad L_1 = \mathbf{v}_i - \frac{1}{|N_i|}\sum_{j\in N_i}\mathbf{v}_j.$$

L_1 seems to be the popular choice for surface deformation. However in our case, we found that L_1 is not appropriate for the estimation of rotations because it almost always degenerates (i.e. is close to zero) in a smooth curve. On the other hand, L_0 always has certain length in an appropriately sampled curve and serves as a reliable guide for estimating rotations. One problem with L_0 based geometry computation is that it causes C^1 discontinuities on the boundaries of the ROI. Therefore, we first use L_0 for the iterative process of rotation estimation, and then switch to L_1 for computing the final vertex positions using the estimated rotations. This combination is a bit complicated, but gives the best results in our experiments.

Physically inspired methods, such as PriMo [BPG06], have become very popular in the context of surface modeling. For comparison and experimentation purposes we have implemented a variant of PriMo tailored to 3D curves; each curve segment defines a (3D) prism. While PriMo is an excellent choice for the simulation of physically plausible deformation, we found it to be unsuitable for our curve editing tool. When the curve is compressed it shows undesirable buckling, while when stretched it loses local detail. Both phenomena result from length preservation, inherent to all physically inspired curve deformation algorithms. In contrast, our uniform discretization of the Laplacian (L_1) tolerates some anisotropic scaling (see Figure 2.14 in Section 2.3). For a comparison of our method to PriMo, see Figure 3.10.

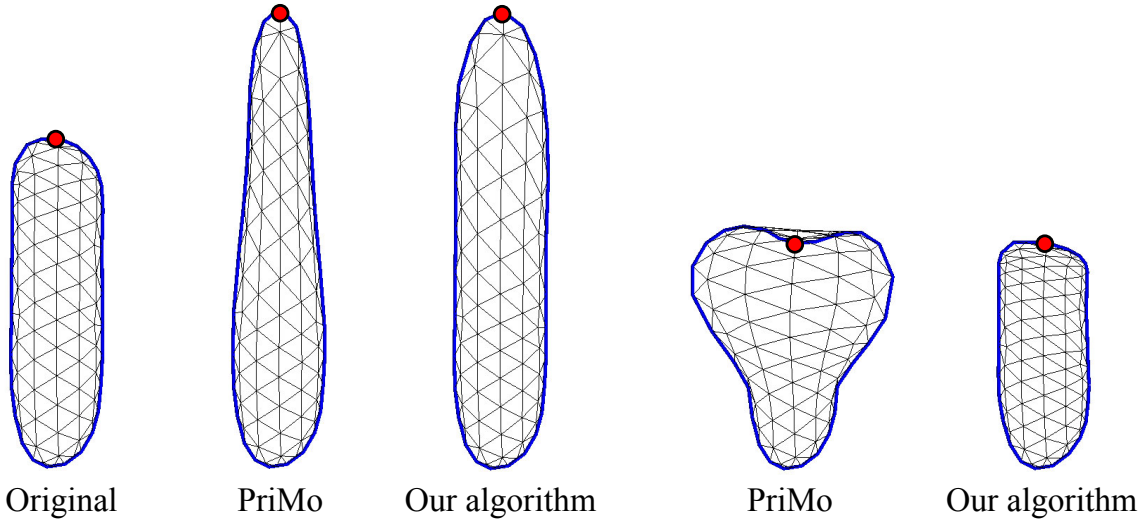


Figure 3.10: Comparison of our curve deformation to a physically based approach inspired by PriMo.

3.3.2 Surface optimization

It is important to provide real-time visual feedback to the user during control curve deformation. This constraint necessitates the use of a fast surface optimization algorithm. An intuitive choice appears to be some discrete surface defined as the solution of a sparse linear system [Sor06, BS07] (see Section 2.1.2 for an overview on linear variational methods). If we kept the system matrix constant during interaction, updating the positions would only require back-substitution, which is very fast. Unfortunately, in our setting we have encountered a shortcoming inherent to these algorithms, which is due to the absence of normal constraints along the curves. Specifically, if the positional constraints lie in a subspace, the solution will also be constrained to lie in this subspace. In our tool, the initially sketched curve is planar, so the resulting mesh geometry is also planar, see Figure 3.11. We prove this property for the constructions of [BK04a] and [SCO04] in Appendix A. Even in the presence of non-planar positional constraints, surfaces from [SCO04] or the linearized thin plate surfaces of [BK04a] seem to concentrate curvature near the curves, see Figure 3.12 (left column).

The problem could be solved by asking the user to specify normal constraints for the curves. In most mesh editing tools, normal constraints are implemented by fixing n-rings of adjacent vertices. We have not considered this as an option, since it is our strict design goal to keep the interface simple. Instead, we have chosen to implement a solution which generates a fair surface S that interpolates the control curves by means of nonlinear functional optimization, also known as (nonlinear) variational surface design. There are a variety of possible objective functions to choose from. Welch and Witkin [WW94] compute surfaces that minimize the

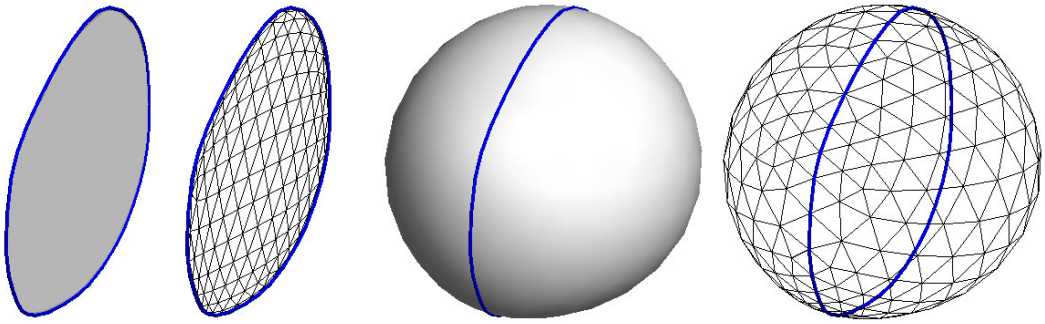


Figure 3.11: The results of least-squares meshes (left) and our non-linear solution (right) for a planar curve.

integral of squared principle curvatures κ_1 and κ_2

$$E_p = \int_S (\kappa_1^2 + \kappa_2^2)\, dA, \tag{3.3}$$

which is also known as thin-plate energy, while Bobenko and Schröder's [BS05] surfaces minimize the closely related Willmore energy

$$E_w = \int_S (\kappa_1 - \kappa_2)^2\, dA, \tag{3.4}$$

implemented as a flow. They use this flow for smoothing and hole filling. Moreton and Séquin's [MS92] surfaces minimize variation of curvature

$$E_c = \int_S \left(\frac{d\kappa_n}{d\hat{e}_1}\right)^2 + \left(\frac{d\kappa_n}{d\hat{e}_2}\right)^2 dA, \tag{3.5}$$

which is the integral of (squared) partial derivatives of normal curvature κ_n w.r.t. the directions $\hat{e}_1$, $\hat{e}_2$ of principal curvatures.

Each objective function has its own strengths and weaknesses, which also heavily depend on how it is implemented. Based on these previous results and our own experiences, we chose to compute a surface, which results from a sequence of optimization problems. This is inspired by a surface construction method presented by Schneider and Kobbelt [SK01]. The partial differential equation (PDE) governing fairness in their work is defined as $\Delta_B H = 0$, where Δ_B is the discrete Laplace-Beltrami operator, and $H = (\kappa_1 + \kappa_2)/2$ is the mean curvature. Their basic idea is to factorize this fourth order problem into two second order problems and solve them sequentially. First they compute target mean curvatures (scalars) that smoothly interpolate the curvatures specified at the boundary, and then move the vertices, one vertex at a time, to satisfy the target curvatures.

However, the second stage of their technique is not fast enough to provide interactive updates of the geometry when the user pulls the curve. In addition, we are lacking curvature information at the boundaries. Our idea for a faster computation,

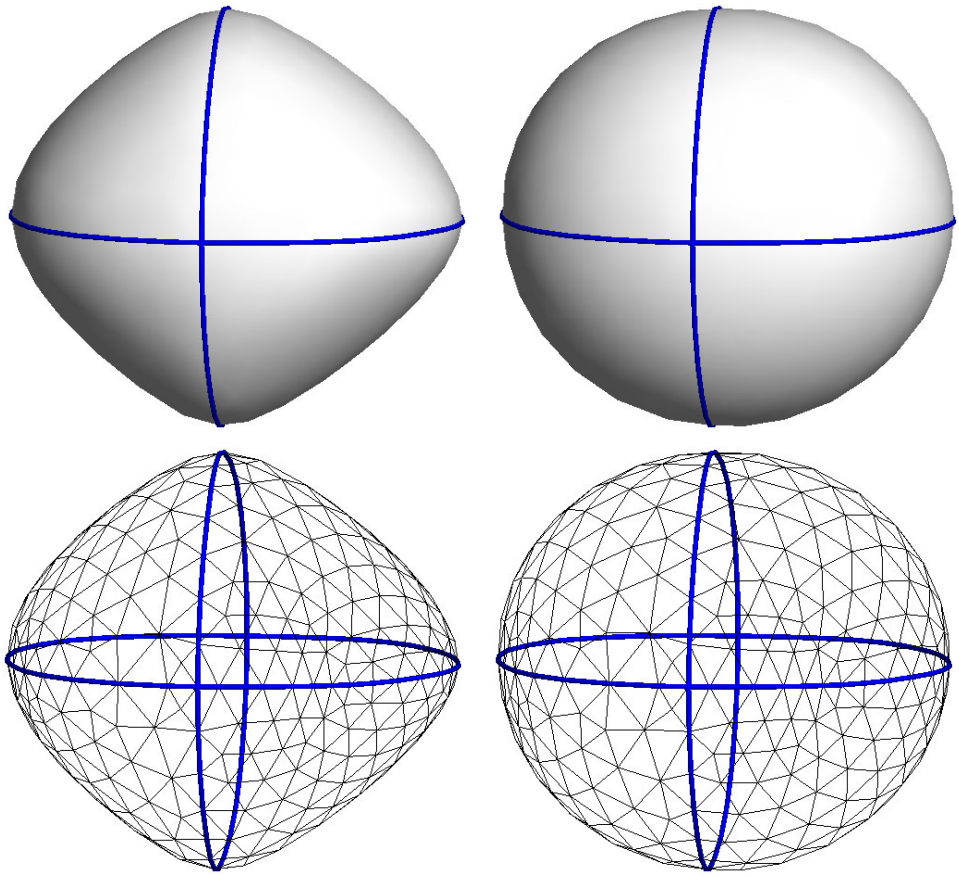


Figure 3.12: Least squares mesh (= linearized thin plate surface $\Delta^2\mathbf{x} = 0$, left) and the results of our nonlinear solution (right).

is to cast both second order problems as sparse linear systems that use a constant system matrix. This allows factoring the matrices once and then performing only back-substitution during the iterations.

In particular, in the first second-order system we replace the geometry dependent Laplace-Beltrami operator by the uniformly discretized Laplace operator and solve the following least-squares minimization problem

$$\arg\min_{c} \left\{ \sum_i \|\mathbf{L}(c_i)\|^2 + \sum_i \|c_i - c_i'\|^2 \right\}, \tag{3.6}$$

where $\mathbf{L}(\cdot)$ denotes the discrete graph Laplacian, to obtain a set of smoothly varying Laplacian magnitudes (LMs) $\{c_i\}$, which approximate scalar mean curvature values. The first term requires that the neighboring LMs vary smoothly and the second term requires the LMs at all vertices to be near the current LM c_i'. In the first iteration we set target LMs only for the constrained curves using the scalar curvatures along these curves. Unlike [SK01], where the curvature is fixed at the boundary, these initial target LMs are likely to change in subsequent iterations.

To obtain a geometry that satisfies these target LMs we use the uniformly discretized Laplacian as an estimator of the *integrated* mean curvature normal [WBH+07]. The integrated target Laplacian $\delta_i = A_i \cdot c_i \cdot \mathbf{n}_i$ per vertex is given as the product of an area estimate A_i for vertex i, the target LM c_i and an estimate of the normal $\mathbf{n}_i$ from the current face normals (see Section 2.3.2). Then new positions could then be computed by solving the following global least-squares system

$$\arg\min_{\mathbf{v}} \left\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \delta_i\|^2 + \sum_{i \in C} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 \right\}, \tag{3.7}$$

where the first term requires that the vertex Laplacians are close to the integrated target Laplacians, and the second term places positional constraints on all vertices in the control curve set C.

However, our assumption that the uniformly discretized Laplacian is a reasonable estimate for the integrated mean curvature normal does not hold when the edges around a vertex are not of equal length. Rather than using a geometry dependent discretization, which would require recomputation of the system matrix in each iteration, we try to achieve equal edge lengths by prescribing target edge vectors. For this, we first compute desired scalar edge lengths, similar to the computation of desired target LMs, by solving

$$\arg\min_{e} \left\{ \sum_i \|\mathbf{L}(e_i)\|^2 + \sum_i \|e_i - e_i'\|^2 \right\}, \tag{3.8}$$

for a smooth set $\{e_i\}$ of target average edge lengths, from the current set of the average lengths e_i' of edges incident on vertex i. Again, we start the iterations by using only the edge lengths along the given boundary curve. Note that the matrix for this linear system is identical to the system for computing target LMs, so that we can re-use the factored matrix.

From these target average edge lengths, we derive target edge 3-vectors η_{ij} for a subset B of the edges in the mesh

$$\eta_{ij} = (e_i + e_j)/2 \cdot (\mathbf{v}_i - \mathbf{v}_j)/\|\mathbf{v}_i - \mathbf{v}_j\|. \tag{3.9}$$

Using this set of target edge vectors, we modify the linear system in Equation 3.7 to derive the updated vertex positions as follows:

$$\arg\min_{\mathbf{v}} \left\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \delta_i\|^2 + \sum_{i \in C} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{(i,j) \in B} \|\mathbf{v}_i - \mathbf{v}_j - \eta_{ij}\|^2 \right\}. \tag{3.10}$$

We have found that it is sufficient to only constrain edges incident to the constrained curves, because setting the uniformly discretized Laplacian equal to vectors in normal direction automatically improves inner fairness at all free vertices (for a detailed description, see Chapter 6).

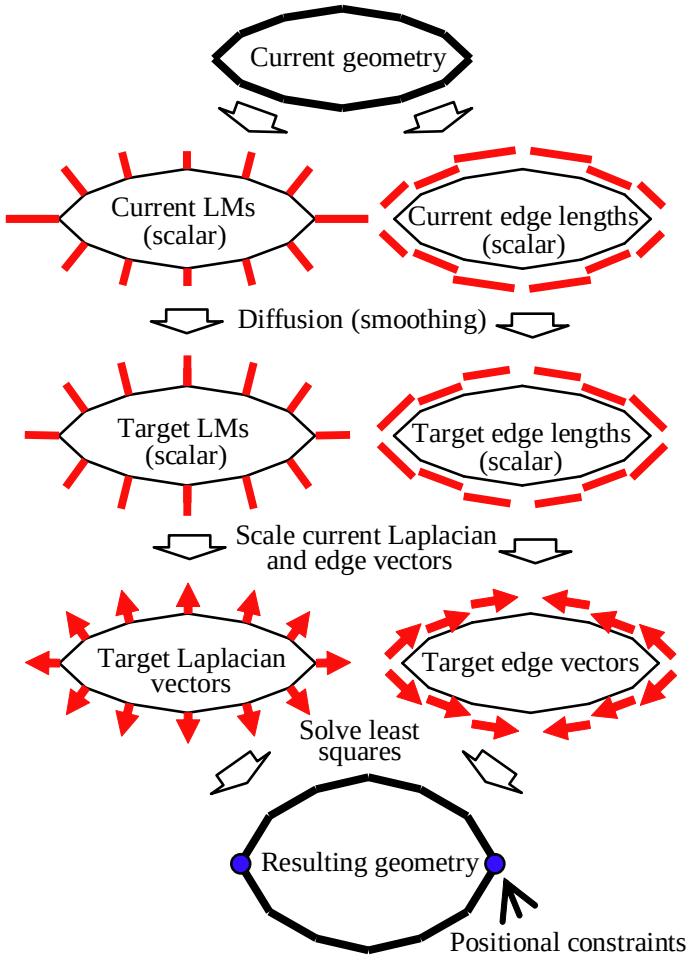


Figure 3.13: A single iteration of our surface optimization algorithm.

The two-step process, consisting of solving for target LMs and edge lengths and then updating the positions, is repeated until convergence. In practice, we observed that the computation converges rather quickly, in approximately 5 to 10 iterations. The system needs to repeatedly solve a few sparse linear systems, but the expensive matrix factorizations are required only once at the beginning (because left-hand side matrices remain unchanged during iteration). The system only needs to run back-substitutions during the iterations, which is very fast. See Figure 3.13 for an overview of one iteration.

While the algorithm described above is stable and robust, it is not entirely independent of tessellation, since we use the uniformly weighted graph Laplacian as an approximation of the integrated mean curvature normal to avoid matrix factorization in every iteration. To overcome this, we could compute a minimal energy surface as Schneider and Kobbelt [SK01] propose. As an experiment with a non-linear solution that is independent of surface tessellation, we have implemented the inexact Newton method for Willmore flow described in [WBH+07].

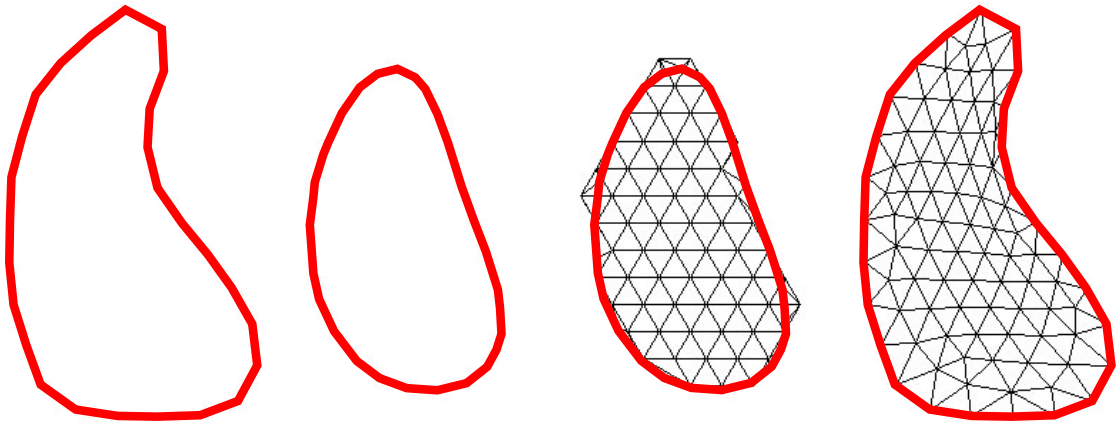


Figure 3.14: Initial mesh generation. The sketch curve (left) is resampled, smoothed (2nd from left) and intersected with a regular triangular grid (3rd from left), resulting in the mesh topology used for surface optimization (right).

We have found that our algorithm tends to generate very similar results if the discretization is near-regular and that, as expected, there are situations where unequal edge lengths along the fixed boundaries would benefit from the discretization-independent solution. However, not only are these techniques significantly slower to an extent that makes them unsuitable for most interactive editing situations, we have also encountered that the solution can become unstable when using insufficient boundary constraints, that is, curves without normals (this is expected and mentioned in [WBH+07]).

3.3.3 Meshing and re-meshing implementations

The system generates a new mesh after the creation, cut, and extrusion operations. In the case of cut, the system flattens the intersection (it is always developable) and generates a 2D mesh inside of it. In the cases of creation and extrusion, the system generates a 2D mesh on the image plane within the region surrounded by the input stroke. The system first resamples the input stroke and then smoothes it by moving each vertex to the mid point of adjacent vertices. It is possible to skip this process, but the resulting mesh is nicer for our purpose because it ends up generating more triangles in high curvature areas. The resampled stroke is intersected with a regular triangular grid mesh, and each point of the resampled stroke is connected to the nearest grid vertex. Both front and back sides are created from the same 2D mesh and stitched together at the common boundary (Figure 3.14). Note that this merely defines the mesh connectivity, not the actual geometry, which is computed subsequently as described in the previous section.

As the geometry is changed by the user, it may become necessary to remesh

the surface. One possible approach is to interleave mesh topology updates and vertex position updates as in [WW94]. Unfortunately, our surface optimization algorithm heavily relies on pre-factorization of the Laplacian matrix, which is defined by mesh topology. If the mesh topology changes during optimization, the factorization must be computed again, which is a significant overhead. Therefore, we apply remeshing only when a large change occurs, and use a constant topology mesh during (continuous) deformation and surface optimization to provide real-time feedback. Specifically, we apply remeshing after each sketch-based modeling operation, and when the user releases the mouse button after a deformation (pulling) operation. We use a modified version of explicit remeshing [SG03] in our system.

3.4 Results

Figure 3.15 shows shapes that are difficult to model with implicit representations [TO02, KHR02, SWSJ05]. CSG operations allow the user to represent *closed* sharp curves along a boundary [SWSJ05], but it is problematic to represent an *open* sharp curve starting in the middle of a smooth surface. It is also difficult to represent *point sharp* (e.g., the tip of a cone) using the standard implicit representation. Both can be modeled with our system (Figure 3.15). Markosian et al. [MCCH99] show that it is possible to include some creases on the surface by tracing the implicit surface with an explicit mesh, but the basic geometry is defined by a user-defined polygonal skeleton, not by surface curves.

Figures 3.16 and 3.17 show some more complex results obtained with our modeling tool. While the models shown in Figure 3.1 each took a trained user approximately 5-10 minutes to create, those depicted in Figures 3.16 and 3.17 took between 10 minutes (arm) and 1 hour (torso). See the accompanying video for more details on the construction process.

We have conducted an informal user study to test FIBERMESH. We trained first-time novice users for approximately 10-15 minutes, and then let them create some models (Figure 3.18). We also asked a professional 2D animation artist to

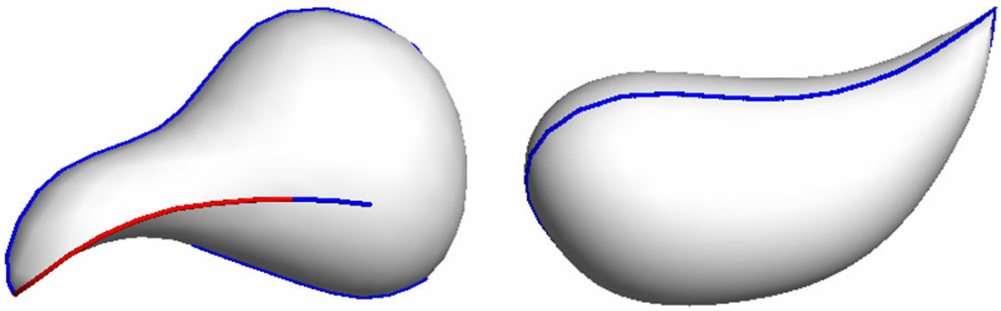


Figure 3.15: Open sharp curve (left) and point-sharp curve (right).

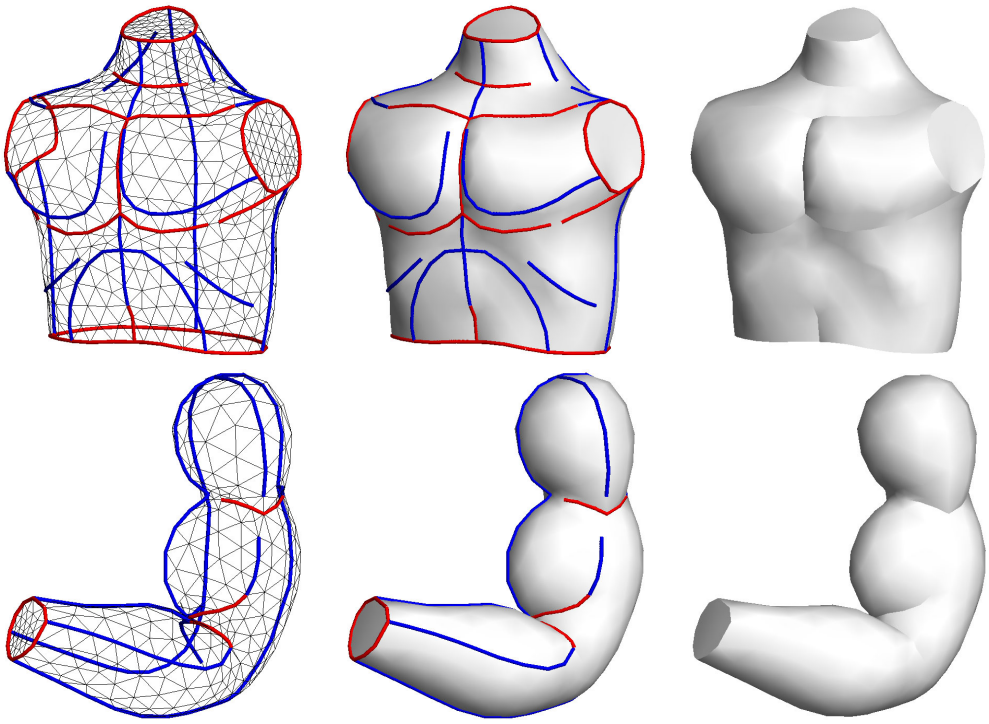


Figure 3.16: Some results obtained using FIBERMESH.

evaluate our system (Figure 3.19). To quote the artist:

> "One great thing about this system is that one can start doodling without having a specific goal in mind, as if doodling on paper. One can just create something by drawing a stroke, and then gradually deform it guided by serendipity, which is very important for creative work. Traditional modeling systems (parametric patches and subdivision surfaces) require a specific goal and careful planning before starting to work on the model, which can hinder the creative process."

Furthermore, we have learned that (a) FIBERMESH indeed supports the skill transfer from traditional 2D sketching to 3D modeling, (b) while the system does require some practice, the amount is reasonable and acceptable and (c) creating separate models first and then merging, as well as animation tools would be very useful.

Our current implementation is written in Java running on the Windows platform. Mesh processing routines are written in Java, but sparse matrix solvers are written in native code (linked via JNI). We are testing the system on an Intel Pentium M 1GHz machine, where it runs in interactive rates. Factorization takes less than a second, and interactive curve deformation (including surface optimization) works in 10-15 fps in most of our examples (600-2000 vertices). We currently process the entire mesh as a single system throughout the deformation, which causes some

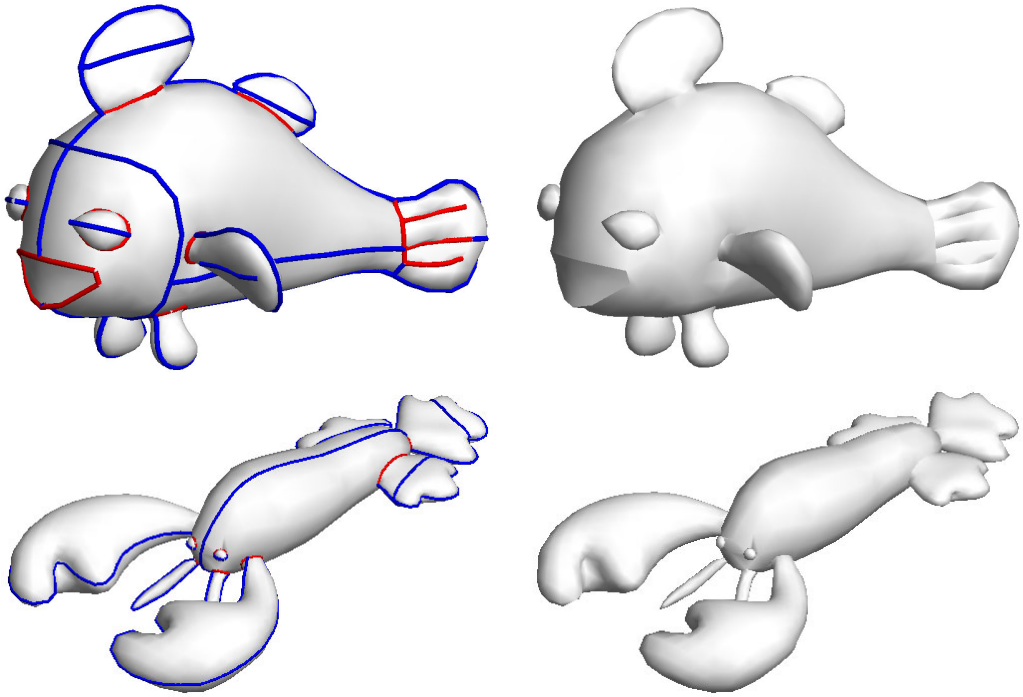


Figure 3.17: Some *fishy* results obtained with the FIBERMESH tool.

slowdown when the model becomes complicated. Note though, that it is straightforward to handle larger meshes by editing only a subset of the mesh, while fixing the rest.

3.5 Discussion

Our current implementation uses a curve only as a series of positional constraints. However, we can expect that curves have more information. For example, when an artist defines a shape with curves, it is often the case that these curves indicate the principal curvature direction of the surface. It is also natural to expect that the character lines form curvature extrema. It might be possible to obtain better (more intuitive and aesthetically pleasing) surfaces by taking these issues into account during optimization. One interesting direction to explore would be to create a quad mesh that follows the direction of the curves. Quad meshes naturally represent principal curvature directions and would make it possible to handle minimum and maximum principal curvatures separately. Quad meshes are also desirable when the user wants to export the resulting model from our system and continue editing it in a standard modeling package.

