

Diss. ETH No. 19140

Constraint-Based Surface Processing for Geometric Modeling and Architecture

A dissertation submitted to
ETH ZURICH

for the degree of
DOCTOR OF SCIENCES

presented by
Michael Eigensatz
MSc CS, ETH Zurich
born 16 December 1979
citizen of Kriens, Lucerne

accepted on the recommendation of
Prof. Mark Pauly, examiner
Prof. Mario Botsch, co-examiner
Prof. Markus Gross, co-examiner
Prof. Helmut Pottmann, co-examiner

2010

*"The more constraints one imposes, the more one frees one's self."
- Igor Fyodorovich Stravinsky*

Abstract

This thesis investigates the application and implementation of geometric constraints to manipulate, approximate, and optimize surfaces for modeling and architecture. In modeling, geometric constraints provide an interface to edit and control the form of a surface. We present a geometry processing framework that enables constraints for positional, metric, and curvature properties anywhere on the surface of a geometric model. Target values for these properties can be specified point-wise or as integrated quantities over curves and surface patches embedded in the shape. For example, the user can draw several curves on the surface and specify desired target lengths, manipulate the normal curvature along these curves, or modify the area or principal curvature distribution of arbitrary surface patches. This user input is converted into a set of non-linear constraints. A global optimization finds the new deformed surface that best satisfies the constraints, while minimizing adaptable measures for metric and curvature distortion that provide explicit control on the deformation semantics. This approach enables flexible surface processing and shape editing operations. In architecture, the emergence of large-scale freeform shapes pose new challenges to the process from design to production. Geometric constraints directly arise from aesthetic, structural, and economical requirements for the fabrication of such structures. A key problem is the approximation of the design surface by a union of patches, so-called panels, that can be manufactured with a selected technology at reasonable cost, while meeting the design intent and achieving the desired aesthetic quality of panel layout and surface smoothness. The production of curved panels is mostly based on molds. Since the cost of mold fabrication often dominates the panel cost, there is strong incentive to use the same mold for multiple panels. Various constraints, such as the limited geometry of mold shapes and tolerances on positional and normal continuity between neighboring panels, have to be considered. We introduce a paneling algorithm that interleaves discrete and continuous optimization steps to minimize production cost while meeting the desired geometric constraints and is able to handle complex arrangements with thousands of panels. The practical relevance of our system is demonstrated by paneling solutions for real, cutting-edge architectural freeform design projects.

Zusammenfassung

Diese Doktorarbeit untersucht den Einsatz und die Implementierung von geometrischen Bedingungen zur Manipulation, Approximation und Optimierung von Flächen in der geometrischen Modellierung und in der Architektur. In der Modellierung bieten geometrische Bedingungen eine Schnittstelle, über welche die Form einer Fläche editiert und optimiert werden kann. Wir präsentieren ein Framework zur Geometrieverarbeitung, welches es ermöglicht, Bedingungen an Positions-, Metrik-, und Krümmungseigenschaften an beliebigen Stellen eines geometrischen Modells vorzuschreiben. Zielwerte für diese Eigenschaften können punktweise oder integriert über Kurven- und Flächenelemente spezifiziert werden. Der Benutzer kann zum Beispiel mehrere Kurven auf eine Fläche zeichnen, die Längen dieser Kurven oder die Normalenkrümmung entlang dieser Kurven vorschreiben oder er kann den Flächeninhalt sowie die Hauptkrümmungen auf beliebigen Flächenelementen modifizieren. Diese Benutzereingabe wird in eine Menge von nichtlinearen Bedingungen übersetzt. Eine globale Optimierung findet die neue, deformierte Fläche, welche die Bedingungen bestmöglich erfüllt. Gleichzeitig geben anpassbare Masse der Metrik- und Krümmungsverzerrung dem Benutzer explizite Kontrolle über die Deformationssemantik. Dieser Ansatz ermöglicht flexible Flächenverarbeitung und Formeditierung. In der Architektur stellt die vermehrte Verwendung von grossen Freiformflächen neue Herausforderungen an den Prozess vom Design zur Produktion. Geometrische Bedingungen entspringen direkt den ästhetischen, strukturellen und ökonomischen Anforderungen der Fabrikation von solchen Gebilden. Ein grundlegendes Problem ist dabei die Approximation der Designfläche durch eine Menge von Flächenelementen, so genannten Panele. Diese Panele sollten mit einer bestimmten Technologie mit angemessenen Kosten gefertigt werden können. Gleichzeitig sollte die Design-Idee und weitere wichtige Qualitätsmerkmale, wie das Layout der Panele und die Glattheit der Fläche, eingehalten werden. Die Produktion von gekrümmten Panele basiert grösstenteils auf Mulden (Press- oder Gussformen). Da die Fertigungskosten einer Mulde oft weit höher liegen als die Kosten, um mit der Mulde ein Panel zu erzeugen, ist es wichtig, eine Mulde für möglichst viele Panele wiederverwenden zu können. Verschiedene

Bedingungen wie die eingeschränkte Geometrie der Muldenform und Toleranzen für Positions- und Normalenkontinuität zwischen benachbarten Panels müssen dabei eingehalten werden. Wir stellen einen Panelisierungs-Algorithmus vor, der diskrete und kontinuierliche Optimierung kombiniert, um die Kosten zu minimieren und gleichzeitig die geometrischen Bedingungen einzuhalten. Der Algorithmus ist fähig, komplexe Anordnungen mit tausenden von Panels automatisch zu berechnen. Wir demonstrieren die praktische Relevanz von unserem System anhand aktueller Projekte der Freiformarchitektur.

Acknowledgements

A PhD means suffering. Not necessarily for the PhD student but for many people around him. Therefore my first and biggest thanks goes to Isa, understanding me, enduring me, and giving me everything I needed to overcome even the hardest times. Thank you Vreni and Jacques, for being the best parents in the world, for giving me unconditional support and love. This PhD is the fruit of your hard work and your trust in me. Thank you Lucien, Livio, and Lea, you are always in my heart. Thank you Matthias, Sam, and Yves. Even to have one of you as a friend is a treasure no money in the world can buy. I but humbly bow to the fortune of having the three of you.

I thank Prof. Mark Pauly for being a supervisor one can only wish for as a PhD. Being an outstanding researcher he supported me to develop my own research qualities and acknowledged both my weaknesses and strengths. He was always open to discuss our agreements and disagreements, was very understanding, open minded, and contagiously enthusiastic which provided an ideal creative environment. I am truly grateful for all this.

I thank Prof. Helmut Pottmann for giving me the opportunity to jump into the fantastic field of Architectural Geometry and believing in my qualities. I thank my other collaborators Prof. Niloy Mitra, Mario Deuss, Martin Kilian, Alexander Schiftner, Heinz Schmiedhofer, and Robert Sumner, without whom this work would not have been possible. And I thank Pierre Alliez, Mirela Ben-Chen, Mario Botsch, David Cohen-Steiner, Joachim Giesen, Eitan Grinspun, Hao Li, Bálint Miklòs, Filip Sadlo, Johannes Wallner, and Camille Wormser for fruitful discussions about my research.

I thank the bachelor and master students Dominik Erni, Cyril Perrig, Moritz Bächer, Gerhard Röthlin, Sandro Löschhorn, and Rolf Weilenmann I had the pleasure to supervise and work with.

I thank Prof. Mario Botsch and Prof. Markus Gross for co-examining my PhD thesis and defense.

A very special warm thanks goes to my former math teacher and mentor Hannes van der Weijden: He kindled my love for mathematics and engineering, enforced my self-confidence, and by that set the first and most important tracks for my successful education.

Thank you all!

This thesis was supported by the Swiss National Science Foundation.

Contents

1	Introduction	1
2	Fundamentals	9
2.1	Surface Geometry in \mathbb{R}^3	9
2.1.1	Curves and Surfaces	10
2.1.2	Tangents and Normals	13
2.1.3	Metric	14
2.1.4	Metric Deformation	16
2.1.5	Curvature	18
2.1.6	Fairness	22
2.2	Least Squares Optimization	24
2.2.1	The Gauss-Newton Algorithm	26
2.2.2	The Levenberg-Marquardt Algorithm	29
2.2.3	Implementation Insights	31
3	Constraint-Based Modeling	41
3.1	Introduction	41
3.1.1	Related Work	43
3.2	Problem Formulation	45
3.2.1	Modeling Constraints	46
3.2.2	Deformation Control	46
3.3	Discretization	49
3.3.1	Surface Energies	50
3.3.2	Modeling Constraints	53
3.3.3	Discussion	56
3.4	Optimization	57
3.5	Constraint-Based Shape Editing	60
3.6	Curvature-Domain Shape Processing	67
3.7	Discussion	75

Contents

4	Paneling Architectural Freeform Surfaces	81
4.1	Introduction	81
4.1.1	Contributions	85
4.1.2	Related Work	86
4.2	Problem Specification	88
4.2.1	Terminology	88
4.2.2	The Paneling Problem	91
4.3	Paneling Algorithm	92
4.3.1	Continuous Optimization Step	94
4.3.2	Discrete Optimization Step	99
4.3.3	Interleaved Iteration	102
4.4	Evaluation and Discussion	103
5	Conclusion and Outlook	117
5.1	Impact	118
5.2	Outlook	119
A	Optimization	125
A.1	Derivatives of Geometric Properties Evaluated on Triangle Meshes	125
A.1.1	Derivatives of Area, Length, and Angle	125
A.1.2	Curvature Derivatives	126
A.2	Element-Weighted Set Cover	130
A.2.1	Problem	130
A.2.2	Algorithm	132
A.2.3	Analysis	133
B	Metric Space for Approximate Panel-Segment Distances	137
	Bibliography	141
	Curriculum Vitae	153

Introduction

The rise of the computer has enabled the digital representation, acquisition, reconstruction, analysis, and processing of geometric objects and the efficient execution of complex geometric computations and algorithms. Over the last century, digital geometry processing has grown to be a significant research area at the intersection of applied mathematics, computer science, and engineering, covering a wide range of applications from multimedia, entertainment, and classical computer-aided design to biomedical computing, machine engineering, and architecture.

In digital geometry, surface processing is concerned with the creation, manipulation, optimization, and approximation of form. The form, in the context of surface processing the shape of a two dimensional surface embedded in three dimensions, is an essential aesthetic and functional attribute. It determines how a geometric object is perceived and how well it performs a given task.

By modeling virtual surfaces artists have created realistic or stylized characters that entertain us in movies and computer games (Figures 1.3, 1.4). Surface models are used to design, simulate, and manufacture ships, airplanes, and cars (Figure 1.1). In architecture, a long

1 Introduction

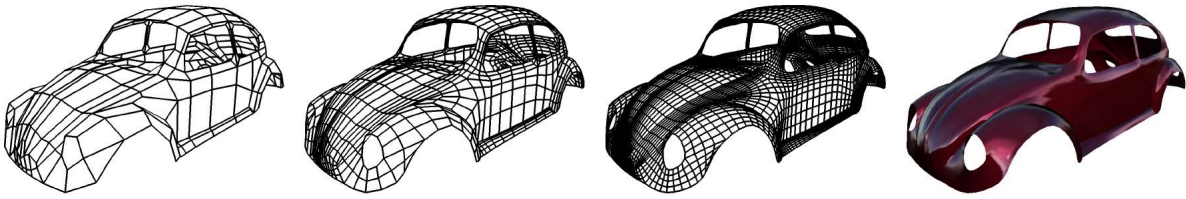


Figure 1.1 *Digital surface model of a car generated by an iterative refinement of a coarse control mesh. (Figure taken from [BPK⁺08], kindly provided by Botsch and colleagues.)*

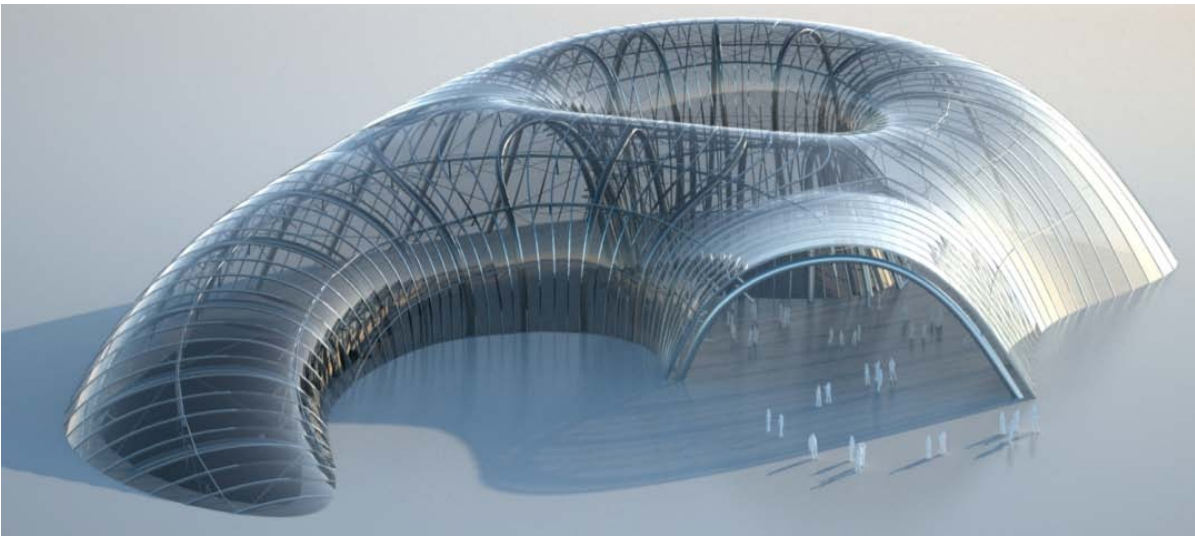


Figure 1.2 *Architectural freeform surface consisting of single-curved strips. (Figure from [PSB⁺08], kindly provided by Pottmann and colleagues.)*

standing source for the application and inspiration of geometric problems, surface processing has reached unprecedented relevance with the emergence of architectural freeform structures where a general smooth surface replaces the classical, geometrically limited composition of flat walls and roof (Figure 1.2).

A central driver in the processing of surfaces is the geometric constraint. A geometric constraint both controls and restricts the form of a surface and can serve as an interface for a user to manipulate form or directly be imposed by structural or economical limitations defined in the problem statement. This thesis reviews existing techniques and presents new approaches to effectively enforce complex geometric constraints for surface processing.

Particularly, this thesis offers the following core contributions:

- We identify the presence and use of geometric constraints in surface modeling and freeform architecture, two principal application domains of digital geometry processing.
- We propose and investigate computational tools and algorithmic frameworks based on a sound mathematical foundation to implement a broad set of advanced geometric constraints for surface processing and, using these tools,
- we evaluate the potential and the limitations of these constraints for modeling and architecture by numerous comparisons and case studies.

In this thesis we have grouped our findings according to their application domain and we present the application-specific technical contributions for effective constraint-based surface processing in the corresponding Chapters 3 for modeling and 4 for freeform architecture.

In surface modeling, geometric constraints offer means of controlling the shape of a geometric model. Current methods mostly rely on position constraints, specified either by a skeleton or a polygonal control cage to perform a deformation of the entire space in which the surface is embedded (e.g., Figure 1.3) or by a control handle directly on the surface itself (e.g., Figure 1.4). See Chapter 3 for an extensive discussion of previous work. In this thesis we broaden the toolset for surface modeling and demonstrate how advanced constraints such as the prescription of length (see Figure 1.5) and further geometric properties provide novel and flexible tools for shape design and exploration.

In freeform architecture, geometric constraints mostly originate from aesthetic, structural, or economical requirements in the production process. For example, freeform surfaces are usually assembled as a collection of smaller elements, called panels (see, e.g., Figure 1.6a). The approximation of an architectural design with such panels underlies practical constraints such as that the maximal gap between neighboring panels and the angle in which two neighboring panels meet (called *kink angle*) should remain below given thresholds. Existing paneling techniques focus on specialized panel types such as planar

1 Introduction

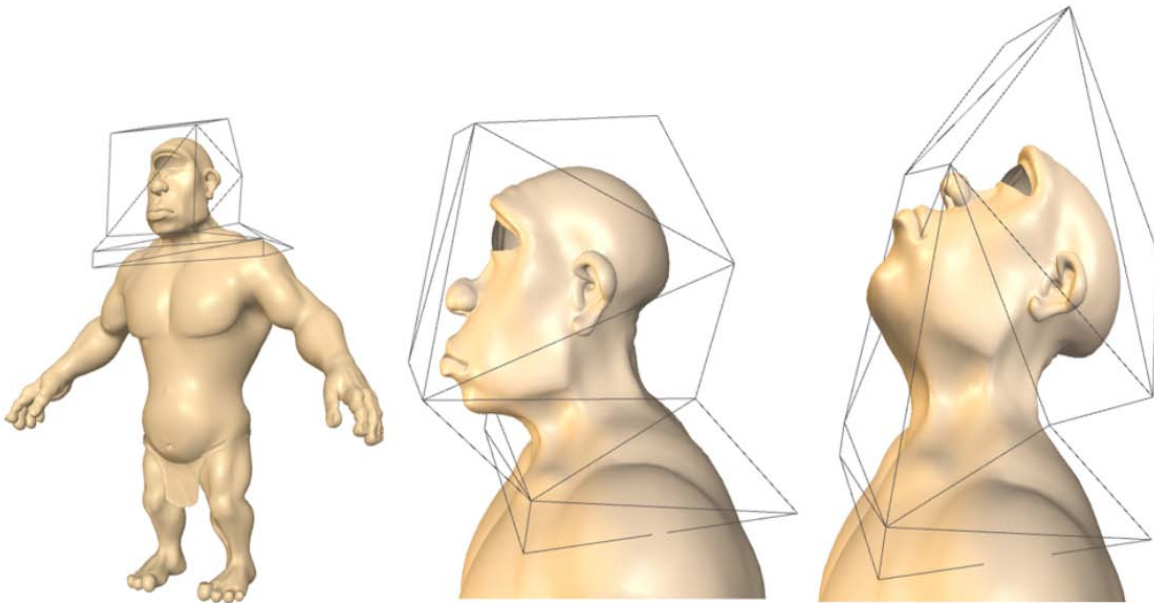


Figure 1.3 *Editing of a surface model by position constraints using a control cage. The position constraints define a deformation of the space in which the surface is embedded. (Figure taken from [LLCOo8], kindly provided by Lipman and colleagues.)*

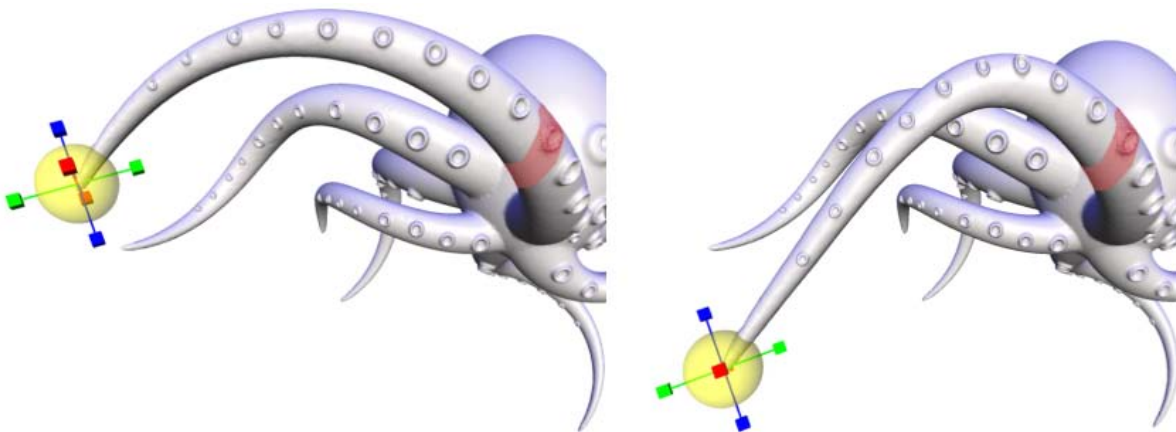


Figure 1.4 *Editing of a surface model by specifying position constraints directly on the surface using a control handle. (Figure taken from [SCOL⁺o4], kindly provided by Sorkine and colleagues.)*

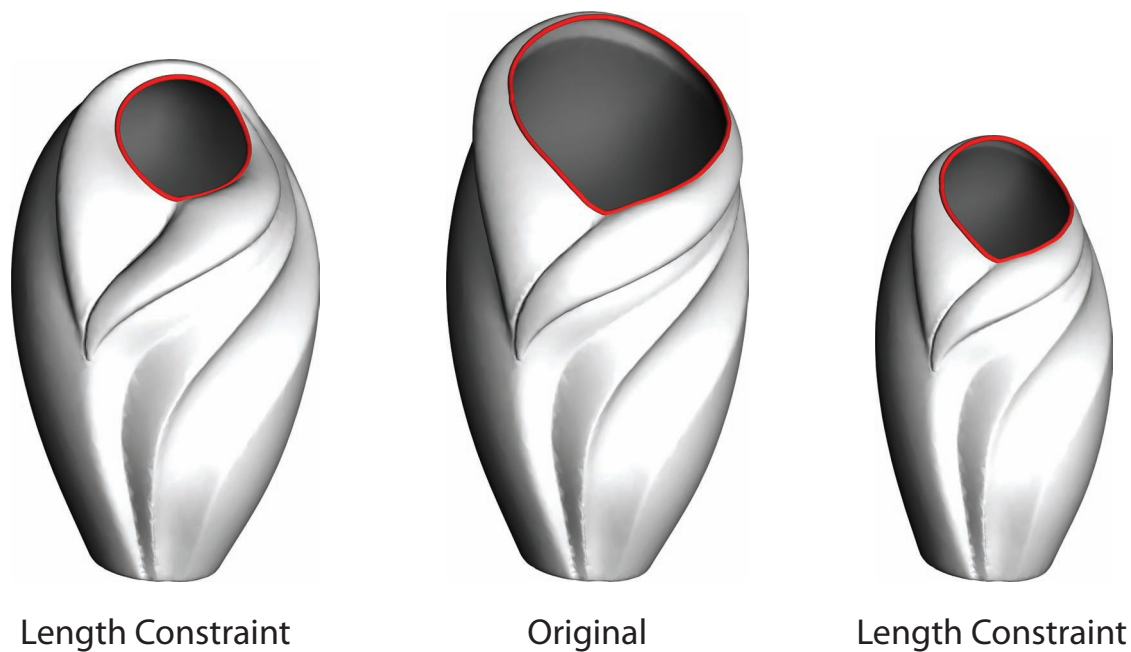
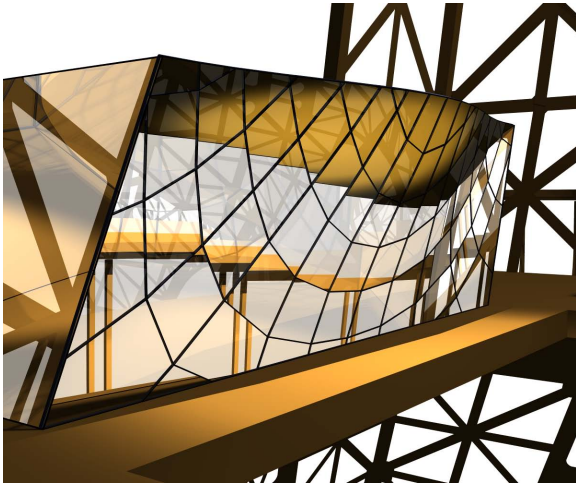


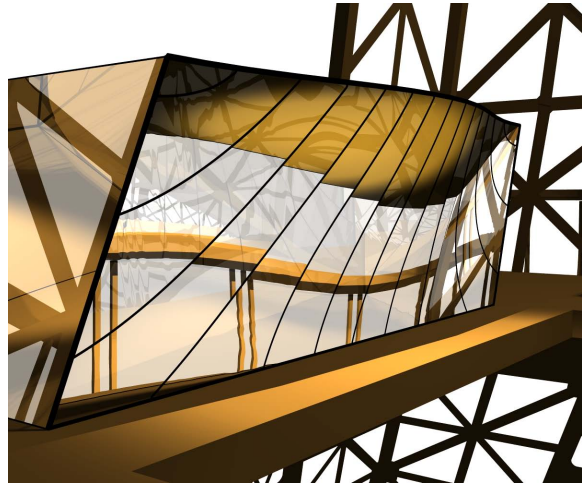
Figure 1.5 *Constraint-based surface modeling using length constraints. Constraining the length of the red curve on the vase does not have a unique solution. The deformations shown on the left and right are but two of infinitely many valid solutions. In this thesis we present techniques to effectively enforce advanced geometric constraints while giving the user explicit control on the deformation outcome.*

panels (e.g., Figure 1.6a) or developable panels (e.g., Figure 1.6b). Due to the limited geometry of these panel types, quality measures like the gap and kink angle between panels can be optimized but cannot be explicitly controlled through constraints. See Chapter 4 for a detailed analysis of previous work.

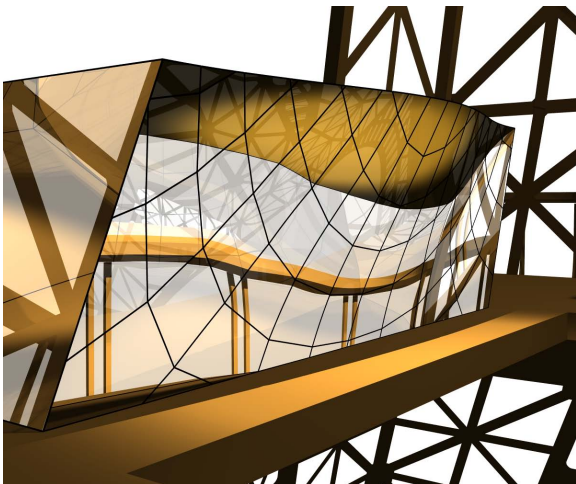
Recent technological advances enable the large-scale production of double-curved panels that allow panelizations of architectural freeform surfaces with superior inter-panel continuity compared to single-curved panels. The fabrication of double-curved panels, however, incurs a higher cost, depending on the complexity of the panel shapes and the material and panel manufacturing process. A cost-effective paneling should use as many simple panels as possible and reuse the same panel shapes for different parts of the surface. With the integration of double-curved panels, therefore, the specification of geometric constraints becomes an essential tool to manage the tradeoff



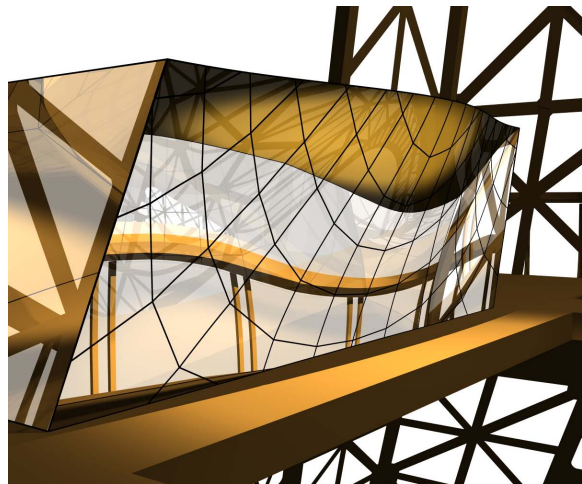
(a) A freeform surface approximation with planar panels according to [LPW⁺06] results in a maximum kink angle of 11°.



(b) A paneling with developable surface strips according to [PSB⁺08] results in a maximum kink angle of 6° between strips.



(c) Paneling algorithm presented in this thesis using a kink angle constraint of 1°.



(d) Paneling algorithm presented in this thesis using a kink angle constraint of 1/4°.

Figure 1.6 Comparison with state-of-the-art algorithms to panelize architectural freeform surfaces. Example taken from a facade designed by architects Moatti et Rivière for an entry to a competition on redesigning the pavilions on the first platform of the Eiffel Tower, panelized by Evolute (www.evolute.at) and RFR (www.rfr-group.com). (a, b) Paneling using planar elements or developable surface strips. The surface reflections indicate high kink angles (angles between neighboring panels). (c, d) The same facade panelized using the paneling algorithm introduced in this thesis that allows the prescription of constraints for the maximal kink angle and other geometric quality measures enabling smooth surface approximations.

between cost and approximation quality. We introduce a general paneling framework that incorporates double-curved panels and enables direct specification of important geometric constraints on quality and fabrication, such as the maximal gap between panels and the maximal kink angle to guarantee a smooth approximation (see Figure 1.6).

Publications. Most of the contributions discussed in this thesis have been presented in the following publications:

MICHAEL EIGENSATZ, ROBERT W. SUMNER, AND MARK PAULY.
Curvature-Domain Shape Processing.
Computer Graphics Forum (Proceedings of EUROGRAPHICS), 2008.

MICHAEL EIGENSATZ AND MARK PAULY.
Positional, Metric, and Curvature Control for Constraint-Based Surface Deformation.
Computer Graphics Forum (Proceedings of EUROGRAPHICS), 2009.

MICHAEL EIGENSATZ, MARTIN KILIAN, ALEXANDER SCHIFTNER, NILOY MITRA, HELMUT POTTMANN, AND MARK PAULY.
Paneling Architectural Freeform Surfaces.
ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2010.

MICHAEL EIGENSATZ, MARIO DEUSS, ALEXANDER SCHIFTNER, MARTIN KILIAN, NILOY MITRA, HELMUT POTTMANN, AND MARK PAULY.
Case Studies in Cost-Optimized Paneling of Architectural Freeform Surfaces.
Advances in Architectural Geometry, 2010.

Organization. Before presenting our research in constraint-based surface processing for modeling (Chapter 3) and freeform architecture (Chapter 4), we provide a detailed review of the underlying fundamental concepts in surface geometry and optimization theory in Chapter 2, including valuable insights gained throughout this PhD. Chapter 5 concludes this thesis and provides an outlook on future work.

Fundamentals

This chapter explains the fundamental concepts of geometry and optimization employed in the subsequent chapters and provides the interested reader with references for further reading. Readers familiar to these topics may skip this chapter or simply use it as a reference.

2.1 Surface Geometry in \mathbb{R}^3

This thesis is concerned with surfaces embedded in 3-dimensional euclidean space. Many concepts described in this chapter, however, translate to higher dimensions and even to surfaces not embedded in euclidean space (see [dC92]).

This chapter offers a qualitative, descriptive introduction to surface geometry in order to maximize readability and provides sufficient background to understand the subsequent chapters. For a thorough mathematical treatment of differential geometry the reader is referred to the classical book *Differential Geometry of Curves and Surfaces* by Do Carmo [dC76]. The work of Botsch et. al. [BPK⁺08, Boto5] provides

2 Fundamentals

a good overview and entry point to the study of surface discretization using polygonal meshes.

2.1.1 Curves and Surfaces

A 1-dimensional **curve** embedded in \mathbb{R}^3 is a map $\mathbf{c} : I \rightarrow \mathbb{R}^3$ of an open interval $I = (a, b)$, $a, b \in \mathbb{R}$, onto \mathbb{R}^3 (Figure 2.1).

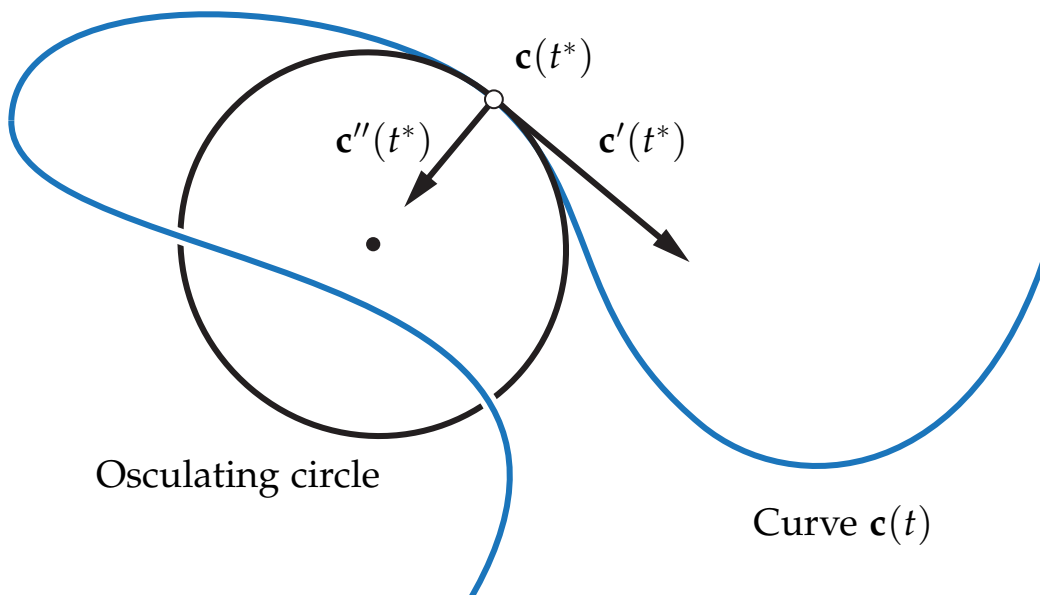


Figure 2.1 A curve $\mathbf{c} \subset \mathbb{R}^3$ parameterized by arc length and derivatives.

Usually, the map $\mathbf{c}(t) = (c_x(t), c_y(t), c_z(t))^T$ is assumed to be differentiable (all three components have well defined derivatives). A **regular curve** is a differentiable curve with $\mathbf{c}'(t) \neq 0$ for all $t \in I$. The regularity of a curve is an important property, because it implies that the curve has a well defined **tangent** for each point on the curve (see Section 2.1.2) and enables the definition of further differential properties. If the length of the curve from some fixed point on the curve to the point $\mathbf{c}(t)$ is always t , the curve is said to be **parameterized by arc length**. Since the derivative of the curve with respect to the parameter t measures the advancement on the curve with respect to the advancement of t , for differentiable curves parameterized by arc length it holds that $|\mathbf{c}'(t)| = 1$ for all $t \in I$. Given an image $\mathbf{c}(I) \subset \mathbb{R}^3$ of a continuous curve there always exists a parameterization by arc length of that curve with the same image.

The arc length parameterization is a canonical parameterization of a curve. For surfaces, such a canonical parameterization does not exist in general. For surfaces, it is not even generally possible to have a global regular parameterization. A simple example is the sphere: The standard parameterization of the sphere using spherical coordinates has non-regular points at the top and bottom, where the derivatives with respect to one variable are all 0.

Surfaces are elegantly defined as subsets of \mathbb{R}^3 : A subset $\mathcal{M} \subset \mathbb{R}^3$ is a **(topological) 2-manifold**, if for every point $\mathbf{p} \in \mathcal{M}$ there exists a local open set neighborhood $\mathcal{V} \subset \mathbb{R}^3$ to cut out a local neighborhood patch $\mathcal{V} \cap \mathcal{M}$ that can be continuously deformed into (is homeomorphic to) a planar disc (Figure 2.2). This simple definition ensures a surface without self intersections. For surfaces with boundaries the definition has to be adjusted by replacing *planar disc* with *planar halfdisc*. A 2-manifold is also a **submanifold** of the 3-dimensional euclidean space, since \mathcal{M} is a subset of \mathbb{R}^3 and \mathbb{R}^3 is itself a manifold.

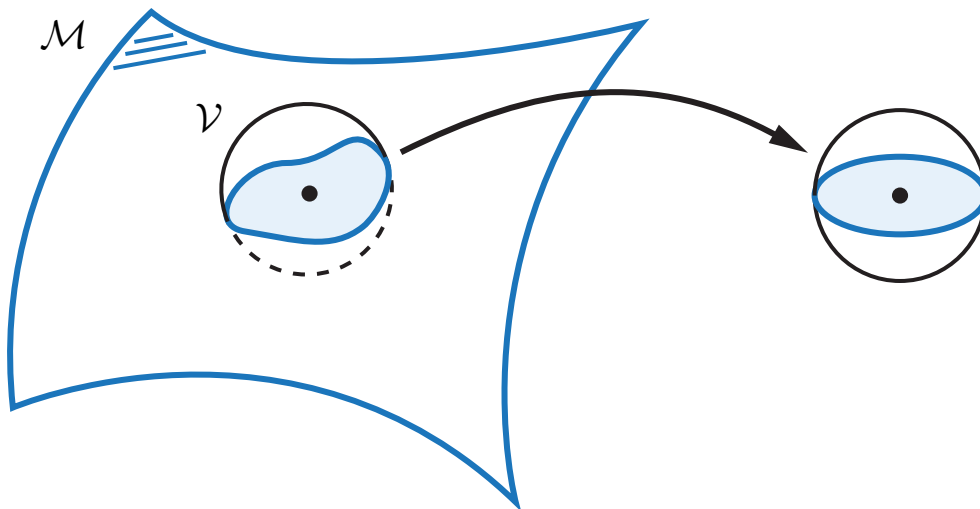


Figure 2.2 For every point on a 2-manifold surface there exists a local neighborhood patch, cut out by a local neighborhood \mathcal{V} , that can be continuously deformed to a planar disc.

A (topological) 2-manifold can have sharp creases and other differential singularities. A subset $\mathcal{M} \subset \mathbb{R}^3$ is a **differentiable 2-manifold**, if for every point $\mathbf{p} \in \mathcal{M}$ there exists a local open set neighborhood $\mathcal{V} \subset \mathbb{R}^3$ to cut out a local neighborhood patch $\mathcal{V} \cap \mathcal{M}$ that can be smoothly deformed into (is diffeomorphic to) a planar disc. Since the

2 Fundamentals

deformations of local neighborhoods to planar discs have to be smooth (differentiable), a differentiable 2-manifold defines a smooth surface. A differentiable 2-manifold in \mathbb{R}^3 is also called a **regular surface**.

As with curves, the regularity of a surface is an important prerequisite for differential geometry, since it enables to define tangent planes and other differential properties of a surface. In this thesis we will assume the surfaces to be piecewise differentiable 2-manifolds with boundaries, meaning that the surfaces are generally smooth but can contain lines of sharp creases.

A **regular local parameterization** of a differentiable 2-manifold is a differentiable map $\mathbf{f} : U \rightarrow \mathbb{R}^3$ from some 2-dimensional domain $U \subset \mathbb{R}^2$ to a local neighborhood $\mathcal{M}_p \subset \mathcal{M}$ around a point $\mathbf{p} \in \mathcal{M}$ (Figure 2.3), where \mathbf{f} and its inverse are continuous (\mathbf{f} is a homeomorphism) and \mathbf{f} is an immersion. The requirement of an immersion means, that the Jacobian of \mathbf{f} has maximal rank (rank 2). The rank of the Jacobian is important since the Jacobian directly relates to the tangent plane and other differential properties of a surface (see Section 2.1.2). Even though it is not generally possible to globally describe a regular surface by a regular parameterization, it is always possible to have regular local parameterizations of a neighborhood around any point on a regular surface.

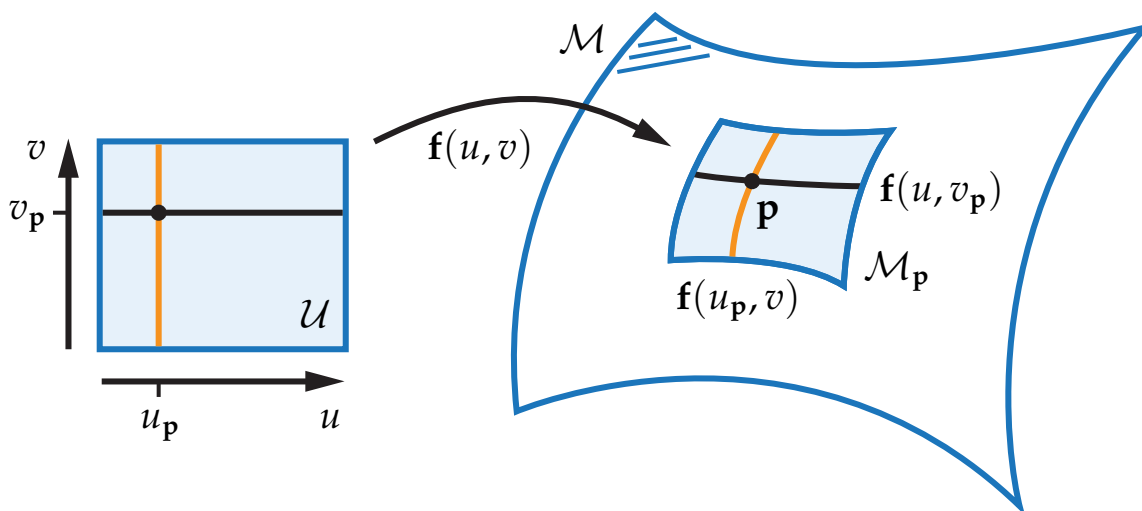


Figure 2.3 Local parameterization $\mathbf{f}(u, v) : U \rightarrow \mathcal{M}_p$ of a surface \mathcal{M} around a point $\mathbf{p} = \mathbf{f}(u_p, v_p)$ and iso-parameter curves $\mathbf{f}(u, u_p)$ and $\mathbf{f}(v_p, v)$.

Local parameterizations are a useful analytic tool since they provide a local 2-dimensional coordinate space at any point of a surface: every point on the surface in the local neighborhood can be uniquely assigned a 2-dimensional coordinate in U . There exists a great variety of literature on how to compute and optimize parameterizations for surfaces (see, e.g., [FH05, SPR06, HLS07] for an overview).

In digital surface processing, curves and surfaces are often discretized to enable efficient computation, representation, and visualization. In this thesis we discretize curves as piecewise linear paths (**polylines**). Surfaces are discretized as piecewise linear **triangular meshes** (Figure 2.4). As demonstrated in [BPK⁺08, Boto5] this piecewise linear discretization provides an efficient and flexible geometry representation.

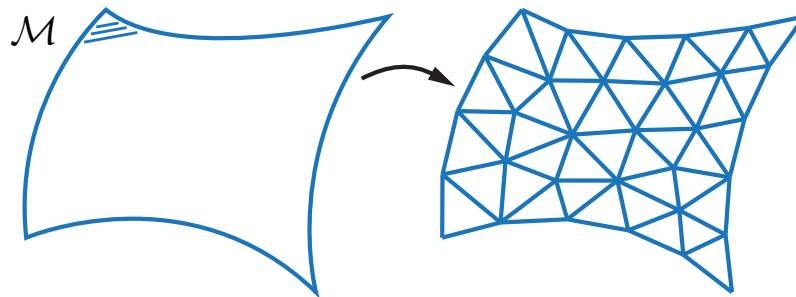


Figure 2.4 *Surface discretization by a triangle mesh.*

2.1.2 Tangents and Normals

The **tangent vector** of a curve at a point $\mathbf{c}(t)$ is the direction in which the curve is advancing and can be computed by the curves derivative $\mathbf{c}'(t)/|\mathbf{c}'(t)|$, or simply $\mathbf{c}'(t)$ for a curve parametrized by arc length (Figure 2.1). The normal of a curve in \mathbb{R}^3 requires the definition of curvature and is thus defined in Section 2.1.5.

The tangent vectors of all curves passing through a point \mathbf{p} on a regular surface \mathcal{M} all lie in (and define) a plane called the **tangent plane** $T\mathcal{M}_{\mathbf{p}}$. Given a regular local parameterization $\mathbf{f}(u, v) : U \rightarrow \mathbb{R}^3$ around \mathbf{p} (i.e., $\mathbf{f}(0, 0) = \mathbf{p}$), the tangent plane is spanned by the two columns of the Jacobian matrix of \mathbf{f} at $(0, 0)$. This holds because the two Jacobian

columns are by definition of regularity linearly independent and each column is the tangent vector of one of the two **isoparameter curves** $\mathbf{f}(u, 0)$ and $\mathbf{f}(0, v)$ (Figure 2.3). The Jacobian $\mathbf{J}_{\mathbf{f}|(0,0)}$ of \mathbf{f} at $(0, 0)$ can thus also be used to map any vector $(u, v)^\top \in \mathbb{R}^2$ to the tangent plane at \mathbf{p} by matrix-vector multiplication: $\mathbf{J}_{\mathbf{f}|(0,0)}(u, v)^\top \in T\mathcal{M}_{\mathbf{p}}$.