Multi-resolution (hierarchical) structure would be necessary to construct more complicated models than those shown in this chapter. Our current implementation

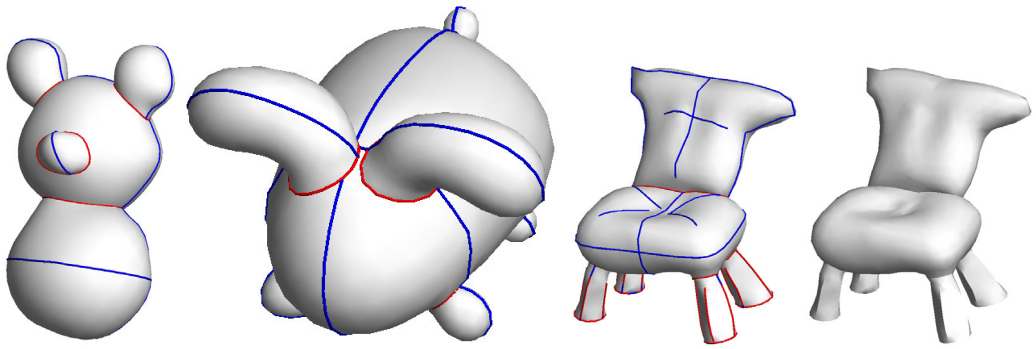


Figure 3.18: Results obtained from first-time novice users. Model creation took 10, 10 and 20 minutes, respectively.

can successfully handle individual body parts such as torso, finger, and face, but the construction of an entire body consisting of these parts would require some mechanism to handle the part hierarchy. One interesting approach would be to allow the user to add a "detailed mesh" on top of a "base mesh" as in multi-resolution approaches. Traditional multi-resolution meshes require fixed mesh topology, but our optimization framework might be able to introduce a topologically more flexible structure.

In a similar vein, we exclusively focused on surface-based control (curves on the surface) in this work. However, in practical modeling purposes, a skeleton based approach might be better in some cases, such as a modeling of simple tube-like arms and legs. Welch and Witkin [WW94] actually combined surface-based control and skeleton based control. It might be interesting to explore further into this direction, especially in the context of character animation.

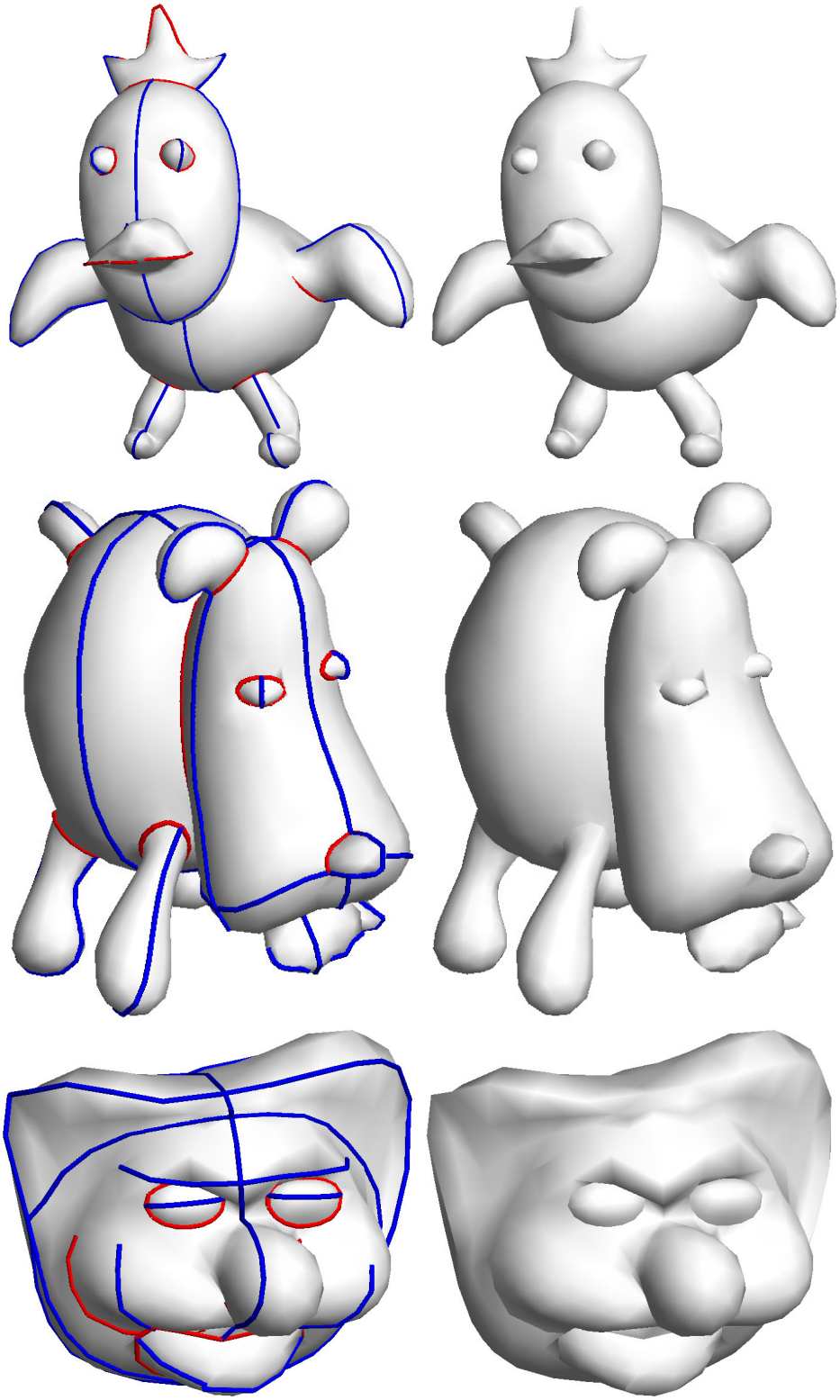


Figure 3.19: Creations from a professional 2D animation artist. Modeling took 10, 20 and 20 minutes, respectively.

Chapter 4

A Sketch-Based Interface for Detail-Preserving Mesh Editing

4.1 Introduction

A few strokes suffice to sketch the main features of a shape. This is why designers still prefer using pen and paper to invent and communicate, and explains the great success of sketch-based shape modeling approaches, such as SKETCH [ZHH96] and Teddy [IMT99]. In this work, we add to the existing toolbox of sketch based shape modeling techniques. Our contribution is a tool for sketching significant shape details on already existing coarse or detailed shapes. We believe the important first step of creating the basic shape from scratch is essentially solved: either based on sketching (apart from the pioneering works mentioned above, see also [CHZ00, KHR02, IH03, BCCD04] and our contribution in Chapter 3) or using other modeling techniques. Ideally, a sketch-based modeling system for 3D shapes should use the very same sketches that designers would draw on a piece of paper to convey the shape. What are these lines? As pointed out by Hoffman and Singh [HS97], the human visual system uses silhouettes as the first index into its memory of shapes, making everyday objects recognizable without color, shading or texture, but solely by their contours. In the area of non-photorealistic rendering (NPR), silhouettes have been used extensively [GG01] and recently they have been extended to *suggestive* contours: curves on the shape that might be silhouettes in nearby views [DFRS03]. The apparent presence of a feature line in a picture of a shape results from an abrupt change in illumination. Apart from view dependent features for which this happens or might happen in a nearby view, change of illumination generally correlates with curvature. Lines of curvature extrema (i.e. ridges and ravines) have, therefore, also been used in NPR for conveying shape.

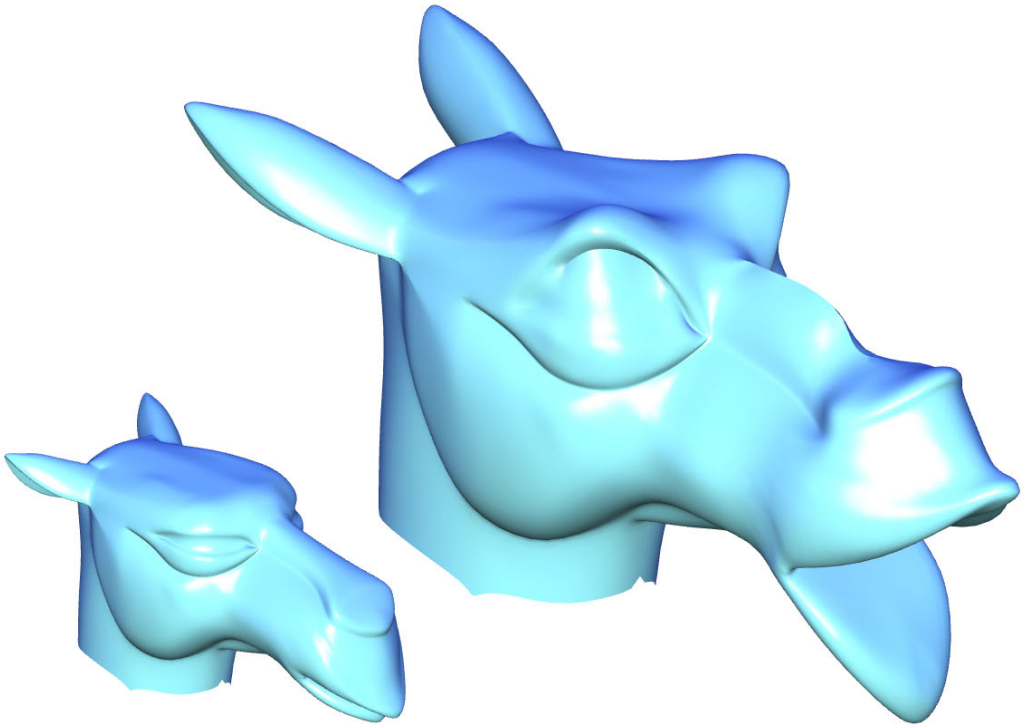


Figure 4.1: With a few strokes we have greatly increased the expressiveness of the CAMEL model (bottom left). See Figure 4.2 for details.

We come to the conclusion that **sketching** a shape is **inverse NPR**. Consequently, we design a sketch-based modeling interface using silhouettes and sketches as input, and producing contours, or suggestive contours, and ridges / ravines. The user can sketch a curve, and the system adapts the shape so that the sketch becomes a feature line on the model, while preserving global and local geometry as much as possible. As the requested properties of the sketch cannot or should not always be accommodated exactly, users only *suggest* feature lines.

It might seem obvious to let users alter contours, or ask for a line in space to be a feature line. Interestingly, our concurrent goals of preserving the global and local geometry during the edit while using feature lines for defining the edit are difficult to implement using traditional approaches: typical sketching tools [IMT99, KHR02, FRSS04] do use silhouettes, however, they create only smooth shapes. Some operations of sketching techniques might preserve geometric detail, however, they are not based on inserting feature lines into the shape [DE03, KG05]. In general modeling environments, such as space deformation techniques (e.g., [SP86, SF98]) and multi-resolution or subdivision mesh modeling approaches [ZSS97, KCVS98, BLZ00], it can be difficult to incorporate the displacement of a feature curve: these approaches provide a basis that spans a space of shapes; the requested displacement has to be translated into coefficients of this basis. In general, this might be impossible, and an approximate solution typically leads to a dif-

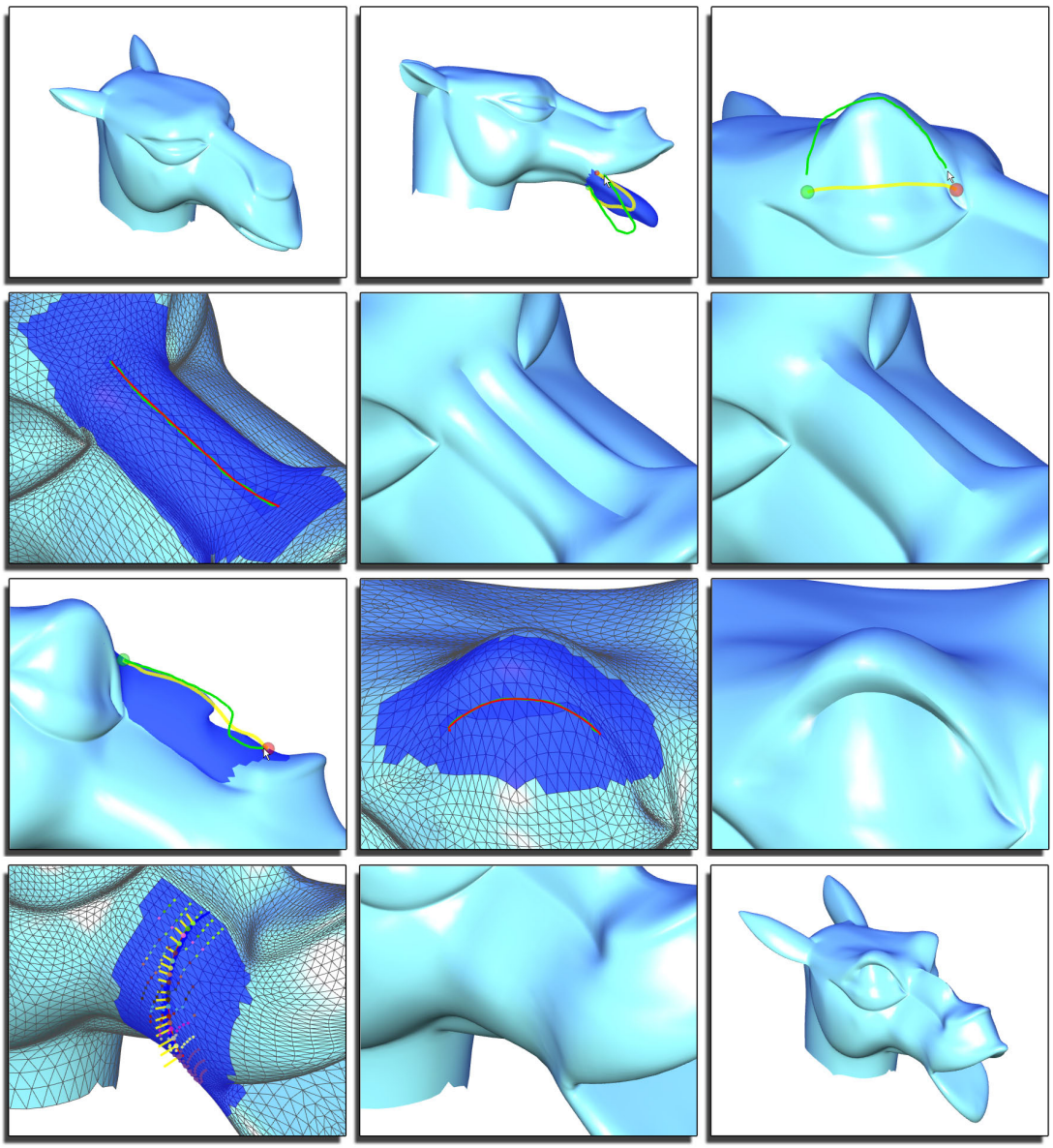


Figure 4.2: Our mesh editing tool in action. First row [(1)-(3)]: First, we open the mouth of the CAMEL model (1) by detecting an object silhouette, and sketching an approximation of the lip shape we want (2) (See Section 4.3). Note that in (2) the yellow curve is the original object silhouette, the green curve is the user sketch, and the dark blue region is the result of a previously placed sketch. By sketching directly onto the model (3) we produce a handle (yellow) by which we can lift the eyebrow with the green sketch. Second row [(4)-(6)]: For the creation of sharp features we sketch the feature line (4) and then scale the affected Laplacians to produce either a ravine (5) or a ridge (6) (See Section 4.4.2). Third row [(7)-(9)]: If we are not yet satisfied with the ridge in (6), we can edit the newly created object contour using our silhouette tool (7). Sketching a ravine under the eye by geometry adjustment (See Section 4.4.1) is shown in (8) and (9). Fourth row [(10)-(12)]: Finally, we sketch a subtle suggestive contour near the corner of the mouth in (10) and (11) (See Section 4.4.3), resulting in the SCREAMING CAMEL model (12), shown in Figure 4.1.

ficult inverse problem (see also Botsch and Kobbelt [BK04a]). Our idea becomes realizable through the recent trend to cast mesh modeling problems as discrete Laplace or Poisson models [Ale03, BK04a, SLCO+04, YZX+04, SP04, Sor06]. Within this framework, it is easy to displace a set of edges (e.g., sketch a new position of an identified contour) while preserving the geometric details of the surface as much as possible. However, most of the feature lines we want to use have specific differential properties, either absolute or relative to the viewing direction, and they might not coincide with edges on the mesh. We therefore extend the framework of Laplace/Poisson mesh modeling in the following ways: (a) we accommodate constraints on the normals and the curvature; (b) we allow constraints to be placed on virtual vertices (vertices placed on edges that only serve to implement the constraints but are never added to the mesh); (c) we incorporate a tangential mesh regularization which moves edges onto sharp features while ensuring well-shaped triangles.

This mesh modeling framework together with a user-interface mostly based on sketching suggested feature lines onto or around a shape, indeed, yields an intuitive shape modeling technique.

4.2 Mesh modeling framework

The basic idea of the modeling framework is to satisfy linear modeling constraints (exactly, or in the least squares sense), while preserving differential properties of the original geometry. This technique has recently been presented in various fashions and we only briefly explain the main concepts. For more detailed explanations see the references given below and Section 2.4. One way of deriving these linear constraints is to ask that the Laplacian of the original geometry be preserved in the least squares sense [Ale03, LSCOL04]. Let the mesh be represented as a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, consisting of vertices $\mathbf{V}$ and edges $\mathbf{E}$. Let $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$, $\mathbf{v}_i = (v_{i_x}, v_{i_y}, v_{i_z}) \in \mathbb{R}^3$ be the original geometry and Δ the Laplace operator, then the deformed geometry $\mathbf{V}'$ is defined by the constrained minimization

$$\mathbf{V}' = \arg\min_{\mathbf{W}} \|\Delta\mathbf{V} - \Delta\mathbf{W}\|^2, \tag{4.1}$$

where the vertices might be weighted differently to trade-off between modeling constraints and the reproduction of original surface geometry. Note that this is equivalent to solving a linear system of the form $\mathbf{AV}' = \mathbf{b}$ in the least squares sense. If the original surface was a membrane, the necessary constraints for the minimizer lead to $\Delta^2\mathbf{V}' = 0$, which has been advocated by Botsch and Kobbelt [BK04a] in the context of modeling smooth surfaces. If, in contrast, the original surface contained some detail, the right-hand side is non-zero and we arrive at a variant of the discrete Poisson modeling approach of Yu et al. [YZX+04].

The modeling operation is typically localized on a part of the mesh. This part of the mesh is selected by the user as the region of interest (ROI) during the in-

teractive modeling session (with a lasso tool). The operations are restricted to this ROI, padded by several layers of anchor vertices. The anchor vertices yield positional constraints $\mathbf{v}'_i = \mathbf{v}_i$ in the system matrix $\mathbf{A}$, which ensure a gentle transition between the altered ROI and the fixed part of the mesh. Based on the constraints formulated so far, local surface detail is preserved if parts of the surface are translated, but changes with rotations and scales. One way of dealing with this is to define local rotations per vertex a priori. Lipman et al. [LSCOL04] compute these rotations from a smoothed solution of Eq. 4.1, Yu et al. [YZX+04] let the user specify a few constraint transformations and then interpolate them over the surface. However, we would like to incorporate the treatment of directions into the modeling phase so that some of the details have a fixed (normal) orientation, while others may rotate. Thus, we adopt the approach of Sorkine et al. [SLCO+04], who define the local rotations and scales by comparing one-rings between $\mathbf{V}$ and $\mathbf{V}'$. However, we discretize the Laplace operator using cotangent weights as recommended by Meyer et al. [MDSB03]. The conditions to be satisfied lead to an overdetermined system of linear equations of the form $\mathbf{AV}' = \mathbf{b}$, which we solve in the least squares sense according to the normal equations $\mathbf{A}^T\mathbf{AV}' = \mathbf{A}^T\mathbf{b}$. For information on how to derive the rows resulting from Eq. 4.1 see [SLCO+04] and Section 2.4.

We extend this framework towards constraints on arbitrary points on the mesh. Note that each point on the surface is the linear combination of two or three vertices. A point on an edge between vertices i and j is defined by one parameter as $(1-\lambda)\mathbf{v}_i + \lambda\mathbf{v}_j$, $0 \leq \lambda \leq 1$. Similarly, a point on a triangle is defined by two parameters. We can put positional constraints $\hat{\mathbf{v}}_{ij}$ on such a point by adding rows to the system matrix $\mathbf{A}$ of the form

$$(1-\lambda)v'_{i_x} + \lambda v'_{j_x} = \hat{v}_{ij_x}, \ldots . \tag{4.2}$$

Furthermore, we extend the framework by using other forms of differentials to achieve some additional effects. Let δ_i be the Laplacian of $\mathbf{v}_i$, the result of applying the discrete Laplace operator to $\mathbf{v}_i$, that is

$$\delta_i = \mathbf{v}_i - \sum_{\{i,j\}\in E} w_{ij}\mathbf{v}_j, \tag{4.3}$$

where $\sum_{\{i,j\}\in E} w_{ij} = 1$, and the weights w_{ij} are determined using the cotangent weights [MDSB03]. An important benefit of this weighting is that δ_i points in the normal direction, and the length $\|\delta_i\|$ is proportional to the mean curvature around vertex i. This allows us to prescribe a certain normal direction and/or curvature δ'_i for a vertex, simply by adding a row to $\mathbf{A}$ of the form

$$\mathbf{v}'_i - \sum_{\{i,j\}\in E} w_{ij}\mathbf{v}'_j = \delta'_i. \tag{4.4}$$

Setting the normal direction is necessary for contours and suggestive contours, setting the curvature – for ridges or ravines.

To access the tangential location of vertices, we use the uniform operator as a discrete Laplacian and relate it to the cotangent weighted Laplace operator. We exploit this for regularizing the mesh in tangential direction, by asking that

$$\mathbf{v}'_i - d_i^{-1} \sum_{\{i,j\}\in E} \mathbf{v}'_j = \mathbf{v}_i - \sum_{\{i,j\}\in E} w_{ij}\mathbf{v}_j, \tag{4.5}$$

where d_i is the degree of vertex i. The rationale behind this operation is this: the uniformly weighted operator generates a tangential component, while the cotangent weighting does not. Asking that they are equivalent is essentially solving the Laplace equation but only for the tangential components. The result is a mesh with well shaped triangles, preserving the original mean curvatures as long as the tangential offset is not too large. Note that we typically restrict this operation to small regions, so that large tangential drift cannot occur.

In the following sections, we explain how to use these basic building blocks for satisfying user-defined feature lines on a mesh.

4.3 Silhouette sketching

Our goal is to identify areas of the model that are easily recognized, and for which our memories hold vast databases of possible variations, and then apply these variations by sketching them. The idea is simple yet effective: after defining a region of interest on the surface and a camera viewpoint, we select (and trim) one of the resulting silhouettes, and then sketch a new shape for this silhouette (see Figure 4.3).

For the computation of silhouettes on polygonal meshes, various methods are available, see [Her99]. We have chosen to use object space silhouettes, and include the ability to switch between edge silhouettes (mesh edges, for which one adjacent face is front-facing and one is back-facing) and smooth surface silhouettes [HZ00]. Hertzmann and Zorin [HZ00] determine the silhouette on mesh edges $e = (\mathbf{v}_i, \mathbf{v}_j)$ by linearly interpolating corresponding vertex normals $\mathbf{n}_i, \mathbf{n}_j$: a silhouette point $\mathbf{p} = (1-\lambda)\mathbf{v}_i + \lambda\mathbf{v}_j$ on e has to satisfy $((1-\lambda)\mathbf{n}_i + \lambda\mathbf{n}_j)\cdot(\mathbf{p}-\mathbf{c}) = 0$, where $\mathbf{c}$ is the viewpoint. Silhouette points on edges are connected by segments over faces.

During editing, the user first picks one of the connected components, and then interactively adjusts the start and end point by dragging them with the mouse. Note that degenerate silhouette edge paths might lead to multiply connected curves, resulting in non-intuitive user interaction. Smooth silhouettes [Her99] remedy this problem on smoothly varying surfaces, and only for models with distinct sharp features (such as CAD models), mesh edges are used as silhouettes. In any case, the selected silhouette segment is represented as a set of points $\mathbf{q}_i$ on the mesh.

After selecting a silhouette segment, the user sketches a curve on the screen, representing the suggested new silhouette segment. The sketch is represented as

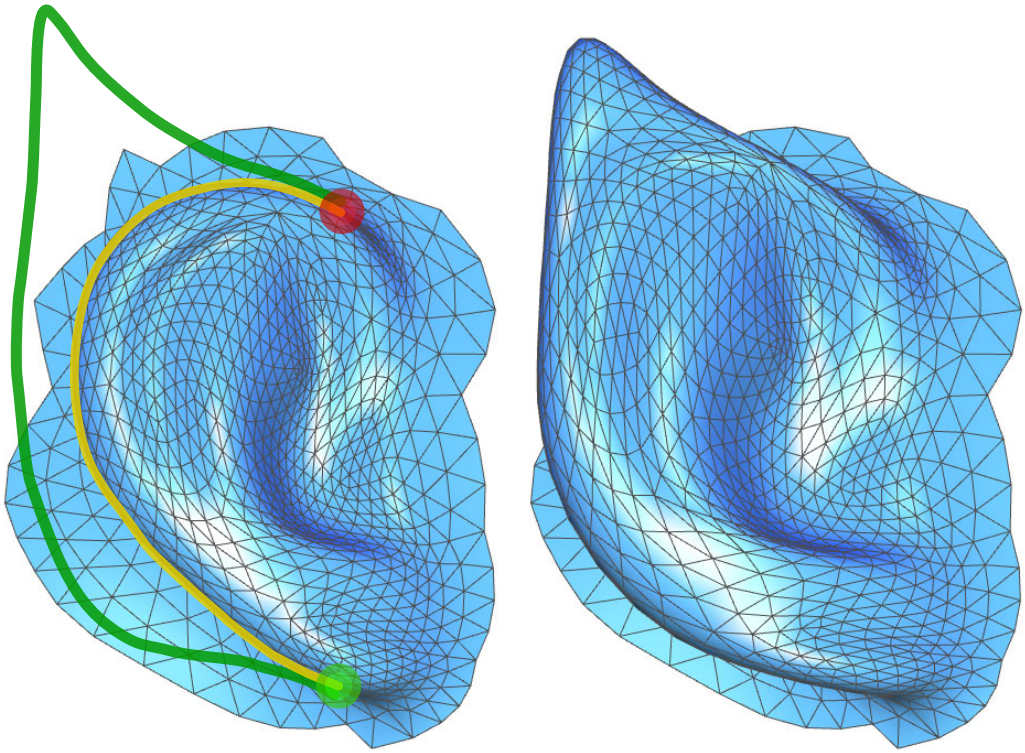


Figure 4.3: Sketching a very recognizable ear silhouette: we detect, select, crop and parameterize an object silhouette (yellow, the green and red balls represent begin and end vertices respectively), and then sketch a new desired silhouette (green).

a polyline in screen space. The vertex locations $\mathbf{s}_i$ on this polyline result in constraints on mesh vertices as follows: First, silhouette vertices $\mathbf{q}_i$ are transformed to screen space (the first two components contain screen space coordinates, while the third contains the z-value). Then, both curves are parameterized over $[0,1]$ based on edge lengths of the screen space polylines. This induces a mapping from $\mathbf{q}_i$ to $\{\mathbf{s}_j\}$, defining a new screen space position $\mathbf{q}'_i$ (note that $\mathbf{q}'_i$ retains the z-value of $\mathbf{q}_i$).

The new position $\mathbf{q}'_i$ in screen space is transformed back to model space and serves as a positional constraint. Note that when using smooth surface silhouettes, on-edge constraints have to be used (see Eq. 4.2). Additionally, varying the weighting of positional constraints along the silhouette against Laplacian constraints leads to a trade off between the accurate positioning of silhouette vertices under the sketch curve, and the preservation of surface details in the ROI. To achieve this, we simply multiply the affected rows in $\mathbf{A}$ and $\mathbf{b}$ with the selected weighting factor. For example, the result in Figure 4.3 follows the sketch closely, whereas the sketch in Figure 4.4 only hints at the desired lip position.

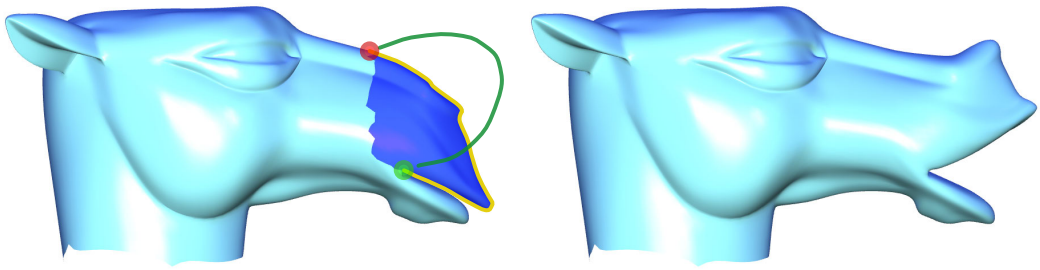


Figure 4.4: Sketching an *approximate* CAMEL lip by reducing the weights on the positional constraints for silhouette vertices.

This method works well even for moderately noisy and bumpy surfaces and preserves details nicely (see Figure 4.5). Note that for very noisy surfaces, object space silhouette paths and loops may become arbitrarily segmented, in which case our silhouette sketching method is no longer applicable. In such cases, sketch editing can be performed relative to any user-defined curve sketched manually onto the surface, as was done for lifting the eyebrows of the CAMEL, see Figure 4.2(3).

The matrix $\mathbf{A}^T\mathbf{A}$ is computed and factored once for each ROI and silhouette curve selection, and we simply solve for each sketch by back substitution [Tol03]. Some editing results in Figure 4.1 were obtained by using the silhouette editing capabilities of our system: sketching larger ears, opening the mouth and modifying the nose contour.

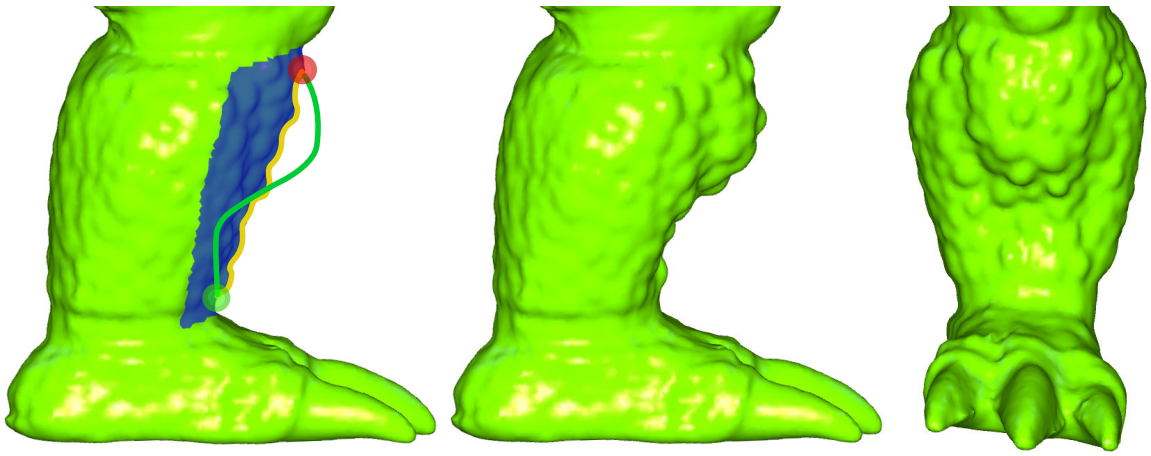


Figure 4.5: Editing the bumpy ARMADILLO leg: although the silhouette (yellow) in the ROI (blue) has substantial depth variation and the desired silhouette (green) is smooth, properly weighting the positional constraints retains the surface characteristics after the edit.