The **surface normal** $\mathbf{n}_{\mathcal{M}}(\mathbf{p})$ at point \mathbf{p} is defined as the unit vector orthogonal to the tangent plane $T\mathcal{M}_{\mathbf{p}}$. The normal can be computed by the cross product of the two columns of the Jacobian matrix. The surface normal has no unique orientation. If there exists a smooth, consistently oriented, normal field $\mathbf{n}_{\mathcal{M}}(\mathbf{p})$ on a surface, the surface is called **orientable**. A famous example of a non-orientable surface is the Möbius strip.

Normals and tangent planes on discrete surfaces such as triangle meshes are not uniquely defined. In general when discretizing differential quantities the choice of discretization depends on the properties of these quantities that should be transferred to the discrete setting. For example one would usually want to define a tangent plane and a normal such that they stand orthogonal to each other. A simple way to define vertex normals on a triangular mesh is as a weighted average of the normals of all the adjacent faces where the face areas or the incident triangle angles are chosen as weights (see [BPK⁺08] for further possibilities).

2.1.3 Metric

The **(local) metric** of a curve or surface describes the local distances between points on curves and surfaces. On surfaces, the metric includes information on angles and areas. Since a surface can be curved, not all relations between metric properties (angles, lengths, area) in \mathbb{R}^2 directly translate to the surface space. The metric of a curve or surface is often described by the metric of its (local) parameterization that relates distances in parameter space to distances on the curve or surface.

On parametrized curves $\mathbf{c}(t) : I \rightarrow \mathbb{R}^3$, the metric information is encoded in the first derivative $\mathbf{c}'(t)$. The **length of a regular curve** can be computed as $\int_I |\mathbf{c}'(t)| dt$.

The local metric properties length, angle, and area on a surface can all be computed with the scalar product of vectors in tangent space. The metric of a regular surface \mathcal{M} is thus defined by the function $g_{\mathbf{p}}$ that assigns to every point $\mathbf{p} \in \mathcal{M}$ the scalar product

$$g_{\mathbf{p}}(\mathbf{x}, \mathbf{y}) := \langle \mathbf{x}, \mathbf{y} \rangle \quad \text{for } \mathbf{x}, \mathbf{y} \in T\mathcal{M}_{\mathbf{p}}. \quad (2.1)$$

The function $g_{\mathbf{p}}$ is called the **First Fundamental Form of a surface**.

The metric of a regular parameterization $\mathbf{f} : U \rightarrow \mathbb{R}^3$ is defined by the function $g_{(u,v)}$ that assigns to every point $(u, v)^{\top} \in U$ the scalar product

$$g_{(u,v)}(\mu, \eta) := \langle \mathbf{J}_{\mathbf{f}|(u,v)}\mu, \mathbf{J}_{\mathbf{f}|(u,v)}\eta \rangle \quad \text{for } \mu, \eta \in \mathbb{R}^2. \quad (2.2)$$

The function $g_{(u,v)}$ is called the **First Fundamental Form of a parametrization**. As described in Section 2.1.2 the Jacobian $\mathbf{J}_{\mathbf{f}|(u,v)}$ of \mathbf{f} at (u, v) is used to map the direction vectors μ and η in parameter space to vectors in the tangent plane. The First Fundamental Form of a parameterization is therefore sometimes also defined as the square symmetric matrix $\mathbf{J}_{\mathbf{f}}^{\top}\mathbf{J}_{\mathbf{f}}$.

With the use of the First Fundamental Form, the **length of a path on the surface** $\mathbf{c}(t) := \mathbf{f}(\mathbf{r}(t))$, where $\mathbf{f} : U \rightarrow \mathbb{R}^3$ is a regular local parameterization and $\mathbf{r} : I \rightarrow U$ is a 2-dimensional path in the parameter domain U , can be computed as

$$L(\mathbf{c}) = \int_I \sqrt{g_{\mathbf{c}(t)}(\mathbf{c}'(t), \mathbf{c}'(t))} dt \quad (2.3)$$

$$= \int_I \sqrt{g_{\mathbf{r}(t)}(\mathbf{r}'(t), \mathbf{r}'(t))} dt. \quad (2.4)$$

Since for the computation of lengths the two arguments of g are the same, the quadratic form $g(\mathbf{x}) = g(\mathbf{x}, \mathbf{x})$ is sometimes called the First Fundamental Form as well.

The **area of a surface patch** $\mathcal{P} \subset \mathcal{M}$ of a surface \mathcal{M} with a regular local parameterization $\mathbf{f} : U \rightarrow \mathcal{P}$ can be computed as

$$A(\mathcal{P}) = \int_U \sqrt{\det(g_{(u,v)})} dudv, \quad (2.5)$$

where the determinant $\det(g_{(u,v)})$ of the First Fundamental Form can be computed as the determinant of the matrix $\mathbf{J}_{\mathbf{f}|(u,v)}^{\top}\mathbf{J}_{\mathbf{f}|(u,v)}$ and is the

2 Fundamentals

square of the area spanned by the two column vectors of $\mathbf{J}_{\mathbf{f}|(u,v)}$. The determinant of the First Fundamental Form defines the infinitesimal area distortion at the coordinates (u, v) . The infinitesimal area element dA is defined as $dA = \sqrt{\det(\mathbf{g}_{(u,v)})} du dv$ and can be used to convert integrals over the parameter domain to integrals over the surface area. Equation 2.5 can therefore also be written as

$$A(\mathcal{P}) = \int_{\mathcal{P}} dA. \quad (2.6)$$

Metric properties of a surface such as the length of a path on the surface or the area of a surface patch are independent on the chosen parameterization and only depend on the geometry of the surface itself. Nevertheless, parameterizations and their First Fundamental Form provide important tools to analytically compute differential properties.

On discrete surface approximations such as triangular meshes that approximate surfaces explicitly via geometric elements, e.g., triangles, metric properties such as lengths, areas, and angles are usually straight forward to compute using vector geometry. In Chapter 3 we investigate ways to compute and use discrete metric quantities for constraint-based surface processing.

2.1.4 Metric Deformation

In this thesis we will extensively work with surface deformations. A **regular deformation** can be defined as a differentiable map $\mathbf{d} : \mathcal{M}_o \rightarrow \mathcal{M}_d \subset \mathbb{R}^3$ that maps one regular surface \mathcal{M}_o to another \mathcal{M}_d . A regular deformation inflicts no self intersections and defines a smooth one-to-one mapping between the original and the deformed surface.

A deformation deforms the local metric of a surface. This metric deformation is again encoded in the First Fundamental Form (compare Section 2.1.3). Since the function domain of \mathbf{d} is not a euclidean space but a (possibly curved) manifold, the definition of the First Fundamental Form of a deformation requires differential calculus on manifolds.

Many notions from differential calculus in euclidean space translate to manifolds (see [dC76] for an extensive discussion). The First Fundamental Form requires the definition of **the differential, or the directional derivative**, $D\mathbf{d}_p$ of the function \mathbf{d} at point $\mathbf{p} \in \mathcal{M}_o$. The differential is a function $D\mathbf{d}_p : T\mathcal{M}_{o,p} \rightarrow T\mathcal{M}_{d,d(p)}$ that maps vectors (directions) in the tangent plane of the original surface to vectors in the tangent plane of the deformed surface (see Section 2.1.2 for the definition of tangent plane).

Analytically, the differential $D\mathbf{d}_p(\mathbf{x})$ at point \mathbf{p} evaluated at vector $\mathbf{x} \in T\mathcal{M}_{o,p}$ can be computed by mapping any curve $\mathbf{c} \subset \mathcal{M}_o$ that passes through \mathbf{p} with tangent \mathbf{x} to the deformed curve $\mathbf{d}(\mathbf{c}) \subset \mathcal{M}_d$ and computing the tangent vector of the deformed curve at $\mathbf{d}(\mathbf{p})$.

For a local surface parameterization \mathbf{f} , we represented the differential with respect to the canonical basis by the Jacobian matrix of \mathbf{f} evaluated at $\mathbf{f}^{-1}(\mathbf{p})$ (see Section 2.1.3). A Jacobian representation of the differential of the deformation function \mathbf{d} at point $\mathbf{p} \in \mathcal{M}_o$ can be constructed geometrically by mapping two orthogonal unit tangent vectors $\mathbf{e}_1, \mathbf{e}_2 \in T\mathcal{M}_{o,p}$ to the corresponding tangent vectors $D\mathbf{d}_p(\mathbf{e}_1), D\mathbf{d}_p(\mathbf{e}_2) \in T\mathcal{M}_{d,d(p)}$ on the deformed surface (Figure 2.5).

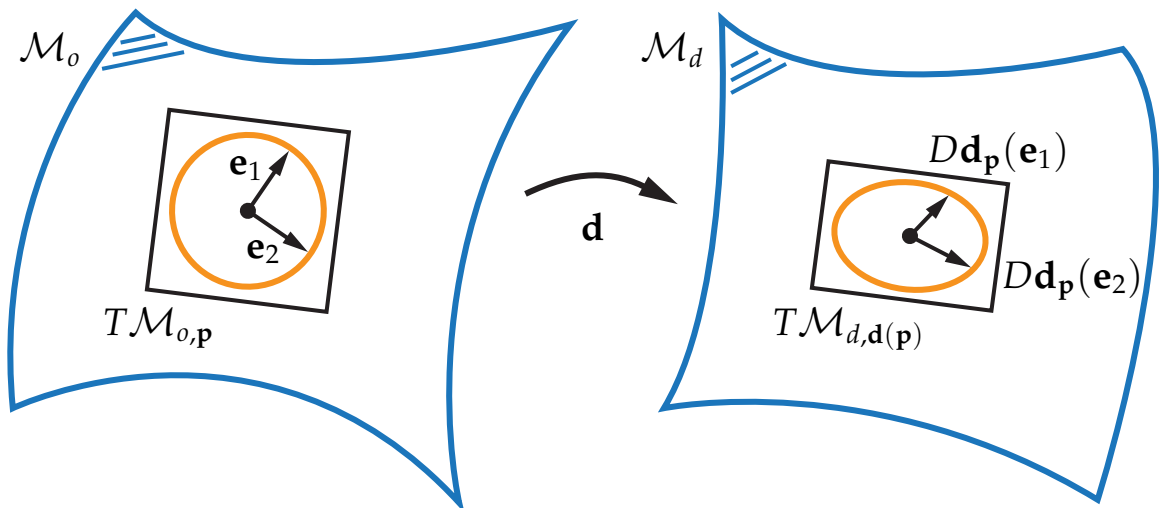


Figure 2.5 The deformation function \mathbf{d} locally deforms circles to ellipses. Original surface \mathcal{M}_o , deformed surface \mathcal{M}_d . Orthogonal basis vectors $\mathbf{e}_1, \mathbf{e}_2$ in the tangent plane $T\mathcal{M}_{o,p}$ of the original surface. The corresponding directional derivatives $D\mathbf{d}_p(\mathbf{e}_1), D\mathbf{d}_p(\mathbf{e}_2)$ of the deformation lie in the tangent plane $T\mathcal{M}_{d,d(p)}$ of the deformed surface.

2 Fundamentals

The two vectors $D\mathbf{d}_{\mathbf{p}}(\mathbf{e}_1), D\mathbf{d}_{\mathbf{p}}(\mathbf{e}_2)$ are the column vectors of the Jacobian $\mathbf{J}_{\mathbf{d}|\mathbf{e}_1, \mathbf{e}_2|_{\mathbf{p}}}$ of the deformation function at \mathbf{p} locally parametrized by the coordinate basis $\{\mathbf{e}_1, \mathbf{e}_2\}$ spanning a 2-dimensional euclidean space. The **First Fundamental Form of a deformation \mathbf{d}** at point \mathbf{p} with respect to the basis $\{\mathbf{e}_1, \mathbf{e}_2\}$ can be represented by the square symmetric matrix

$$\mathbf{J}_{\mathbf{d}|\mathbf{e}_1, \mathbf{e}_2|_{\mathbf{p}}}^{\top} \mathbf{J}_{\mathbf{d}|\mathbf{e}_1, \mathbf{e}_2|_{\mathbf{p}}} \quad (2.7)$$

As with the First Fundamental Form of local surface parameterizations discussed in Section 2.1.3, the metric information encoded in the First Fundamental Form of a deformation is independent on the specific choice of the parameterization basis $\{\mathbf{e}_1, \mathbf{e}_2\}$. The First Fundamental Form of a deformation defines a scalar product of tangent vectors in the deformed surface with respect to tangent vectors in the original surface and encodes how tangent vectors are stretched and turned by the deformation. As depicted in Figure 2.5 a regular surface deformation locally deforms discs into ellipses. The shape of the ellipse is described by the First Fundamental Form of the deformation. The square roots of the eigenvalues of the First Fundamental Form (i.e., the singular values of the Jacobian of \mathbf{d}) define the major and minor radii of the ellipse. The local area change induced by the deformation, for example, can be computed by the square root of the determinant of the First Fundamental Form. In Chapter 3 we discuss ways to use and discretize metric distortion induced by surface deformation.

2.1.5 Curvature

Curvature measures how curved, or bent, a curve or a surface is. It is defined as the change of the tangent, or equivalently the change of the normal, in a given direction. The **curvature κ of a curve $\mathbf{c}(t)$** parameterized by arc length can be computed as the derivative of the tangent vector $\mathbf{c}'(t)$:

$$\kappa(t) = |\mathbf{c}''(t)| \quad (2.8)$$

The second derivative $\mathbf{c}''(t)$ is called the **curvature vector** and is always orthogonal to the tangent (Figure 2.1). The **normal** $\mathbf{n}_c(t)$ of a curve is thus defined as

$$\mathbf{n}_c(t) = \frac{\mathbf{c}''(t)}{|\mathbf{c}''(t)|} = \frac{\mathbf{c}''(t)}{\kappa(t)} \quad (2.9)$$

wherever the curve has non-zero curvature. The curvature of a circle is the inverse of its radius $1/r$. The circle that touches the curve at point $\mathbf{c}(t)$ with tangent vector $\mathbf{c}'(t)$ and that has the same curvature as the curve at t is called the osculating circle (Figure 2.1). The osculating circle has center $\mathbf{c}(t) + \mathbf{c}''(t)/|\mathbf{c}''(t)|^2$ and is the best second-order local approximation of the curve.

Since a surface is 2-dimensional, there are several measures of curvature at a point on the surface. The **normal curvature** $\kappa_n(\mathbf{p}, \mathbf{x})$ at a point \mathbf{p} on a regular surface \mathcal{M} in the tangent direction $\mathbf{x} \in T\mathcal{M}_p$ is defined as the curvature of the curve $\mathbf{c}_{p,x} \subset \mathcal{M}$ obtained by intersecting \mathcal{M} with the normal plane $N_{p,x}$ spanned by \mathbf{x} and the surface normal $\mathbf{n}_{\mathcal{M}}(\mathbf{p})$ (see Figure 2.6).

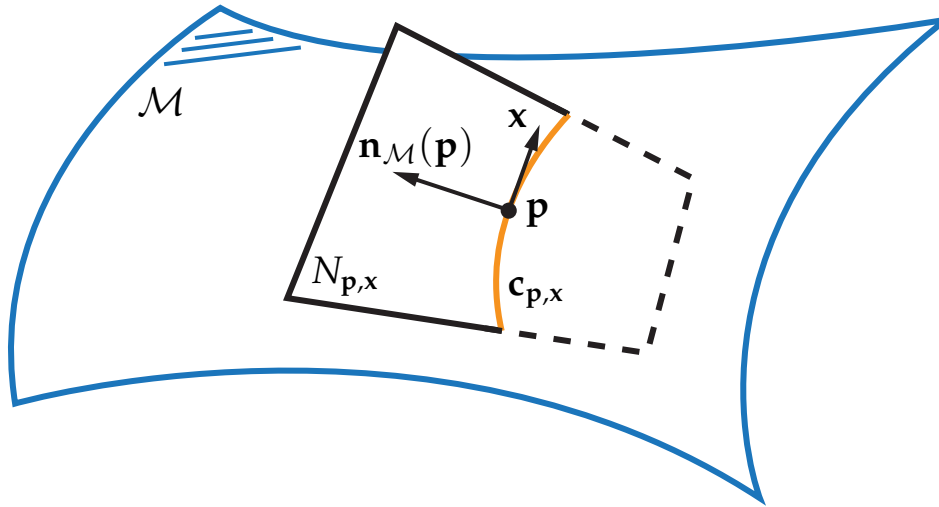


Figure 2.6 The normal curvature is the curvature of the curve $\mathbf{c}_{p,x}$ obtained by intersecting the normal plane $N_{p,x}$ at \mathbf{p} , spanned by surface normal $\mathbf{n}_{\mathcal{M}}(\mathbf{p})$ and direction \mathbf{x} , with the surface \mathcal{M} .

Without loss of generality we assume $\mathbf{c}_{p,x}$ to be parameterized by arc length and $\mathbf{c}(0) = \mathbf{p}$. The normal curvature can therefore be computed as

$$\kappa_n(\mathbf{p}, \mathbf{x}) = |\mathbf{c}_{p,x}''(0)| = \left\langle \mathbf{c}_{p,x}''(0), \mathbf{n}_{\mathcal{M}}(\mathbf{c}_{p,x}(0)) \right\rangle. \quad (2.10)$$

2 Fundamentals

Since $\langle \mathbf{c}'_{\mathbf{p},\mathbf{x}}(t), \mathbf{n}_{\mathcal{M}}(\mathbf{c}_{\mathbf{p},\mathbf{x}}(t)) \rangle = 0$, it also holds $\left(\langle \mathbf{c}'_{\mathbf{p},\mathbf{x}}(t), \mathbf{n}_{\mathcal{M}}(\mathbf{c}_{\mathbf{p},\mathbf{x}}(t)) \rangle \right)' = 0$ and therefore:

$$\kappa_n(\mathbf{p}, \mathbf{x}) = \left\langle \mathbf{c}''_{\mathbf{p},\mathbf{x}}(0), \mathbf{n}_{\mathcal{M}}(\mathbf{c}_{\mathbf{p},\mathbf{x}}(0)) \right\rangle \quad (2.11)$$

$$= - \left\langle \mathbf{c}'_{\mathbf{p},\mathbf{x}}(0), \mathbf{n}'_{\mathcal{M}}(\mathbf{c}_{\mathbf{p},\mathbf{x}}(0)) \right\rangle \quad (2.12)$$

$$= \left\langle \mathbf{c}'_{\mathbf{p},\mathbf{x}}(0), -D\mathbf{n}_{\mathcal{M},\mathbf{p}}(\mathbf{c}'_{\mathbf{p},\mathbf{x}}(0)) \right\rangle \quad (2.13)$$

Equation 2.13 makes again use of the differential D (directional derivative - see Section 2.1.4) and directly confirms the statement at the beginning of this section that curvature measures the normal change in a given tangent direction. The differential $L_{\mathbf{p}} := -D\mathbf{n}_{\mathcal{M},\mathbf{p}} : T\mathcal{M}_{\mathbf{p}} \rightarrow \mathbb{R}^3$ of the normal field of a surface is called the **Weingarten Map**. An interesting fact is that the range of the Weingarten Map is again the tangent plane of the surface, that is, the Weingarten Map is actually a function $L_{\mathbf{p}} : T\mathcal{M}_{\mathbf{p}} \rightarrow T\mathcal{M}_{\mathbf{p}}$.

Using the Weingarten Map and Equation 2.13 one can derive the general function

$$h_{\mathbf{p}}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, -D\mathbf{n}_{\mathcal{M},\mathbf{p}}(\mathbf{y}) \rangle = g_{\mathbf{p}}(\mathbf{x}, L_{\mathbf{p}}(\mathbf{y})), \quad \text{for } \mathbf{x}, \mathbf{y} \in T\mathcal{M}_{\mathbf{p}}. \quad (2.14)$$

The function h is called the **Second Fundamental Form** of a surface. The Second Fundamental Form measures the normal change at point \mathbf{p} in tangent direction \mathbf{y} restricted to tangent direction \mathbf{x} . The normal curvature in unit direction \mathbf{x} can therefore be computed using the Second Fundamental Form as

$$\kappa_n(\mathbf{p}, \mathbf{x}) = h_{\mathbf{p}}(\mathbf{x}, \mathbf{x}). \quad (2.15)$$

In fact, as we shall see now, the Second Fundamental Form stores all information on surface curvature: The Weingarten Map is a linear map and symmetric with respect to the scalar product $g_{\mathbf{p}}$, that is $g_{\mathbf{p}}(\mathbf{x}, L_{\mathbf{p}}(\mathbf{y})) = g_{\mathbf{p}}(L_{\mathbf{p}}(\mathbf{x}), \mathbf{y})$ and therefore also h is symmetric $h_{\mathbf{p}}(\mathbf{x}, \mathbf{y}) = h_{\mathbf{p}}(\mathbf{y}, \mathbf{x})$. This implies, that the extremal values of $h_{\mathbf{p}}(\mathbf{x}, \mathbf{x})$ are the eigenvalues $\kappa_1(\mathbf{p})$ and $\kappa_2(\mathbf{p})$ of the Weingarten Map $L_{\mathbf{p}}$ and the directions $\mathbf{x}_1(\mathbf{p})$ and $\mathbf{x}_2(\mathbf{p})$ to obtain these values are the orthogonal eigenvectors of $L_{\mathbf{p}}$, i.e., $L_{\mathbf{p}}(\mathbf{x}_{\{1,2\}}) = \kappa_{\{1,2\}}\mathbf{x}_{\{1,2\}}$. The normal curvatures $\kappa_1(\mathbf{p})$ and $\kappa_2(\mathbf{p})$ are the maximum and minimum of the normal

curvatures at point \mathbf{p} in all directions and are called the **principal curvatures**. The corresponding orthogonal tangent directions $\mathbf{x}_1(\mathbf{p})$ and $\mathbf{x}_2(\mathbf{p})$ are called the **principal directions**. The **mean curvature** $H(\mathbf{p})$ is the average, the **Gaussian curvature** $K(\mathbf{p})$ is the product of the principal curvatures:

$$H(\mathbf{p}) := \frac{\kappa_1(\mathbf{p}) + \kappa_2(\mathbf{p})}{2} = \frac{1}{2}\text{trace}(L_{\mathbf{p}}) \quad (2.16)$$

$$K(\mathbf{p}) := \kappa_1(\mathbf{p})\kappa_2(\mathbf{p}) = \det(L_{\mathbf{p}}) \quad (2.17)$$

Similar to the First Fundamental Form, the Second Fundamental Form can also be represented by a square symmetric matrix $\mathbf{W}_{\mathbf{p}}$

$$h_{\mathbf{p}}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{W}_{\mathbf{p}}\mathbf{y}. \quad (2.18)$$

The matrix $\mathbf{W}_{\mathbf{p}}$ is representing the Weingarten Map in the canonical basis of \mathbb{R}^3 and is sometimes called the **Weingarten Matrix** or the **Shape Operator**. The two non-zero eigenvalues of the matrix \mathbf{W} and the corresponding eigenvectors are again the principal curvatures and directions. The Weingarten Map, the Second Fundamental Form, and the Weingarten Matrix can also be defined for local parameterizations (in which case \mathbf{W} is a 2×2 matrix, [dC76]). It is a classical result from differential geometry that the First and Second Fundamental Form together uniquely determine the geometry of a regular surface [dC76]).

There are numerous methods of how to discretize curvatures and the Second Fundamental Form on triangle meshes. See [BPK⁺08] for an overview. Discrete curvatures are often derived by locally fitting smooth elements to the discrete representations (e.g., [CP05, Glo4]), or using the Taylor approximation and other approximation strategies (e.g., [CS92, Tau95a, HSo2, MDSBo2, CSMo3, Rus04] and for curves [LBS05]). The emerging field of discrete differential geometry [BS09] provides a sound mathematical framework to transfer properties and techniques known from classical differential geometry to discrete surfaces.

2.1.6 Fairness

In many applications of digital geometry processing the fairness, that is, the aesthetic quality of curves and surfaces, is an important objective. High aesthetic quality of curves or surfaces often means low variation of first or higher order derivatives. Fairness of derivatives is sometimes qualitatively called *smoothness* of a curve or surface. A low variation of derivatives is related to and motivated by physical properties of real world materials like the bending behavior of thin plates of metal or the area minimizing property of membrane surfaces such as soap bubbles. There exists a great amount of literature on how to ensure and optimize the aesthetic quality of curves and surfaces (see [BPK⁺08] for an overview).

A simple example is the differential variation of curves. k -th order fairness of a curve parameterized by arc length can be defined as minimizing the functional

$$E_{fair}^k(\mathbf{c}) = \int_I |\mathbf{c}^{(k)}(t)|^2 dt. \quad (2.19)$$

Minimizing E_{fair}^1 produces curves with minimal length and corresponds to the physical behavior of rubber bands. Minimizing E_{fair}^2 tries to make the curve locally as straight as possible, given other possibly conflicting objectives, and corresponds to the bending behavior of thin wooden rods. Minimizing E_{fair}^3 means low variation of curvature, that is, the curve should maintain its curvature as well as possible along I . And so on.

On polylines, the variational (energy minimization) problem of minimizing E_{fair}^k can be solved by discretizing the derivatives of \mathbf{c} using finite differences, transforming the problem to a (linear) least squares problem (see Section 2.2). E_{fair}^2 for a polyline through the points $\{\mathbf{p}_i\}$, for example, can be discretized as

$$E_{fair}^2(\{\mathbf{p}_i\}) = \sum_i \left(\frac{\mathbf{p}_{i-1} - 2\mathbf{p}_i + \mathbf{p}_{i+1}}{h^2} \right)^2 h, \quad (2.20)$$

where h is the average edge length

$$h = \frac{|\mathbf{p}_{i+1} - \mathbf{p}_i| + |\mathbf{p}_i - \mathbf{p}_{i-1}|}{2}. \quad (2.21)$$

In order for the minimization problem to be well defined, additional objectives (or boundary conditions) are required.

On surfaces, this general concept of k -th order fairness applies as well and is in practice often solved by transforming the variational problem to a partial differential equation involving the laplacian using the Euler-Lagrange equation [[Tau95b](#), [DMSB99](#), [BPK⁺08](#)].

The notion of fairness is also related to deformations: A deformation should often maintain or even improve the fairness of a curve or surface, which may contradict other deformation objectives. In [Chapters 3 and 4](#) we will apply and discuss various fairness measures for constraint based surface processing.

2.2 Least Squares Optimization

This section describes techniques and provides practical insights to solve the following optimization problem:

The Least Squares Problem. Given a vector of m functions $\mathbf{f} = (f_1, \dots, f_m)^\top : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that depend on n variables $\mathbf{x} = (x_1, \dots, x_n)^\top$ with $m \geq n$. Find the vector \mathbf{x}^* that minimizes all the functions simultaneously in a least squares sense:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} F(\mathbf{x}) \quad (2.22)$$

$$F(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^m (f_i(\mathbf{x}))^2 = \frac{1}{2} \|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2} \mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}). \quad (2.23)$$

The factor $1/2$ does not influence the result but is introduced for notational convenience in the following discussion.

Motivation. Many optimization problems in digital geometry processing are least squares problems as defined by Equations 2.22-2.23. Fitting a set of points with a curve or a surface, for example, involves minimizing the squared distances between the points and the curve or surface. The square norm is used since it represents the most even and natural combination of different objectives. The iso-regions of the squared distance to a point in Euclidean space are spheres. The use of the squared distances for the application of point fitting can also be motivated by the assumption that the points represent discrete *measurements* of the curve or surface that where subject to gaussian distributed measurement errors [Mar09]. Another reason for the squared nature of the objective function F can be the approximation of physical properties of real-world objects as for example the fairness functionals discussed in Section 2.1.6. Qualitatively speaking, squaring the functions f_i penalizes larger errors significantly more than small ones, leading to an even distribution of the total error over all functions f_i . The functions f_i can also be seen as a set of possibly conflicting constraints $f_i = 0$. The least squares problem defines a possible way to resolve the issue that not all constraints can be met simultaneously. From this viewpoint it is also clear why there have to be more constraints than variables ($m \geq n$) for the optimization problem to be well defined.

Context. The least squares problem is a special instance of the general optimization problem $\mathbf{x}^* = \arg \min_{\mathbf{x}} F(\mathbf{x})$, since the objective function F has the special structure of a sum of squares. This special structure of F enables efficient solutions with superlinear convergence without the need of computing second derivatives as required by more general optimization methods. The fact that second derivatives are not required can be very advantageous, especially if the derivatives of the involved terms f_i are very hard and costly to compute. If the constraints f_i have a simple form and the second derivatives can be efficiently computed, however, more general optimization schemes relying on second derivatives can produce better results (see e.g., [DS87, GMW89, Fle00, Rus06, PFT04]). Since the least squares problems introduced in this thesis involve very complex constraints f_i , this chapter focuses on algorithms exploiting the least squares structure and requiring first derivatives only. For further details on least squares problems and algorithms we direct the interested reader to the comprehensive literature on this topic, e.g., [Bjö96, MNT04].

Linear Least Squares

If all the functions f_i in Equation 2.23 are linear, the problem is called a linear least squares problem. In this case, the vector \mathbf{f} has the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x}, \quad (2.24)$$

where \mathbf{A} is a rectangular $m \times n$ matrix and \mathbf{b} is a vector with m entries. At the minimum \mathbf{x}^* the gradient of the objective Function $\nabla F(\mathbf{x})$ will be the zero vector. The gradient of F can be computed as

$$\nabla F(\mathbf{x}) = -\mathbf{A}^\top (\mathbf{b} - \mathbf{A}\mathbf{x}). \quad (2.25)$$

Therefore, the solution \mathbf{x}^* can be obtained by solving the linear system of equations called the **Normal Equations**

$$(\mathbf{A}^\top \mathbf{A})\mathbf{x}^* = \mathbf{A}^\top \mathbf{b}. \quad (2.26)$$

One problem with the normal equations is that the square symmetric matrix $(\mathbf{A}^\top \mathbf{A})$ is often ill-conditioned. This can be resolved by replacing \mathbf{A} with its QR-decomposition (or *thin QR-decomposition*, see [GL96])

$$\mathbf{A} = \mathbf{Q}\mathbf{R}, \quad (2.27)$$

where \mathbf{Q} is an $m \times n$ matrix with orthonormal columns (that is, $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$) and \mathbf{R} is an upper triangular $n \times n$ matrix. Applying the QR-decomposition to \mathbf{A} the normal equation transforms to

$$\mathbf{R}^\top \mathbf{Q}^\top \mathbf{Q} \mathbf{R} \mathbf{x}^* = \mathbf{R}^\top \mathbf{Q}^\top \mathbf{b} \quad (2.28)$$

$$\Leftrightarrow \mathbf{R}^\top \mathbf{R} \mathbf{x}^* = \mathbf{R}^\top \mathbf{Q}^\top \mathbf{b} \quad (2.29)$$

$$\Leftrightarrow \mathbf{R} \mathbf{x}^* = \mathbf{Q}^\top \mathbf{b}. \quad (2.30)$$

The linear Equation 2.30 can be efficiently solved by back substitution. Solving the linear least squares problem via the QR-decomposition is more accurate and stable than via the normal equations.

The least squares problems introduced in this thesis are all **Non-Linear Least Squares Problems**. We will thus focus on this more general case for the remainder of this Chapter. Interestingly, however, the solution of non-linear least squares problems usually involves the repeated solution of linear least squares problems, as we will see in the following.

2.2.1 The Gauss-Newton Algorithm

The Gauss-Newton Algorithm is an iterative algorithm to solve non-linear least squares problems (Equations 2.22, 2.23) without the need of second derivatives. The Gauss-Newton Algorithm forms the basis of more specialized algorithms like the Levenberg-Marquardt Algorithm discussed in Section 2.2.2. The underlying concept, however, remains the same for all Gauss-Newton type methods:

The Gauss-Newton Algorithm linearly approximates the functions (constraints) $f_i(\mathbf{x})$ using the Taylor expansion at a current solution \mathbf{x}_i as

$$\mathbf{f}(\mathbf{x}_i + \mathbf{d}) \simeq \mathbf{f}(\mathbf{x}_i) + \mathbf{J}(\mathbf{x}_i) \mathbf{d}, \quad (2.31)$$

where $\mathbf{J}(\mathbf{x}_i)$ is the Jacobian matrix of $\mathbf{f}(\mathbf{x})$ evaluated at \mathbf{x}_i . With this approximation $\mathbf{f}(\mathbf{x}_i + \mathbf{d})$ is linear and $F(\mathbf{x}_i + \mathbf{d})$ is quadratic in \mathbf{d} . Thus the least squares problem is locally approximated with a linear least squares problem for small \mathbf{d} around \mathbf{x}_i and we can find the solution \mathbf{d}^* using the normal equations (see paragraph Linear Least Squares in Section 2.2)

$$(\mathbf{J}(\mathbf{x}_i)^\top \mathbf{J}(\mathbf{x}_i)) \mathbf{d}^* = -\mathbf{J}(\mathbf{x}_i)^\top \mathbf{f}(\mathbf{x}_i). \quad (2.32)$$

The solution \mathbf{d}^* can then be added to the current solution to obtain an improved estimation $\mathbf{x}_i + \mathbf{d}^*$ of the minimum of $F(\mathbf{x})$.

With $\mathbf{J} := \mathbf{J}(\mathbf{x}_i)$, $\mathbf{f} := \mathbf{f}(\mathbf{x}_i)$, the core step of the Gauss-Newton Algorithm is thus

$$\boxed{\begin{aligned} \text{Solve } (\mathbf{J}^\top \mathbf{J}) \mathbf{d}^* &= -\mathbf{J}^\top \mathbf{f} \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \alpha \mathbf{d}^* \end{aligned}} \quad (2.33)$$

where α is called the **Step Size** and is a value in the interval $(0, 1]$.

Discussion

- **Initialization:** The Gauss-Newton algorithm requires an initialization \mathbf{x}_0 of the optimization variables as input. Since the objective function is non-linear and mostly has a complex shape with many local minima, the performance of the Gauss-Newton algorithm as well as the quality of the result critically depends on this initial variable vector. Obviously, \mathbf{x}_0 should already be as close as possible to the minimum of $F(\mathbf{x})$. The specific choice of \mathbf{x}_0 , however, entirely depends on the application. We will discuss initializations of the non-linear least squares problems presented in this thesis in Chapters 3 and 4.
- **Convergence, Step Size, and Line Search:** \mathbf{d}^* is always a descend direction at $F(\mathbf{x}_i)$, that is, $F(\mathbf{x}_i + \alpha \mathbf{d}^*) < F(\mathbf{x}_i)$ for a sufficiently small α . Given that F has a well defined minimum and that the Jacobian $\mathbf{J}(\mathbf{x}_i)$ has full rank in all steps, the algorithm will thus converge if α is properly chosen in every iteration. There are various, so-called **Line Search**, strategies to choose α (see [MNT04]). In the implementation details below (Section 2.2.3) we will describe a simple but efficient line search strategy we use throughout this thesis.
- **Speed of Convergence:** There are two interesting interpretations of the Gauss-Newton step (Equation 2.33): The first interpretation, already mentioned above, is that the non-linear least squares problem is reduced to a linear least squares problem in every iteration by linearly approximating $\mathbf{f}(\mathbf{x})$ at the current so-

lution \mathbf{x}_i . From this analogy it is clear that if all the constraints $f_i(\mathbf{x})$ are linear, the Gauss-Newton algorithm will converge to the optimal solution in only one step, as the problem is in this case a linear least squares problem. The second interpretation is that the Gauss-Newton algorithm is nothing but the classical Newton algorithm (see [PFT04, MNT04]) with an approximated Hessian matrix of F instead of the exact Hessian \mathbf{H}_F . The classical Newton algorithm solves the general optimization problem $\mathbf{x}^* = \arg \min_{\mathbf{x}} F(\mathbf{x})$ by iteratively solving the linear system

$$\mathbf{H}_F(\mathbf{x}_i) \mathbf{d}^* = -\nabla F(\mathbf{x}_i) \quad (2.34)$$

compared to the Gauss-Newton system

$$(\mathbf{J}(\mathbf{x}_i)^\top \mathbf{J}(\mathbf{x}_i)) \mathbf{d}^* = -\mathbf{J}(\mathbf{x}_i)^\top \mathbf{f}(\mathbf{x}_i). \quad (2.35)$$

The right hand sides of Equations 2.34 and 2.35 are identical because $\mathbf{J}(\mathbf{x}_i)^\top \mathbf{f}(\mathbf{x}_i)$ is exactly the gradient of F at \mathbf{x}_i . Therefore, the Gauss-Newton step is the same as the Newton step except of using the matrix $(\mathbf{J}(\mathbf{x}_i)^\top \mathbf{J}(\mathbf{x}_i))$ instead of the Hessian of F . The matrix $(\mathbf{J}(\mathbf{x}_i)^\top \mathbf{J}(\mathbf{x}_i))$ is the Hessian of the quadratic approximation of $F(\mathbf{x}_i + \mathbf{d})$ around \mathbf{x}_i using the linearized vector \mathbf{f} (see Equation 2.31) and is therefore an approximation of the true Hessian of F . Using the special structure of F (Equation 2.23) one can show that the Hessian of F can indeed be computed as

$$\mathbf{H}_F(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}) + \sum_{i=1}^m f_i(\mathbf{x}) \mathbf{H}_{f_i}(\mathbf{x}), \quad (2.36)$$

where $\mathbf{H}_{f_i}(\mathbf{x})$ are the Hessian matrices of the components $f_i(\mathbf{x})$. Equation 2.36 offers very interesting insights to the Gauss-Newton algorithm. The classical Newton algorithm is known to have quadratic convergence speed in contrast to the linear convergence of steepest descent methods. The Gauss-Newton algorithm exploits the special structure of F to obtain an approximation of the Hessian only requiring first order derivatives and uses this approximate Hessian to perform the Newton iteration. With this adapted Newton scheme the Gauss-Newton algorithm can achieve superlinear or close to quadratic convergence without the need of second derivatives. Interestingly, Equation 2.36

also indicates that the quality of the Hessian approximation, and therefore also the speed of convergence, depends on two factors: the values and the second derivatives of the functions $f_i(\mathbf{x})$. The closer the values of $f_i(\mathbf{x})$ are to zero, that is, the closer F is to its best possible minimum of zero, the faster the algorithm will converge, which is a surprising relation. The second relation is more intuitive and also coincides with the first interpretation of iteratively approximating the problem by a linear least squares problem: the smaller the curvature of the functions $f_i(\mathbf{x})$, that is, the more linear the functions, the faster the convergence. In Chapter 3 we will see a specific example of how this insight can be exploited to improve the efficiency of the optimization.

2.2.2 The Levenberg-Marquardt Algorithm

The Levenberg-Marquardt Algorithm is a **damped** version of the Gauss-Newton algorithm. A damped version of the normal equations is solved in each step which replaces the step size of the standard Gauss-Newton method:

$$\begin{aligned} \text{Solve } (\mathbf{J}^\top \mathbf{J} + \mu \mathbf{I}) \mathbf{d}^* &= -\mathbf{J}^\top \mathbf{f} \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{d}^* \end{aligned} \quad (2.37)$$

\mathbf{I} is the identity matrix and $\mu \in \mathbb{R}^+$ is the damping factor. The Levenberg-Marquardt algorithm is essentially a hybrid version between the gradient descend and the Gauss-Newton methods. If the damping factor μ is large we get

$$\mathbf{d}^* \simeq -\frac{1}{\mu} \mathbf{J}^\top \mathbf{f} = -\frac{1}{\mu} \nabla F(\mathbf{x}) \quad (2.38)$$

and therefore a small step in the direction of steepest descent is performed. If μ is small, the Levenberg-Marquardt step moves the current solution in a similar direction as the Gauss-Newton step. The damping factor μ therefore not only controls the step size but also the step direction and is automatically adjusted in every iteration to optimally adapt to the stage of the optimization (see [MNT04, Nie99] on how to initialize and adapt μ).

Levenberg-Marquardt versus Gauss-Newton. Even though Levenberg-Marquardt is an extension of the Gauss-Newton algorithm it is not in all cases the superior algorithm. As detailed in Section 2.2.3 the decision of which Gauss-Newton variant to choose should always be based on a good understanding and careful evaluation of the optimization problem. In our experience, the standard Gauss-Newton algorithm performs generally very well on a broad variety of non-linear least squares optimization problems, whereas the Levenberg-Marquardt algorithm can have better convergence for specific problem instances but is also more difficult to control. The main difficulty of the Levenberg-Marquardt algorithm is that the step control parameter μ controls both the step size and the step direction at the same time. This can be very hindering, especially if the initial value of the variable vector \mathbf{x}_0 is far apart from the desired minimum. The Levenberg-Marquardt algorithm will then mostly follow the steepest descent which can lead to convergence problems well known for gradient descent methods. One cause of such convergence problems is when the functions f_i describe very local objectives but the final solution requires a drastic global change of the variables. A specific example is the smoothing of a surface: assume we wish to drastically smooth a surface discretized by a triangular mesh. We implement the smoothing by simultaneously minimizing functions f_i describing the smoothness of local surface patches consisting of the i -th vertex on the mesh and its one-ring neighborhood of adjacent vertices. The gradient descent will only minimize the f_i in a very local manner, smoothing out the high frequencies of the surface very quickly, but requiring a lot of time for the lower frequencies to smooth out. The problem is even more dramatic if a small change at one point on the surface should propagate larger changes to the whole surface. Such situations are often encountered in constraint based surface modeling (see Chapter 3). Additionally, in our experience the Levenberg-Marquardt scheme is more sensitive to changes in the problem configuration, since small changes in the adaption of μ change the step direction and size and can lead to rather different solutions. For these reasons and experiences, we prefer to use the standard Gauss-Newton algorithm for the optimization problems presented in this thesis.

2.2.3 Implementation Insights

We conclude this Chapter with a collection of insights and thoughts on solving non-linear least squares problems collected in several years of implementation and application experience.

1. Which XYZ should I use?

Which solver should I use? Which step size control mechanism should I use? When implementing a solution to an optimization problem one faces many design decisions. The basis of these decisions should always be the underlying application. This means: understand your user well and understand your problem well.

In applied research, providing tools that will serve a user in industry is often the driving goal. It is therefore invaluable to obtain as much understanding on the user requirements as possible. Should the solution allow interactive control and adjustment by the user? Or is accuracy and quality of the result more important than computation time? What means and level of control are desired by the user? The answer to these questions can validate or invalidate design decisions down to the smallest implementation details.

An understanding of the problem is crucial for the more technical decisions. The discussion on Gauss-Newton versus Levenberg-Marquardt in Section 2.2.2 is a good example of how insights on the details of the problem domain can help to determine high level design questions like which solver should be used. Non-linear least squares problems in digital geometry processing often have a complex objective function with ten or hundred thousands of unknowns and constraints and with a complicated energy landscape. It is important to obtain as much understanding about the objective function, its shape and its variables, as possible through testing, visualizations, and breaking the objective down to its components. A deep understanding of the problem helps both making the right design decisions and evaluating the implementation (see Item 2).

2. Never Stop Testing!

Non-linear least squares problems are often so complex that they are very hard to evaluate. The iterative solvers will often converge and might give reasonable solutions, even if the implementation of derivatives or other computations is incorrect. In our experience we have learned the following guidelines: Understand your problem (see Item 1). Be suspicious if the results are not as expected. Try to understand rather than speculate why the results are not as expected. Take the time it needs to carefully investigate the details that might cause the problems, it will eventually pay off. Never blame the numerics too quickly, the problems are much more likely to originate from mistakes in implementation or design. If the problems are numerical problems you should be able to understand and reproduce exactly where the numerical errors originate. As mentioned above, non-linear least squares problems are often very complex. The absolutely best tool to understand and evaluate the implementation is therefore the TEST. Testing should be done frequently and in small blocks. Test every component. Testing five things simultaneously is tedious and error-prone. If the implementation is extended by adding new code, gradually compare the results with the old implementation whenever possible, since the older implementation is often the only reference implementation you have. Test your mathematical computations by comparing the results with results from established mathematics software. The computation of derivatives should always be tested by comparing them to numerical derivatives obtained for example by finite differences. To our experience it is extremely helpful to have a general automated system to test derivatives that can be reused by further non-linear least squares implementations. Testing will not only help you find and prevent errors but also provide you with a deeper understanding of the problem domain.