4.4 Feature and contour sketching

4.4.1 Geometry adjustment

Suppose we intend to create a potentially sharp feature where we have drawn our sketch *onto* the mesh. To create a meaningful feature (i.e. a ridge, ravine or crease) on a mesh, we must first adjust the mesh geometry to accommodate such a feature directly under the sketch, since in our setting the sketch need not run along an edge path of the mesh. To illustrate this, see Figure 4.6(a), where the sketch path $\{\mathbf{s}_i\}$ (green) follows the edges on the left, but runs perpendicular to them on the right. By applying repeated subdivision we could have locally adjusted the mesh resolution, but for situations similar to the one in Figure 4.6(a), many levels of subdivision would be necessary to properly approximate the sketch with an edge path. Another option would be to cut the mesh along the sketch; however, we have found a simpler method that avoids increasing the mesh complexity, yields nice feature lines and well-shaped triangles while retaining the original mesh topology. In detail:

1. The triangles in the ROI are transformed to screen space; triangles intersecting $\{\mathbf{s}_i\}$ are gathered (Figure 4.6(a), dark triangles) and the begin and end mesh vertices are identified.

2. An edge path $\mathbf{V}_p = (\mathbf{v}_{p_1}, \mathbf{v}_{p_2}, \ldots, \mathbf{v}_{p_n})$ that is close to $\{\mathbf{s}_i\}$ is computed by solving a weighted shortest path problem in the edge graph of the ROI. The weight for each edge is the sum of its vertices' screen space distance to $\{\mathbf{s}_i\}$. The resulting edge path vertices are generally not on, but close to $\{\mathbf{s}_i\}$ (shown in red in Figure 4.6(a)).

3. The path vertices $\mathbf{V}_p$ are mapped onto closest edges of the sketch path $\{\mathbf{s}_i\}$ in screen space; corresponding z-values are computed from restricting each vertex to move on its tangent plane, as defined by the original vertex normal (Figure 4.7, left). The resulting edge path closely follows the sketch curve (Figure 4.6(b)), yet may introduce badly shaped triangles.

We improve triangle shapes by relaxing vertices close to the sketch so that their uniform Laplacian equals the cotangent Laplacian in the least squares sense (See Figure 4.7, right, and Section 4.2). For the vertex relaxation we must solve a linear system, much like the actual editing solver, but with constraints given by Eq. 4.5. Obviously, the edge path vertices must remain under the sketch path during this procedure. To ensure this, while also giving the edge path vertices a valid degree of freedom, we add them as positional constraints (Section 4.2), and additionally add averaging constraints of the form

$$\mathbf{v}'_{p_i} - \frac{1}{2}\mathbf{v}'_{p_{i-1}} - \frac{1}{2}\mathbf{v}'_{p_{i+1}} = 0, \tag{4.6}$$

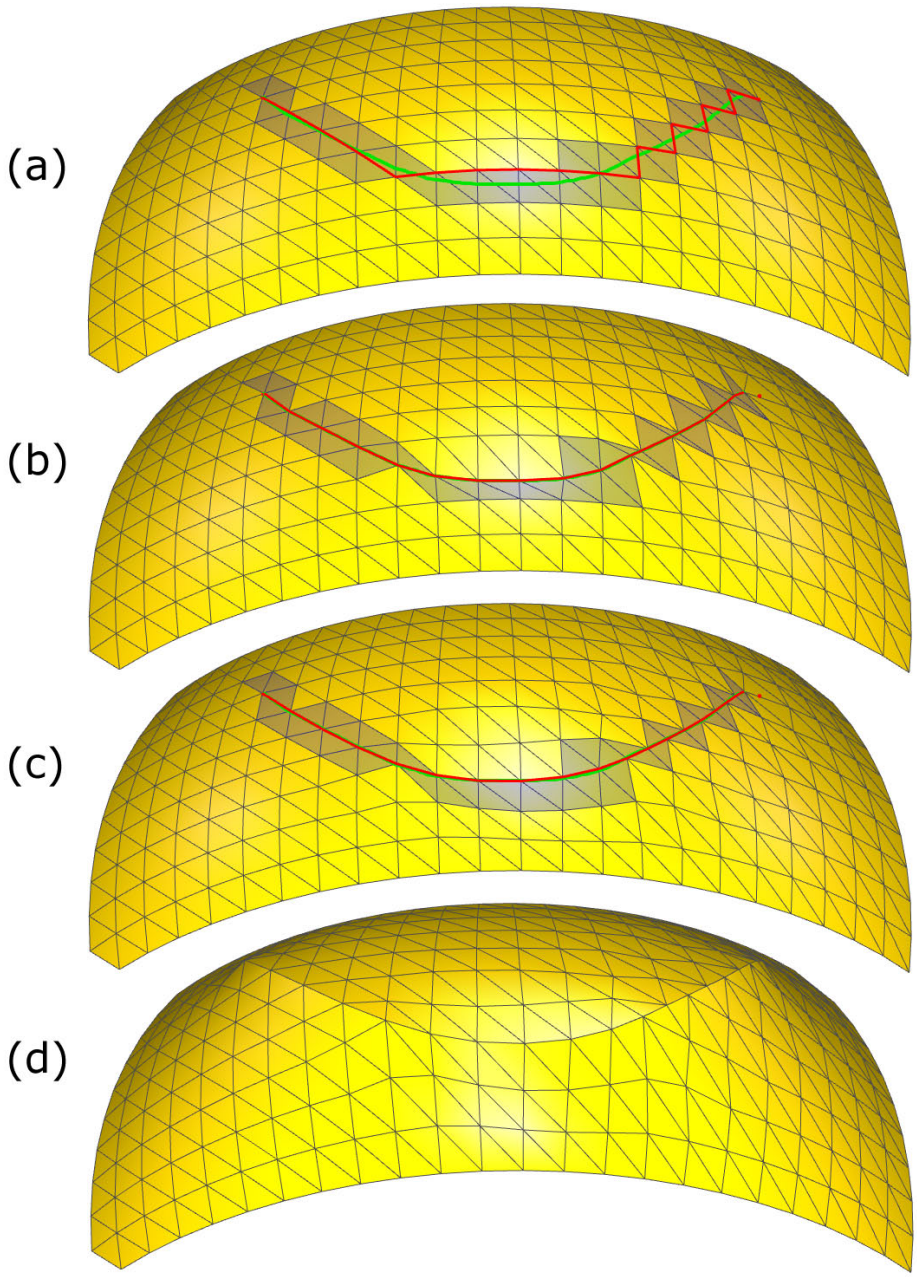


Figure 4.6: Creating a ravine-like crease: in (a) the green sketch given by the user is approximated by the red edge path on the original geometry. We adjust the geometry to lie directly under the sketch by orthogonal projection along the tangent plane (b), and then relax the area around the sketch (c). Now we can create the crease by scaling the Laplacians along the edge path (d), resulting in a sharp feature, even for this coarsely sampled surface.

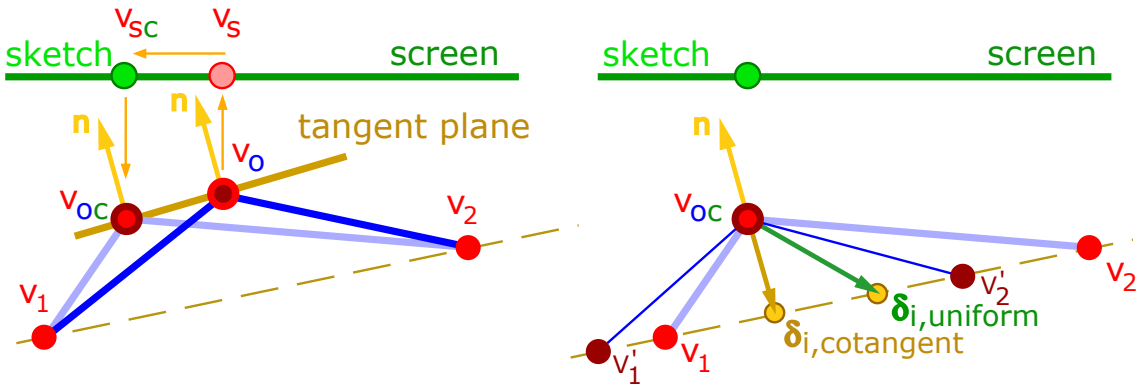


Figure 4.7: Adjusting edge path vertices to lie under the sketch curve (left): an object-space edge path vertex $\mathbf{v}_o$ is projected to $\mathbf{v}_s$ in screen space, from there orthogonally projected onto $\mathbf{v}_{sc}$ on the sketch curve, and then projected back onto the tangent plane defined by the normal at $\mathbf{v}_o$, yielding the new vertex position $\mathbf{v}_{oc}$. Relaxing the sketch region (right): to ensure a good triangulation after adjusting the geometry, we perform a relaxation of the edge-path vertices (allow them to move along the sketch path) and nearby vertices by constraining $\delta_{i,uniform}$ to $\delta_{i,cotangent}$ in the least squares sense. Qualitatively, this moves $\mathbf{v}_1$ and $\mathbf{v}_2$ to $\mathbf{v}_1'$ and $\mathbf{v}_2'$, while keeping $\mathbf{v}_{oc}$ under the sketch.

for all vertices in $\mathbf{V}_p$ excluding the begin and end vertices. The averaging constraint *loosens* the positional constraint, allowing edge path vertices to move between their adjacent vertices in the path. Adjusting the ratio of weights between positional and averaging constraints leads to a trade-off between accurately approximating the sketch, and some possibly desired path smoothing.

We have experienced no detrimental effects when applying this procedure on meshes that approximate the underlying smooth surface well, even in areas of high curvature. Also, small changes might be tolerable, as this region will be subsequently edited.

After the geometry adjustment step, the surface is prepared for editing operations in the vicinity of the sketch.

4.4.2 Sharp features

To create a sharp feature along the edge path, we adjust the Laplacians of path vertices when constructing the $\mathbf{A}$ matrix by prescribing the Laplacian transform for sketch vertices without flexibility to rotate or scale (i.e., as in Eq. 4.4). Since we discretize the Laplacian using the cotangent weights, we can simply scale the Laplacians of edge path vertices, resulting in a ridge or ravine, depending on the sign. If the Laplacian evaluates to zero, as is the case for flat surfaces, we instead

scale the surface normal and prescribe it as the new Laplacian. As described in Section 4.3, we factor the matrix $\mathbf{A}^T\mathbf{A}$ once we have selected a sketch, and can then quickly evaluate the results of varying scales by dragging the mouse up and down. The creation of a sharp ridge is shown in Figure 4.6(d). Alternatively, we can add some amount to the Laplacians, making the change absolute rather than relative. This works well in regions with high curvature variation along the sketch.

We have found it to be very convenient to create a ridge using our modeling framework, and thereafter treat it as a silhouette from a different camera position and edit it as outlined in Section 4.3. This technique was applied in the creation of the wavy ridge along the nose of the CAMEL model in Figures 4.1 and 4.2(7).

4.4.3 Smooth features and suggestive contours

Applying the editing metaphor described in the previous section can only create sharp features. To enable smooth features or suggestive contours, we need to in-

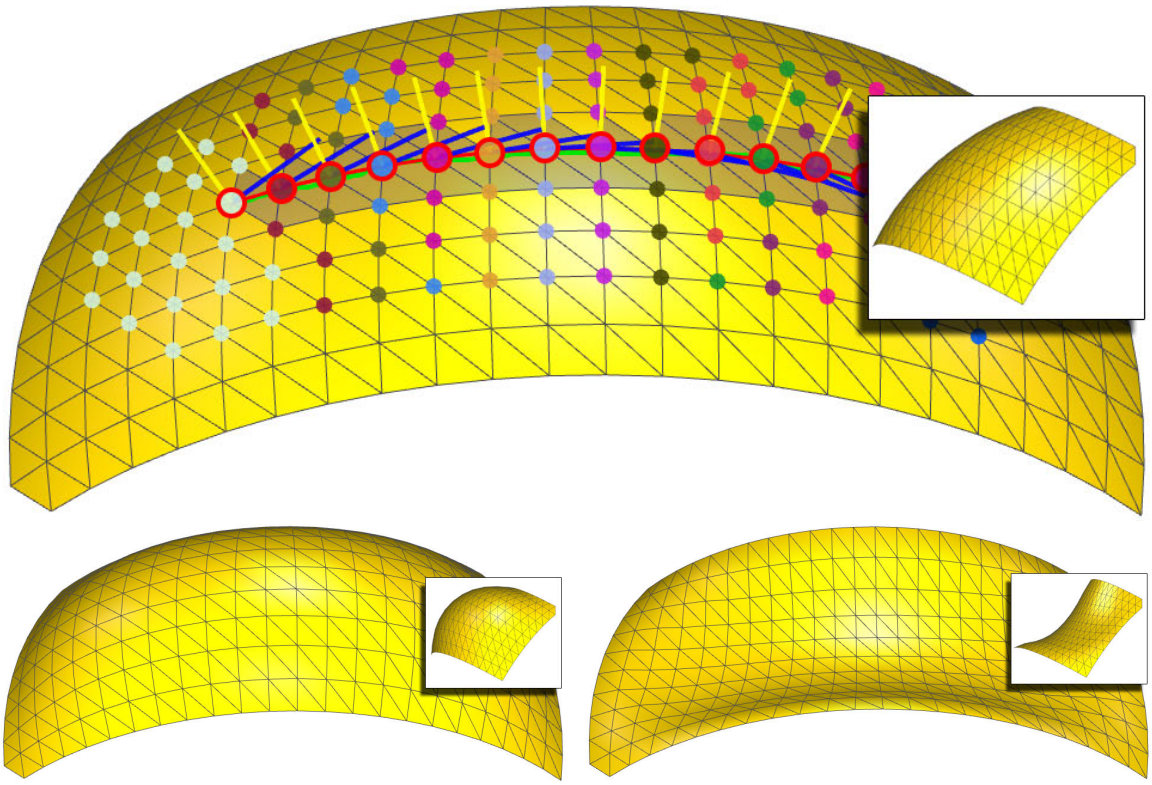


Figure 4.8: Top: view dependent vertex segmentation and rotation axis assignment. Bottom left: scaling all Laplacians in the sketch region by the same factor produces smooth ridges and ravines. Bottom right: rotating all Laplacians by an angle of $-\pi/2$ w.r.t. the blue rotation axes results in a suggestive contour.

fluence the Laplacians of more vertices than only those lying on the edge path. Additionally, for suggestive contours, we intend to manipulate curvature in the viewing direction. Thus, we need to rotate the Laplacians w.r.t. an axis that is orthogonal to both viewing and normal vectors. After performing the geometry adjustment of Section 4.4.1, given the viewing position $\mathbf{c}$, we gather and segment vertices within a user-defined sketch region around the edge path as follows (Figure 4.8, top):

- For each path vertex $\mathbf{v}_{p_i}$ with normal $\mathbf{n}_{p_i}$ (the yellow vectors in Figure 4.8) we compute the radial plane $\mathbf{r}_i$ that passes through $\mathbf{v}_{p_i}$ with plane normal $\mathbf{n}_{r_i} = (\mathbf{v}_{p_i} - \mathbf{c}) \times \mathbf{n}_{p_i}$ (the blue vectors in Figure 4.8). Now we can segment the vertices in the sketch region $\mathbf{V}_s = (\mathbf{v}_{s_1}, \mathbf{v}_{s_2}, \ldots, \mathbf{v}_{s_n})$ such that each sketch region vertex is associated with one such plane (ergo, each vertex in $\mathbf{V}_s$ *belongs* to one edge path vertex).
- Each vertex in $\mathbf{V}_s$ is assigned to the radial plane it is closest to, where the distance of $\mathbf{v}_{s_j}$ to plane $\mathbf{r}_i$ is measured as $d_j = orthodist(\mathbf{r}_i, \mathbf{v}_{s_j}) + dist(\mathbf{v}_{p_i}, \mathbf{v}_{s_j})$. Here, *orthodist* measures orthogonal distance to the plane, and *dist* is the Euclidean distance between $\mathbf{v}_{p_i}$ and $\mathbf{v}_{s_j}$. We take Euclidean distance into account to avoid problems that occur when two different path vertices have similar radial planes, and furthermore to limit the support of the sketch region.

In Figure 4.8 (top image), we show one such segmentation, where the edge path vertices are highlighted with red circles and the segmentation is color coded (i.e. all vertices of the same color are associated with the path vertex of that color).

Once we have this segmentation, one possible operation is to uniformly scale (or add to) the Laplacians of all sketch region vertices. Complementing the sharp features of Section 4.4.2, this operation gives us smooth bumps and valleys (Figure 4.8, bottom left). By setting the Laplacians to zero we can flatten specific regions of the mesh.

An alternative editing behavior results from rotating all Laplacians w.r.t. their respective rotation axes (given by above segmentation) by a user-defined angle, determined by dragging the mouse left or right. Note that rotation by π is identical to scaling by minus one. For angles in the ranges $[0, \pi)$ and $(\pi, 2\pi]$ we create varying radial curvature inflection points (Figure 4.8, bottom right), resulting in suggestive contours [DFRS03] such as the cheekbone shown in Figure 4.9. Note that these inflection points are not necessarily directly under the sketch, since they result from the Laplacian surface reconstruction and the boundary constraints around the ROI.

4.5 Discussion

Generating plausible and visually pleasing shapes and deformations is far from trivial: while our capability to derive a mental model from everyday shapes around

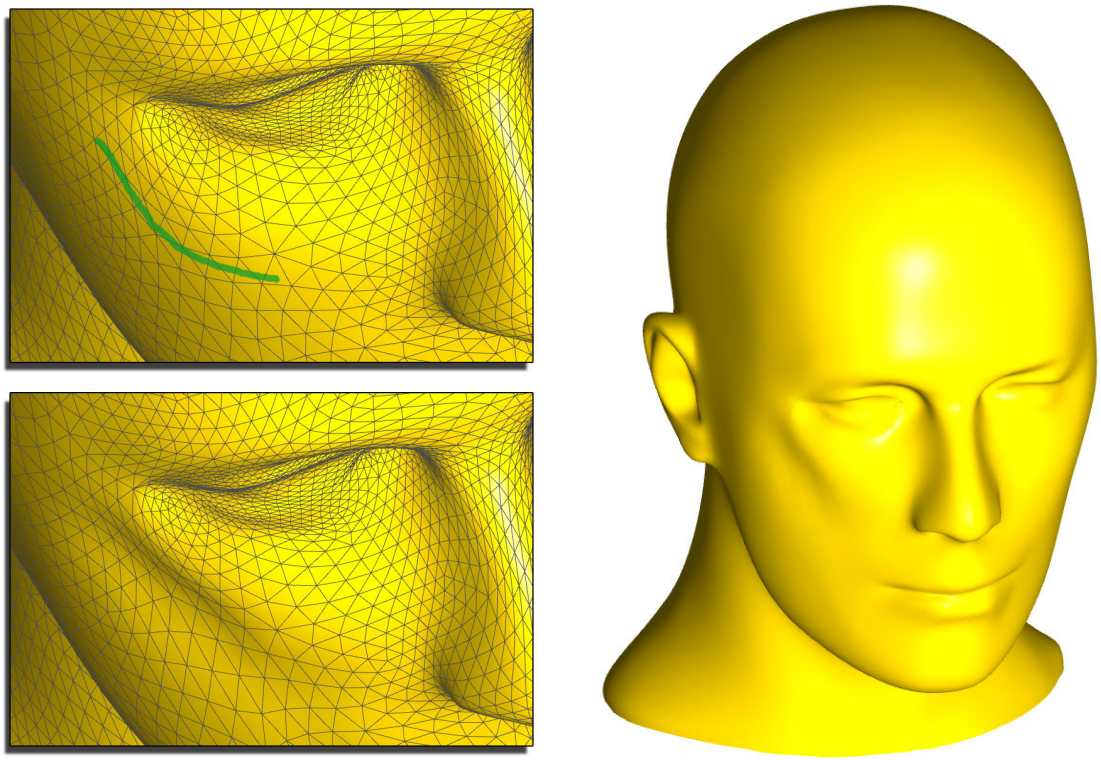


Figure 4.9: Adding a strong cheekbone to the MANNEQUIN model by sketching a suggestive contour.

us is well developed, we fail to properly communicate this to a machine. This is why we have to model in a loop, constantly correcting the improper interpretation of our intentions.

The quality of shape editing, therefore, depends on two factors: the time required by the system to update the shape after user commands and how well the shape change reflects our mental model of that process. The update time is a potential bottleneck in our approach, as the necessary matrix factorization and back substitution depend on the number of vertices and not the complexity of the edit operation. For example, ROI sizes of 5.5K/12K/33K vertices require 0.7/2.5/7.0 seconds for factorization and 0.035/0.07/0.25 seconds for back substitution on an Intel P4/2.0 GHz. On the other hand, we believe we have improved the match between the mental model and shape updates, though this is obviously hard to quantify.

From a user's point of view, our system is similar to other sketch-based editing interfaces [IMT99, KHR02, DE03, KG05], while it differs algorithmically: the above methods are based on space warps and variational implicits, whereas our representation is aimed at surface detail preservation. Our method inherits the simplicity of the user interface, and enables the creation of interesting and useful surface edits, both for inexperienced users and modeling professionals.

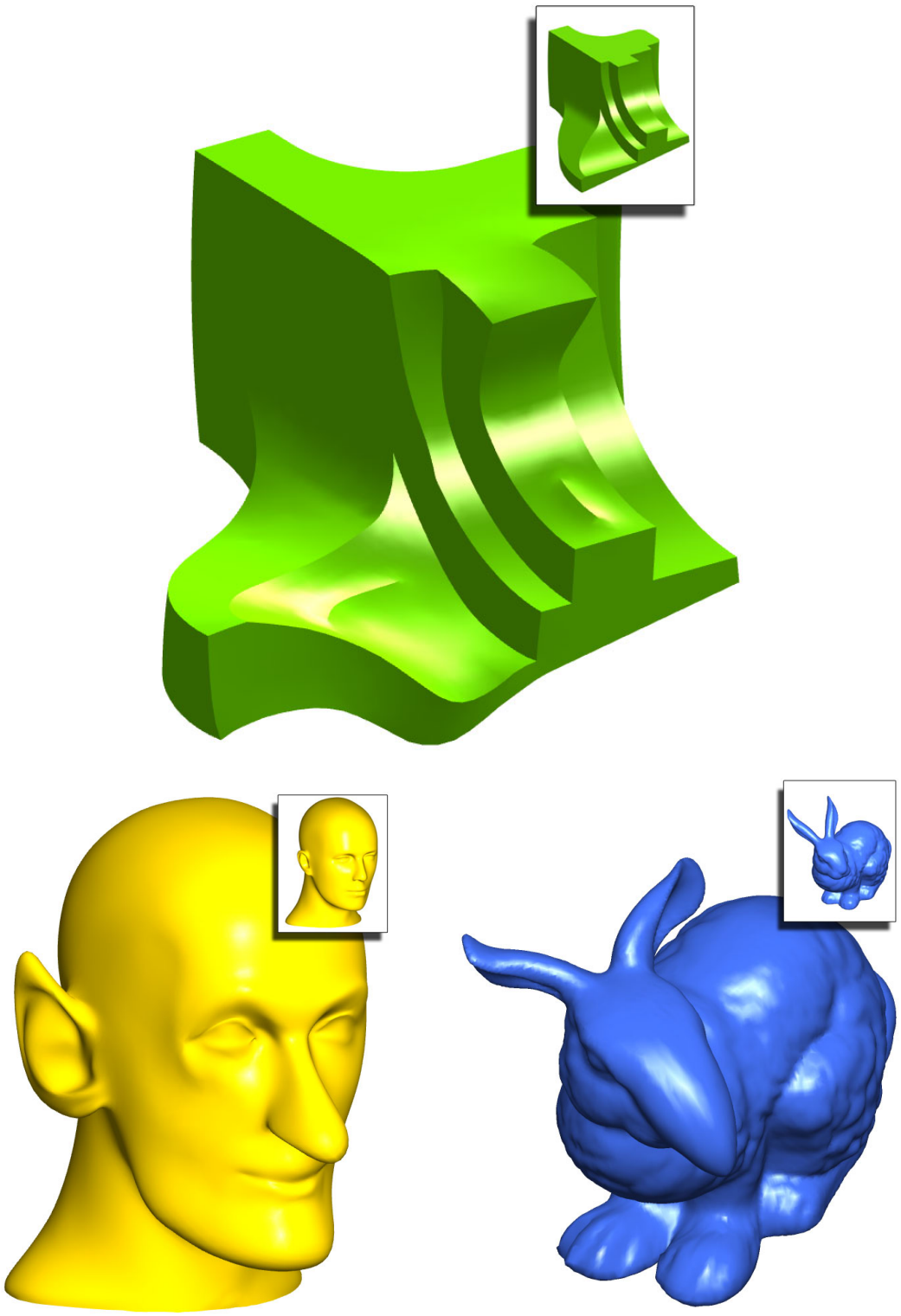


Figure 4.10: Some results: a deformed FANDISK with a few more sharp features, a rather surprised MANNEQUIN with more than just an extra contour around the eye, and droopy-eared, big-nose BUNNY with large feet.

Chapter 5

Automated Sketch-Based Editing of Surface Meshes

5.1 Introduction

The process of generating 3D shapes in engineering or content creation typically goes through several *design reviews*: renderings of the shapes are viewed on paper or a screen, and designers indicate necessary changes. Oftentimes designers sketch replacements of feature lines onto the rendering. This information is then taken as the basis of the next cycle of modifications to the shape.

We present a surface mesh editing system motivated by design reviews: given nothing but the over-sketch of a feature line, it automatically deforms the mesh geometry to accommodate the indicated modification. Building on existing mesh deformation tools [SLCO+04, NSACO05] (see Section 2.4 and Chapter 4), the main feature of this chapter is the *automatic* derivation of all necessary parameters that these systems require as input in *real-time*.

In particular, Laplacian Surface Editing [SLCO+04], but also most other recent mesh deformation techniques (e.g., [YZX+04, BPG06]) require the selection of: handle vertices, the displacement for these handle vertices and a region of interest (ROI), representing the part of the mesh to be modified to accommodate the displaced handle vertices. For our system, we need to compute this information from the over-sketched feature line alone; and we do this in fractions of a second. The steps described below comprise our system (see also Figure 5.1) – breaking down the problem into these steps and performing each step in few milliseconds are the main contributions:

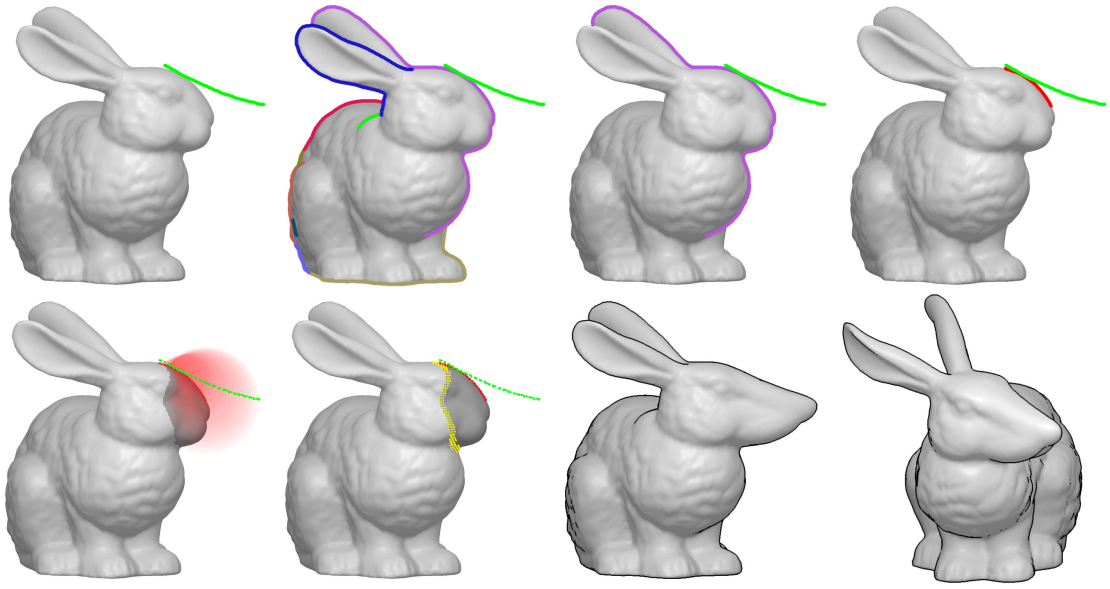


Figure 5.1: Algorithm pipeline. Top row, from left to right: a) user-sketch, b) image-space silhouettes, c) retained silhouettes after proximity culling, d) handle estimation; Bottom row, left to right: e) correspondences and ROI estimation by bounding volumes, f) setup for Laplacian Surface Editing, g) and h) deformation result. Note that the user only sees a), g) and h).

1. Based on the screen projection of the shape, a subset of pixels lying on potential feature lines is identified. These pixels are then segmented and converted to image-space polylines as the set of candidate feature lines.

2. The user-sketch is matched against all polylines to find the corresponding part on a feature line.

3. Based on the correspondence in image-space, a set of handle vertices in the surface mesh is selected. The image-space projection of these vertices covers the detected part of the feature line.

4. New positions for the handle vertices are derived from the displacements in image-space between the projection of the handle vertices and the user's sketch; these are the necessary displacements.

5. A part of the surface mesh around the handle vertices, computed by region growing, is defined as the ROI.

Note that in steps 3,4, and 5 we compute the necessary input for shape deformation, while steps 1 and 2 are required to identify the input, based only on the user-sketch.