3. Sparse Problems

If all the functions f_i in Equation 2.23 only depend on a small subset of the variables x_j , the least squares problem is called sparse. For the Gauss-Newton algorithm this implies that the linear system to be solved in every iteration is sparse, because the i -th row

of the Jacobian matrix \mathbf{J} only has a non-zero entry in the j -th column, if the function f_i depends on variable x_j . The sparsity structure of the normal equation matrix $\mathbf{J}^\top \mathbf{J}$ is a bit more complicated: the entry (i, j) is non-zero, if there exists at least one function f_k that depends on both variables x_i and x_j . In practice $\mathbf{J}^\top \mathbf{J}$ is usually denser than \mathbf{J} but still sparse if \mathbf{J} is sparse. In Chapter 3 we present a detailed analysis of the sparsity structure of \mathbf{J} and $\mathbf{J}^\top \mathbf{J}$ for a specific least squares problem instance.

The sparsity can be exploited by using a sparse linear system solver, which brings the cubic ($\mathcal{O}(n^3)$) computation times of dense linear system solvers down to linear time in practice. "In practice" because the effective computing time depends on the quality of the ordering of the matrix rows, the sparsity structure, and the matrix itself. Those are impossible to predict. In practice, however, the solvers based on the sparse Cholesky factorization achieve computation times linear in the number of non-zero entries of the matrix describing the linear system [GL81, BPK⁺08]. All least squares problems presented in this thesis are sparse problems and we use the solvers PARISO [SG04] and CHOLMOD [CDHR09] which achieve similar performance for our problems.

4. QR-Decomposition

We have stated at the beginning of Section 2.2 that the QR-decomposition provides more accurate solutions to the linear least squares problem than the normal equations. Since the Gauss-Newton algorithm solves a linear least squares problem in every iteration, we could make use of this fact by solving the linear system using the QR-Decomposition instead of the normal equations in each iteration. The QR-decomposition has complexity $\mathcal{O}(mn^2)$ and is usually slightly slower than solving the normal equations. The difference becomes more significant if the Jacobian matrix is very large and sparse, since sparse QR-decompositions (even though very efficient implementations exist, e.g., SuiteSparseQR [Dav]) tend to be slower than the sparse linear system solvers discussed in Item 3. In practice, the QR-Decomposition is the preferred method for dense or small prob-

lems, since it provides higher accuracy, and the normal equations are preferred for large sparse problems and if fast computation times are required. The least squares problems presented in Chapters 3 and 4 of this thesis are large and sparse and for all instances we tested the linear systems were well conditioned. Thus the Gauss-Newton method using the normal equations as described in Section 2.2.1 is applied for all problems in this thesis.

5. Line Search

The Gauss-Newton algorithm outlined in Section 2.2.1 requires an adaptive scheme to choose and control the step size α (Equation 2.33) in every iteration. We have found the following simple but efficient scheme to be sufficient for the problems investigated in this thesis. To choose α , in every iteration solve the normal equations to obtain \mathbf{d}^* and then perform a), b), and c) in this order:

a) $\alpha = 1$

b) while ($F(\mathbf{x}_i + \alpha\mathbf{d}^*) > F(\mathbf{x}_i)$) do $\alpha = \alpha/2$

c) while ($F(\mathbf{x}_i + \alpha\mathbf{d}^*) > F(\mathbf{x}_i + (\alpha/2)\mathbf{d}^*)$) do $\alpha = \alpha/2$

This scheme is called a line search strategy, since it evaluates F along the line described by point \mathbf{x}_i and direction \mathbf{d}^* . Step b) ensures convergence by only accepting an α that improves the current solution. Step c) is based on the observation that the function $F_{\mathbf{x}_i, \mathbf{d}^*}(\alpha) = F(\mathbf{x}_i + \alpha\mathbf{d}^*)$ locally often has a quadratic form. It can therefore pay off to further decrease α until we reach a minimum of F along the search line. If α gets too small during the line search process, the variable vector \mathbf{x} will only change insignificantly and the Gauss-Newton algorithm has converged to a minimum (see Item 8).

6. Computing \mathbf{f} and \mathbf{J}

In many least squares problems, there are different logic blocks of similar objectives f_i that belong together. For example, there might be a set of f_i dedicated to guarantee the smoothness of a surface, while another set of f_i pulls the surface towards a set of points that should be fitted. It has proven to be very convenient

to implement the Gauss-Newton solver in such a way, that blocks of objectives can easily be added or removed from the problem in a plugin-like system. For example, the fitting of points can then easily be tested with different ways to ensure smoothness by simply exchanging the blocks of smoothness objectives. Such a block of objectives is responsible for filling one block of entries in the vector $\mathbf{f}(\mathbf{x})$ and the corresponding derivatives as a block of rows in the Jacobian $\mathbf{J}(\mathbf{x})$.

Often the computation of the derivatives requires values also needed in the computation of the objectives. It therefore makes sense to compute f_i and the corresponding Jacobian row of derivatives in the same routine. Depending on the line search strategy and the speed of convergence, particularly on how often the function value F has to be queried during line search, the computation of the objective vector \mathbf{f} is typically needed much more often than the the computation of the Jacobian \mathbf{J} . It therefore usually makes sense to skip the derivative computation and only compute \mathbf{f} during line search.

7. Computing Sparse $\mathbf{J}^\top \mathbf{J}$

Building the normal equation matrix $\mathbf{J}^\top \mathbf{J}$ is computationally expensive. If \mathbf{J} is a dense $m \times n$ matrix, computing $\mathbf{J}^\top \mathbf{J}$ in a straight forward way has complexity $\mathcal{O}(mn^2)$. If \mathbf{J} is sparse, the complexity not only depends on the dimensions $m \times n$ but also on the number of non-zero entries. In the following, we denote by $\mathbf{A}_{i,j}$ the entry in the i -th row and j -th column of matrix \mathbf{A} and by $\mathbf{A}_{i,*}$ and $\mathbf{A}_{*,j}$ the i -th row and j -th column vector of \mathbf{A} . For a multiplication of an $l \times m$ matrix \mathbf{A} and an $m \times n$ matrix \mathbf{B} it holds

$$(\mathbf{AB})_{i,j} = \mathbf{A}_{i,*} \mathbf{B}_{*,j} = \sum_{k=1}^m \mathbf{A}_{i,k} \mathbf{B}_{k,j} \quad (2.39)$$

An important observation for the multiplication of sparse matrices is that the product can thus be written as a sum of outer products:

$$\mathbf{AB} = \sum_{k=1}^m \mathbf{A}_{*,k} \mathbf{B}_{k,*} \quad (2.40)$$

This enables the following sparse matrix multiplication scheme:

Basic Scheme

```

for  $k = 1$  to  $m$ 
  for all  $i$  with  $\mathbf{A}_{i,k} \neq 0$ 
    for all  $j$  with  $\mathbf{B}_{k,j} \neq 0$ 
       $\mathbf{AB}_{i,j} = \mathbf{AB}_{i,j} + \mathbf{A}_{i,k}\mathbf{B}_{k,j}$ 
    
```

The crucial advantage of this scheme is that the multiplications are only performed with the non-zero entry pairs of $\mathbf{A}_{*,k}$ and $\mathbf{B}_{k,*}$. As a consequence, the number of multiplications and write accesses to \mathbf{AB} do only roughly linearly depend on the number of non-zeros in the matrices (see below for a detailed analysis).

For the multiplication $\mathbf{J}^\top \mathbf{J}$ this means

$$\mathbf{J}^\top \mathbf{J} = \sum_{k=1}^m (\mathbf{J}_{k,*})^\top \mathbf{J}_{k,*} \quad (2.41)$$

that is, the sum goes over all rows $\mathbf{J}_{k,*}$ of the Jacobian.

Let us consider two specific implementations of $\mathbf{J}^\top \mathbf{J}$ following the general scheme described above. One way to represent sparse matrices is as a list of column vectors, where each entry in the vector stores the row index and the value of a non-zero entry. This representation can efficiently provide all non-zero entries of a column of a sparse matrix. Efficient filling of the matrix, however, requires the fill order to have monotonously increasing row indices. This does not hold for the general scheme described above. Therefore the scheme has to be adapted such that the entries $\mathbf{AB}_{i,j}$ are filled with increasing i . An efficient way to do this is to explicitly store both matrices \mathbf{J} as well as \mathbf{J}^\top in the sparse format. Let us assume \mathbf{J} is a $m \times n$ matrix. The multiplication algorithm is then

Algorithm 1

```

for  $i = 1$  to  $n$ 
  for all  $k$  with  $\mathbf{J}_{k,i} \neq 0$ 
    for all  $j$  with  $(\mathbf{J}^\top)_{j,k} \neq 0$ 
       $(\mathbf{J}^\top \mathbf{J})_{i,j} = (\mathbf{J}^\top \mathbf{J})_{i,j} + (\mathbf{J}^\top)_{j,k} \mathbf{J}_{k,i}$ 
    
```

The symmetry of the matrix $\mathbf{J}^\top \mathbf{J}$ leads to the advantage that Algorithm 1 can be applied to sparse matrices stored as lists of columns or as lists of rows. The symmetry of $\mathbf{J}^\top \mathbf{J}$ can further be exploited so that only half of the entries actually have to be computed. Algorithm 1 requires

$$\sum_{i=1}^n \sum_{\{k: \mathbf{J}_{k,i} \neq 0\}} \text{nz} \left((\mathbf{J}^\top)_{*,k} \right) \quad (2.42)$$

computations, where the function `nz` measures the number of non-zeros in a vector. In practice, the non-zero entries are usually evenly distributed over the number of rows and columns. Assume the non-zero values are perfectly distributed, that is, there are NZ/m non-zero values in each row and NZ/n non-zero values in each column, where NZ is the total number of non-zero values in \mathbf{J} . In this case the number of computations required to compute $\mathbf{J}^\top \mathbf{J}$ by Algorithm 1 is

$$\sum_{i=1}^n \frac{\text{NZ}}{m} \frac{\text{NZ}}{n} = \frac{\text{NZ}^2}{m} \quad (2.43)$$

which is essentially linear in NZ if the number of non-zero entries per row is a small constant.

The major drawback of Algorithm 1 is that it requires explicitly storing the transpose of the Jacobian \mathbf{J}^\top . An alternative that does not require this is:

Algorithm 2

```

for  $k = 1$  to  $m$ 
  for all  $i$  with  $\mathbf{J}_{k,i} \neq 0$ 
    for all  $j$  with  $\mathbf{J}_{k,j} \neq 0$ 
       $(\mathbf{J}^\top \mathbf{J})_{i,j} = (\mathbf{J}^\top \mathbf{J})_{i,j} + \mathbf{J}_{k,i} \mathbf{J}_{k,j}$ 

```

This algorithm simply computes the sum all possible pairs of non-zero entries for every row vector $\mathbf{J}_{k,\cdot}$, and adds the result to $\mathbf{J}^\top \mathbf{J}$. There are two issues that have to be addressed with this algorithm: First, the two inner for loops require efficient access to the non-zero entries of the Jacobian rows. Second, the order in

which the entries of $\mathbf{J}^\top \mathbf{J}$ are accessed is arbitrary. The first issue could be resolved by storing the Jacobian by a list of rows instead of columns. The second issue can only be handled efficiently if the sparse matrix $\mathbf{J}^\top \mathbf{J}$ is stored using a hash table that guarantees random access with amortized constant cost. For large matrices, however, hash maps might be too memory consuming.

The option that has proven to be very efficient in practice is to neither store \mathbf{J} nor \mathbf{J}^\top explicitly. Instead, only the current row of the Jacobian is stored. This is possible for most problems, since the derivatives for each function f_i can usually be computed one at a time, that is, the Jacobian can be computed row by row. Once a row of the Jacobian is computed, all possible non-zero entry pairs are multiplied and the result is added to the matrix $\mathbf{J}^\top \mathbf{J}$ as described by Algorithm 2. Since neither the Jacobian nor its transpose have to be stored, this solution saves a lot of memory. For maximum memory efficiency we still store the matrix $\mathbf{J}^\top \mathbf{J}$ as a simple list of column vectors.

The critical operation is the write access to $\mathbf{J}^\top \mathbf{J}$. Inserting a new non-zero entry into a column vector costs time linear in the number of non-zero entries of the column. This does usually not pose a problem in practice, especially if the number of non-zero entries per column is a small constant. In most problems, the sparsity structure of $\mathbf{J}^\top \mathbf{J}$, that is, where the non-zero entries are in the matrix, does not change after the first iteration of the Gauss-Newton algorithm. The access time to $\mathbf{J}^\top \mathbf{J}$ can thus be further reduced by keeping the sparsity structure fixed after the first iteration and performing a binary search on the column vector to find the entry (i, j) leading to a logarithmic complexity.

Since the number of non-zero entries per column is usually small and constant, this algorithm is almost as efficient as Algorithm 1 but needs much less memory. If we again assume the number of non-zero entries per row and column of the Jacobian to be equal

for all rows and columns, Algorithm 2 with the sparse matrix representations discussed above requires

$$\sum_{i=1}^m \left(\frac{\text{NZ}}{m}\right)^2 \log\left(\frac{\text{NZ}}{n}\right) = \frac{\text{NZ}^2}{m} \log\left(\frac{\text{NZ}}{n}\right) \quad (2.44)$$

computations, which is again essentially linear in NZ, the number of non-zero entries of \mathbf{J} , if the number of non-zero entries per row and column is constant.

Another advantage of Algorithm 2 and the separate computation of the Jacobian rows is that it nicely supports parallelism: Each objective function f_i and its derivatives can be computed and added to $\mathbf{J}^\top \mathbf{J}$ in parallel, independently of the other objectives and derivatives.

8. Stopping Criteria

The Gauss-Newton algorithm outlined in Section 2.2.1 is an iterative scheme to gradually improve the solution \mathbf{x} . It is unlikely that the algorithm exactly reaches a minimum of F . Therefore the algorithm should be stopped, if either the objective $F(\mathbf{x})$ at the current solution, or the solution vector \mathbf{x} itself only change insignificantly within one iteration. There are various possible stopping criteria for Gauss-Newton type algorithms (see [MNT04]). In our experience, the following stopping criteria have proven to be a good choice: The algorithm has reached convergence, if one of the following criteria are met

$$\|\mathbf{J}(\mathbf{x}_i)^\top \mathbf{f}(\mathbf{x}_i)\|_\infty \leq \epsilon_\nabla (1 + F(\mathbf{x}_i)) \quad (2.45)$$

$$|\alpha d^*| \leq \epsilon_x (|\mathbf{x}_i| + \epsilon_x) \quad (2.46)$$

$$|F(\mathbf{x}_{i+1}) - F(\mathbf{x}_i)| \leq \epsilon_F F(\mathbf{x}_i) \quad (2.47)$$

where ϵ_∇ , ϵ_x , and ϵ_F are small tolerance thresholds. Criterion 2.45 is tested at the beginning of each iteration before solving the normal equations. If the maximum absolute value of the gradient $\nabla F = \mathbf{J}^\top \mathbf{f}$ is too small, the objective function is too shallow and the algorithm stops. Criterion 2.46 is tested during line search. If the step size α and thus the induced variable change gets too small, but condition b) in the line search scheme (see Item 5) is

2 *Fundamentals*

still not met, the algorithm stops. Criterion 2.47 is tested after the line search has evaluated a new variable vector \mathbf{x}_{i+1} . If no significant improvement on the objective function F could be achieved, the algorithm stops.

Constraint-Based Modeling

Positional, Metric, and Curvature Control for
Constraint-Based Surface Deformation

3.1 Introduction

Effective algorithms for surface deformation are of central importance in digital geometry processing. One of the most popular interaction metaphors allows the user to select subsets of the model as a control handle and specify an affine transformation for each handle region. The deformed surface is then computed such that the resulting positional constraints are satisfied, while preserving important properties of the original shape.

While intuitive and easy-to-learn, certain tasks remain difficult to achieve with handle-based interaction. For example, preserving or explicitly modifying first or second order properties of the surface, such as lengths, areas, or curvature, is cumbersome when deforming a shape by specifying positional constraints only.

3 Constraint-Based Modeling

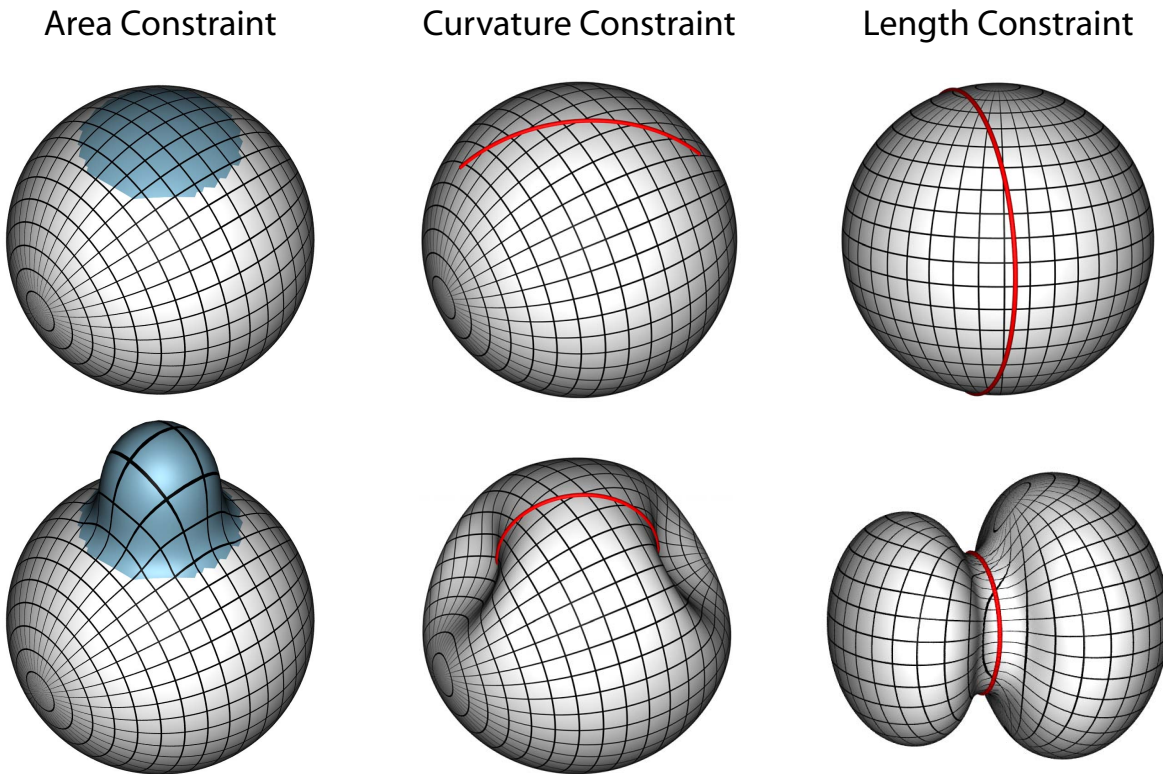


Figure 3.1 *Basic constraint-based surface edits supported by our system.*

To address this problem, we introduce a new framework for surface processing using advanced surface constraints beyond the direct prescription of positions. Our method allows the user to directly constrain positions, lengths, areas, and curvatures (Figure 3.1). Target values for these quantities can be specified anywhere on the model, both point-wise or integrated over embedded curves and surface patches. Our system solves for a deformation of the surface that best satisfies the user constraints, while preserving the original shape of the model as well as possible. We argue that the concept of *shape preservation* comes in different flavors that strongly depend on the user’s editing intent. To provide the necessary design flexibility, we enable explicit control over the shearing (conformal distortion), stretching (area distortion) and bending (curvature distortion) induced by the deformation mapping. We define the corresponding non-linear energies for each of these properties, and show how a consistent discretization can be obtained for surfaces represented by triangle meshes.

3.1.1 Related Work

There exists a vast amount of literature on techniques to edit and manipulate geometry and we refer to recent surveys such as [ASo6], [BPK⁺o8], or [BSo8] for an overview. Positional constraints are used successfully for shape manipulation with curves [WW92, SF98], multi-resolution editing [ZSS97, KCVS98, BKo4], and surface deformation based on differential coordinates [SCOL⁺o4, LSCO⁺o4, YZX⁺o4, LSLCOo5]. The latter achieve intuitive shape deformations by manipulating derived properties such as mesh gradients or Laplacian coordinates and reconstructing the deformed surface by solving a linear system. Sketch-based tools have become popular due to an intuitive shape design process centered around a sketching metaphor [IMT99]. The work by Nealen et al. [NISAo7] allows the user to freely draw and modify control curves on the model and reconstructs the surface defined by these curve constraints using an optimization approach.

Various papers extended the idea of positional constraints to also support a more direct specification of derived properties. For example, constraint based design tools (e.g. [MS92]) have successfully used differential normal or curvature constraints at fixed points in space to create a set of interpolating parameterized surface patches from scratch. Miura and colleagues [MCW01] allow the user to scale derivatives (i.e. local lengths) of analytic curves and surfaces. Tosun and co-workers propose a system for shape optimization using reflection lines that allows modifying surfaces by specifying a target reflection function gradient [TGRZo7]. This approach has been extended in [GZo8] to support stroke-based editing of the shaded image to drive the deformation of the surface.

Our work is also related to optimization methods in surface parameterization [HLSo7]. Ben-Chen and colleagues [BCGBo8] and Springborn et al. [SSPo8] have proposed algorithms for computing a conformal mapping based on metric scaling and curvature prescription. While conceptually similar in the sense that these methods 'deform' the surface into a plane while preserving conformality, the specific context of surface parameterization warrants a fundamentally different approach than our more general shape deformation setting.

3 Constraint-Based Modeling

Our general constraint-based surface processing framework enables the manipulation of whole distributions of surface properties (e.g. principal curvatures - see Section 3.6) leading to operations in the broader context of surface optimization. Previous work in this context are most commonly based on energy minimizing flows, where a given surface is progressively evolved to decrease an energy functional that quantifies the desired surface properties. Taubin [Tau95b] proposed an iterative Laplacian scheme to implement surface diffusion for low-pass filtering of discrete surfaces. Desbrun and colleagues [DMSB99] perform mean curvature flow to remove geometric noise on a surface and propose an implicit scheme to stabilize the computation. Ohtake and co-workers [OBS02] apply diffusion to the mesh normals and reconstruct the smoothed surface using a fitting approach. Bobenko and Schröder [BS05] introduced a version of discrete Willmore flow that preserves important symmetries of the continuous setting. A variety of different energy functionals, including Willmore and minimum variation of curvature energies, are studied by Pushkar and Sequin [PS07] in the context of fair surface design. Similarly, Pinkall and Polthier [PP93] construct discrete minimal surfaces based on an area minimizing flow.

Various researchers have extended this class of shape optimization methods by adding more direct control of the flow evolution. Hildebrandt and Polthier [HP04] present a method for feature-preserving noise removal on surface meshes based on an anisotropic mean curvature flow. Their method allows mean curvatures to be prescribed as targets for the flow evolution. Eckstein and co-workers [EPT⁺07] generalize geometric surface flows by tailoring the inner product of the underlying vector field to the requirements of specific applications. This extension provides a design tool for controlling the flow, which has been successfully applied for surface fairing and deformable shape matching.

3.2 Problem Formulation

We wish to broaden the toolset for surface deformation by enabling the user to specify advanced constraints on the surface beyond the direct prescription of positions. This involves three main challenges:

1. **Modeling Constraints:** What constraints provide effective, intuitive, and flexible tools for surface deformation?
2. **Deformation Control:** The constraints will not uniquely determine the deformation. There will be additional degrees of freedom. We wish to exploit this fact to give the user explicit control on the deformation outcome.
3. **Optimization Framework:** a practical framework for constraint-based surface deformation mandates an efficient implementation that is able to simultaneously optimize for various, possibly conflicting, constraints requested by the user.

We address challenge 3 by formulating constraint-based surface deformation as a nonlinear least squares optimization problem: the user input is converted into nonlinear least squares energies that are to be minimized. This section proposes a set of modeling constraints (challenge 1) and adaptable measures that provide control on the deformation outcome (challenge 2). For all these constraints and measures we first derive the corresponding optimization energies for continuous surfaces in general. Sections 3.3 and 3.4 discuss the discretization of the proposed energies and how to simultaneously minimize these energies to obtain the deformation result.

Notation. In the following, a prime denotes quantities of the unknown deformed surface \mathcal{S}' we wish to find, a hat denotes target values, and plain symbols denote quantities of the original input surface \mathcal{S} .

3.2.1 Modeling Constraints

We propose three basic types of advanced shape editing constraints, directly derived from the three most natural measures on a surface: length, area, and curvature. The user can draw curves and surface patches onto the surface and modify their global length and area, respectively, or change the normal curvature of the surface along the selected curves (see Figure 3.1). In addition, similar to existing handle-based deformation tools, our system offers position constraints that are computed from the displacement of a control handle.

The corresponding nonlinear least squares energies for these modeling constraints are all of the same simple form:

$$E_{constraint} = \left(value' - \widehat{value} \right)^2, \quad (3.1)$$

where *value* either corresponds to the length of a curve, the area of a surface patch, the normal curvature at a point along a curve, or the position in space of a specific surface point. Minimizing the energy $E_{constraint}$ means trying to match the user-specified target \widehat{value} with the *value'* of the deformed surface as well as possible within the limits of all the other, possibly conflicting, constraints.

3.2.2 Deformation Control

Imagine a simple constraint-based shape edit: decreasing the length of a curve around a sphere. Figure 3.2 indicates that there are (infinitely) many solutions achieving exactly the same target curve length. Which one should be chosen? Given no additional information, the best one can do is to preserve the properties from the original surface as well as possible. A standard approach in surface modeling is to preserve distances on the surface. Various surface modeling methods apply such a deformation strategy, often referred to as "as-rigid-as-possible deformation" (see [ASo6, BPK⁺o8, BSo8] for an overview of existing approaches). Although this deformation metaphor usually leads to intuitive deformations, mainly because it mimics the physically correct behavior of thin shells, we argue that it is not in all cases what the user

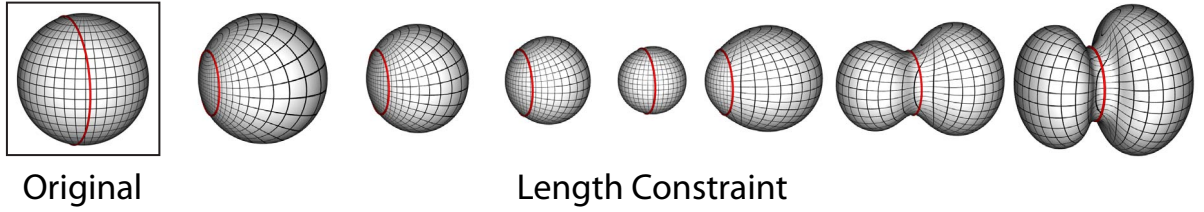


Figure 3.2 Reducing the curve length of the model on the left has infinitely many solutions.

wants. It restricts the search space spanned by the underdetermined problem of constraint-based surface deformation to a single solution. For example: of all the solutions to decreasing the length of a curve around a sphere as shown in Figure 3.2 only one would automatically be chosen by the system.

We improve the modeling flexibility by providing the user explicit control on which properties of the original surface should be preserved to what extent. It is a classical result from differential geometry that a surface is defined by its metric and curvature properties (see Section 2.1). From parameterization theory we know that metric distortion can be divided into areal and angular (conformal) distortion [Horo1]. In our framework we therefore implement deformation control with metric and curvature energies that can be weighted to specify the relative importance of area, angle, and curvature preservation.

Metric Energies. Let $J_{\mathbf{p}}$ be the Jacobian of the deformation function $\mathcal{S} \rightarrow \mathcal{S}'$ restricted to the tangent space at a point \mathbf{p} on the original surface \mathcal{S} . The metric distortion can be measured through the local anisotropic scaling that is encoded in the singular values σ_1, σ_2 of $J_{\mathbf{p}}$ (see Section 2.1). The product $\sigma_1\sigma_2$ quantifies the change in area and the ratio σ_1/σ_2 measures the angular (conformal) distortion. We use the symmetric energies

$$E_{areal_{\mathbf{p}}} = \sigma_1\sigma_2 + \frac{1}{\sigma_1\sigma_2} \quad (3.2)$$

$$E_{conf_{\mathbf{p}}} = \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1} \quad (3.3)$$

3 Constraint-Based Modeling

to measure the areal and conformal distortion, respectively. Integrating these local measures over the whole surface leads to the total metric energies:

$$E_{areal_S} = \frac{1}{A_S} \int_S E_{areal_{\mathbf{p}}} dA \quad (3.4)$$

$$E_{conf_S} = \frac{1}{A_S} \int_S E_{conf_{\mathbf{p}}} dA \quad (3.5)$$

Dividing by the total surface area A_S ensures independence of the result from global scaling. As we will illustrate below, a weighted sum of these two energies provides flexible control on the deformation semantics. If both E_{areal} and E_{conf} are weighted equally, their sum corresponds to an isometric distortion measure.

Curvature Energy. We denote with κ_1 the signed maximum and with κ_2 the signed minimum curvature at a point $\mathbf{p} \in \mathcal{S}$ (see Section 2.1.5)). Quantities of the unknown surface \mathcal{S}' are defined analogously and denoted by a prime. We define an energy that quantifies curvature preservation by measuring the deviation of principal curvatures κ'_1, κ'_2 of the deformed surface \mathcal{S}' from the curvatures on the original surface as

$$E_{pc_S} = \int_S (\kappa'_1 - \kappa_1)^2 + (\kappa'_2 - \kappa_2)^2 dA. \quad (3.6)$$

Combined Surface Energy. The total surface energy is computed as a weighted combination of the metric and curvature energies

$$E_{surf_S} = k_{areal} E_{areal_S} + k_{conf} E_{conf_S} + k_{pc} E_{pc_S} \quad (3.7)$$

with scalar weights k_{areal} , k_{conf} , and k_{pc} . The effect of different choices for these weights is illustrated in Figure 3.3. A dominant conformal weight leads to a uniform scaling of the sphere. A dominant areal term yields a constricting deformation with fairly strong distortion of angles and curvature, while a dominant curvature term leads to the curve 'sliding' along the sphere. By varying the three weights the user is able to achieve any of the possible deformations shown in Figure 3.2 and all the stages in between. This simple example illustrates how our approach supports different notions of *shape preservation*. The user can

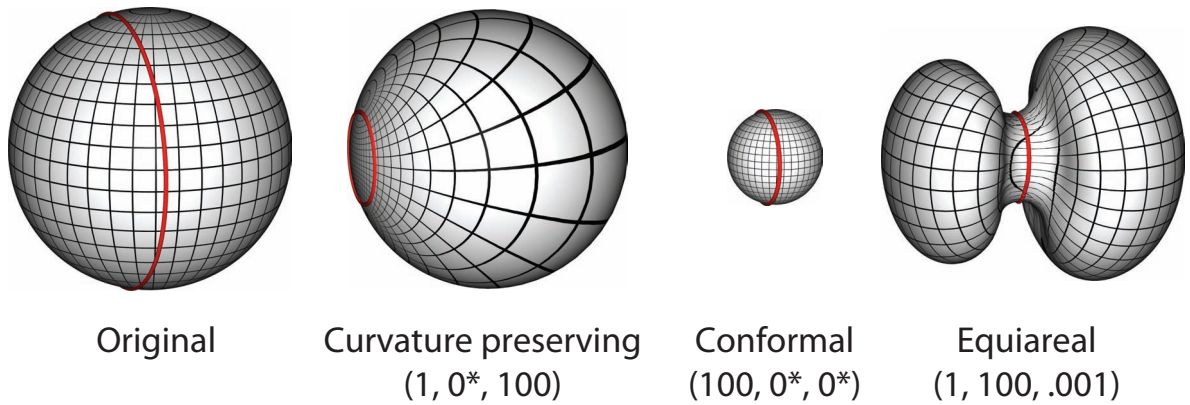


Figure 3.3 *Effect of different weights for the local surface energies. As a modeling constraint, the target length of the red curve is set to one third of its original value. The numbers in brackets denote the weighting terms $(k_{conf}, k_{areal}, k_{pc})$, where 0^* denotes a very small contribution (we use $0^* = 10^{-5}$) of the corresponding energy added for regularization.*

adapt the behavior of the optimization to preserve properties of the original shape that are important in the specific application context, enabling flexible surface processing and shape design.

Surface Energies as Modeling Constraints. Besides shape preservation, these surface energies also provide direct control for shape deformation. For example, we can locally scale the metric to grow or shrink the shape, or control surface bending by prescribing principal curvatures. The latter can lead to particularly interesting surface processing operations when applying filters to modify the whole distribution of principal curvatures on the surface. This special application leads to the concept of Curvature-Domain Shape Processing and is presented in Section 3.6.

3.3 Discretization

To enable an efficient implementation, we derive the discrete energies from their continuous counterparts presented in Section 3.2. The surface \mathcal{S} is discretized with a triangle mesh $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V} = \{v_i\}$ denotes the set of vertices, $\mathcal{E} = \{e_{ij}\}$ the edge set, and $\mathcal{F} = \{f_{ijk}\}$ the face set with $1 \leq i, j, k \leq |\mathcal{V}|$. The position of ver-

3 Constraint-Based Modeling

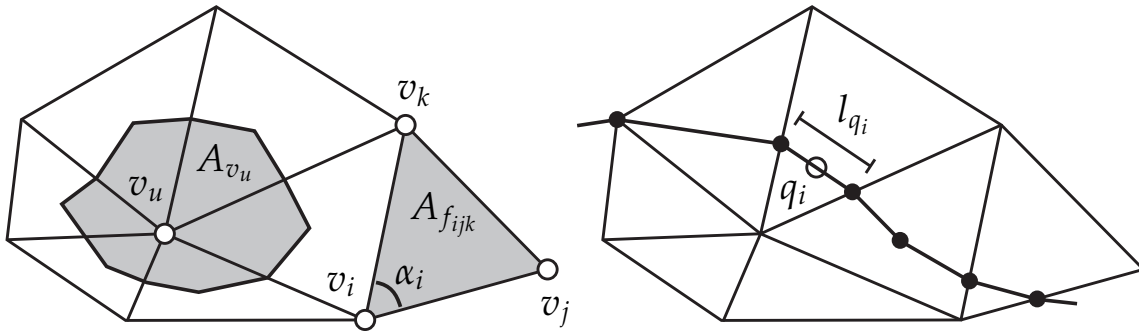


Figure 3.4 Vertex and face areas, inner triangle angle (left). Length and center point of a discrete path segment (right).

text v_i is given by $\mathbf{v}_i \in \mathbb{R}^3$. As before, a prime denotes quantities of the deformed mesh, a hat specifies target values, all other values are computed on the input surface. While our method uses triangle meshes as a discrete representation of smooth surfaces, the basic concepts are independent of this choice of discretization.

3.3.1 Surface Energies

Metric Energies. In the discrete setting both energies E_{areal_p} (3.2) and E_{conf_p} (3.3) are constant for all points on each triangle f and can be written as [Horo1]

$$E_{areal_f} = \frac{A_f}{A_{f'}} + \frac{A_{f'}}{A_f} \quad (3.8)$$

$$E_{conf_f} = \frac{\cot(\alpha_i) \|\mathbf{e}'_{jk}\|^2 + \cot(\alpha_j) \|\mathbf{e}'_{ik}\|^2 + \cot(\alpha_k) \|\mathbf{e}'_{ij}\|^2}{2A_{f'}}, \quad (3.9)$$

where A_f denotes the area of triangle $f = f_{ijk}$, α_i , α_j , and α_k are the inner angles of f at the corresponding vertices, and $\mathbf{e}_{ij} = \mathbf{v}_j - \mathbf{v}_i$ is the edge vector between the corresponding vertices (Figure 3.4). Summing the per-triangle energies over the entire mesh \mathcal{M} with surface area $A_{\mathcal{M}}$ leads to the total areal energy

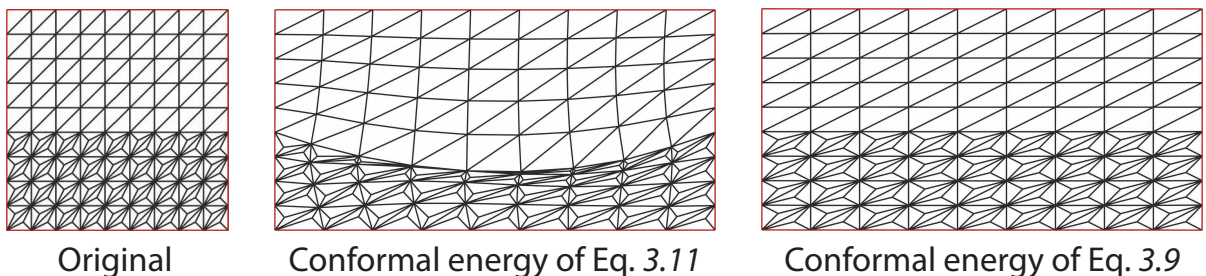
$$E_{areal_{\mathcal{M}}} = \frac{1}{A_{\mathcal{M}}} \sum_{f \in \mathcal{F}} A_f E_{areal_{f'}}, \quad (3.10)$$

and, analogously, the total conformal energy $E_{conf_{\mathcal{M}}}$. Energies similar to E_{areal} have been used for surface parameterization and physically based shell models [DMK03, GHDS03]. The conformal energy E_{conf} corresponds to the MIPS energy introduced by Hormann in the context of mesh parameterization [Horo1, HGo0]. To the best of our knowledge, this energy has never been used for surface deformation, partly because the corresponding optimization is rather involved. We resolve this issue by introducing a novel formulation of the energy in Section 3.4.

An alternative, slightly more efficient formulation of a conformal energy is based on inner triangle angles α_i (Figure 3.4):

$$E_{conf_f} = (\alpha_i - \alpha'_i)^2 + (\alpha_j - \alpha'_j)^2 + (\alpha_k - \alpha'_k)^2. \quad (3.11)$$

A distinct advantage of E_{conf} as defined in Equation 3.9 is illustrated in the figure below. Compared to the conformal energy of Equation 3.11 the one of Equation 3.9 is not sensitive to the specific discretization of the mesh. When stretching the non-uniformly tessellated patch along the horizontal axis, the local distortion is independent of the triangulation, which makes the method robust under re-sampling of the surface.



Curvature Energy. We discretize the integral of Equation 3.6 as a sum of discrete curvatures:

$$E_{pc_{\mathcal{M}}} = \sum_{v_i \in \mathcal{V}} A_{v_i} [(\kappa'_{1,i} - \kappa_{1,i})^2 + (\kappa'_{2,i} - \kappa_{2,i})^2], \quad (3.12)$$

where $\kappa_{1,i}$ and $\kappa_{2,i}$ denote the signed maximal and minimal principal curvatures at vertex v_i .

3 Constraint-Based Modeling

A variety of techniques have been proposed to estimate curvatures on piece-wise linear surfaces (cf., [MDSBo2, CSMo3, CPo5, GGRZo6, KSNSo7]). In our implementation, we use the curvature tensor proposed by Cohen-Steiner and Morvan [CSMo3] as it provides a robust and theoretically well-founded estimation of the principal curvatures at multiple scales, which we exploit in our framework for Curvature-Domain Shape Processing (Section 3.6) to implement multi-scale processing operations. However, our method is not dependent on this specific curvature discretization and other techniques could be used instead.

We adopt the notation used by Alliez and colleagues [ACSD⁺o3] and summarize the curvature computation here. The curvature tensor $\mathcal{K}(v)$ for vertex v is found by averaging an edge-based tensor for all edges e that fall within a region B surrounding the vertex. The curvature tensor is represented by the 3×3 matrix

$$\mathcal{K}(v) = \frac{1}{|B|} \sum_{e \in B} \beta(e) |e \cap B| \bar{\mathbf{e}} \bar{\mathbf{e}}^\top, \quad (3.13)$$

where $|B|$ is the surface area of the region B , $\beta(e)$ represents the signed angle between the normals of the triangles incident to edge e and is positive for a convex crease and negative for a concave one, $|e \cap B|$ is the length of the portion of the edge within the region B , and $\bar{\mathbf{e}}$ is a unit-length vector aligned with edge vector \mathbf{e} . The principal curvatures are found by computing the eigenvalues of the curvature tensor $\mathcal{K}(v)$. The eigenvalue closest to zero is discarded, and the remaining two form the signed maximum and minimum curvatures: the larger signed eigenvalue is κ_1 and the smaller κ_2 . The size of B determines the scale of the curvature estimation. As the smallest scale we use the barycentric area A_{v_i} of a vertex v_i [MDSBo2] (Figure 3.4). We use barycentric areas rather than generalized Voronoi regions to simplify derivative computations. For larger scales, we compute the union of the barycentric areas of all vertices within a certain distance to v_i . For all modeling operations where the curvature energy represents a shape preservation energy (i.e. all operations except Curvature-Domain Shape Processing discussed in Section 3.6) we use the smallest scale for B .

Combined Surface Energy. As in the continuous case, the total surface energy on a mesh is computed as a weighted combination of the metric and curvature energies

$$E_{surf_{\mathcal{M}}} = k_{areal}E_{areal_{\mathcal{M}}} + k_{conf}E_{conf_{\mathcal{M}}} + k_{pc}E_{pc_{\mathcal{M}}} \quad (3.14)$$

with scalar weights k_{areal} , k_{conf} , and k_{pc} . The effect of different choices for these weights is discussed in Section 3.2.2.

3.3.2 Modeling Constraints

The surface energies defined above aim at preserving important properties of the input surface, i.e., control how the shape resists to change dictated by the user's modeling constraints. However, as indicated in Section 3.2.2, the energies can also be used to drive the deformation by replacing the values of the original shape by arbitrary target values. One of the unique features of our method is the ability to explicitly control the surface metric, i.e., set constraints for lengths and areas on the surface. This can be done locally by specifying target values for each triangle in Equations 3.8 and 3.9, or as global least-squares constraints over curves and surface patches, as described below. All energies below follow the general form stated in Section 3.2.1 (as always, user-specified target values are indicated by a hat symbol).

Area Constraints. A global area constraint for a patch $\mathcal{P} \subset \mathcal{S}$, represented by a set of faces $\mathcal{F}_{\mathcal{P}} \subseteq \mathcal{F}$, is defined as

$$E_{area_{\mathcal{P}}} = \frac{1}{A_{\mathcal{P}}^2} (A_{\mathcal{P}'} - \hat{A}_{\mathcal{P}})^2, \quad (3.15)$$

i.e., the total area $A_{\mathcal{P}'}$ of the patch \mathcal{P} on the deformed surface should be equal to the specified target value $\hat{A}_{\mathcal{P}}$ provided by the user. In contrast to local area constraints, scaling the global area offers more degrees of freedom to satisfy conflicting user constraints, as illustrated in Figure 3.5.

Length Constraints. Another integrated measure is the length of a curve (Figure 3.3), discretized as a piecewise linear path $\mathcal{D} \subset \mathcal{M}$ (Figure 3.4). The corresponding energy is defined as

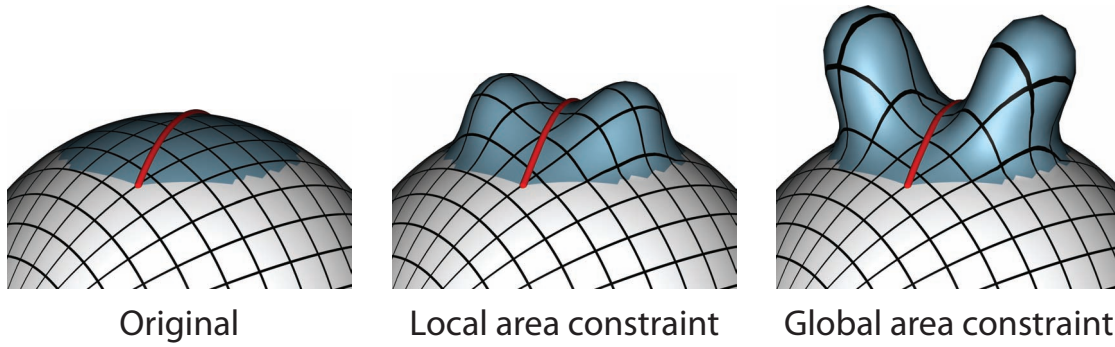


Figure 3.5 Comparison of local vs. global area constraints. The area of the blue patch is scaled by a factor of 2.5 while constraining the length of the red curve to remain constant. Specifying the area change locally for each triangle does not offer enough degrees of freedom to satisfy both constraints. The global area constraint for the entire patch offers more flexibility to satisfy the constraints.