5.2 Related work and system design

Sketch-based interfaces are a very popular method for creation and deformation of 3D surface meshes [IMT99, KSvdP07, KS07]. Deriving the parameters for mesh deformation from sketches only is not new: Kho and Garland [KG05] derive ROI and handle vertices from sketching onto the projected shape, essentially implying a skeleton for a cylindrical part. A second stroke then suggests a modification of the skeleton, and the shape is deformed according to the deformed skeleton. However, according to Hoffman and Singh [HS97], we recognize objects mainly by a few feature lines, namely silhouettes and concave creases. Since the process of paper-based sketching relies exactly on these features, we feel it is more natural to use them as the basis for our over-sketching mesh deformation tool. Note that this line of thought is congruent to Chapter 4: we have enhanced Laplacian Surface Editing techniques to work in the setting of prescribing new silhouettes. In particular, this requires positional constraints defined on mesh edges and finding the correspondence between a pre-selected silhouette of the mesh and the over-sketched silhouette. In Chapter 4 the user manually selects the ROI and a part of one of the silhouettes as a pre-process. In the work presented in this chapter, all these manual selections are now automated; the user only provides a single stroke, from which handle and ROI are estimated (Figures 5.1 and 5.2).

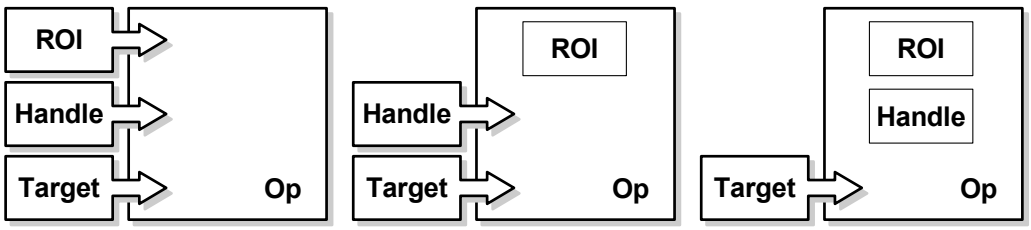


Figure 5.2: Required user interaction (from left to right): Chapter 4 [NSACO05], Kho and Garland [KG05], and our approach .

We have also observed that computing silhouettes from the mesh representation (i.e. in object-space) has problems: the silhouette path on the mesh might fold onto itself when projected to image-space – specifically, a point of the silhouette in image-space could map to several pieces of the silhouette on the mesh. As a result, the mapping from the sketch to handle vertices could be ill-defined. More generally, the complexity of the silhouette path on the surface is not necessarily reflected in its image-space projection, making a reasonable mapping from the sketch to vertices on the mesh difficult.

Because of these problems we detect silhouettes in image-space, and then try to identify vertices in the mesh that would map onto the detected region in image-space. Image-space silhouettes are usually obtained using edge detection filters on the depth map and/or normal map of the shape [Her99]. Typically, the conversion from raster-based edge pixels to vector-based polylines is then achieved by

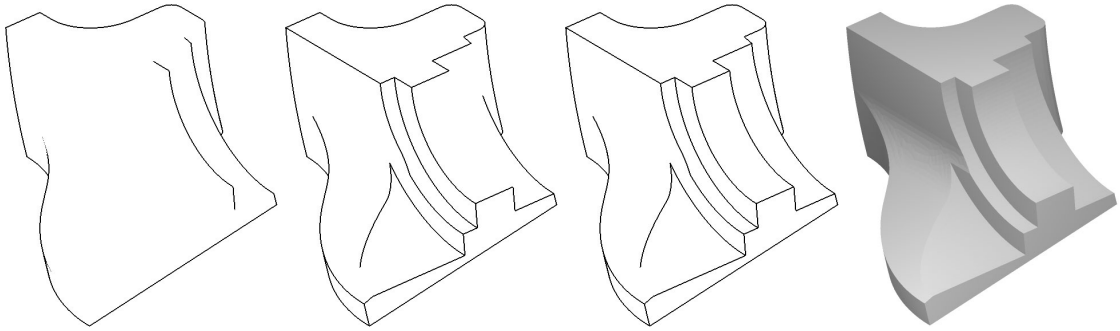


Figure 5.3: Depth map discontinuities (left), Normal map discontinuities (2nd left), combined discontinuities (3rd left), flat shaded scene (right)

applying some morphological operations (e.g. thinning) and finally tracing (e.g. chain codes). We have decided to restrict the set of feature lines to discontinuities in the depth map. This approach shows a feasible trade-off between quantity of feature lines vs. their significance (see Figure 5.3).

Matching a segment of a silhouette in image-space to the user-sketch requires a metric, defining the distance between polylines. This metric should resemble human perception of similarity. We have found that the important features are proximity to the candidate feature lines and intrinsic shape (see Figure 5.4). By intrinsic shape we mean similarity regardless of position and orientation in space. To maximize this intrinsic shape similarity we use a method by Cohen and Guibas [CG97].

We determine the handle mesh vertices corresponding to the silhouette segment by selecting vertices that are close to the handle in image-space. The displacements for these vertices are derived from displacements in image-space.

We consider defining the ROI as a form of mesh segmentation, for which var-

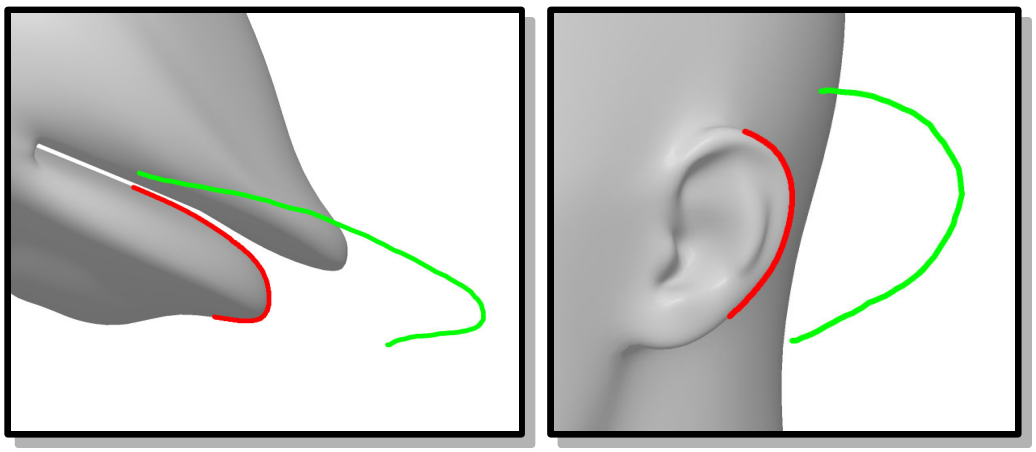


Figure 5.4: Handle estimation due to the similarity of handle candidate (red) and targeted deformation (green).

ious geometry-based methods are described (see [KT03, JLCW06]), and even image-based approaches are conceivable (see [HJBJ+96, PP93b]). Whereas image-based approaches obviously suffer from occlusion, geometry-based methods are only restricted by the requirement for interactive response times. Generally, topologically growing the ROI from the handle vertices is a feasible method.

Once we have defined handle vertices, their transformed target positions and the region of interest, the application of Laplacian surface editing is straightforward. Note that the user only provides 2D input and we have found that preserving the scale in depth leads to more intuitive results than scaling isotropically in 3D. Interestingly, several of the refinements of Laplacian Surface Editing (such as [SLCO+04]) favor isotropic scaling. For this reason, here we use an approach in the spirit of [LSCOL04], where local transformations of each frame are estimated a priori. As previously, we like to stress that other mesh deformation tools could be used as well.

5.3 Interface

Our user interface consists of a single rendering window with an orthogonal projection, embedded controls for navigation, and the capability of drawing viewport-aligned strokes (enabled by default). Holding some meta key activates the embedded navigation controls, with which the user can drag the mesh along the horizontal and vertical axis, rotate it by tapping beside it and dragging the mouse, and scale the current projection by clicking and dragging two invisible sliders on the left and right screen boundaries.

If the user has determined an appropriate view, placing a sketch near the silhouette implies a deformation. The system identifies the appropriate parameters (see following sections) and then displays the result. The user has the option to approve this deformation or to apply refinements by oversketching the new silhouette path.

5.4 Algorithm

The user sketches the desired deformation result as a view-dependent polyline. This polyline simply consists of tracked mouse events, and we apply the Douglas-Peucker algorithm [DP73] to obtain a simplified version. In the following sections we detail the steps of our algorithm.

5.4.1 Image-space silhouettes

In this section, we describe how to retrieve image-space 2D polylines that describe discontinuities in the depth map (and therefore silhouettes) of the scene using two

steps of detection and extraction. We developed a method that exploits the properties of a synthetic scene (= absence of noise) to speed up our algorithm, rather than relying on well established methods like the Canny edge detector [Can86] or morphological operations.

Silhouette detection

We determine discontinuities in the depth map by applying a 4-neighborhood Laplacian edge detection filter on each pixel p, along with some threshold θ_p:

$$sil(p) := D^2_{xy}[depth(p)] > \theta_p \tag{5.1}$$

We retrieve only edge pixels that describe the foreground of a discontinuity, since we define the depth range of the scene to (near, far) [0, 1] and use θ_p as a threshold for the *signed* filter response. Depending on the choice of θ_p (we recommend 0.005), the binary images retrieved consist of continuous silhouette paths (Figure 5.5, left). Note though, that these paths can be more than a single pixel wide, especially in areas of high curvature.

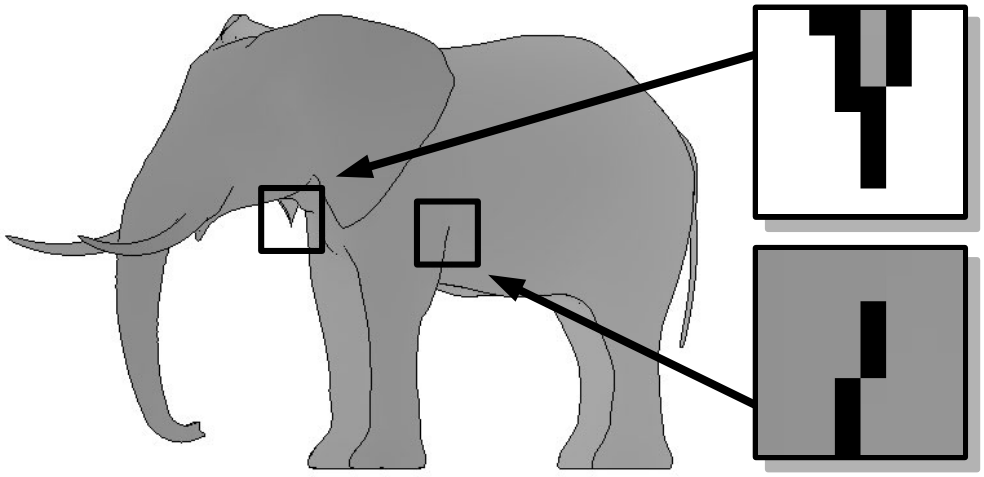


Figure 5.5: Depth map with binary overlay from Equation 5.1 (left), degenerated silhouette feature (top, right), silhouette caused by a surface crease (bottom, right)

Silhouette extraction

For the subsequent handle estimation (Section 5.4.2), we need to convert the silhouette pixel paths into a set of image-space polylines. Aiming for simplicity and speed, we developed a greedy segmentation algorithm that relies only on local criteria for silhouette tracing.

The basic idea of tracing connected components of the silhouettes is that silhouette pixels in the image are neighbors on a silhouette segment if they have similar depth. In other words, two neighboring silhouette pixels a and b are depth continuous if

$$cont(a,b) := \|depth(a) - depth(b)\| < \theta_n. \tag{5.2}$$

Remember that the silhouette pixels form a path that could be wider than a single pixel, making the conversion to a poly-line ambiguous. Some approaches use the morphological operation of thinning to correct this problem. However, applying morphological operations on the binary silhouette image may result in silhouette paths that are continuous in 2D, but discontinuous in depth. This is illustrated in Figure 5.6b: the silhouette terminates on pixel f_c if n_7 is removed by erosion, and $\|depth(f_c) - depth(n_0)\|$ exceeds θ_n. In this case, n_7 is exactly the pixel that stitches the silhouette together. Instead of developing depth sensitive morphological operations , we solve this issue by using a local tracing criterion.

The idea for the local tracing is to favor silhouette paths with lower curvature in image-space – that is, straight silhouettes are favored over ones with sharp corners. The criterion is implemented as a priority map relative to the direction from which we entered the current silhouette pixel (see Figures 5.6 and 5.7: a smaller number in the mask around f_c indicates higher priority). Based on the priority mask, silhouette edge paths are formed by selecting from depth continuous silhouette pixels.

However, correctly identifying endpoints of silhouette paths requires extra attention. A silhouette path ends in surface creases; and it might appear to end in sharp creases of the silhouette (see Figure 5.5). It also ends in image-space when the silhouette is obstructed by another part of the surface, in which case it connects to another silhouette (see Figure 5.7). Our basic tracing algorithm would correctly

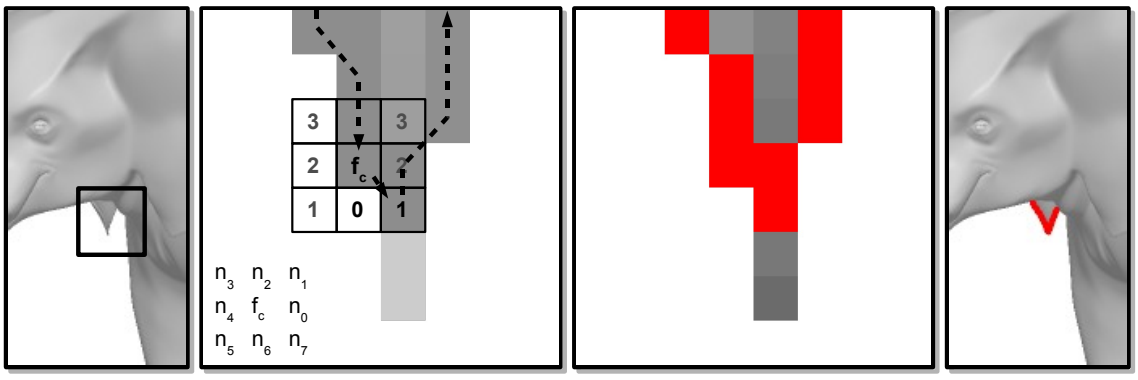


Figure 5.6: Tracing the silhouette path near a degenerate feature (from left to right): a) Elephant's ear, b) tracing step ($f_c \rightarrow n_7$) with priority map, neighborhood index (bottom left) and a degenerate feature in light grey (which is removed in a preprocessing step), c) final silhouette path, d) extracted silhouette.

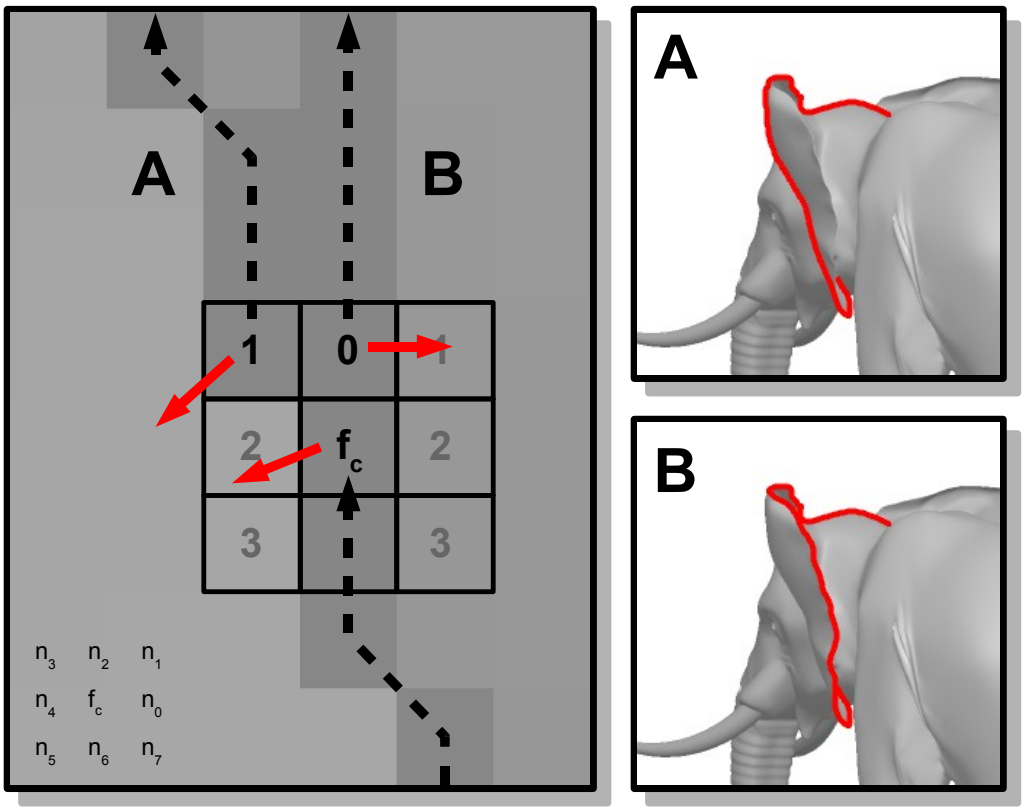


Figure 5.7: Maintaining depth map gradient orientation. Path A shows how our tracing algorithm maintains depth map gradient orientation with respect to the tracing direction (gradients shown as arrows per pixel) . If we disregard these gradients, the tracing algorithm will track a bogus silhouette, in this case path B, due to the preferred tracing direction. Note though, that the silhouette part from path B, which is missing in path A, will be a separate silhouette segment after all silhouettes have been traced.

identify endpoints in surface creases, however, it might also classify sharp corners as endpoints and could connect unconnected parts of the silhouettes if they happen to have almost similar depth. To avoid terminating in sharp corners, we remove the tips of silhouettes. Note that surface creases are surrounded by pixels with almost similar depth in the depth image, while tips of the silhouette are not (see Figure 5.5). So we remove tips by repeatedly removing silhouette pixels if they have less than two depth continuous 8-neighbors in the depth image (see Figure 5.6, second image). As an additional criterion for identifying connected silhouette pixels we use consistency of the surface normals along the silhouette (see Figure 5.7). As we are only interested in the orientation of the normals, it is sufficient to consider the gradients of the depth map.

In detail, our silhouette extraction algorithm creates silhouette polylines $S : \{(v_1, d_1), ..., (v_n, d_n)\}$ described by vertices $v_i \in \mathbb{R}^2$ and depth values $d_i \in \mathbb{R}$, by

scanning the binary silhouette image row by row, and extracting feature paths for any encountered silhouette pixel $f_c : (v_c, d_c)$ according to the following algorithm:

1. Create $S = \emptyset$.
2. Append f_c to S.
3. Determine next silhouette pixel f_n, where
   a) f_n is adjacent to f_c,
   b) f_n is depth continuous to f_c according to Equation5.2,
   c) f_n maintains the orientation of depth map gradients w.r.t. the current tracing direction (see Figure 5.7), and
   d) the tracing direction turn caused by f_n is minimal.
4. Mark f_c as a non-silhouette pixel.
5. Assign f_n to f_c.
6. Repeat on 2. until $f_c = NIL$.

Note that a) and b) are determined by Equations 5.1, and 5.2 respectively, whereas c) ensures continuity of the normals along the silhouette paths (Figure 5.7). Furthermore, d) is the tracing criterion, navigating the tracing algorithm through silhouette paths wider than a single pixel.

Since scanning the silhouette image row by row typically encounters a silhouette somewhere inside its path, the tracing algorithm is applied twice for any initial pixel, in opposite directions.

5.4.2 Handle estimation

To derive the actual handle polyline (a subset of all silhouette polylines), we introduce an estimation metric which reflects the likelihood that an arbitrary silhouette segment is a good handle w.r.t. the user-sketch (target polyline). As pointed out before, this scoring function relies on both proximity and similarity.

First, we substitute the silhouette polylines by simplified delegates (polylines as well, see [DP73]), and reduce the silhouettes by culling according to a proximity criterion (see Figures 5.1b and 5.1c).

The criterion on similarity is derived from the *Polyline Shape Search Problem (PSSP)* described by Cohen and Guibas [CG97]. First, we compute Turning Angle Summaries (TASs) $\{(s_0,t_0),...,(s_n,t_n)\}$ from the edges $\{e_0,...e_n\}$ of the target and silhouette polylines by concatenating tuples of edge lengths s_i and cumulative turning angles t_i, where

$$s_i = \| e_i \|, \quad t_i = \begin{cases} \measuredangle(e_0, 0) & \text{if } i = 0 \\ \measuredangle(e_{i-1}, e_i) + t_{i-1} & \text{if } i > 0 \end{cases} \tag{5.3}$$

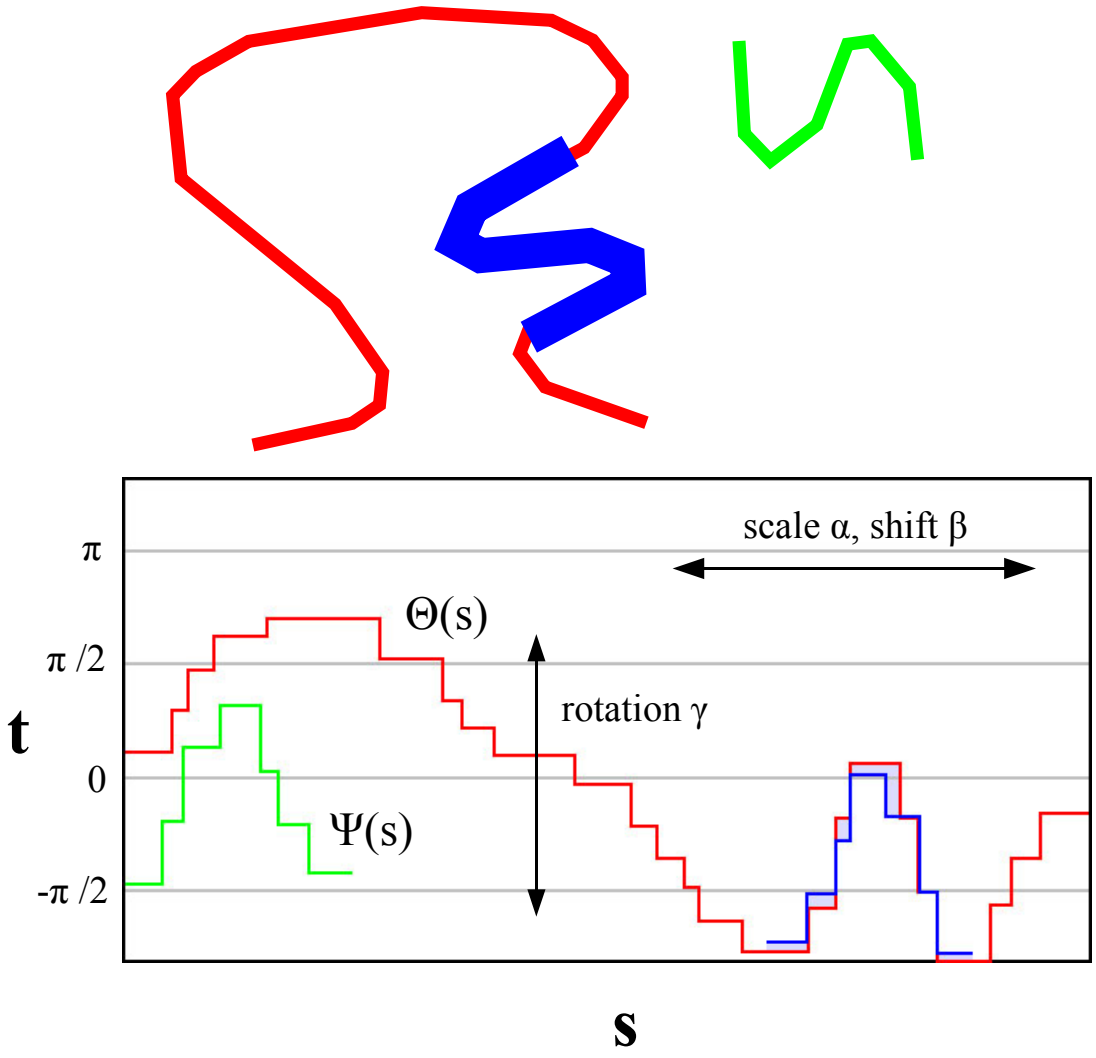


Figure 5.8: Top: the short, green target polyline, red silhouette, and best-match (blue/thick) shown as a subset of the red silhouette polyline. Bottom: arclength vs. cumulative turning angle representations of target $\Psi(s)$, silhouette $\Theta(s)$, and best-match polylines (bottom).

Please note that these summaries lack the representation of absolute coordinates, but they do retain the polyline arclength. Furthermore, rotating a polyline relative to its head results in a shift of its TAS along the turning angle axis, whereas isotropic scaling results in stretching its TAS along the arclength axis (see Figure 5.8).

We match the target polyline onto a single silhouette polyline, described by its (isotropic) scale α and position (shift) β, by matching their Turning Angle Summaries (Figure 5.8). The match result $M_{PSSP}: (\alpha, \beta, \gamma, R_{*mod})$ is described by a prescribed α and β, an optimal rotation γ, and the matching score R_{*mod}.

Optimal rotation and matching score are computed by a modified version of the scoring function from [CG97]. Using finite sums of differences, I_1 and I_2 describe the linear and squared differences between the piecewise constant TASs $\Psi(s)$ of the target and $\Theta(s)$ of the sihouette polylines (Figure 5.8):

$$\begin{aligned} I_1(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \left(\Theta(s) - \Psi\left(\frac{s-\beta}{\alpha}\right)\right) ds, \\ I_2(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \left(\Theta(s) - \Psi\left(\frac{s-\beta}{\alpha}\right)\right)^2 ds. \end{aligned} \tag{5.4}$$

Given the arclength l of the target polyline, we compute optimal rotation

$$\gamma = \gamma_*(\alpha,\beta) = \frac{I_1}{\alpha l}, \tag{5.5}$$

and matching score

$$R_{*mod}(\alpha,\beta) = \frac{1}{\alpha l}\left(\frac{I_2(\alpha,\beta)}{\alpha l} - \left(\frac{I_1(\alpha,\beta)}{\alpha l}\right)^2\right). \tag{5.6}$$

Cohen and Guibas retrieve matches for all segments (α,β) by using a topological sweep algorithm [EG86] to match the respective Turning Angle Summaries in scale/position space. However, since this approach needs $O(m^2n^2)$ time for m silhouette edges and n target edges, we decided to probe only a discrete number of sample segments in Equation 5.6 in $O(m+n)$ time per segment. Specifically, we match the target polyline to sample segments of a silhouette polyline by discretely sampling α and β respectively.

For the proximity criterion we compute the distances of corresponding endpoints of the two polylines, retrieving a near and far value $Prox_{near}$, $Prox_{far}$. Then we apply a final scoring function on the obtained per-silhouette match results:

$$R := 1/(1 + w_1 Prox_{near} + w_2 Prox_{far} + w_3 R_{*mod})^2 \tag{5.7}$$

Iterating over all silhouettes, we select the segment with the highest score, and extract the deformation handle from the respective full-res silhouette by using (α,β) of its matching record M_{PSSP}.

5.4.3 Finding handle/target correspondences

Given the polylines of deformation handle and target, we need to determine the corresponding mesh vertices and their transformed positions respectively.

Using both the image-space handle pixels, as well as the corresponding depth map, we construct an object-space bounding volume for each handle pixel (see Figure 5.9). A mesh vertex is classified as a handle vertex if it lies in the union of these bounding volumes.

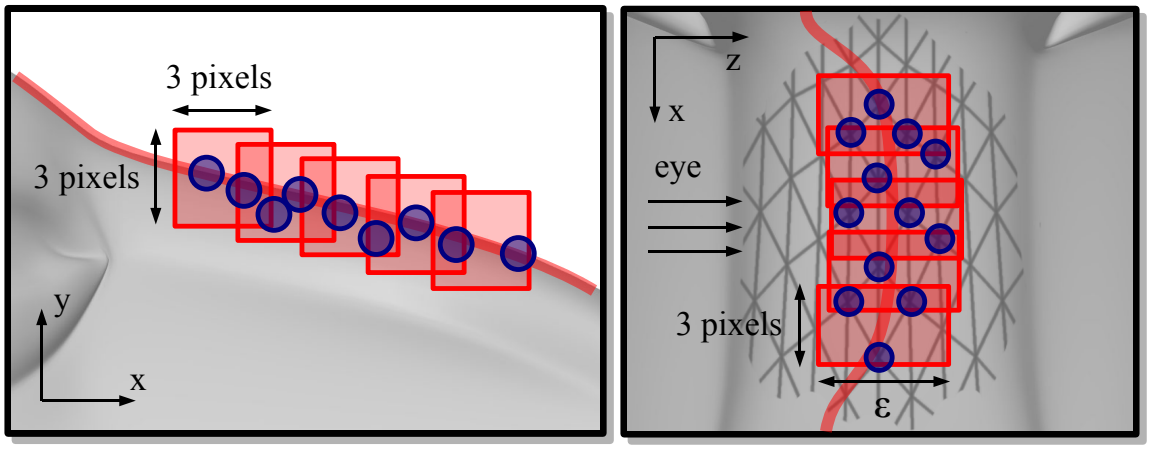


Figure 5.9: Mesh vertices that are classified as handle members (blue circles) using one bounding volume (red box) for each image-space handle pixel. Left: view from the editor, right: view from top (silhouette indicated as a red line in both views).