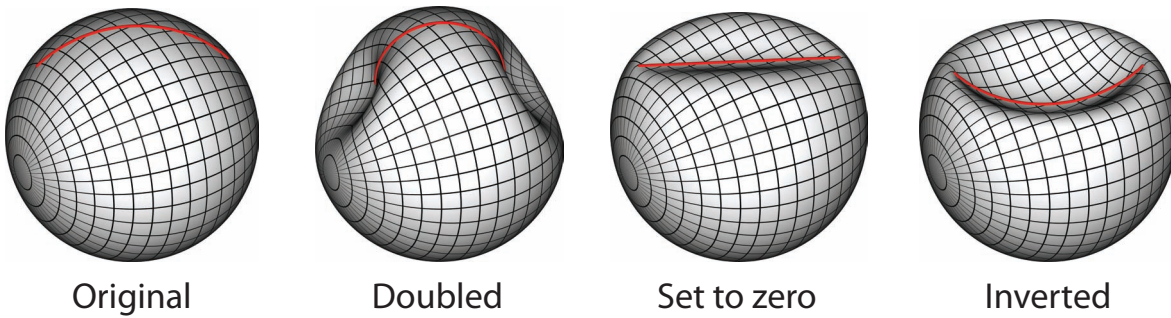


Figure 3.6 Deformation of a sphere by prescribing normal curvature of the surface along the marked curve.

$$E_{length_{\mathcal{D}}} = \frac{1}{l_{\mathcal{D}}^2} \left(l_{\mathcal{D}'} - \hat{l}_{\mathcal{D}} \right)^2, \quad (3.16)$$

where $l_{\mathcal{D}}$ denotes the total length of the polyline.

Normal Curvature Constraints. Using the curvature energy of Equation 3.12, the user can directly modify principal curvatures (see Section 3.6). Access to principal curvatures, however, does not provide *directional control* of surface bending. We therefore additionally allow prescribing the normal curvature of the surface (see Section 2.1) along arbitrary tangent directions. In order to do so, the user can freely draw

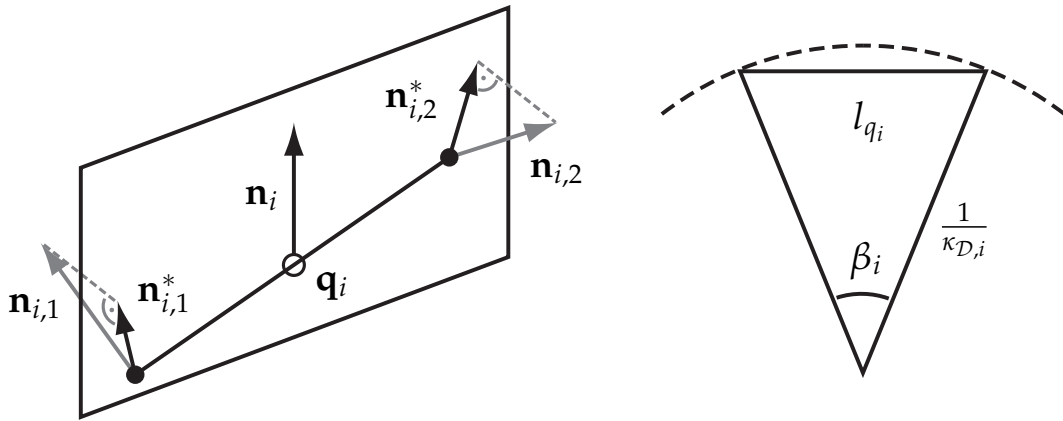


Figure 3.7 *Left: Discrete path segment with center point \mathbf{q}_i , interpolated mesh normals \mathbf{n} and projections \mathbf{n}^* onto the osculating plane. Grey elements do not lie in the plane. Right: Approximation of the normal curvature.*

curves on the surface and specify the desired target values for normal curvature (Figure 3.6).

We measure signed normal curvature as the change of surface normal along the curve. Continuous surface normal vectors are defined using barycentric interpolation of vertex normals. At a given point \mathbf{q}_i on the curve we compute the osculating plane spanned by the surface normal vector \mathbf{n}_i at \mathbf{q}_i and the curve's tangent vector. For a discrete approximation of normal curvature $\kappa_{\mathcal{D}}$ we consider a small curve segment i of length l_{q_i} around \mathbf{q}_i (Figure 3.4) and project the interpolated normal vectors at the ends of the curve segment onto the osculating plane (Figure 3.7), leading to

$$\kappa_{\mathcal{D},i} = \frac{2 \sin(\beta_i/2)}{l_{q_i}}, \quad (3.17)$$

where β_i is the angle between the projected surface normals $\mathbf{n}_{i,1}^*$, $\mathbf{n}_{i,2}^*$. The normal curvature energy is then defined as

$$E_{nc_{\mathcal{D}}} = \sqrt{A_{\mathcal{M}}} \sum_i l_{q_i} (\kappa_{\mathcal{D}',i} - \hat{\kappa}_{\mathcal{D},i})^2, \quad (3.18)$$

Position Constraints. Our system also offers position constraints based on the standard handle paradigm. Since the vertex positions are the unknowns of the optimization, fixing vertices to a specified position in space is trivial. The affected vertices are displaced to the target

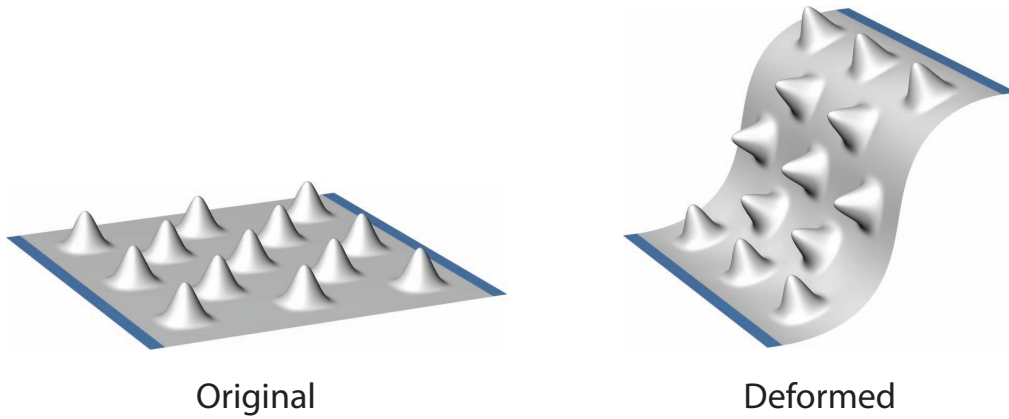


Figure 3.8 Our system incorporates the handle-based deformation metaphor using position constraints (marked in blue). Local detail is accurately preserved due to the non-linear surface energies.

location and the corresponding variables are removed from the optimization. To provide more flexibility when accommodating conflicting constraints we also include soft constraints, formulated using the least squares energy

$$E_{pos_{\mathcal{M}}} = \frac{1}{A_{\mathcal{M}}^2} \sum_{v_i \in \mathcal{V}} A_{v_i} \|\mathbf{v}'_i - \hat{\mathbf{v}}_i\|^2, \quad (3.19)$$

where A_{v_i} is the barycentric area around vertex v_i (Figure 3.4). Since all of the above energies except E_{pos} are invariant to rigid transformations, for all the examples shown we either fixed a number of vertices to their original position or added soft position constraints to all vertices using a small weight $k_{pos} = 0.001$. Figure 3.8 shows a test case for position constraints that illustrates how our non-linear surface energies lead to intuitive detail preservation, i.e., accurate rotation of surface detail even for purely translation-based handle displacements (cf. Figure 2 of [BPGKo6]).

3.3.3 Discussion

An important feature of our approach is the systematic definition of all shape preservation and constraint energies based on the following three requirements: (i) *Convergence*: the discrete energies should converge to the continuous ones for a suitable refinement of the mesh,

(ii) *Local symmetry*: energies should be locally invariant when interchanging deformed and original surface, (iii) *Scale invariance*: energies should be invariant under global re-scaling. The last property is achieved using global normalization terms, while local symmetry follows immediately from the definition of the energy terms. Cohen-Steiner and Morvan have proved convergence for their estimation of principal curvatures [CSM03] that we apply to evaluate Equation 3.12. The area of a set of triangles naturally converges to the area of the corresponding patch if a smooth parameterized surface is sampled regularly with increasing density to generate a mesh. The same argument holds for the derived metric energies since they are constant for a triangle face and converge to the functions evaluated at a point of the smooth surface, if the three vertices converge to that point. Similarly, convergence for the measures of normal curvature and length of a curve is assured if the sampling density of the curve increases.

3.4 Optimization

Editing constraints and shape preservation control are encoded in the energies introduced in the previous section. We find the corresponding deformed surface by solving for the vertex positions $\mathbf{v}_1^* \dots \mathbf{v}_n^*$ such that

$$\mathbf{v}_1^* \dots \mathbf{v}_n^* = \operatorname{argmin}_{\mathbf{v}'_1 \dots \mathbf{v}'_n} \sum_i k_i E_i, \quad (3.20)$$

where E_i and k_i is short-hand notation for the different energies and weighting constants defined in Section 3.3. We solve this nonlinear least squares problem using a Gauss-Newton solver which enables superlinear convergence without the need for second derivatives. See Section 2.2 for an extensive discussion on nonlinear least squares problems. The initial values of the variables are the vertex positions of the input surface. In all our experiments the results converge within about fifteen iteration steps.

Since the energies each only depend on a subset of the vertices the linear system to be solved in a Gauss-Newton iteration is sparse (see Section 2.2.3 for details). Figure 3.9 shows the sparsity structure of

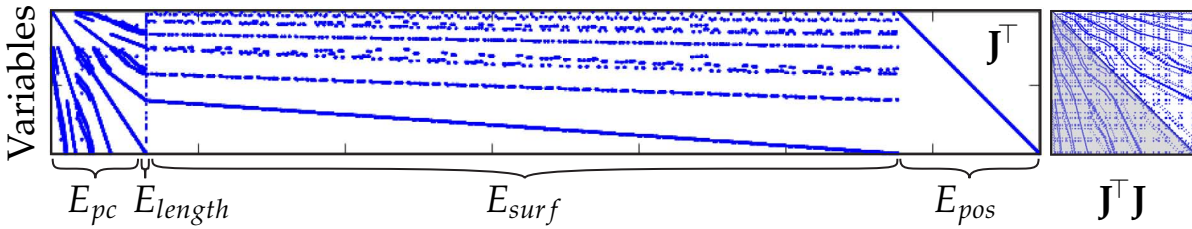


Figure 3.9 Left: Sparsity structure of the Jacobian matrix J^T (transposed for convenience) showing which rows of J correspond to which energy terms for the optimization problem shown in Figure 3.3. Right: Sparsity structure of the corresponding normal equations system $J^T J$.

the Jacobian matrix J and the normal equations matrix $J^T J$ for a specific problem instance, namely the manipulation of curve length on the sphere shown in Figure 3.3. If the curve covers a big percentage of the surface, the path length energy depends on many of the vertices and the density of the matrix $J^T J$ increases. One completely dense (all values non-zero) row in the Jacobian matrix is enough to render $J^T J$ completely dense. Since a path usually only covers a small part of the surface, the matrices are still sufficiently sparse. For large area constraints, however, it can happen that the matrices become rather dense and thus require a lot of memory (see Section 3.7 for a discussion).

The derivation of analytic expressions for the gradients of the energies is discussed in Appendix A.1. If curvatures are manipulated (Section 3.6), the optimization falls into the category of composite non-smooth optimization [WF86] due to discontinuities in the derivatives of the principal curvatures at umbilic points. As detailed in Appendix A.1, we are able to derive suitable approximations for the derivatives at these points so that efficient methods designed for smooth functions can be applied.

Usually, there exists a conflict between the modified properties (for example the length of a path) and the properties that should preserve their values (for example metric properties). For some applications the user might want to achieve the modified properties as exactly as possible. To support this, the optimization procedure can be adapted to allow the preservation energies to evolve in an iterative fashion: the optimization problem (3.20) is solved repetitively until convergence, each time reinitializing the property preservation energies while keeping

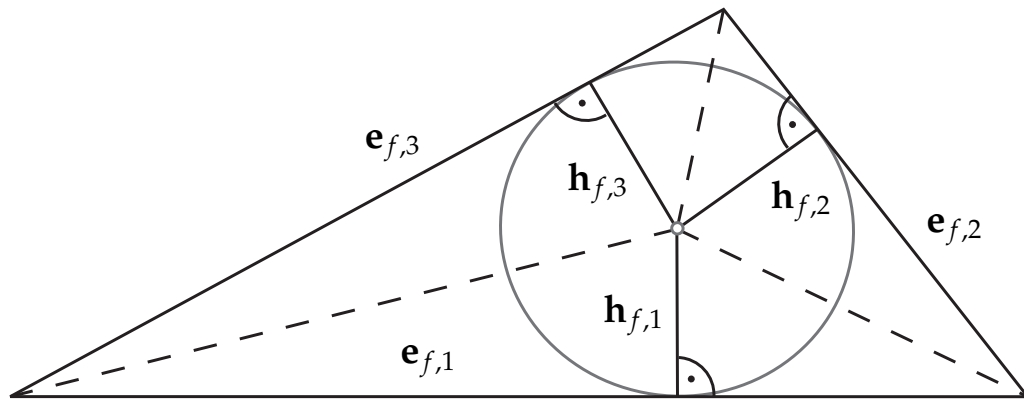


Figure 3.10 *Virtual subdivision of a triangle f avoids negative cotangent weights, since all angles are $\leq \pi/2$.*

the original energies for the manipulated properties. We only make use of this possibility for Curvature-Domain Shape Processing (Section 3.6).

Conformal Energy. In our experiments we found that the convergence of the optimization is largely dominated by the conformal energy E_{conf} . In fact, a direct implementation of Equation 3.9 leads to prohibitively slow convergence. Since the rather involved formulas for the curvature energies make the use of an advanced solver that requires second derivatives undesirable, we make use of the following observation to obtain a practical implementation: As written in Equation 3.9, E_{conf} is the sum of three rational quadratic terms. Since the Gauss-Newton solver locally approximates the objective function with a quadratic and the convergence critically depends on the curvature of the terms in the objective vector (see Section 2.2), splitting the energy into three separate terms significantly improves the convergence. However, this separation is only admissible, if the single terms (i.e. the cotangent weights) are all positive as required by the Gauss-Newton Solver. To address this issue, we propose a re-formulation for the conformal energy based on a virtual subdivision of a triangle using its incircle center to generate six smaller triangles as depicted in Figure 3.10. By construction, negative cotangent weights cannot occur, hence we can perform the above split into separate terms. By exploiting symmetry, right angles, and invariance of the conformal energy

3 Constraint-Based Modeling

under triangle subdivision, the resulting formula simplifies to a sum of six *positive* rational quadratics for each triangle:

$$E_{conf_f} = \sum_{u=1}^3 \left(\frac{A_f}{3\|\mathbf{e}_{f,u}\|^2} \frac{\|\mathbf{e}'_{f,u}\|^2}{A_{f'}} + \frac{\phi_f^2}{12A_f} \frac{\|\mathbf{h}'_{f,u}\|^2}{A_{f'}} \right), \quad (3.21)$$

where ϕ_f denotes the circumference of the triangle, $\mathbf{h}_{f,u}$ are the vectors connecting points on the edges and the center of the incircle on the triangle of the original mesh (Figure 3.10). On the deformed mesh, these points are expressed with the barycentric coordinates of the points on the original mesh. The barycentric coordinates are computed as a preprocessing step at the beginning of the optimization. This reformulation is essential to achieve a practical speed of convergence with a Gauss-Newton type solver, as we demonstrate with a simple example: Scaling the normal curvature of the sphere along the curve depicted in Figure 3.6 takes 9 iteration steps using the virtual subdivision compared to 108 iterations without it. Since the terms of the conformal energy are better approximated by quadratic functions when using the subdivision, the stepsize has only to be scaled by an average factor of 0.43 compared to 0.00027 in the unsubdivided formulation. This reduces the cost of stepsize control in our implementation. For this simple test case, the overall speedup factor achieved by virtual subdivision was 47. For more complex operations, these performance differences become even more extreme which renders the unsubdivided energy impractical for use with a Gauss-Newton type optimization. Note that the straightforward way of subdivision by dividing the obtuse angle with a line orthogonal to the opposite edge is not suitable since the virtual triangles can become arbitrarily thin leading to numerical instabilities.

3.5 Constraint-Based Shape Editing

Figures 3.3 to 3.8 illustrate specific properties of our framework for simple geometric shapes. In the following we show various constrained deformations performed on more complex models focusing

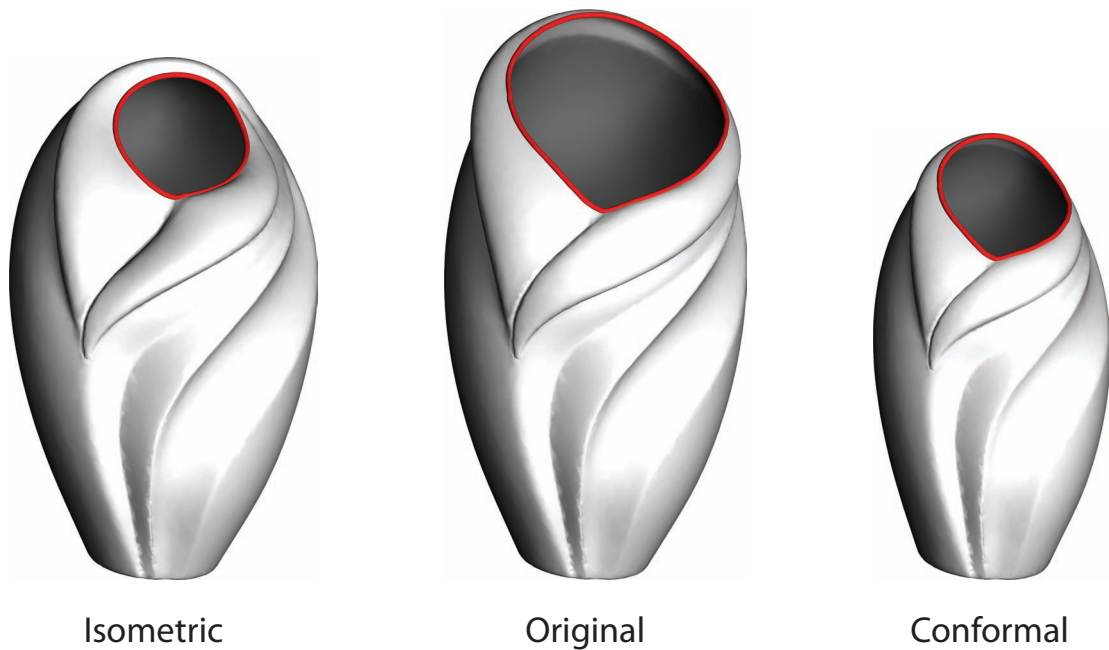


Figure 3.11 *The vase has been edited by reducing the length of the marked curve by a factor of two while keeping the bottom fixed. Different notions of shape preservation lead to different editing semantics.*

on editing operations that are difficult to achieve with traditional, purely position-based approaches.

Figure 3.11 shows an editing operation using length constraints. The user draws a curve along the rim of the vase and interactively modifies the desired target length while the bottom of the vase is kept fixed. Two settings for the surface energies, one aiming at isometric, the other at conformal deformation, illustrate how different notions of shape preservation lead to different editing semantics. A benefit of our approach is that the user has explicit control of the shape-preserving energies and can thus easily adapt the deformation to a specific application context.

Figure 3.12 shows the creation of a cartoon horse. The belly and flanks have been enhanced by scaling the area of the blue patches. Length constraints are used to constrict the waist and enlarge the head. The normal curvature for each point of the curve along the spine has been set to the curve average, leading to a deformation that pushes the spine towards a circular arc.