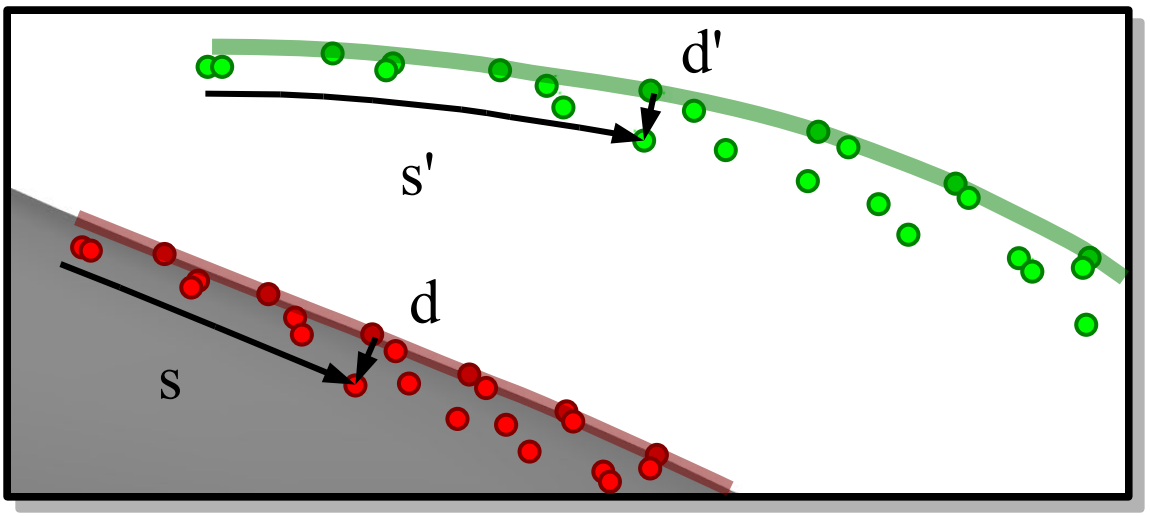


Figure 5.10: Mapping of handle relative arclength position s and diplacement d (red) onto the target polyline (green).

The transformed positions for these handle vertices are computed by mapping their handle-relative positions onto the target polyline. Specifically, we determine the position (s,d) for each handle vertex, where the arclength position s is given by its orthogonal projection of length d. Both handle and target polylines are parameterized uniformly in $[0,1]$ and the target position (s',d') is scaled accordingly (see Figure 5.10).

5.4.4 ROI estimation

To complete the deformation setup, we have to select the final ROI of the mesh according to some context sensitive criterion. We grow the ROI from the handle vertices. To control the expansion, we constrain the ROI to lie within a union of bounding volumes, which consists of one volume per handle vertex.

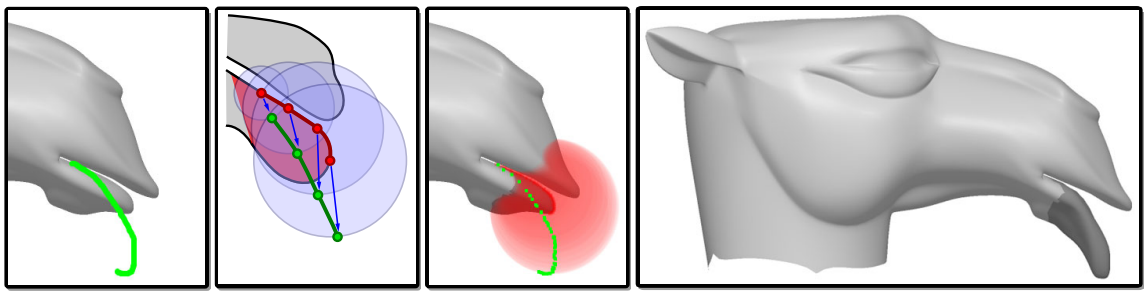


Figure 5.11: Automatic ROI selection (from left to right): a) After the user places a sketch, the handle is estimated and correspondences are established. b) From these correspondences, the ROI is grown within the union of spheres, starting from the handle vertices (dark/red region, lower lip). c) Shows this for the camel lip example. d) We use the obtained vertex sets *handle*, *transformed handle* and *ROI* as input to the Laplacian surface editing algorithm. See text for more details.

Specifically, we create a union of spheres, where each sphere center is located at the position of the respective handle vertex. Each sphere radius is set to the Euclidean distance $d_{h,s}$ between handle vertex and its transformed position. We have experimented with a variety of functions $r_s = f(d_{h,s})$, but have found that using $r_s = d_{h,s}$ already yields satisfying results: when the user sketch is far from the handle, using a larger sphere results in a larger ROI, yielding more deformable material (Figure 5.11), which is a reasonable heuristic. To determine the ROI, we define the handle vertices to be the initial ROI vertex set, and grow this set by subsequently adding vertices of the mesh that are (a) adjacent to the current ROI border, and (b) are inside the union of spheres.

5.5 Results

The modeling session shown in Figure 5.12 illustrates ease of use: after the user places a stroke, the system responds interactively, presenting a deformation that generally corresponds to the users intent. All algorithmic details, which are shown in various figures in this chapter, are absent from the actual user interface.

Table 5.1 shows some timings obtained on a Intel Core 2 Duo 6600 processor with 2.4 GHz and 2GB memory. Extracting and segmenting the image-space silhouettes (column *Sil*) takes between 5-20% of the processing time. Handle

Model	Feature	Sil*)	Handle*)	ROI	FacLSE	SolLSE*)	Sum	LSE size
Bunny	Ear	109	297	15	1032	500	1953	4911^2
CamelHead	Lip	110	250	15	250	140	765	1533^2
Mannequin	Nose	188	219	15	485	156	1063	2013^2
	Ear	94	62	16	609	156	937	3627^2
all timings in msec; *) unoptimized code								

Table 5.1: Some timings of our system.

estimation and finding handle/target correspondence (column *Handle*) depends on the density of silhouettes, as well as the number of model vertices (=5-25% overall). The column *LSE size* shows the dimensions of the sparse linear system (= number of ROI vertices), which is factored (FacLSE) and solved (SolveLSE) every time the user places a new stroke. This works interactively for ROIs up to a few thousand vertices. Of course we can also reuse the factorization as described in Chapter 4. Note that in all cases, our algorithms (Sil + Handle + ROI) use less time than LSE setup, factorization and solve (FacLSE + SolveLSE).

5.6 Discussion

Each of the steps in our approach presents a trade-off between fidelity and speed. And while the requirement of real-time interaction certainly restricts the algorithmic possibilities, it should also be clear that almost all over-sketches are potentially ambiguous, even in the case of communication among humans – so it is unlikely that an algorithm could consistently guess correctly according to the user's expectation.

We find that the extraction and segmentation of feature lines (silhouettes) works in almost all practical cases. It might be interesting to extend the extraction to discontinuities in the normals of the shape, or even to more subtle feature lines such as suggestive contours [DFRS03]. Another set of feature lines, though invisible from the rendering but known to more experienced users, are the projections of skeleton curves used in models rigged for animation. The information deduced by

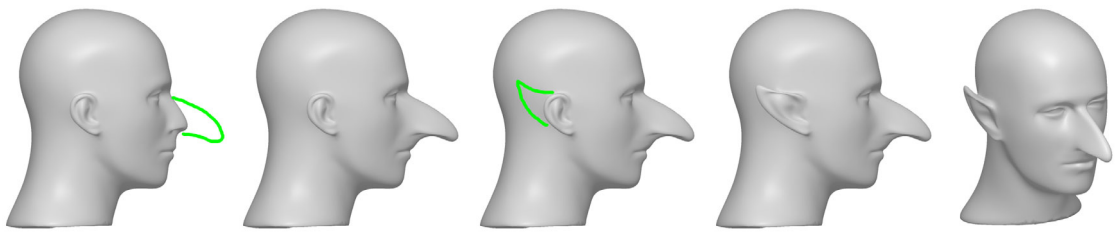


Figure 5.12: The MANNEQUIN modeling session.

our system could then be fed into modeling systems controlled by skeletons.

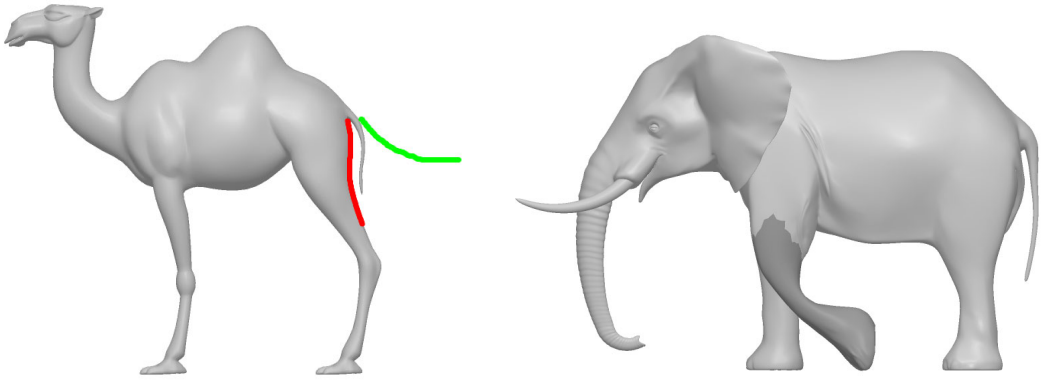


Figure 5.13: Left: ambiguous handle estimation at the CAMEL's tail. Right: unnatural deformation of the ELEPHANT's leg due to the limitation of Laplacian surface editing regarding large rotations.

Matching the user-sketch against the feature lines works nicely, however, it might be interesting to experiment with different functions for measuring proximity and shape similarity to overcome ambiguous handle estimations (see Figure 5.13 left). More fundamentally, matching is performed only against connected segments of the feature lines. The user might want to sketch something that the system has identified as different parts of the feature lines. It is unclear to us how to extend the matching process to this case.

The scores gathered when finding the best matching part of a curve could also be used to improve the subsequent steps of determining the ROI and computing the necessary deformation. For example, the rotation and scale for the best match could be used in the mesh deformation tool.

The ROI is selected based on proximity between user-sketch and feature line in image-space. This turned out to be simple and effective, yet it disregards apparent features of the shape. We believe the results could be improved by including information such as curvature and other features in image-space into our region growing approach. Another way of improving on the selection of the ROI would be to involve the user, perhaps by defining a special stroke indicating parts that may not move.

Looking at the deformation example in Figure 5.13 (right), it is clear that Laplacian surface editing is not a universally applicable deformation tool. However, it should be feasible to use the information gathered by the handle estimation such as rotation and scale of the best handle match in the deformation step.

Finally, as the system is almost generic with regard to the type of surface representation and the deformation tool, it would be very interesting to also try this approach in other settings.

Chapter 6

Laplacian Mesh Optimization

6.1 Introduction

Polygonal, and specifically triangular meshes are ubiquitous in computer graphics. Whether they have been generated by hand, from real world data, or by other means, obtaining a well-behaved mesh with respect to sampling rate and triangle quality is of great importance for a variety of applications. Our goal is to establish a simple and efficient framework, with which we can optimize triangle shapes, or smooth an input mesh, solely by means of vertex relocation, while maintaining both sampling rate and connectivity. The underlying assumption is that the input mesh is the *ground truth*: the sampling rate and connectivity information hold all information on the underlying smooth shape (possibly with implicitly defined or user-tagged sharp features). Furthermore, since our technique could be easily incorporated into existing sampling and connectivity optimizing algorithms (such as [Tur92, HDD+93, SG03, BK04b]), we have chosen to focus on vertex relocation.

Our framework is inspired by recent methods that compute vertex positions as the weighted least-squares solution to positional and Laplacian constraints. In particular, several methods try to preserve all original vertex Laplacians while imposing new positions for a few vertices [LSCOL04, SLCO+04, YZX+04], which allows detail preserving modeling. Prescribing new vertex Laplacians has several additional applications: a smooth surface has a Laplacian with small magnitude. This observation has lead Sorkine and co-workers to represent the geometry of a mesh by assuming the vertex Laplacians vanish and additionally fixing the positions of a few control vertices [SCO04, SCOIT05]. By prescribing vertex Lapla-

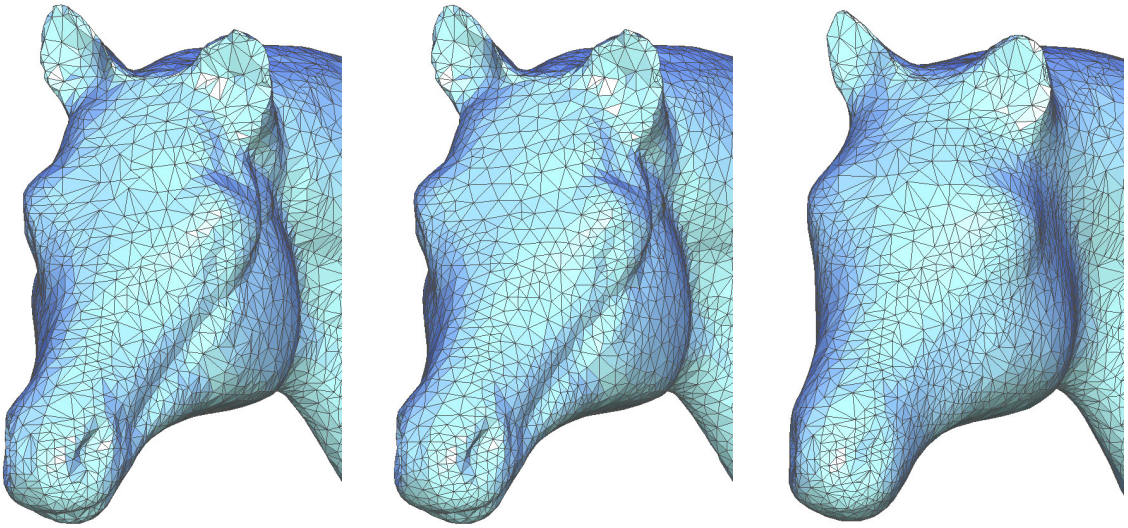


Figure 6.1: Original HORSE mesh (left) triangle shape optimization (middle) and feature preserving smoothing (right)

cians with modified magnitude or direction, we achieve a variety of modeling effects in our sketch-based tool (Chapter 4). We explain some of these techniques in Section 6.3.

We take these ideas a step further and use positional and Laplacian constraints for *all* vertices. Figure 6.2 illustrates our basic idea. Constraining all vertices with both positions and vertex Laplacians leads to a simple, flexible, and powerful framework for mesh optimization (Section 6.4). Inspired by our local triangle shape optimization, which we use to embed the user drawn sketches into the mesh (Chapter 4), we show that in this framework *all* triangles can be optimized at the same time (Section 6.5). Figure 6.3 shows that fixing only a few vertices (as in Chapter 4) would not suffice. Similarly, the idea of Sorkine and Cohen-Or [SCO04] to generate fairly smooth meshes by constraining a few positions and assuming the vertex Laplacians vanish turns into a global mesh smoothing technique. This idea is detailed in Section 6.6. Figure 6.1 demonstrates both techniques. Furthermore, by adjusting the weights on the different terms of the optimization it is possible to preserve sharp features, if this is desired. Sections 6.5 and 6.6 introduce several weighting schemes for the positional as well as the Laplacian constraints and discuss the variety of effects that can be achieved.

6.2 Related work

Since our framework performs vertex relocation for both triangle shape optimization and smoothing, we briefly list recent work in these fields that is most related to our approach. An extensive overview is beyond the scope of this dissertation, but can be found on remeshing [AUGA05] and geometric sig-

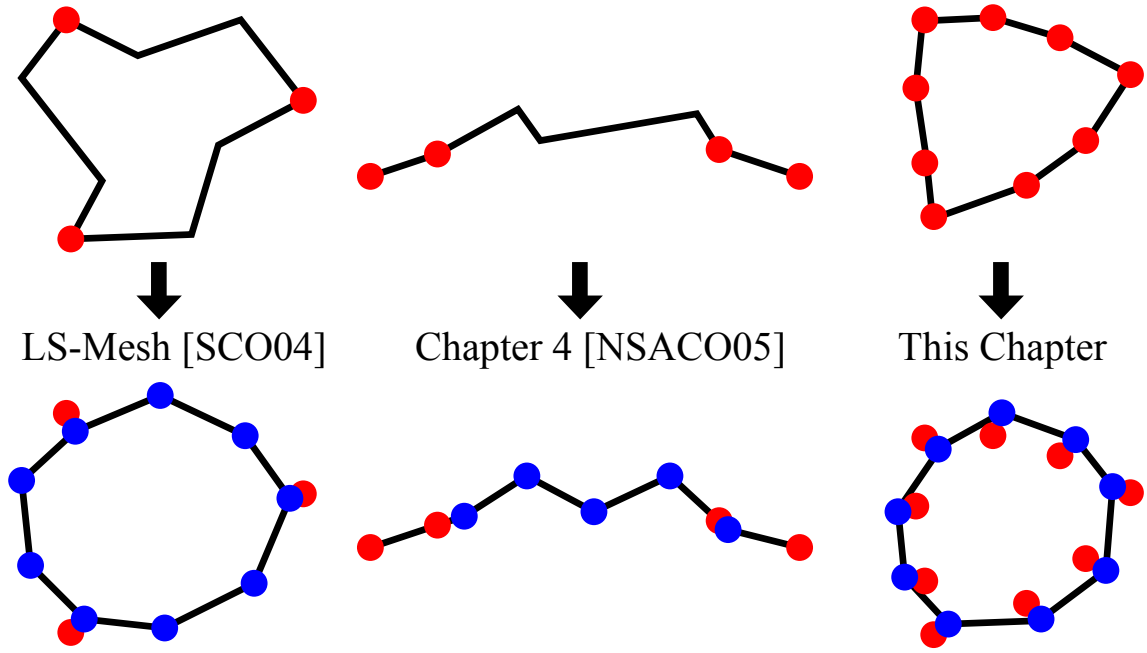


Figure 6.2: Left: Sorkine and Cohen-Or [SCO04] reconstruct the surface from a subset of control vertices (red). Middle: in Chapter 4 we constrain the boundary vertices (red), and optimize internal triangle shapes. Right: in the work presented in this chapter, we constrain all vertices and optimize triangle shapes and/or smooth the mesh, with optional feature preservation.

nal processing [Tau00]. More recent surveys on mesh smoothing are available in [HP05], [BS05] and [CC05].

Vertex relocation for remeshing. Repositioning vertices is generally treated as a subproblem in the context of *remeshing* [AUGA05], motivated by the need to improve triangle shapes with respect to one or more of the extensively studied quality metrics [PB03]. There are quite a few algorithms that circumvent the problem of relocating original mesh vertices entirely, by resampling the surface [Tur92] using a global parameterization of the mesh [AMD02, AECDI03]. Remeshing algorithms, which perform connectivity optimization with vertex relocation, construct a global or a per-vertex, local parameterization, and use this parameterization to lift the vertex to the original surface after relocation in the parameter domain [VRKS01, SG03, VRS03, SAG03]. When exact error bounds are not required, the relocated vertex can also be simply projected back onto its original tangent plane [BK04b], instead of constructing a parameterization. What is common to most of these algorithms (except for the "pure" halftoning approach [AMD02]), is that they are inherently iterative, that is, they apply the relocation step-by-step, one vertex at a time, until some stoppage criterion is reached. In contrast, all vertex relocations in our system result from the unique solution of one sparse linear system.

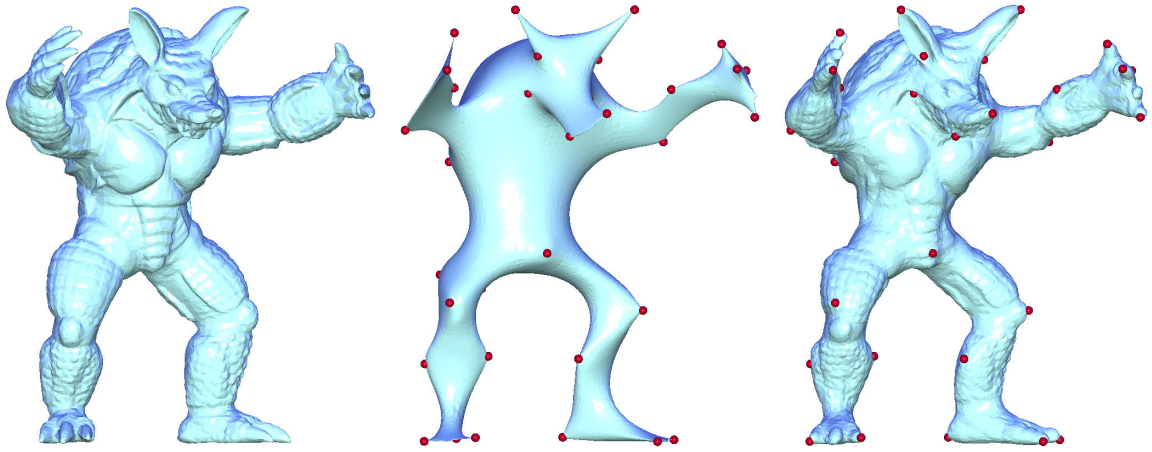


Figure 6.3: Least squares mesh [SCO04] (middle) and our detail preserving triangle shape optimization from Chapter 4 (right) of the ARMADILLO (left), each with the same set of 40 control vertices (red).

Mesh smoothing. The bulk of existing work in mesh smoothing deals with discrete filter design. Taubin's [Tau95] pioneering work introduces a two-step Laplacian operator to inflate the mesh after smoothing, thereby reducing shrinkage. Desbrun et al. [DMSB99] use implicit integration with Fujiwara [Fuj95] or cotangent weights [PP93a] for scale-dependent, unconditionally stable smoothing, which leaves triangle shapes and sizes mostly unchanged (also known as *mean curvature flow* or MC flow for short). This approach is extended to handle anisotropy and to preserve features by Meyer et al. [MDSB03], and further improved to *prescribed* MC flow by Hildebrandt and Polthier [HP05]. Recently, researchers have applied the bilateral filter, known from image processing, to discrete surfaces [FDCO03, JDD03]. Our approach to mesh smoothing differs significantly from all of these methods. We rely on the least-squares meshes algorithm [SCO04] to perform inner (triangle shape) and/or outer (surface smoothness) fairing, while applying soft positional constraints to all mesh vertices – where the weights depend on, for example, discrete curvature distribution – to retain specific features. Our optimization technique has similarities with the geometric fairing of Schneider and Kobbelt [SK01]; their method is oriented towards freeform surface design and thus prescribes positional constraints only to the boundary vertices, whereas our technique optimizes an existing mesh geometry and/or triangle quality. Other mesh fairing work includes discrete smooth interpolation (D.S.I.) by Mallet [Mal89] and the constrained version thereof [LM99], which is related to our own work, since they also fit a mesh to given data points in the least-squares sense. The main difference is that, like Schneider and Kobbelt [SK01], they do not generally constrain all vertices, as is the case in our technique. For another related constraint-based fairing method, see [HP07].

6.3 Basics and notation

The mesh is represented as a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, with vertices $\mathbf{V}$ and edges $\mathbf{E}$, where $\mathbf{V} = [\mathbf{v}_1^T, \mathbf{v}_2^T, \ldots, \mathbf{v}_n^T]^T$, $\mathbf{v}_i = [v_{ix}, v_{iy}, v_{iz}]^T \in \mathbb{R}^3$ is the original geometry, and $\mathbf{V}'$ denotes the displaced geometry. Furthermore, δ_i is the Laplacian of $\mathbf{v}_i$, the result of applying the discrete Laplace operator to $\mathbf{v}_i$

$$\delta_i = \sum_{\{i,j\} \in \mathbf{E}} w_{ij}(\mathbf{v}_j - \mathbf{v}_i) = \left[\sum_{\{i,j\} \in \mathbf{E}} w_{ij}\mathbf{v}_j \right] - \mathbf{v}_i, \tag{6.1}$$

where $\sum_{\{i,j\} \in E} w_{ij} = 1$, and the choice of weights

$$w_{ij} = \frac{\omega_{ij}}{\sum_{\{i,k\} \in \mathbf{E}} \omega_{ik}} \tag{6.2}$$

defines the nature of δ_i. Some popular choices are

$$\omega_{ij} = 1, \tag{6.3}$$

$$\omega_{ij} = cot\ \alpha + cot\ \beta, \tag{6.4}$$

where (6.3) are the uniform and (6.4) the cotangent weights. The angles used in these equations are shown in Figure 6.4. In the remainder of this chapter, we will refer to the uniform and cotangent Laplacians with normalized weights as δ_u and δ_c respectively. To obtain the Laplacians for the entire mesh, we use the $n \times n$ Laplacian matrix $\mathbf{L}$, with elements

$$\mathbf{L}_{ij} = \begin{cases} -1 & i = j \\ w_{ij} & (i,j) \in \mathbf{E} \\ 0 & \text{otherwise} \end{cases}, \tag{6.5}$$

and we denote as $\mathbf{L}_u$ and $\mathbf{L}_c$ the Laplacian matrices with uniform and cotangent weights respectively. With $\mathbf{V}_d = [v_{1d}, v_{2d}, \ldots, v_{nd}]^T, d \in \{x,y,z\}$, the $n \times 1$ vector containing the x,y or z coordinates of the n vertices, we can compute the x, y and z coordinates of the Laplacians $\Delta_d = [\delta_{1d}, \delta_{2d}, \ldots, \delta_{nd}]^T, d \in \{x,y,z\}$ separately as

$$\Delta_d = \mathbf{L}\mathbf{V}_d. \tag{6.6}$$

The uniform Laplacian of $\mathbf{v}_i$ points to the centroid of its neighboring vertices, and has the nice property that its weights do not depend on the vertex positions. The cotangent Laplacian is known to be a good approximation of the surface normal, although the weights can become negative and depend on the vertex positions. When properly scaled by the Voronoi region (see Figure 6.4)

$$\overline{\kappa}_i \mathbf{n}_i = \delta_{i,c\overline{\kappa}} = \frac{1}{2A(\mathbf{v}_i)} \sum_{\{i,j\} \in \mathbf{E}} (cot\ \alpha + cot\ \beta)(\mathbf{v}_j - \mathbf{v}_i), \tag{6.7}$$

as described by Meyer et al. [MDSB03], we obtain the discrete mean curvature normal $\bar{\kappa}_i \mathbf{n}_i$, which is the unit length surface normal $\mathbf{n}_i$ scaled by the discrete mean curvature $\bar{\kappa}_i$. Furthermore, we will use Equation 6.7 to discretize the Laplacian matrix $\mathbf{L}$, denoted as $\mathbf{L}_{c\bar{\kappa}}$, in the context of mesh smoothing (Section 6.6).

We now describe two interesting applications, which will lead to our optimization in Section 6.4.

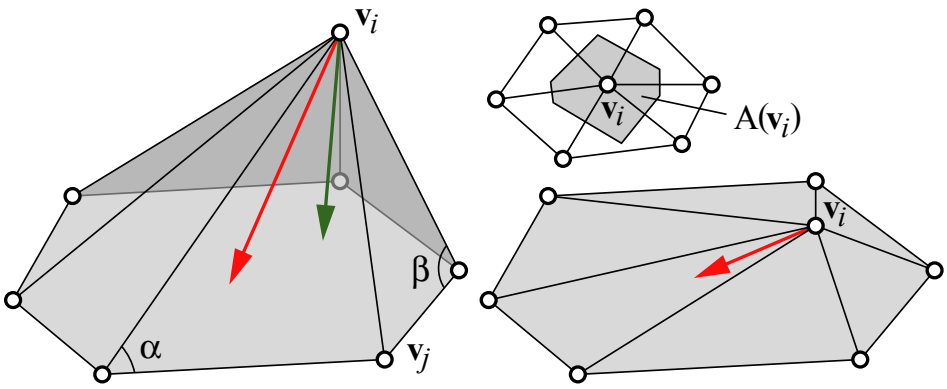


Figure 6.4: Left: uniform (red) and cotangent (green) Laplacian vectors for a vertex $\mathbf{v}_i$ and its (in this case planar) 1-ring, as well as the angles used in Equation 6.4 for one $\mathbf{v}_j$, Bottom right: the effect of flattening $\mathbf{v}_i$ into the 1-ring plane. While the cotangent Laplacian vanishes, the uniform Laplacian generally does not. Right top: the Voronoi region $A(\mathbf{v}_i)$ around a vertex.

Least squares meshes. Sorkine and Cohen-Or [SCO04] demonstrate how a mesh can be reconstructed from connectivity information alone, using a small subset $\mathbf{C} \subset \mathbf{V}$ of m geometrically constrained vertices (anchors). They solve for x, y and z positions ($\mathbf{V}'_d = [v'_{1d}, v'_{2d}, \ldots, v'_{nd}]^T, d \in \{x, y, z\}$) separately by minimizing the quadratic energy

$$\|\mathbf{L}_u \mathbf{V}'_d\|^2 + \sum_{s \in \mathbf{C}} w_s^2 |v'_{sd} - v_{sd}|^2, \tag{6.8}$$

where the v_{sd} are the stored anchor positions and the w_s^2 are weighting factors. In practice, with the anchors as the first m vertices (w.l.o.g.), the $(n+m) \times n$ overdetermined linear system $\mathbf{A}\mathbf{V}'_d = \mathbf{b}$

$$\left[\begin{array}{c|c} \mathbf{L}_u & \\ \hline \mathbf{I}_{m \times m} & \mathbf{0} \end{array}\right] \mathbf{V}'_d = \left[\begin{array}{c} \mathbf{0} \\ \hline \mathbf{V}_{(1\ldots m)d} \end{array}\right] \tag{6.9}$$

is solved in the least squares sense using the method of normal equations $\mathbf{V}'_d = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$. Note that the first n rows of $\mathbf{A}\mathbf{V}'_d = \mathbf{b}$ are the Laplacian constraints, while the last m rows are the positional constraints. The reconstructed shape is

generally smooth, with the possible exception of small areas around anchor vertices. The minimization procedure moves each vertex to the centroid of its 1-ring, since the uniform Laplacian $\mathbf{L}_u$ is used, resulting in good inner fairness.

Detail preserving triangle shape optimization. In Chapter 4, we show how a least squares optimization can improve triangle quality in a small mesh region (including boundary constraints) with negligible vertex drift. Essentially, we modify the linear constraints in Equation 6.9 as

$$\left[\begin{array}{c|c} \mathbf{L}_u & \\ \hline \mathbf{I}_{m\times m} & \mathbf{0} \end{array}\right] \mathbf{V}'_d = \left[\begin{array}{c} \Delta_{d,c} \\ \hline \mathbf{V}_{(1\ldots m)d} \end{array}\right], \tag{6.10}$$

where the uniform Laplacian of each new vertex position is asked to resemble its *undeformed* cotangent Laplacian as closely as possible. The idea here is that while the uniform Laplacian has a tangential component, the cotangent Laplacian does not (see also [DMSB99]), and thus the optimization attempts to remove the tangential components, while preserving the the surface details in the normal direction. As shown in Figure 6.3, this method fails to preserve the overall shape when the region is large, or there are insufficient boundary constraints, but works well for local modifications. We have experimented with the anchor selection process of Sorkine and Cohen-Or [SCOIT05] for Equation 6.10, an iterative process, which, although fast, still introduces a significant computational overhead. Yet, while this attempt did produce fairly good results after many iterations, it lead us to the *"all vertices are anchors"* approach, which requires no selection process, and will be described and discussed in the next sections.

6.4 Global vertex relocation framework

We propose a modification of Equations 6.9 and 6.10, which generalizes both least-squares meshes [SCO04] and detail preserving triangle shape optimization [NSACO05], and gives rise to two interesting applications. Our general $2n \times n$ system $\mathbf{A}\mathbf{V}'_d = \mathbf{b}$ is

$$\left[\begin{array}{c} \mathbf{W}_L\mathbf{L} \\ \hline \mathbf{W}_p \end{array}\right] \mathbf{V}'_d = \left[\begin{array}{c} \mathbf{W}_L\mathbf{f} \\ \hline \mathbf{W}_p\mathbf{V}_d \end{array}\right]. \tag{6.11}$$

The main modification is that we no longer have a subset of positional constraints, instead, *all* vertices appear both as Laplacian, and positional constraints. As we will describe in more detail further below, we can modify the result of the operation and introduce a good trade-off between triangle quality and geometric error by (non-)uniformly weighting the positional constraints with the diagonal matrix $\mathbf{W}_p$. In some cases, it will also be beneficial to weight the Laplacian matrix $\mathbf{L}$ and the corresponding right-hand side $\mathbf{f}$ with the diagonal matrix $\mathbf{W}_L$. In general, larger weights in $\mathbf{W}_p$ enforce positional constraints and thus preserve the original geometry, which can be useful for high-curvature regions and sharp features. On

the other hand, larger weights in $\mathbf{W}_L$ enforce regular triangle shapes and/or surface smoothness. As we describe in the following sections, solving the general system in Equation 6.11 results in

- **detail preserving triangle shape optimization**, when setting $\mathbf{L} = \mathbf{L}_u$ and $\mathbf{f} = \Delta_{d,c}$, or
- **mesh smoothing**, when setting $\mathbf{L} = \mathbf{L}_{c\overline{\kappa}}$ or $\mathbf{L}_c$ (outer fairness) or $\mathbf{L} = \mathbf{L}_u$ (inner and outer fairness) and $\mathbf{f} = \mathbf{0}$.

6.5 Global triangle shape optimization

Our goal here is to optimize triangle shapes. We measure our success with the *radius ratio* [PB03], mapped to $[0,1]$ as

$$t_i = 2\frac{r}{R}, \tag{6.12}$$

where R and r are the radii of the circumscribed and inscribed circles respectively. This way, $t_i = 1$ indicates a well shaped, $t_i = 0$ – a degenerate triangle. To perform this optimization, we discretize $\mathbf{L}$ in Equation 6.11 using the uniform Laplacian $\mathbf{L}_u$, set $\mathbf{f}$ on the right hand side to $\Delta_{d,c}$, and use $\mathbf{W}_L = \mathbf{I}$, similar to Equation 6.10. A straightforward, yet naive choice, is to use uniform weights $\mathbf{W}_p = \mathbf{W}_{const} = s\mathbf{I}$, where s denotes an arbitrary positive scalar. While this does somewhat improve the overall triangle quality (see Figure 6.5(b)), and has the lowest geometric error (Hausdorff distance to (a) [CRS98]), we can do better in terms of triangle quality.

It can be observed that large geometric error is likely to occur in the vicinity of vertices with high discrete mean curvature $\overline{\kappa}$. Therefore, a more sensitive choice of weights is a linear curvature-to-weight transfer function (Figure 6.5(a), bottom), which maps discrete mean curvature $\overline{\kappa}_i$ to weight w_i for each vertex, resulting in $\mathbf{W}_p = \mathbf{W}_{linear}$. More precisely, given the minimal and maximal discrete mean curvature over the mesh, $\overline{\kappa}_{min}$ and $\overline{\kappa}_{max}$, we linearly map the interval $[\overline{\kappa}_{min}, \overline{\kappa}_{max}]$ to $[0,s]$ (Figure 6.5(c)). Since the values for $\overline{\kappa}$ can be very large for a small set of vertices, we truncate these outliers before computing the linear mapping. The threshold value $\overline{\kappa}_\tau$ is computed using a simple box plot test [Tuk77], and works well for all models we have encountered. The weights of vertices where $\overline{\kappa} > \overline{\kappa}_\tau$ are clamped to s.

As can be seen in Figures 6.5 (b) and (c), while using $\mathbf{W}_{const}$ constrains low curvature vertices excessively, $\mathbf{W}_{linear}$ has a tendency to give them too much freedom, and therefore also has relatively high geometric error. Since we wish to take the relative frequency of mean curvature into account, we have chosen to use the cumulative density function (cdf) of $\overline{\kappa}$ (Figure 6.5(a)) to map from $\overline{\kappa}_i$ to weight $w_i \in [0,s]$, resulting in $\mathbf{W}_p = \mathbf{W}_{cdf}$. In detail, given the normalized distribution

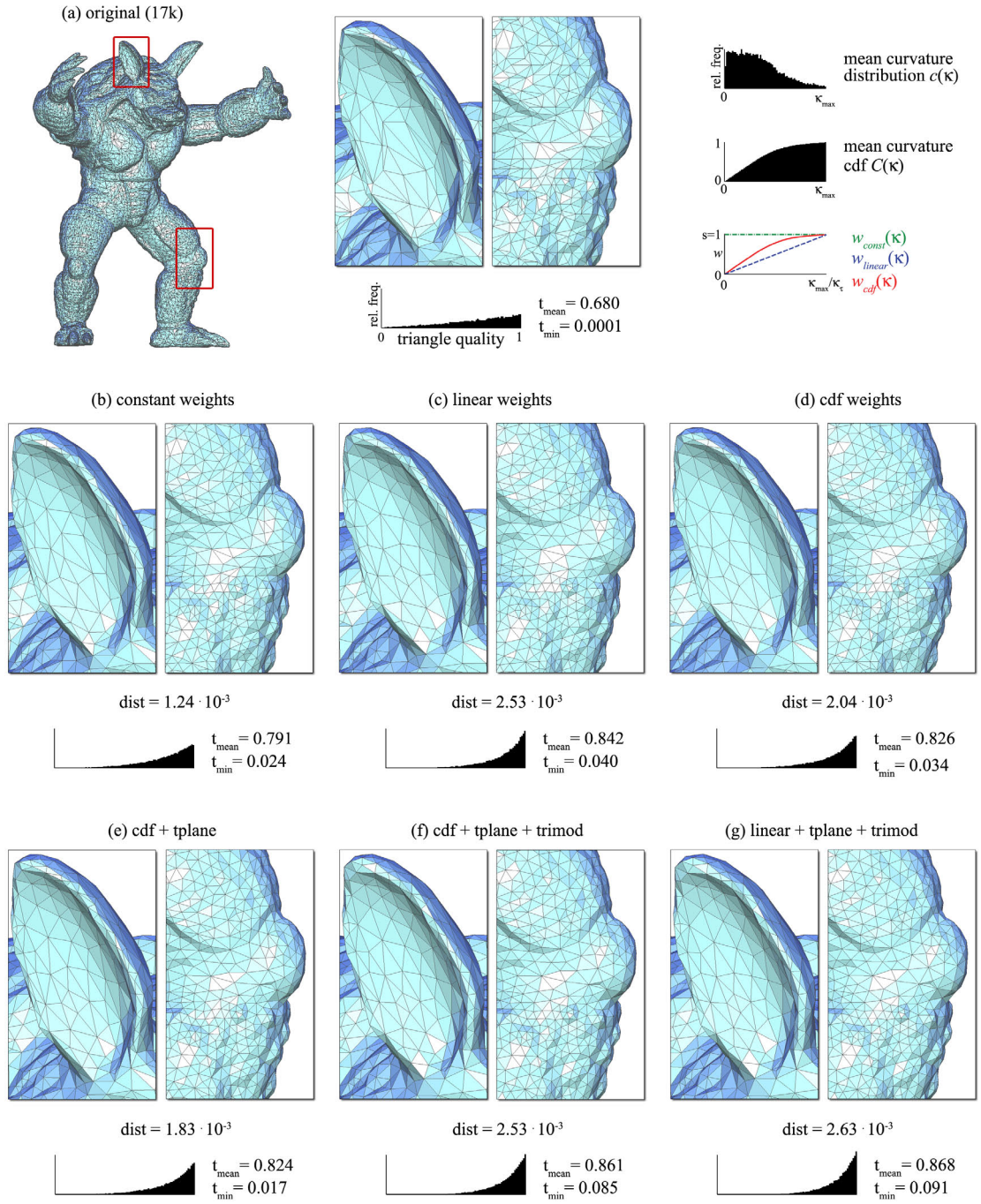


Figure 6.5: Comparison of triangle optimization weights. Top row: a 17k vertices ARMADILLO mesh (left), with the same two close-ups used in (b)-(g) and the triangle quality distribution (see Equation 6.12) (center), the distribution function of discrete mean curvature $c(\overline{\kappa})$, the associated cumulative density function (cdf) $C(\overline{\kappa})$, and the three weighting functions, used in (b)-(g) to map from curvature to positional weight for each vertex (right). Each optimized mesh (b)-(g) shows its triangle quality distribution, its mean (t_{mean}) and minimal (t_{min}) triangle quality, as well as the Hausdorff distance ($dist$) to the original mesh (a) with respect to the bounding box diagonal.

function of mean curvature $c(\overline{\kappa})$ for a mesh (Figure 6.5(a)), we compute the cdf $C(\overline{\kappa})$ as

$$C(\overline{\kappa}) = \int_0^{\overline{\kappa}} c(t)\, dt, \tag{6.13}$$

shown in Figure 6.5(a) (in practice, $C(\overline{\kappa})$ is precomputed as a discrete sum over the vertex curvatures). The rationale behind using $C(\overline{\kappa})$ is that if we have a mesh with a large amount of low curvature vertices (such as the ARMADILLO in Figure 6.5), these vertices should be assigned a larger weight than they would have if $\mathbf{W}_{linear}$ was used, thus reflecting the need to retain the detail in these regions and reduce vertex drift. As can be seen from the values in Figure 6.5(d), using $\mathbf{W}_{cdf}$ is somewhere between $\mathbf{W}_{const}$ and $\mathbf{W}_{linear}$, and is a good trade-off between triangle quality and geometric error. The cumulative density function also has the advantage of implicitly dealing with outliers without the need to compute $\overline{\kappa}_\tau$.

6.5.1 Tangent plane constraints

To further reduce the geometric error, planar (2D) constraints can be incorporated in our system. Essentially, we add the following term to the energy in Equation 6.8:

$$\sum_{i=1}^{n} |\mathbf{n}_i \cdot (\mathbf{v}_i' - \mathbf{v}_i)|^2. \tag{6.14}$$

This term penalizes displacement perpendicular to the tangent plane defined by the original vertex position $\mathbf{v}_i$ and the local surface normal $\mathbf{n}_i$ given in Equation 6.7, assuming a first order approximation of the surface around $\mathbf{v}_i$. Since constraints of this form require solving for x, y and z positions simultaneously, we need to couple the $2n \times n$ system (Equation 6.11) for each coordinate x, y and z into a single $6n \times 3n$ system, and add n constraints of the form

$$\mathbf{n}_i \cdot \mathbf{v}_i' = \mathbf{n}_i \cdot \mathbf{v}_i, \tag{6.15}$$

resulting in a $7n \times 3n$ system. Although this involves significant computational overhead, Figure 6.5(e) shows that tangent plane constraints reduce geometric error and have nearly the same mean triangle quality compared to cdf weights alone (Figure 6.5(d)).

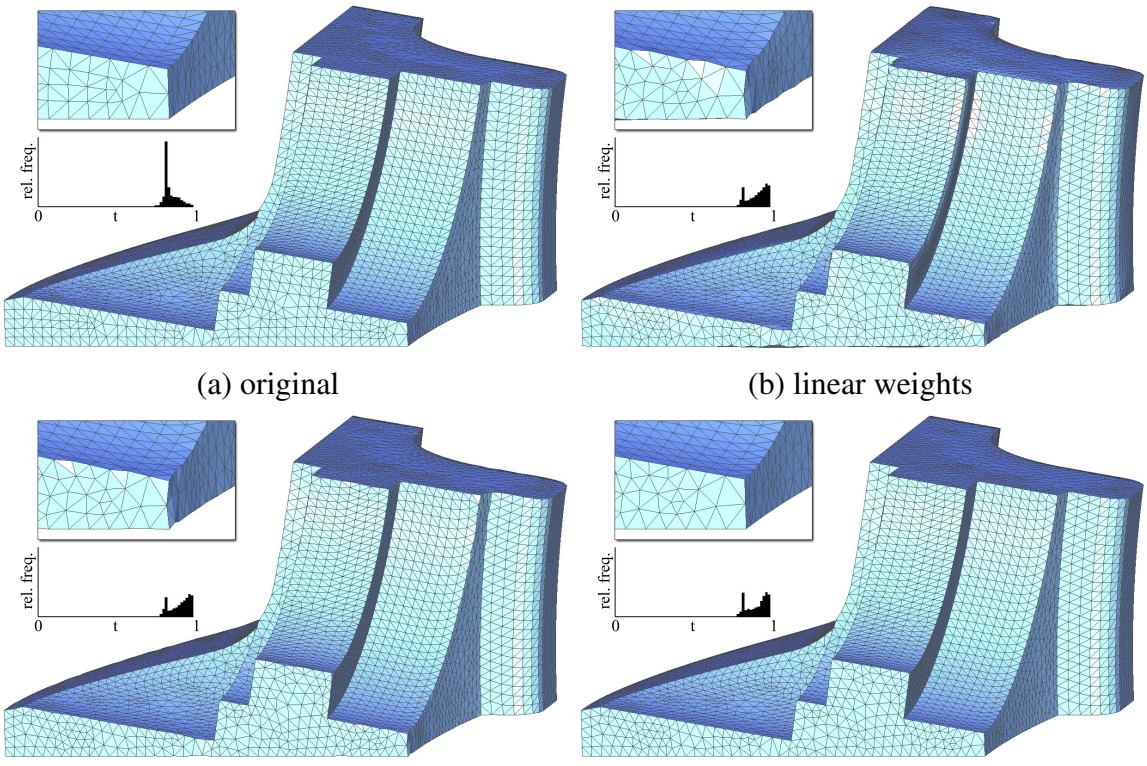
6.5.2 Triangle quality modulation

The quality of meshes for numerical simulations, such as the Finite Element Method (FEM), is heavily influenced by t_{min}, the minimal triangle quality. To maximize t_{min}, the positional weights $\mathbf{W}_p$ are modulated with the diagonal matrix $\mathbf{W}_t$, where the entry $w_{t,i}$ for vertex $\mathbf{v}_i$ is set to the minimal triangle quality t of its adjacent triangles. This way, if there is a triangle with small t attached to $\mathbf{v}_i$ it will be allowed to move more than without the modulation. The effect for cdf

and linear weighting can be observed in Figures 6.5(f) and (g): triangle quality has improved, especially around the ear of the ARMADILLO. The cost for this improved triangle quality is increased geometric error, which can be a worthwhile trade-off if a large t_{min} is required by the application.

6.5.3 Sharp features

For meshes with distinct sharp features – edges (1D) and corners (0D) – the method described so far produces visible artifacts (Figure 6.6(b) and (c)). Clearly, what we want is that vertices on edges only move along the edge (1D constraint), while corners remain fixed in place (0D constraint). This requires solving an $3n \times 3n$ system, as in Section 6.5.1. We have found a very simple solution that achieves a similar effect, and only requires solving an $n \times n$ system; by reducing the weight on the *Laplacian* constraint of high curvature vertices (and using



(a) original (b) linear weights

(c) linear + tplane (d) linear + reduced Laplacian

Figure 6.6: Triangle optimization on an input mesh (a) with distinct sharp features. Using linear weights (b) breaks the sharp features, and using (2D) tangent plane constraints (c) cannot resolve the problem, since these features would require 1D or zero dimensional (positional) constraints. Instead, we reduce the weight on Laplacian constraints of feature vertices (d).

$\mathbf{W}_p = \mathbf{W}_{linear}$), feature vertices are no longer inclined to move, and their neighboring triangles will be optimized by their non-feature 1-ring vertices. Specifically, we use $\mathbf{W}_L = s\mathbf{I} - \mathbf{W}_{linear}$ in Equation 6.11 (when using weights $\mathbf{W}_{linear} \in [0,s]$) and scale all values to [0,1]. The result of this modification can be seen in Figure 6.6(d), where all feature vertices remain in place, while vertices in planar regions are moved towards their respective 1-ring centroids.

6.6 Mesh smoothing

Sorkine and Cohen-Or [SCO04], propose a procedure for smooth hole filling by fitting a thin-plate surface ($\mathbf{L}^2\mathbf{V}'_d = 0$, for infinitely large anchor weights), while simultaneously improving the triangle quality of the fitted mesh. Since only boundary vertices are constrained, the fitted surface is smooth without detail or features. Our framework (Section 6.4) can be easily adjusted to perform global mesh smoothing, optionally with feature preservation, simply by setting $\mathbf{f} = 0$, and adjusting the positional- ($\mathbf{W}_p$) and Laplacian ($\mathbf{W}_L$) weights. Also, we can perform simultaneous inner and outer fairing with $\mathbf{L}_u$, or outer fairing alone with $\mathbf{L}_c$ or $\mathbf{L}_{c\overline{\kappa}}$. Note that using the non-normalized $\mathbf{L}_{c\overline{\kappa}}$ discretization is similar to the curvature flow approach [DMSB99]: the weights on Laplacian smoothness constraints are scaled by discrete mean curvatures.

To make the magnitude and behavior of our smoothing procedure user-tuneable in a simple and intuitive way, we utilize and adjust three parameters.

- **Positional weights.** The weighting matrices $\mathbf{W}_{const}$, $\mathbf{W}_{linear}$ and $\mathbf{W}_{cdf}$ introduced in Section 6.5 are applied in the setting of mesh smoothing. These either uniformly smooth the mesh, shown in Figure 6.7(b) and (e), or attempt to retain features by placing more (positional) weight on high curvature vertices, as seen in Figure 6.7(c) and (f).

- **Scale factor.** While the scale factor s was set to $s = 1$ for all of Section 6.5, we can now use this scaling to adjust the overall amount of smoothing. Compare the top and bottom rows of Figure 6.7, where the scale factors differ by an order of magnitude.

- **Laplacian weights.** To further increase feature preservation, practically any function that reduces the weight $\mathbf{W}_L$ on Laplacian smoothness constraints of feature vertices can be applied. In this work we have experimented with Gaussian falloffs, and variants of the function used in Section 6.5.3 to automatically deduce $\mathbf{W}_L$ from the mean curvature cdf (see Figures 6.8 and 6.7(d) and (g)), but many other choices are conceivable, for example user-tagged feature vertices.

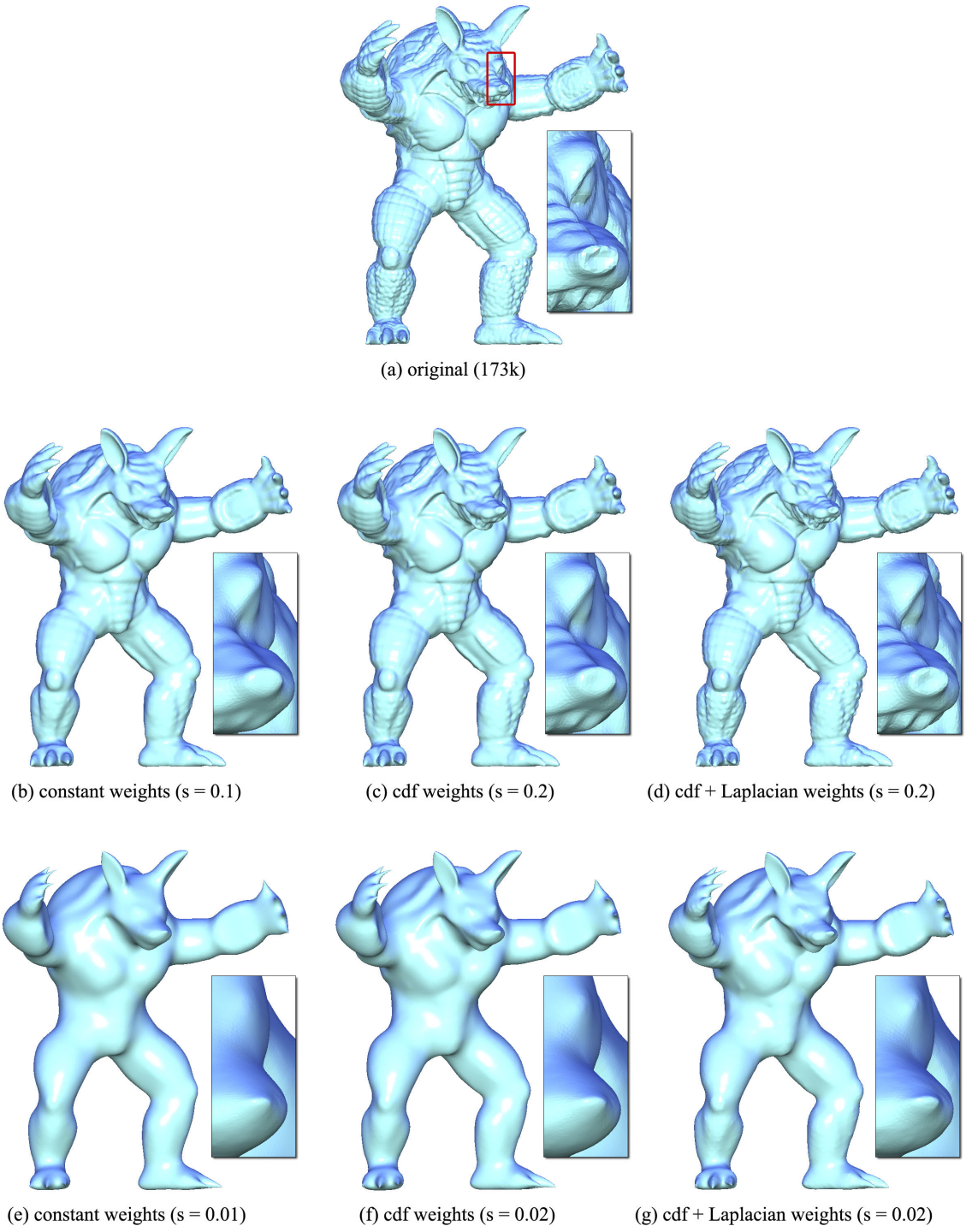


Figure 6.7: Comparison of smoothing weights. Columns 2 and 3 are generated using the weight functions introduced in Section 6.5, with different scaling factors s. The right column shows the effect of reducing the weights on the (Laplacian) smoothness constraints of high curvature vertices. All results in this figure were generated with $\mathbf{L} = \mathbf{L}_u$. In the pdf, please zoom in on the insets to see the differences.

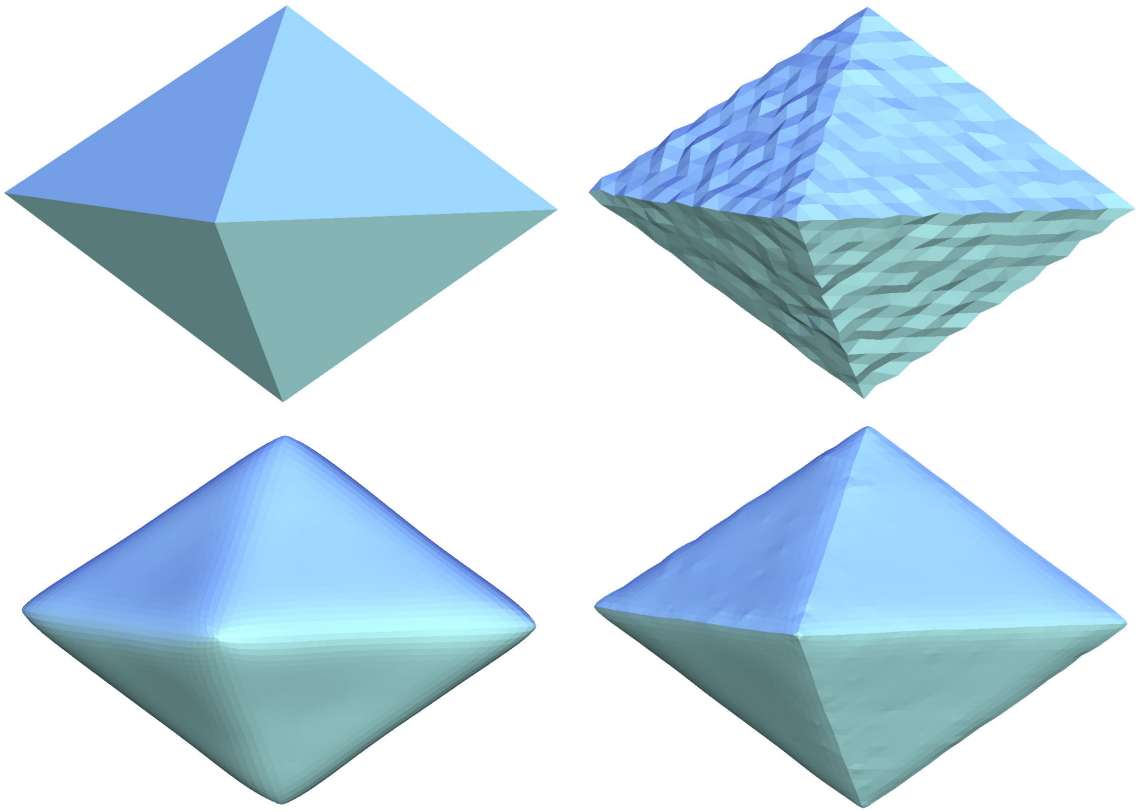
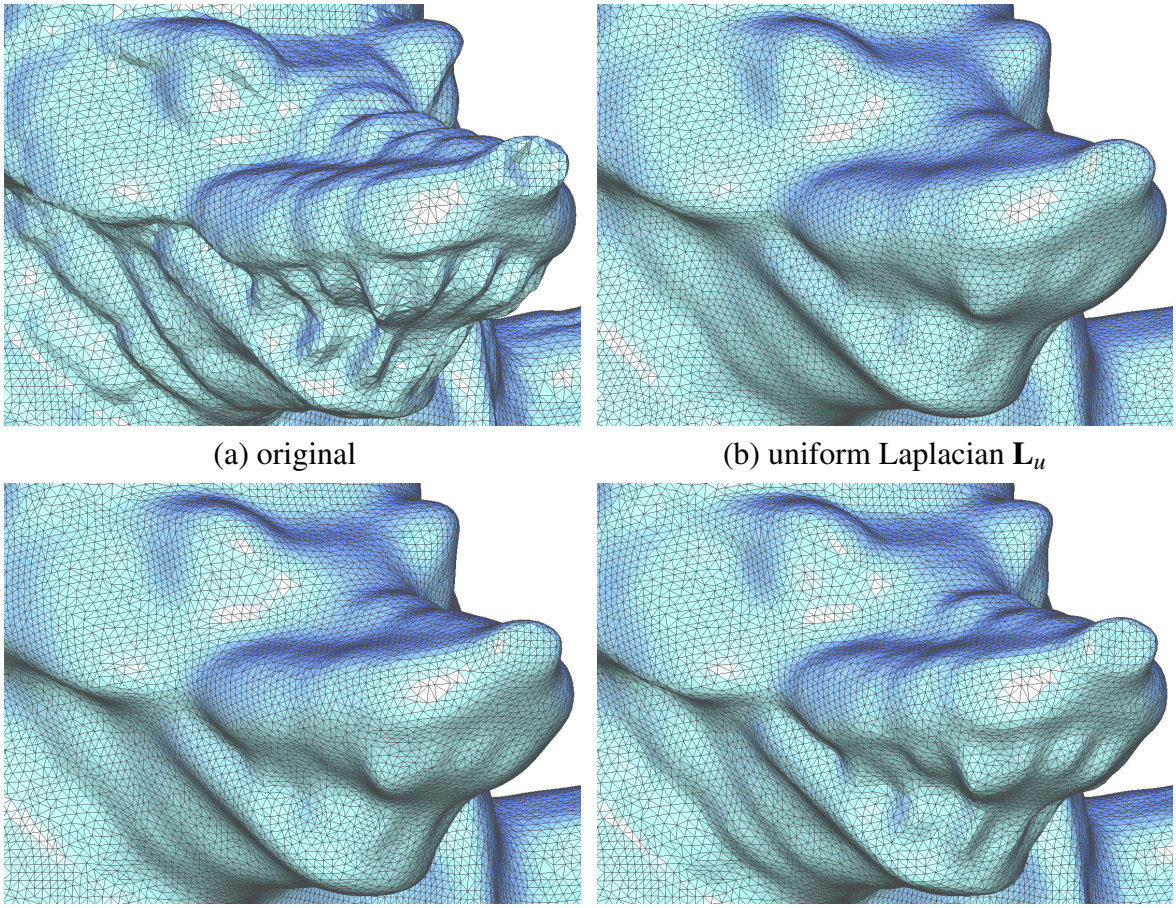


Figure 6.8: Smoothing a pyramid. Top row: original DOUBLEPYRAMID and the noisy version. Bottom row: Using $\mathbf{W}_p = \mathbf{W}_{linear}$ alone smoothes out sharp features, while additionally reducing the weights on Laplacian constraints of feature vertices recovers most of the original shape.

The results of using these three parameters are shown in Figure 6.7, where the full ARMADILLO model is smoothed. Note how varying the weights and the scale factor results in certain feature preservation, which can be seen in the closeup of the snout and teeth. For a detailed analysis of using $\mathbf{W}_{const}$ with varying s, see Section B. A typical example of feature preserving smoothing is shown in Figure 6.8. The original DOUBLEPYRAMID model in the upper left is contaminated with Gaussian noise, shown in the upper right. In the lower left the result of using $\mathbf{W}_{linear}$ is shown, where the edges are visible, albeit heavily smoothed. If we additionally use Laplacian weights $\mathbf{W}_L$, thereby relaxing Laplacian constraints as described above, we can successfully recover most of the original model.

The results shown in Figures 6.7 and 6.8 use $\mathbf{L} = \mathbf{L}_u$, resulting in a coupling of inner and outer fairness. While this works well for regularly sampled meshes, it has been pointed out by Desbrun et al. [DMSB99] that this no longer holds for irregularly sampled meshes, and unwanted object deformation can occur. This effect is somewhat reduced in our framework, thanks to positional constraints, yet



(a) original (b) uniform Laplacian $\mathbf{L}_u$

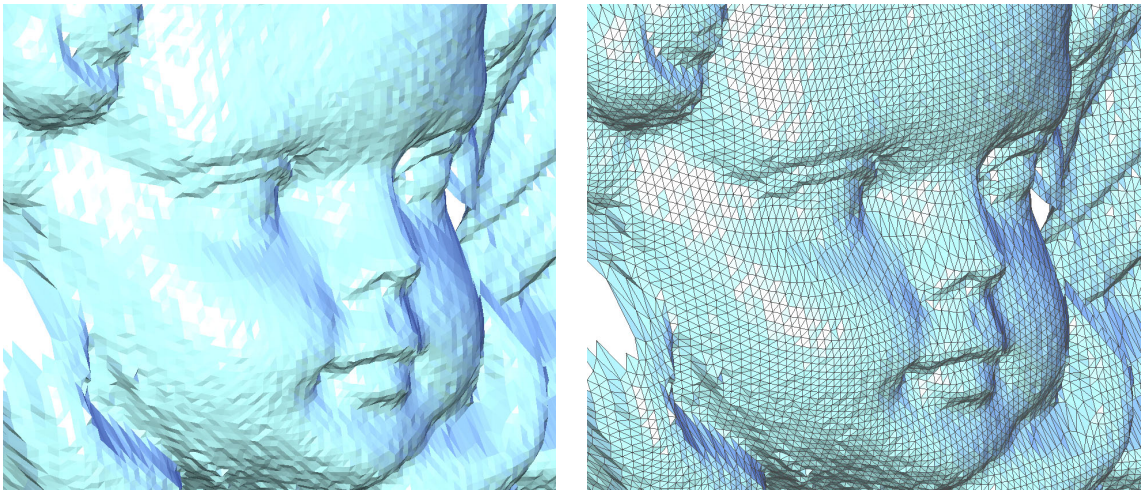
(c) cotangent Laplacian $\mathbf{L}_c$ (d) weighted Laplacian $\mathbf{W}_L\mathbf{L}_c$

Figure 6.9: Comparison of uniform (b) and cotangent (c) discretization of the $\mathbf{L}$ matrix. In (d), Laplacian constraints $\mathbf{L}_c\mathbf{V}'_d = 0$ are relaxed on feature vertices.

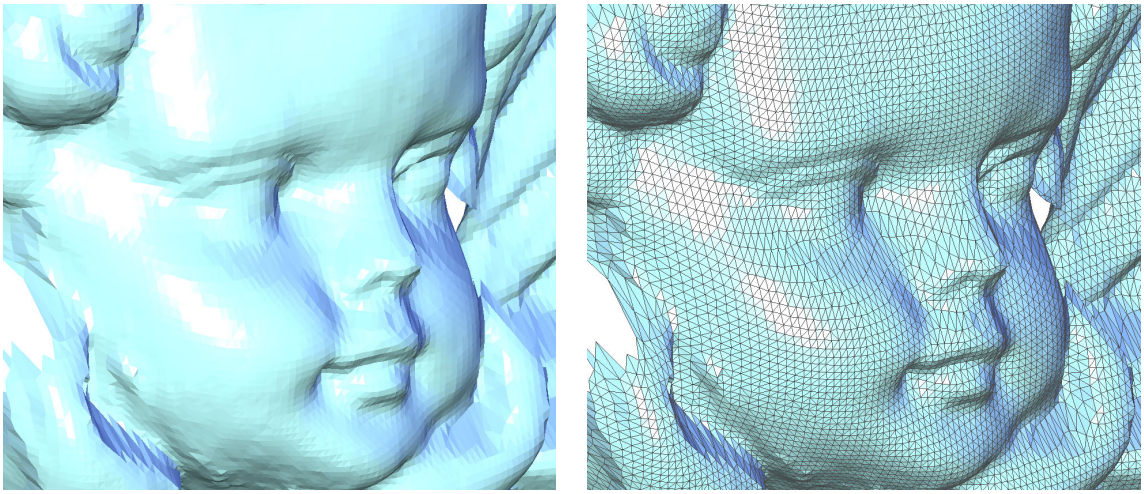
we can also decouple inner and outer fairness simply by using L_c or $L_{c\overline{\kappa}}$. As in other work on curvature flow, this moves each vertex along its normal, while leaving the tangential component unchanged. The effect is clearly visible, when comparing Figure 6.9 (b) and (c); while the mesh in (b) has nicely shaped triangles, the triangle shapes in (c) mostly reflect the original triangulation in a smoothed version. Note that $\mathbf{L}_c$ can also be coupled with the feature preserving Laplacian weighting matrix $\mathbf{W}_L$ (Figure 6.9(d)). For an analysis of the least-squares smoothing algorithm see appendix B.

6.7 Discussion

Our framework can optimize triangle shapes and smooth meshes with results similar to existing methods, but has the advantage that the solution is well-defined and



(a) original

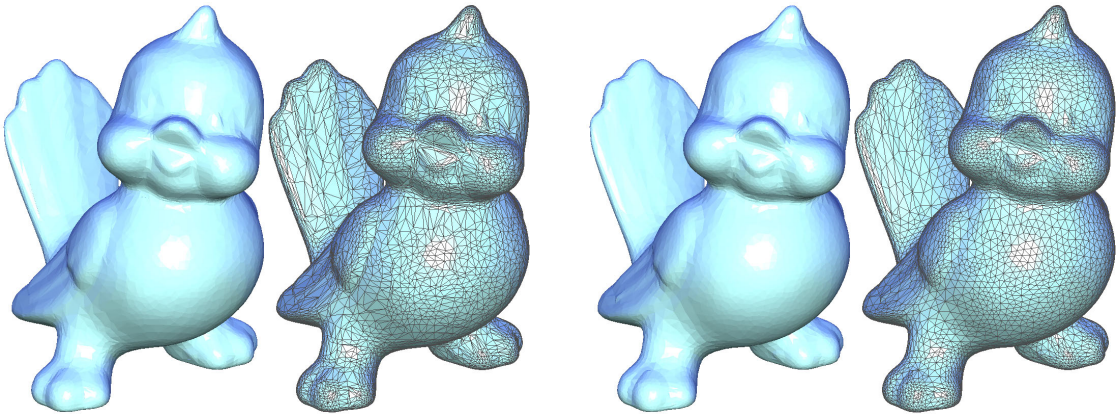


(b) smoothed

Figure 6.10: Smoothing the noisy ANGEL dataset, using $\mathbf{W}_{cdf}$ and $\mathbf{W}_L\mathbf{L}_c$.

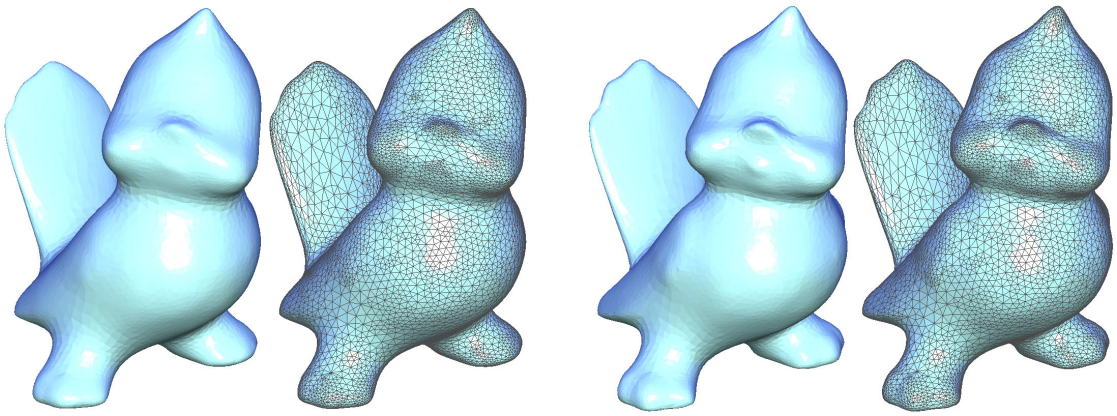
can be computed using optimized sparse linear solvers [Tol03]. Timing our algorithm amounts to measuring the time for the factorization of the system matrix, since this part makes up approximately 70-80% of the entire runtime. On an Intel Centrino Duo with 2.0 GHz we can factorize the $n \times n$ system matrix ($3n \times 3n$ for tangent plane constraints) for meshes with 173k, 43k and 17k vertices in 27s(43s), 3s(25s) and 1s(6s) respectively.

In addition, setting specific least-squares weights allows optimizing the results with respect to the particular model and the application at hand. The parameters we expose allow either trading off geometric error for triangle quality, or determine the intensity of the smoothing operation, including the amount of detail preservation. While the user may decide which weights and parameters to chose,



(a) original

(b) triangle optimization with $\mathbf{W}_{cdf}$



(c) smoothing with $\mathbf{W}_{cdf}$ and $\mathbf{L}_u$ (inner and outer fairness)

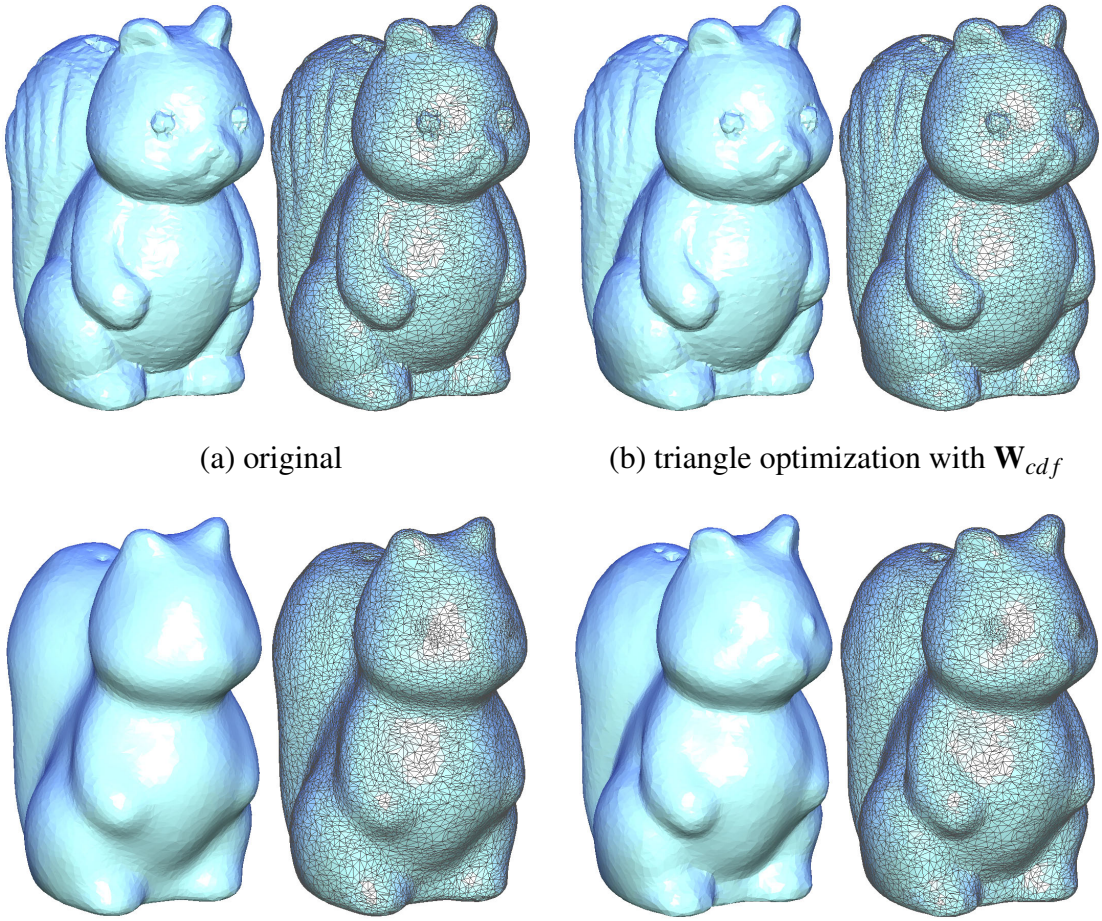
(d) feature preserving smoothing with $\mathbf{W}_{cdf}$ and $\mathbf{W}_L\mathbf{L}_u$ (inner and outer fairness)

Figure 6.11: TWEETY results.

we suggest the following combinations.

- **Smooth models.** Models, for which the mean curvature distribution is fairly smooth, such as the ARMADILLO model used throughout this chapter, generally benefit from using $\mathbf{W}_{cdf}$ for triangle shape optimization, since details in mid-range curvature regions are retained, the overall geometric error is low, and it produces a good distribution of triangle shapes. For smoothing, $\mathbf{W}_{cdf}$ nicely preserves detail in high curvature regions to a certain degree, which can be improved by additionally reducing weights on specific Laplacian smoothness constraints with $\mathbf{W}_L$.

- **CAD models.** CAD models, or models with mostly flat surfaces and some sharp features, are weighted with $\mathbf{W}_{linear}$ in Figures 6.6 and 6.8. Using $\mathbf{W}_{cdf}$ has little effect in these cases, since the associated cdf has a high



(a) original

(b) triangle optimization with $\mathbf{W}_{cdf}$

(c) smoothing with $\mathbf{W}_{cdf}$ and $\mathbf{L}_{c\overline{\kappa}}$

(outer fairness only, inner fairness untouched)

(d) feature preserving smoothing with $\mathbf{W}_{cdf}$ and $\mathbf{W}_L\mathbf{L}_c$
(outer fairness only, inner fairness untouched)

Figure 6.12: SQUIRREL results.

weight value for the numerous low (or zero) curvature vertices. This fixes these vertices in place, where they should actually be allowed to move freely in their tangent plane. The situation is similar for the smoothing application; here, we wish to de-noise the planar regions, thus necessitating a low weight on the positional constraints of these vertices, while holding feature curves and points in place, again making $\mathbf{W}_{linear}$ the better choice.

We ran the algorithm on some real world noisy data, and a few more meshes with bad triangle shapes. For the ANGEL dataset, acquired using 3D photography [BP98], using $\mathbf{W}_{cdf}$ and $\mathbf{W}_L\mathbf{L}_c$ succeeds in removing the noise while retaining the prominent features around the nose and eyes (Figure 6.10). The improvement in triangle quality on the TWEETY model in Figure 6.11 is clearly visible, while a

significant change in (flat) shading is not. Figures 6.11 and 6.12 illustrate the components of our algorithm: the original mesh (a) can be optimized with respect to triangle shapes (b) and adaptively smoothed with curvature dependent positional weights $\mathbf{W}_{cdf}$ (c) and reduced Laplacian weights $\mathbf{W}_L$ (d). Note that while in Figures 6.11(c),(d) both inner and outer fairness is improved, Figures 6.12(c),(d) leave inner fairness untouched, since $\mathbf{L}_{c\overline{\kappa}}$ and $\mathbf{L}_c$ decouple the fairness criteria.

We have not explicitly treated the issue of volume preservation, but the simple fact that the original mesh geometry is a substantial part of the result, and can be granularly controlled with $\mathbf{W}_p$, reduces shrinkage significantly.

We would like to stress the fact that all these optimizations based on adjusting the weight matrices $\mathbf{W}_p$ and $\mathbf{W}_L$ have no adverse effect on timing. It is conceivable that better feature detection could steer the weighting, leading in turn to better feature preservation, again at no additional cost (in the optimization computation). Consequently, we are interested in improved automated feature detection in our ongoing work. Other potential avenues for future work are volume preservation by analyzing (and adjusting) the spectral properties of the system matrix, and preservation of higher order discrete surface properties.

Chapter 7

Conclusion

Creating 3D shapes and characters from scratch is still an inherently difficult task. We think of this as the human/computer I/O problem: digital computers are excellent in both input (mouse, keyboard, video-in, etc.) as well as output (monitor, printer, projector, etc.) of data, while, at least to this day, humans are only equipped with a *video-in* (= visual sensory system), yet lack an appropriate *video-out* interface to digital computers. It is therefore nontrivial to convey a *mental model* of shape – that is, transfer the mental model into a format that can be digitally processed and displayed using commodity computer/graphics hardware. Many previous attempts rely on the fact that the user has some domain knowledge and/or is familiar (to a certain degree) with the intricate mathematical subtleties of the modeling tool, thus rendering these systems unusable for inexperienced users. The work presented in this dissertation eases the use of 3D modeling tools for first-time users, and expands the possibilities for experienced modelers. By utilizing various abstractions, such as silhouettes, ridges, ravines, and other general surface curves – and their 2D projections – we have strived to make our modeling interfaces feel more like traditional 2D painting, sketching, and manipulation. We like to think of our abstractions as *flexible handles*, as described in Chapter 1.

7.1 Contributions

Designing Freeform Surfaces with 3D Curves allows creating initial model components by sketching the silhouette onto an empty canvas, adding connected components by extrusion, and changing the topology by tunneling and merging. The model can subsequently be modified by either adding / removing / smoothing control curves, changing curve properties, or deforming existing curves. We think of these curves as flexible handles, as mentioned in the introduction (Chapter 1). The underlying discrete polyline and triangle-mesh representations are hidden from the user, and while our curve and surface optimization algorithms rely on this representation, the interface does not. Overall, this interface bridges the gap between traditional 3D modeling tools, such as parametric patches and sub-

division surfaces, where initial patch layout can be problematic, and sketch-based modeling systems, where detailed, careful editing of initial surface components has not yet been fully explored. To achieve real-time response for reasonably sized meshes, we have implemented fast curve and surface optimization algorithms based on minimizing discrete differential properties of the curves and surfaces. The surface construction and curve deformation methods we present utilize recent insights in the fields of discrete differential geometry and numerical linear algebra. These insights had previously been used in the context of mesh editing. Our system is the first to apply contemporary discrete differential geometry to 3D model creation *from scratch*.

Feature Oversketching and Manipulation enables simple modification of existing meshes by leveraging the human index of possible shapes [HS97]. Asking the user to generate a specific shape by transforming a vertex, or even sets of vertices, is a potentially ill-posed problem. On the other hand, our silhouette editing tool, another instance of our flexible handle metaphor, leverages the vast cultural knowledge on shapes and their silhouettes. This allows any user to modify the silhouette of a 3D model, which we see as a first step towards a human *video-out*. Furthermore, we extend the existing Laplacian surface editing framework [SLCO$^+$04] to accommodate feature scaling, with which the user can create ridges and ravines anywhere on the surface. Finally, we automate the entire process of silhouette oversketching: the user only needs to find an appropriate 2D view of the 3D model and place a sketch, thereby hinting at the desired silhouette modification. Using partial matching and guided mesh segmentation techniques, our system automatically finds the model silhouette segment corresponding to the user-sketch, as well as the subregion on the mesh that undergoes deformation. This system comes very close to mimicking 2D oversketching, while presenting a smooth 3D model edit as a result.

Laplacian Mesh Optimization exposes a small set of intuitive parameters, with which the user can perform – local or global – detail preserving triangle shape optimization or mesh smoothing. The proposed framework (Section 6.4) generalizes both least-squares meshes [SCO04] (also known as geometry-aware bases [SCOIT05]), as well as our local detail preserving triangle shape optimization described in Chapter 4. To ensure a good trade-off between inner fairness and geometric error w.r.t. to the original mesh, we introduce various weighting schemes for the positional constraints, which are modeled as transfer functions from discrete mean curvature to scalar weight, and show that simply using the cumulative distribution function of discrete mean curvature over the (sub)mesh is a good (and fully automatic) choice. Our mesh smoothing algorithm, also a special case of the general framework, formulates mesh smoothing as the unique solution to a least-squares problem. Since this formulation incorporates non-uniformly weighted positional constraints, as well as Laplacian smoothness constraints, it is easy to combine these weights, such that shrinking is minimal and prominent features are preserved.

7.2 Future directions

Character and Shape Animation. The most obvious extension we see, is utilizing our flexible handle metaphor for character and shape animation. One possibility would be to use the same set of sketches an illustrator uses to convey motion in static imagery [McC94]. One could also use 2D image space features to track animations from given sequences [JT95], similar to Bregler and co-workers [BLCD02]. In this setting, user sketches could be used to modify features and shapes in specific keyframes of the animation sequence, from which new 2D/3D animation sequences could be synthesized. A first foray into applying this idea to 3D editing is given by Kraevoy and co-workers [KSvdP07]. This principle could be extended to spatial keyframing [IMH05b]. We also see great potential in automatically generating skeletons and skin-weights (also known as *rigging*), and allowing the user to deform the skin directly, and indirectly by sketching skeletal animations. For recent developments in this field, see [BP07, SLSK07, YBS07, WSLG07, WPP07]. We are furthermore considering the use of multi-body hierarchies for such an animation and skinning tool.

Deforming and Sketching other Features. Currently, our tools allow the user to deform silhouettes, and create ridges and ravines. It seems straightforward to let the user deform these ridges and ravines as well, and this is possible if the user modifies the view such that a ridge becomes a silhouette. Unfortunately, since there is already much ambiguity in the user sketch, adding to the selection of possible modification handles in our automatic modeling tool makes things worse. Instead, we would like to make only the most prominent and likely features from the sets of silhouettes and surface creases modifiable, thereby creating a tractable problem with little to no ambiguity, yet a good range of flexibility. Other ideas include sketching features onto the surface, such as cross hatches (e.g. [HZ00]) to indicate directions of principle curvature, and optimize the surface such that it incorporates these features. Perhaps a quad-mesh based surface representation would be more suitable for this task [LPW+06, PLW+07].

Multiresolution Editing. To reduce the computational complexity of our surface optimization algorithm, we would like to include multiresolution editing paradigms to the system. We are considering the possibility of non-conforming mesh hierarchies: at runtime, the user sketches a detailed mesh on top of the base mesh, which is stored in local coordinate frames – in the spirit of traditional multiresolution modeling [ZSS97, KCVS98] – yet is not connected to the base mesh. edits and deformations can then be carried out on each level individually, without modifying the other levels. Once the user is done with the modeling session, the mesh is stitched together, starting from the base mesh all the way up the hierarchy. This method was considered for the model creation tool presented in Chapter 3, but was dropped due to time constraints. Nevertheless, we would like to revisit the idea.

Application to other Curve and Surface Representations. Currently, our representations are all mesh and polyline based. It might be interesting to apply the interface principles and the flexible handle metaphor to other surface representations, such as points [ABCO+03, ZPKG02, AA06], implicit surfaces [SWSJ05] and space-warps [DE03, LKG+03].

Appendix A

A Note on Boundary Constraints for Linear Variational Surface Design

In the process of creating FiberMesh [NISA07] (see Chapter 3), we have implemented both Least-Squares Meshes (LSM) [SCO04, SCOIT05] and the Botsch/Kobbelt (BK) framework [BK04a] for fair surface computation. In 2D MATLAB implementations, this works fine as long as the positional constraints (also known as the *anchors*) are not collinear, as we can see from the examples in Figure A.1.

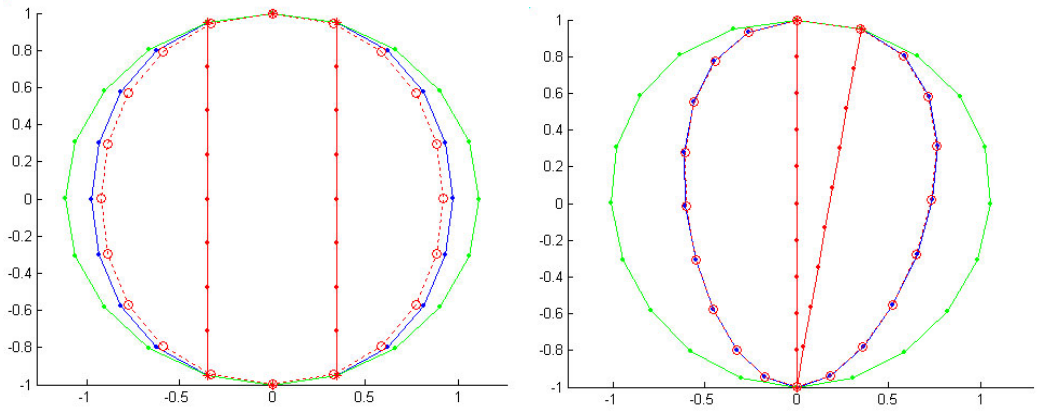


Figure A.1: MATLAB examples where anchor points (red crosses) are not collinear. The red dotted line is the LSM solution, while the red, blue and green solid lines are the BK solutions for $k = 1$, $k = 2$ and $k = 3$ respectively.

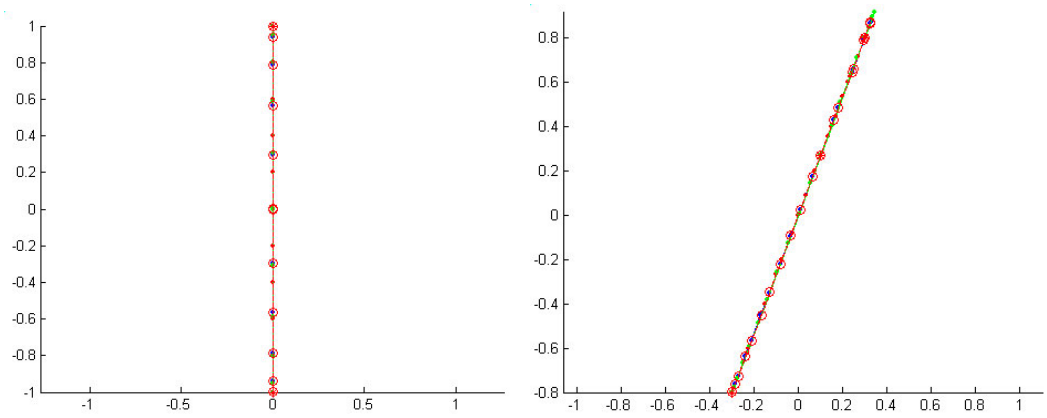


Figure A.2: Degenerate MATLAB examples with two (left) or 3 collinear anchors (right). Color coding as in Figure A.1

When the anchors lie in a linear subspace (such as a line in 2D), all other vertices of this shape are forced to lie in this subspace as well (see Figure A.2). Intuitively, this means that all vertices are computed as affine combinations of the anchor vertices (see Figure A.3). In the following, we show that the linear solutions of [SCO04] and [BK04a] both have this property, which makes them unsuitable for *inflating* a mesh defined by a planar curve: in a 2D sketching tool we will generally have constraints that lie on a plane (the two anchors in Figure A.2 can be seen as the cross section of a planar 2D sketch). Adding positional constraints perpendicular to the sketch plane works, which is analogous to implementing higher order boundary constraints, yet requires manual placement, which is a nontrivial task in a 3D modeling environment (see Figure A.1 for the cross section analogy).

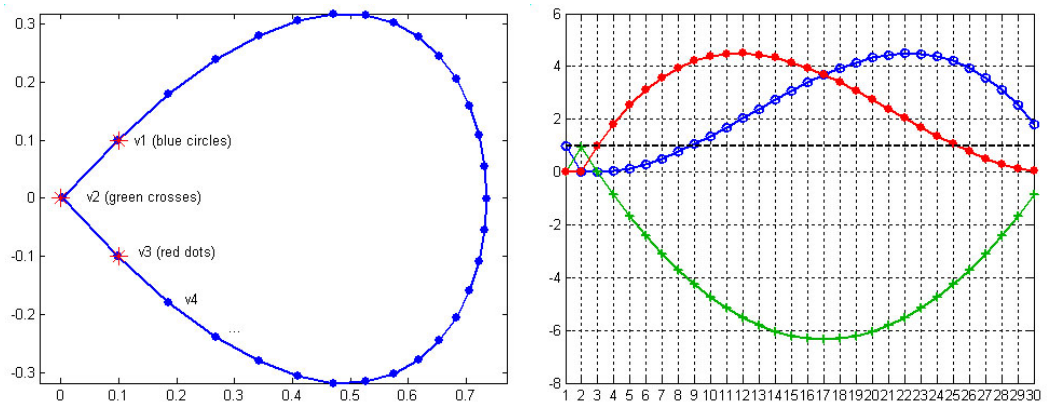


Figure A.3: MATLAB LSM example with three adjacent anchor points. The weights are high, therefore the smoothness constraints cannot be met (right, first three indices). Note that the horizontal dashed line is the sum of the three bases.

A.1 Least-squares meshes

Suppose we have connected mesh topology M with n vertices. Denote by $\mathbf{L}$ the Laplacian matrix with $\mathbf{L}(1,\ldots,1)_n^T = \mathbf{0}_n$; all rows of $\mathbf{L}$ sum up to zero. Some k vertices are tagged as anchor vertices; w.l.o.g. assume that the anchors are vertices $\{1,2,\ldots,k\}$. In [SCO04] the positions of all vertices are obtained as the solution of the following least-squares problem

$$\begin{aligned} &\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \left(\begin{array}{c|c} \mathbf{L} & \\ \hline \mathbf{I}_{k\times k} & \mathbf{0} \end{array} \right) \mathbf{x} - \begin{pmatrix} \mathbf{0}_n \\ \mathbf{x}'_k \end{pmatrix} \right\|^2 \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{A}\mathbf{x} - \mathbf{b} \right\|^2, \end{aligned} \tag{A.1}$$

where $\mathbf{I}_{k\times k}$ is the k by k identity matrix and $\mathbf{x}'_k = (x'_1,\ldots,x'_k)^T$ is the vector of prescribed anchor positions. While the matrix $\mathbf{L}$ has rank $n-1$, the rectangular matrix $\mathbf{A}$ involved in the least-squares problem above has full column rank after adding a single positional constraint (or more). Therefore, we can write the least-squares solution explicitly as

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}. \tag{A.2}$$

Observation A.1.1. *The solution to Equation A.2 is an affine combination of the anchor positions* $\mathbf{x}'_k$.

Proof. First we expose the structure of the vector $\mathbf{A}^T\mathbf{b}$ of length n, which has anchor positions in the first k entries, and zeros elsewhere

$$\mathbf{A}^T\mathbf{b} = \left(\mathbf{L}^T \left| \begin{array}{c} \mathbf{I}_{k\times k} \\ \hline \mathbf{0}_{(n-k)\times k} \end{array} \right. \right) \begin{pmatrix} \mathbf{0}_n \\ \mathbf{x}'_k \end{pmatrix} = \begin{pmatrix} \mathbf{x}'_k \\ \mathbf{0}_{n-k} \end{pmatrix}. \tag{A.3}$$

It is easy to see that:

$$\mathbf{A}^T\mathbf{A} = \left(\mathbf{L}^T\mathbf{L} + \left(\begin{array}{c|c} \mathbf{I}_{k\times k} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right) \right). \tag{A.4}$$

Denote $\mathbf{1}_n = (1,\ldots,1)_n^T$. Since $\mathbf{L}\mathbf{1}_n = \mathbf{0}_n$, also $\mathbf{L}^T\mathbf{L}\mathbf{1}_n = \mathbf{0}_n$. Thus:

$$(\mathbf{A}^T\mathbf{A})\mathbf{1}_n = \left(\mathbf{L}^T\mathbf{L} + \left(\begin{array}{c|c} \mathbf{I}_{k\times k} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right) \right) \mathbf{1}_n = \begin{pmatrix} \mathbf{1}_k \\ \mathbf{0}_{n-k} \end{pmatrix} \tag{A.5}$$

Let us multiply the last equation by $(\mathbf{A}^T\mathbf{A})^{-1}$ on the left side:

$$\begin{aligned} (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{A})\mathbf{1}_n &= (\mathbf{A}^T\mathbf{A})^{-1} \begin{pmatrix} \mathbf{1}_k \\ \mathbf{0}_{n-k} \end{pmatrix} \\ \mathbf{1}_n &= (\mathbf{A}^T\mathbf{A})^{-1} \begin{pmatrix} \mathbf{1}_k \\ \mathbf{0}_{n-k} \end{pmatrix} \end{aligned} \tag{A.6}$$

The last equality tells us that the sum of the first k elements of each row of $(\mathbf{A}^T\mathbf{A})^{-1}$ is equal to 1. These multiply the first k elements of $\mathbf{A}^T\mathbf{b}$ in Equation A.2, which shows that the solution $\mathbf{x}$ is an affine combination of the anchors $\mathbf{x}'$.

□

A.2 Botsch/Kobbelt bases

In [BK04a] the anchors are interpolated instead of approximated. Therefore, we are dealing with disjoint sets of vertices (in contrast to [SCO04] where the anchors are a subset of all vertices). We denote the set of k fixed vertices as $\mathbf{F}$, the set of n free vertices as $\mathbf{P}$ and $N = n + k$. Note that the fixed vertices are only required to enforce boundary constraints around the free region, since their positions are known a priori (their smoothness constraints are also dropped from the system, as we will see below).

First, the $N \times N$ Laplacian matrix $\mathbf{L}$ is raised to the desired power p (where $p = 1$ for the membrane, $p = 2$ for the thin-plate and $p = 3$ for the minimum variation solution). W.l.o.g. assume that the k fixed vertices have indices $1, \ldots, k$ in the system, and the free vertices have indices $k+1, \ldots, N$, then for the smoothness constraint we have

$$\mathbf{L}^p \begin{pmatrix} \mathbf{F} \\ \mathbf{P} \end{pmatrix} = \mathbf{0}. \tag{A.7}$$

The smoothness constraints on vertices in $\mathbf{F}$ are removed from the system by omitting the first k rows of $\mathbf{L}^p$, resulting in the $n \times N$ matrix Δ^p. By adding the boundary conditions back to the system, we arrive at the formulation used in [BK04a]

$$\left(\begin{array}{c|c} \Delta^p & \\ \hline \mathbf{I}_{k\times k} & \mathbf{0} \end{array}\right) \begin{pmatrix} \mathbf{F} \\ \mathbf{P} \end{pmatrix} = \left(\begin{array}{c|c} \Delta^p_{\mathbf{F}} & \Delta^p_{\mathbf{P}} \\ \hline \mathbf{I}_{k\times k} & \mathbf{0} \end{array}\right) \begin{pmatrix} \mathbf{F} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{F} \end{pmatrix}. \tag{A.8}$$

This is for notational convenience only. Here, we use $\Delta^p_{\mathbf{F}}$ for the upper left $n \times k$ matrix and $\Delta^p_{\mathbf{P}}$ for the upper right $n \times n$ matrix. By dropping the bottom k rows of Equation A.8 and rearranging we get

$$\mathbf{P} = \left(\Delta^p_{\mathbf{P}}\right)^{-1} \left(-\Delta^p_{\mathbf{F}}\right) \mathbf{F} = \mathbf{B}_{BK} \mathbf{F}. \tag{A.9}$$

The $n \times k$ matrix $\mathbf{B}_{BK}$ can be interpreted as the matrix of basis functions. This basis suffers from the subspace property, which is equivalent to saying that the basis vectors (the columns of $\mathbf{B}_{BK}$) sum to 1.

Observation A.2.1.

$$\left(\Delta^p_{\mathbf{P}}\right)^{-1} \left(-\Delta^p_{\mathbf{F}}\right) \mathbf{1}_k = \mathbf{1}_n. \tag{A.10}$$

Proof. We know that the rows of Δ^p sum to zero, since these are rows of the original Laplacian matrix. This can also be written as

$$\left(\Delta^p_{\mathbf{F}}\right) \mathbf{1}_k + \left(\Delta^p_{\mathbf{P}}\right) \mathbf{1}_n = \mathbf{0}_n. \tag{A.11}$$

Rearranging and multiplying from the left with $(\Delta^p_{\mathbf{P}})^{-1}$ yields

$$\mathbf{1}_n = \left(\Delta^p_{\mathbf{P}}\right)^{-1} \left(-\Delta^p_{\mathbf{F}}\right) \mathbf{1}_k \tag{A.12}$$

which shows that the bases in [BK04a] sum to 1 on each mesh vertex.

□

Appendix B

Analysis of the Least-Squares Smoothing Algorithm

In this Section we outline the connection to signal theory, specifically Wiener filters [Wie49], and Tikhonov regularization [Tik63].

Let $\mathcal{M}$ be a triangular mesh with n vertices. Denote by $\mathbf{x} = (x_1, \ldots, x_n)^T$ the x-coordinates of $\mathcal{M}$'s vertices (similarly for $\mathbf{y}$ and $\mathbf{z}$). A widely used class of Laplacian operators L on meshes have the general form:

$$L(x_i) = \sum_{j \in N(i)} w_{ij}(x_i - x_j),$$

where $N(i)$ is the set of 1-ring vertices. The Laplacian can be represented by an n-by-n matrix L, such that the off-diagonal entries are w_{ij} when there is an edge between i and j and zero otherwise; the diagonal entries are simply the row sums. Prominent weighting schemes are the uniform weights $w_{ij} = 1$ and the cotangent weights [PP93a, DMSB99]; the latter allow better approximation of the surface Laplace-Beltrami operator. If the weighting scheme is symmetric and $w_{ij} > 0$, which we assume in the following (we use uniform weights), then L is symmetric positive semidefinite and has an orthonormal eigenbasis $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ with corresponding eigenvalues $0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_n$. The eigenvectors corresponding to small eigenvalues are smooth, low frequency functions, and those corresponding to large eigenvalues are highly oscillatory. In fact, when the mesh is a regular grid, the Laplacian eigenbasis (LEB) is the discrete cosine basis (DCB). The LEB is viewed as an extension of DCB for irregular meshes [Tau95]. Since the eigenbasis V is orthonormal, the mesh geometry can be represented in this basis as

$$\mathbf{x} = \sum_{i=1}^{n} a_{i,x} \mathbf{v}_i,$$

with coefficients $a_{i,x} = \mathbf{v}_i^T \mathbf{x}$, obtained by simple projection (same for the $\mathbf{y}$ and $\mathbf{z}$ functions). By truncating the high-frequency components, we can obtain a smoothed version of $\mathcal{M}$'s geometry:

$$\tilde{\mathbf{x}} = \sum_{i=1}^{k} (\mathbf{v}_i^T \mathbf{x})\, \mathbf{v}_i\,, \qquad k \ll n, \tag{B.1}$$

where the value of k determines how much high-frequency detail we want to preserve. Lévy [Lév06] shows several results of this smoothing approach. Computing the spectral decomposition is, however, very time-consuming and may be impractical for typical detailed meshes. In his seminal work, Taubin [Tau95] has shown that an approximation of this smoothing result can be obtained by explicitly iterating $\mathbf{x}_{m+1} = (I - \lambda L)\mathbf{x}_m$. Such filtering, as well as the implicit fairing algorithm [DMSB99] leads to shrinking of the shape volume, and volume restoration steps must be interleaved with the smoothing steps. Our non-iterative smoothing approach presented in Chapter 6 realizes smoothing by minimizing the following energy functional:

$$\mathbf{x}_{\text{smooth}} = \underset{\mathbf{x}'}{\operatorname{argmin}} \left(\|L\mathbf{x}'\|^2 + w^2 \|\mathbf{x}' - \mathbf{x}\|^2 \right), \tag{B.2}$$

where $\mathbf{x}$ is the original mesh geometry. The magnitude of the weight w determines the amount of smoothing. The optimization (B.2) tries to simultaneously satisfy the smoothness condition (i.e. minimize the discrete Laplacian magnitude) and to retain the original mesh geometry. As a result, the smoothed mesh has negligible volume loss; the optimization can be solved very efficiently using sparse linear solvers.

The quadratic optimization problem (B.2) can be formalized as the least-squares solution of the following over-determined linear system $A\mathbf{x}' = \mathbf{b}$:

$$\left(\frac{L}{wI} \right) \mathbf{x}' = \left(\frac{\mathbf{0}_n}{w\mathbf{x}} \right),$$

where I is the n-by-n identity matrix and $\mathbf{0}_n$ denotes the column vector of n zeros. The least-squares solution of the system above can be obtained using the normal equations $\mathbf{x}' = (A^T A)^{-1} A^T \mathbf{b}$ since A has full column rank:

$$\begin{aligned} \mathbf{x}' &= \left(\left(\, L \mid wI \, \right) \left(\frac{L}{wI} \right) \right)^{-1} \left(\, L \mid wI \, \right) \left(\frac{\mathbf{0}_n}{w\mathbf{x}} \right) = \\ &= \left(L^2 + w^2 I \right)^{-1} \left(w^2 \mathbf{x} \right). \end{aligned} \tag{B.3}$$

This simplified formula was obtained by considering that $L = L^T$ and performing block-matrix multiplication. Note that Equation B.3 represents an instance of Tikhonov regularization [Tik63].

What can we say about the solution $\mathbf{x}'$? Let us analyze the matrix in Eq. (B.3), that is, $(L^2 + w^2 I)^{-1}$. We know that L has the spectral decomposition $L = V \cdot$

$\mathrm{diag}(\lambda_1,\ldots,\lambda_n)\cdot V^T$, where V is orthogonal; the matrix L^2 has the same eigenvectors and squared eigenvalues: $L^2 = V\cdot\mathrm{diag}(\lambda_1^2,\ldots,\lambda_n^2)\cdot V^T$. Interestingly, the matrix $B = \left(L^2 + w^2 I\right)$ also has the same eigenvectors, and its eigenvalues are incremented by w^2:

$$B\,\mathbf{v}_i = \left(L^2 + w^2 I\right)\mathbf{v}_i = L^2\,\mathbf{v}_i + w^2\,\mathbf{v}_i = \lambda_i^2\,\mathbf{v}_i + w^2\,\mathbf{v}_i = (\lambda_i^2 + w^2)\,\mathbf{v}_i\,.$$

The matrix $B^{-1} = V\cdot\mathrm{diag}\left(\frac{1}{\lambda_1^2+w^2},\ldots,\frac{1}{\lambda_n^2+w^2}\right)\cdot V^T$ exists since $w>0$. Therefore the solution $\mathbf{x}' = B^{-1}\,w^2\,\mathbf{x}$ in Eq. (B.3) can be written as follows:

$$\mathbf{x}' \;=\; V\cdot diag\left(\frac{1}{\lambda_i^2+w^2}\right)\begin{pmatrix} w^2\,\mathbf{v}_1^T\mathbf{x} \\ \vdots \\ w^2\,\mathbf{v}_n^T\mathbf{x}\end{pmatrix} = \sum_{i=1}^{n}\frac{w^2}{\lambda_i^2+w^2}(\mathbf{v}_i^T\mathbf{x})\,\mathbf{v}_i\,.$$

This shows that the smoothed solution $\mathbf{x}'$ is a combination of the Laplacian eigenvectors with modified coefficients $\frac{w^2}{\lambda_i^2+w^2}\,\mathbf{v}_i^T\mathbf{x}$, where the $\frac{w^2}{\lambda_i^2+w^2}$ are the Wiener filter coefficients [Wie49] (see [PSZ01] for another application of Wiener filter coefficients). When w approaches infinity, we have $\mathbf{x}' = \mathbf{x}$ because

$$\lim_{w\to\infty}\frac{w^2}{\lambda_i^2+w^2} = \lim_{w\to\infty}\frac{1}{\lambda_i^2/w^2+1} = 1\,.$$

On the other hand, when w is small, the low-frequency components are preserved and the high-frequency components are attenuated: for small i, λ_i is very small (in the order of $\Theta(n^{-1.5})$ for typical meshes, see [GM00]) and thus $w^2/(\lambda_i^2+w^2)\approx 1$; in contrast, the last λ_i are relatively large ($\lambda_i > 1$ and in particular, λ_n is in the order of twice the maximal vertex degree of the mesh), and therefore $w^2/(\lambda_i^2+w^2)\ll 1$. The smaller w is, the more the high-frequency components are attenuated. This is indeed the behavior observed in the practical experiments in Chapter 6.

This analysis shows that the optimization of a particular quadratic energy can lead to a mesh smoothing algorithm whose properties are very similar to classic spectral smoothing, while the computational effort is significantly lower. Figure B.1 visually supports our results by comparing the smoothing outcomes of both schemes.

Figure B.1: Comparison between L-eigenbasis truncation and least-squares smoothing, denoted as SPECTRAL and LS, respectively (uniformly-weighted Laplacian was used). The value of k denotes the number of eigenfunctions used in the spectral reconstruction (see Eq. (B.1)), and w is the weight used for least-squares smoothing. It can be observed that very similar smoothing results are obtained.

Bibliography

[AA06] Anders Adamson and Marc Alexa. Point-sampled cell complexes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 671–680, 2006.

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[ACP03] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph.*, 22(3):587–594, 2003.

[ACWK06] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: constant volume modeling. *Grap. Models*, 68(4):324–332, 2006.

[AECDI03] Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. Isotropic surface remeshing. In *SMI '03: Proceedings of the Shape Modeling International 2003*, 2003.

[Ale03] Marc Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2):105–114, 2003.

[AMD02] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 347–354, 2002.

[AUGA05] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. Part of the state-of-the-art report of the AIM@SHAPE EU network, 2005.

[Bær02] J. Andreas Bærentzen. *Manipulation of volumetric solids with applications to sculpting*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2002.

[Bar84] Alan H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, 1984.

[Bau94] Thomas Baudel. A mark-based interaction paradigm for free-hand drawing. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 185–192, 1994.

[BCCD04] David Bourguignon, Raphaëlle Chaine, Marie-Paule Cani, and George Drettakis. Relief: A modeling by drawing tool. In *First Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 151–160, 2004.

[BK04a] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[BK04b] Mario Botsch and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 185–192, 2004.

[BK05] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005.

[BLCD02] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. Turning to the masters: motion capturing cartoons. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 399–407, 2002.

[BLZ00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of SIGGRAPH 2000*, pages 113–120, July 2000.

[BP98] Jean-Yves Bouguet and Pietro Perona. 3D photography on your desk. In *ICCV'98 proceedings*, pages 43–50, 1998.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), 2007. to appear.

[BPG06] Mario Botsch, Mark Pauly, and Markus Gross. PriMo: coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, pages 11–20, 2006.

[BS05] Alexander I. Bobenko and Peter Schroeder. Discrete Willmore flow. In *Eurographics Symposium on Geometry Processing*, pages 101–110, 2005.

[BS07] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 2007. To appear.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 109–116, 1990.

[BW99] E. Galin B. Wyvill, A. Guy. Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, 1999.

[Can86] John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[CC05] Chun-Yen Chen and Kuo-Young Cheng. A sharpness dependent filter for mesh smoothing. *Comput. Aided Geom. Des.*, 22(5):376–391, 2005.

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 257–266, 1991.

[CG97] Scott D. Cohen and Leonidas J. Guibas. Partial matching of planar polylines under similarity transformations. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[CHZ00] Jonathan M. Cohen, John F. Hughes, and Robert C. Zeleznik. Harold: a world made of drawings. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 83–90, 2000.

[Coq90] Sabine Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, 1990.

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[Dav04] Timothy A. Davis. UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004.

[DE03] Geoffrey Draper and Parris Egbert. A gestural interface to free-form deformation. In *Proceedings of Graphics Interface*, pages 113–120, 2003.

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

[dM07] 3ds Max. Autodesk, `http://www.autodesk.com/3dsmax`, 2007.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proccedings of ACM SIGGRAPH*, pages 317–324, 1999.

[do 76] Manfredo do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

[DP73] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a hne or its caricature. *The Canadian Cartographer*, (10(2)):112–122, 1973.

[EG86] Herbert Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 389–403, 1986.

[Far90] Gerald Farin. *Curves and surfaces for computer aided geometric design*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[FAT07] Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. Effective derivation of similarity transformations for implicit Laplacian mesh editing. *Computer Graphics Forum*, 26(1):34–45, 2007.

[FCG02] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Resolution adaptive volume sculpting. *Graphical Models (GMOD)*, 63:459–478, 2002.

[FDCO03] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, 22(3):950–953, 2003.

[Fel07] Carlos Felippa. Nonlinear finite element methods. `www.colorado.edu/engineering/CAS/courses.d/NFEM.d/`, 2007.

[Fie88] David A. Field. Laplacian smoothing and delaunay triangulations. *Communications in Applied Numerical Methods*, 4:702–712, 1988.

[FM03] Thomas-Peter Fries and Hermann G. Matthies. Classification and overview of meshfree methods. Technical report, TU Brunswick, Germany Nr. 2003-03, 2003.

[FRSS04] Timo Fleisch, Florian Rechel, Pedro Santos, and André Stork. Constraint stroke-based oversketching for 3D curves. In *First Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2004.

[Fuj95] Koji Fujiwara. Eigenvalues of Laplacians on a closed Riemannian manifold and its nets. In *Proc. AMS.*, pages 2585 – 2594, 1995.

[GDP+06] Eitan Grinspun, Mathieu Desbrun, Konrad Polthier, Peter Schröder, and Ari Stern. Discrete differential geometry: An applied introduction. SIGGRAPH 2006 Course Notes, `http://ddg.cs.columbia.edu`, 2006.

[GG01] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. A.K. Peters, 2001.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 267–274, 1991.

[GH95] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 359–368, 1995.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[Gin07] Chaim Gingold. SPORE's magic crayons. Game Developers Conference, 2007.

[GM97] Sarah Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics, 1997.

[GM00] Stephen Guattery and Gary L. Miller. Graph embeddings and Laplacian eigenvalues. *SIAM Journal on Matrix Analysis and Applications*, 21(3):703–723, 2000.

[GSS99] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334, 1999.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26, 1993.

[Her99] Aaron Hertzmann. Introduction to 3D non-photorealistic rendering: Silhouettes and outlines. In *Non-Photorealistic Rendering. SIGGRAPH 99 Course Notes.*, 1999.

[HHK92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 177–184, 1992.

[HJBJ+96] Adam Hoover, Gillian Jean-Baptiste, Xiaoyi Jiang, Patrick J. Flynn, Horst Bunke, Dmitry B. Goldgof, Kevin K. Bowyer, David W. Eggert, Andrew W. Fitzgibbon, and Robert B. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):673–689, 1996.

[HP05] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum (Eurographics Proceedings)*, 23(3):101–110, 2005.

[HP07] Klaus Hildebrandt and Konrad Polthier. Constraint-based fairing of surface meshes. In *Eurographics/ACM Symposium on Geometry Processing*, 2007.

[HS97] Donald D. Hoffman and Manish Singh. Salience of visual parts. *Cognition*, (63):29–78, 1997.

[HSL+06] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134, 2006.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of SIGGRAPH 2000*, pages 517–526, 2000.

[IH03] Takeo Igarashi and John F. Hughes. Smooth meshes for sketch-based freeform modeling. In *ACM Symposium on Interactive 3D Graphics*, pages 139–142, 2003.

[IMH05a] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005.

[IMH05b] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. Spatial keyframing for performance-driven animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 107–115, 2005.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *ACM SIGGRAPH*, pages 409–416, 1999.

[JBS06] Mark W. Jones, J. Andreas Bærentzen, and Milos Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[JDD03] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph.*, 22(3):943–949, 2003.

[JLCW06] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy Mesh Cutting. *Computer Graphics Forum*, 25(3):283–291, 2006.

[JMD+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 71, 2007.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, 2005.

[JT95] Ollie Johnston and Frank Thomas. *The Illusion of Life: Disney Animation*. Disney Editions, 1995.

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

[KG05] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 147–154, 2005.

[KH06] Olga A. Karpenko and John F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

[KHR02] Olga A. Karpenko, John F. Hughes, and Ramesh Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3):585–594, 2002.

[Kob00] Leif Kobbelt. Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer*, 16(3-4):142–158, 2000.

[KS07] Levent Burak Kara and Kenji Shimada. Sketch-based 3D shape creation for industrial styling design. *IEEE Computer Graphics and Applications*, 27(1):60–71, 2007.

[KSvdP07] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. Contour-based modeling using deformable 3d templates. Tech Report TR-2007-13, CS, 2007.

[KT03] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 954–961, 2003.

[LCOGL07] Yaron Lipman, Daniel Cohen-Or, Ran Gal, and David Levin. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, 26(1), 2007.

[Lev98] David Levin. The approximation power of moving least-squares. *Math. Comp.*, 67(224):1517–1531, 1998.

[Lév06] Bruno Lévy. Laplace-Beltrami eigenfunctions: Towards an algorithm that understands geometry. In *Proceedings of SMI*, 2006. Invited talk.

[LKG+03] Ignacio Llamas, Byungmoon Kim, Joshua Gargus, Jarek Rossignac, and Chris D. Shaw. Twister: a space-warp operator for the two-handed editing of 3d shapes. *ACM Trans. Graph.*, 22(3):663–668, 2003.

[LM99] Bruno Levy and Jean-Laurent Mallet. Constrained discrete fairing for arbitrary meshes. *Tech report*, 1999.

[LPW+06] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graph.*, 25(3):681–689, 2006.

[LS96] Hod Lipson and Moshe Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer Aided Design*, 28(8):651–663, 1996.

[LSCOL04] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, and David Levin. Differential coordinates for interactive mesh editing. In *International Conference on Shape Modeling and Applications*, pages 181–190, 2004.

[LSLCO05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[Mal89] Jean-Laurent Mallet. Discrete smooth interpolation. *ACM Trans. Graph.*, 8(2):121–144, 1989.

[Max07] Maxis. SPORE ™. Electronic Arts, `www.spore.com`, 2007.

[May07] Maya. Autodesk, `http://www.autodesk.com/maya`, 2007.

[McC94] Scott McCloud. *Understanding Comics: The Invisible Art*. Harper, 1994.

[MCCH99] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: a constructive approach to modeling free-form shapes. In *ACM SIGGRAPH*, pages 393–400, 1999.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, 2003.

[MJ96] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, 1996.

[MKL05] Mark Masry, Dong-Joong Kang, and Hod Lipson. A pen-based freehand sketching interface for progressive construction of 3d objects. *Computers and Graphics*, 29:563–575, 2005.

[MS92] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *ACM SIGGRAPH*, pages 167–176, 1992.

[Nay90] Bruce Naylor. Sculpt: an interactive solid modeling tool. In *Proceedings on Graphics interface '90*, pages 138–148, 1990.

[Nea04] Andrew Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. Technical Report, TU Darmstadt, 2004.

[NISA06] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In *ACM GRAPHITE*, pages 381–389, 2006.

[NISA07] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. FiberMesh: Designing freeform surfaces with 3d curves. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), 2007. to appear.

[NMK+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[NS07] Andrew Nealen and Olga Sorkine. A note on boundary constraints for linear variational surface design. Technical Report, TU Berlin, 2007.

[NSACO05] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[PB03] Philippe P. Pébay and Timothy J. Baker. Analysis of triangle quality measures. *Math. Comput.*, 72(244):1817–1839, 2003.

[Pen55] Roger Penrose. A generalized inverse for matrices. In *Cambridge Phil. Soc. 51*, pages 406–413, 1955.

[PF01] Ronald N. Perry and Sarah F. Frisken. Kizamu: a system for sculpting digital characters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 47–56, 2001.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 313–318, 2003.

[PLW+07] Helmut Pottmann, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. Geometry of multilayer freeform structures for architecture. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), 2007. to appear.

[PP93a] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PP93b] Kari Pulli and Matti Pietikäinen. Range image segmentation based on decomposition of surface normals. In *8th Scandinavian Conference on Image Analysis (SCIA'93)*, Tromso, May 1993.

[PSZ01] Jianbo Peng, Vasily Strela, and Denis Zorin. A simple algorithm for surface denoising. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 107–112, 2001.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C - The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[Rho93] John S. Rhoades. *Shaping Curved Surfaces*. PhD thesis, University of North Carolina at Chapel Hill, 1993.

[SAG03] Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. Isotropic remeshing of surfaces: a local parameterization approach. In *proceedings of 12th International Meshing Roundtable*, pages 215–224, 2003.

[SCO04] Olga Sorkine and Daniel Cohen-Or. Least-squares meshes. In *Shape Modeling International*, pages 191–199, 2004.

[SCOIT05] Olga Sorkine, Daniel Cohen-Or, Dror Irony, and Sivan Toledo. Geometry-aware bases for shape approximation. *IEEE Transactions on Visualization and Computer Graphics*, 11(2):171–180, 2005.

[SF98] Karan Singh and Eugene L. Fiume. Wires: A geometric deformation technique. In *ACM SIGGRAPH*, pages 405–414, 1998.

[SG03] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Eurographics Symposium on Geometry Processing*, pages 20–30, 2003.

[SK01] Robert Schneider and Leif Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design*, 18(4):359–379, 2001.

[SK04] Alla Sheffer and Vladislav Kraevoy. Pyramid coordinates for morphing and deformation. In *Second International Symposium on 3DPVT (3D Data Processing, Visualization, and Transmission)*, pages 68–75, 2004.

[SLCO+04] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Eurographics Symposium on Geometry Processing*, pages 179–188, 2004.

[SLSK07] Andrei Sharf, Thomas Lewiner, Ariel Shamir, and Leif Kobbelt. On-the-fly curve-skeleton computation for 3d shapes. *Computer Graphics Forum (to appear)*, 26(3), 2007.

[SMW06] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[Sor06] Olga Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.

[SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, volume 20, pages 151–160, August 1986.

[SP04] Robert Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, 2004.

[Str98] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1998.

[Sut63] Ivan Sutherland. *Sketchpad: A man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, 1963.

[SWSJ05] Ryan Schmidt, Brian Wyvill, Mario Costa Sousa, and Joaquim A. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–62, 2005.

[SWZ05] Scott Schaefer, Joe Warren, and Dennis Zorin. Lofting curve networks with subdivision surfaces. In *Symposium on Geometry Processing*, pages 105–116, 2005.

[SYBF06] Lin Shi, Yizhou Yu, Nathan Bell, and Wei-Wen Feng. A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph.*, 25(3):1108–1117, 2006.

[SZBN03] Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. T-Splines and T-NURCCs. *ACM Trans. Graph.*, 22(3):477–484, 2003.

[Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, 1995.

[Tau00] Gabriel Taubin. Geometric signal processing on polygonal meshes. EUROGRAPHICS state-of-the-art report, 2000.

[Tik63] Andrey N. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet Math Dokl*, 4:1035–1038, 1963.

[TO99] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 335–342, 1999.

[TO02] Greg Turk and James F. O'brien. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph.*, 21(4):855–873, 2002.

[Tol03] Sivan Toledo. TAUCS*: A Library of Sparse Linear Solvers*. Tel Aviv University, 2003.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214, 1987.

[Tuk77] John Wilder Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 55–64, 1992.

[vFTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[VRKS01] Jens Vorsatz, Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Feature sensitive remeshing. *Computer Graphics Forum*, 20(3):393–401, 2001.

[VRS03] Jens Vorsatz, Christian Rössl, and Hans-Peter Seidel. Dynamic remeshing and applications. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 167–175, 2003.

[WBH+07] Max Wardetzky, Miklos Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. Discrete quadratic curvature energies. *CAGD (to appear)*, 2007.

[Wie49] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. Wiley, 1949.

[Wil90] Lance Williams. 3d paint. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 225–233, 1990.

[WK95] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 151–ff., 1995.

[WPP07] Robert Y. Wang, Kari Pulli, and Jovan Popović. Real-time enveloping with rotational regression. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), 2007. to appear.

[Wri06] Will Wright. What's next in content keynote. Game Developer Conference, `http://www.sporeuniverse.net/gdc2006.php`, 2006.

[WSLG07] Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. Context-aware skeletal shape deformation. *Computer Graphics Forum (to appear)*, 26(3), 2007.

[WW94] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *ACM SIGGRAPH*, pages 247–256, 1994.

[YBS07] Shin Yoshizawa, Alexander G. Belyaev, and Hans-Peter Seidel. Skeleton-based variational mesh deformations. *Computer Graphics Forum (to appear)*, 26(3), 2007.

[YZX+04] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3D scenes. In *Proceedings of SIGGRAPH 96*, pages 163–170, August 1996.

[ZHS+05] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph Laplacian. *ACM Trans. Graph,*, 24(3):496–503, 2005.

[ZNA07] Johannes Zimmermann, Andrew Nealen, and Marc Alexa. Silsketch: Automated sketch-based editing of surface meshes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2007.

[ZPKG02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3d: an interactive system for point-based surface editing. *ACM Trans. Graph.*, 21(3):322–329, 2002.

[ZRKS05] Rhaleb Zayer, Christian Rössl, Zachi Karni, and Hans-Peter Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24(3):601–609, 2005.

[ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *ACM SIGGRAPH*, pages 259–268, 1997.