3 Constraint-Based Modeling

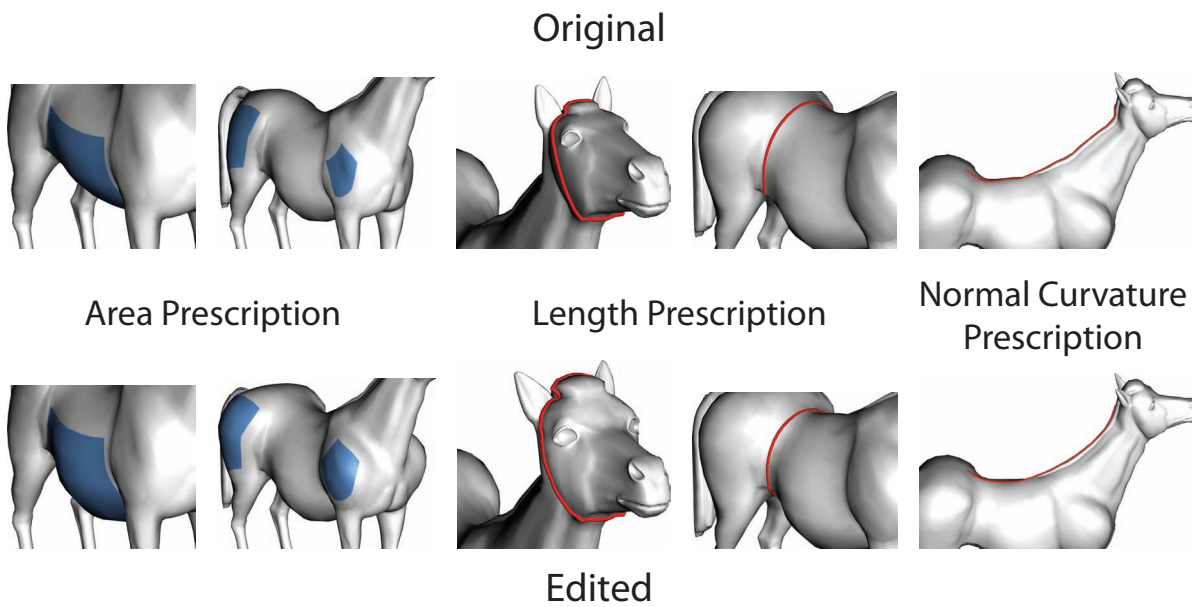
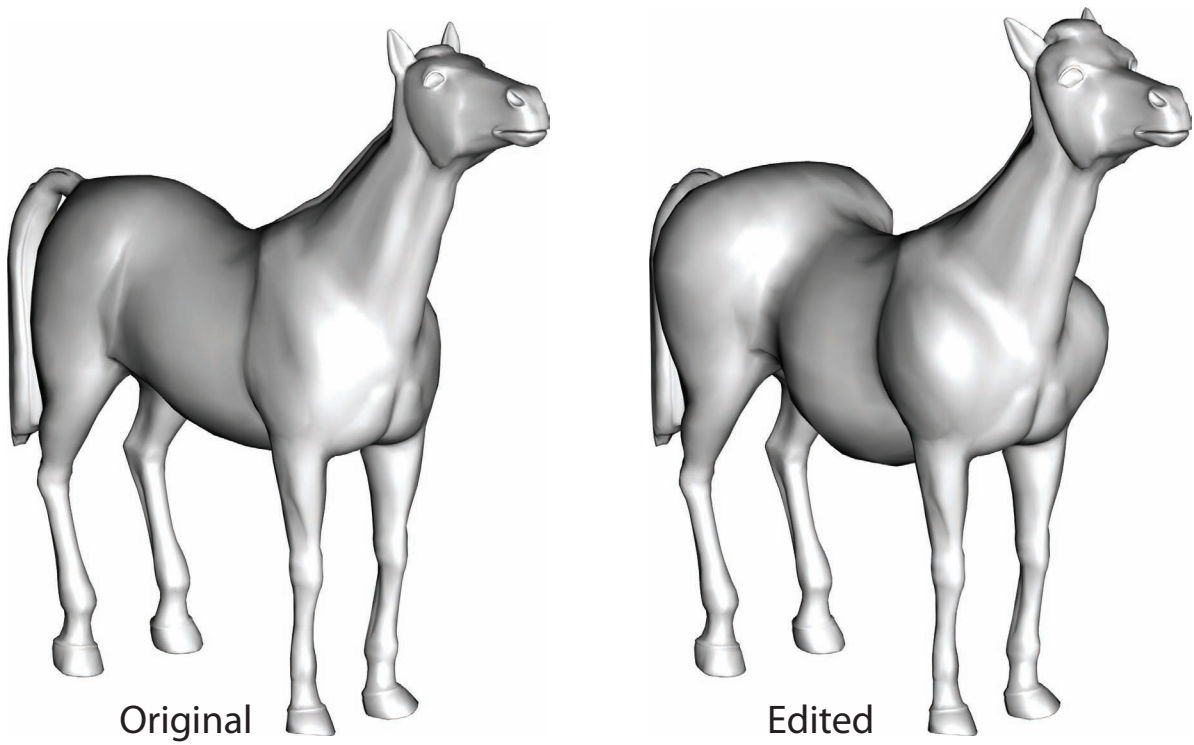


Figure 3.12 *The shape of the horse model has been edited by direct manipulation of length, normal curvature, and area of different curves and surface patches.*

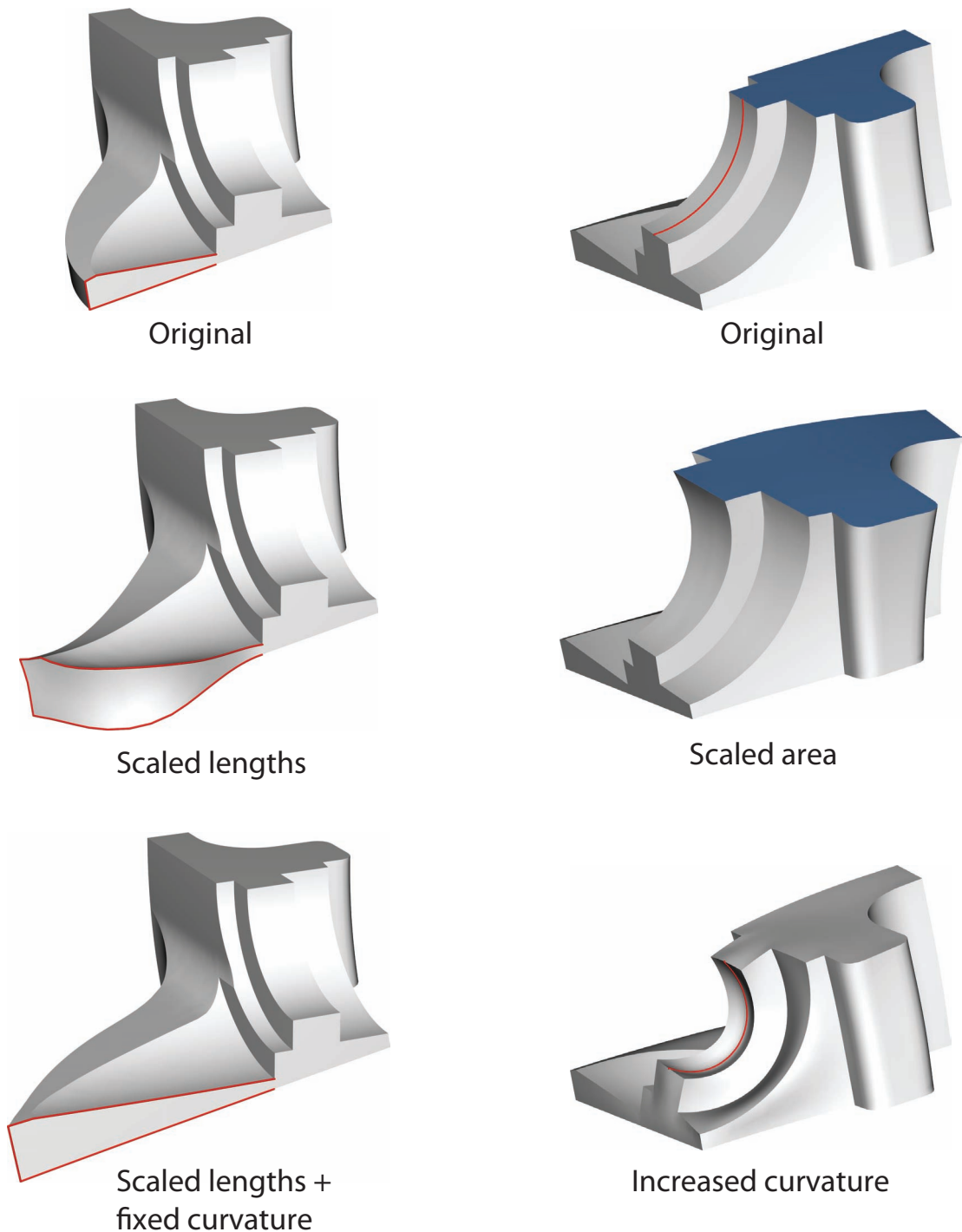


Figure 3.13 *Processing the fandisk model. Length, normal curvature, and area of the marked curves and patch have been scaled by a factor of two in the corresponding images. For the bottom left image, the curvature of the curves has additionally been constrained to remain zero for each of the marked segments.*

Figure 3.13 shows various editing operations on the fandisk model. Direct control of metric and curvature is particularly useful in a CAD context, where the semantics of the model are often directly linked to lengths, areas, and curvature. In the special case of fixing the curvature along a curve to zero, it might be more desirable to constrain the curvature of the curve itself instead of the normal curvature of the surface, which significantly simplifies computation. This technique was used to generate the bottom left result of Figure 3.13.

Image Deformation. Our general constraint-based modeling framework is not restricted to surface deformation and can also be used to edit images: The image domain is triangulated by a 2-dimensional triangle mesh and the energies introduced above can be directly applied to specify advanced constraints for image deformation. Figures 3.14 and 3.15 demonstrate how curvature constraints for curves drawn on the image can be used to intuitively correct distorted images similar to [CAA09] without any prior information on how the image was distorted. The prescription of area (Figure 3.16) and length (Figure 3.17) provides novel tools for image manipulation. As for surface deformation, the metric shape preservation energies can be used to control how local angles and areas are preserved (obviously, the principal curvature energy is not needed in this 2D setting). Figure 3.17 investigates the potential of adaptive weights for shape preservation, where the user can specify important regions that should be preserved as well as possible. This is implemented by using separate optimization weights k_{areal} and k_{conf} for every mesh vertex instead of two global values. Figure 3.18 indicates that not only the principal curvature energy (see Section 3.6) but also the metric preservation energies can serve as constraints by prescribing angles and local areas for every triangle, which offers potential for future work.

3.5 Constraint-Based Shape Editing

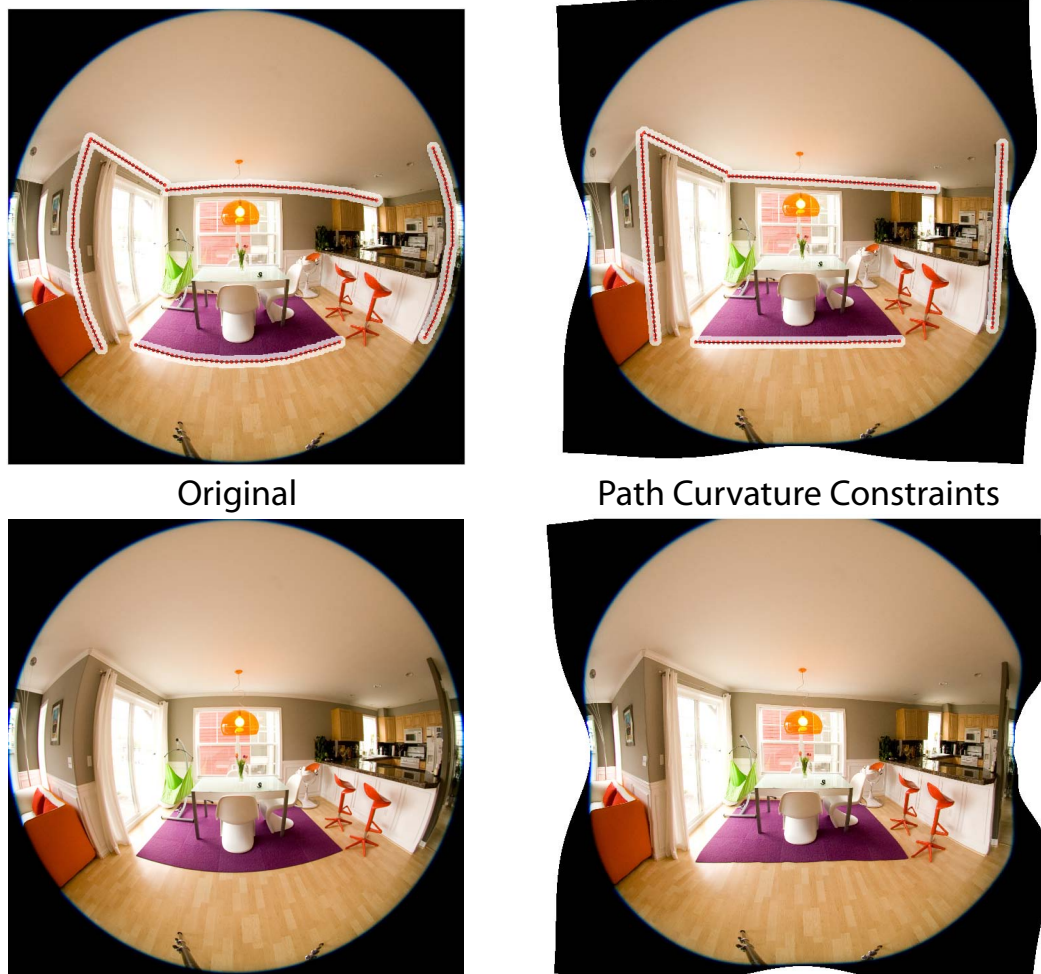


Figure 3.14 Correct "fisheye" lens distortion by prescribing zero path curvature. The user specified paths are shown in the top row. (Original image taken from [CAA09] and kindly provided by Aseem Agarwala.)



Figure 3.15 Correct "fisheye" lens distortion by combining path curvature and position constraints. (Original image taken from [CAA09] and kindly provided by Aseem Agarwala.)

3 Constraint-Based Modeling



Figure 3.16 Area constraints for interactive image deformation.

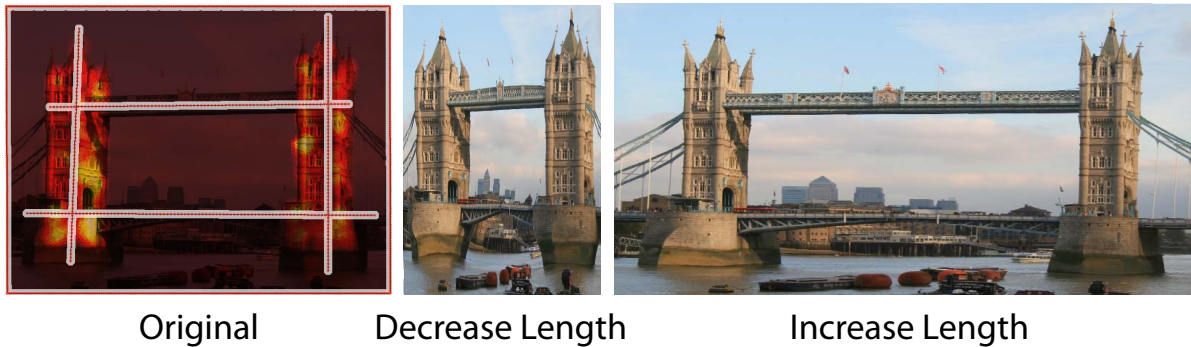


Figure 3.17 Length constraints for interactive image deformation. The user can draw adaptive weights (shown by the dark-red to yellow color scheme on the left) to specify salient regions (light red and yellow) that should be distorted less by the deformation.

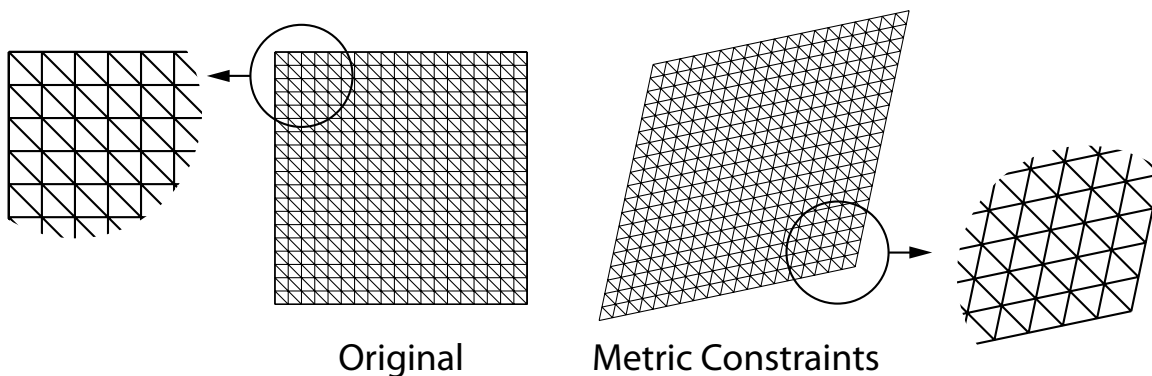


Figure 3.18 Prescription of metric properties on a 2-dimensional mesh. For each triangle the angles are constrained to be 60° , while the area should be preserved, leading to a more uniform triangle mesh.

3.6 Curvature-Domain Shape Processing

Curvature is an essential concept in geometry and plays a crucial role in surface optimization, geometric modeling, and shape classification. Many geometry processing operations strive to optimize the curvature distribution of a surface based on energy functionals that measure surface fairness [BPK⁺08]. Shape classification, feature extraction, and segmentation algorithms [Shao6, AKM⁺06] heavily rely on curvature information to identify meaningful geometric structures, such as ridges, valleys, and corners, that are mapped to features or used as segmentation boundaries. Similarly, non-photorealistic rendering approaches often make use of surface curvature to determine the position and style of rendered strokes [MHIL02]. Typically, these methods use curvature either as a tool for analysis, or indirectly in the optimization of fairness energies that are defined as curvature integrals over the entire surface.

Our constraint-based surface deformation framework provides direct access to curvature as a geometry processing tool. Instead of editing a shape through modeling constraints defined on curves and surface patches, as demonstrated in Section 3.5, we now investigate the potential of editing and filtering the distribution of principal curvatures over the whole surface in order to alter the shape of an object. The conceptual work-flow of this Curvature-Domain Shape Processing is illustrated in Figure 3.19. Our system first computes a mapping from the spatial domain to the curvature domain by evaluating principal curvatures for each point on the surface of a given object. In this domain, important geometric features and properties are more directly accessible and can be manipulated by setting specific curvatures to desired target values or by applying filtering operations on the curvature distribution. The mapping to the curvature domain is then inverted to reconstruct the modified object geometry using our optimization framework that computes a deformation of the input surface that best approximates the prescribed curvature constraints in a least-squares sense. As shown in Figure 3.20 and on further results shown below, this approach facilitates a variety of geometry processing operations

3 Constraint-Based Modeling

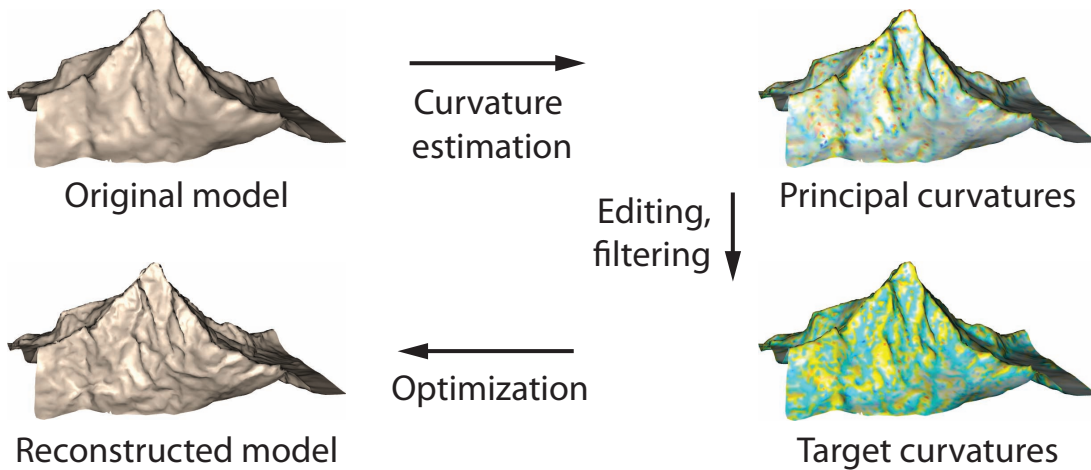


Figure 3.19 The central idea of *Curvature-Domain Shape Processing* is to process the geometry of a model by altering its curvature distribution.

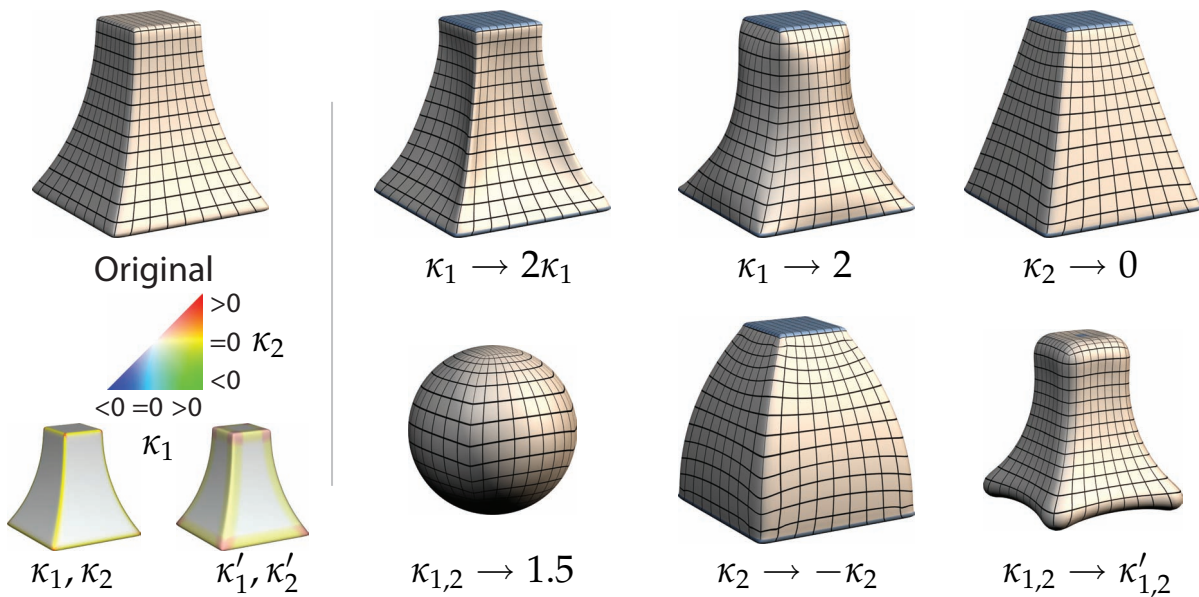


Figure 3.20 Geometry processing in the curvature domain. The models on the right (blue regions indicate constrained vertices) have been reconstructed by modifying the signed maximum and minimum curvatures, κ_1 and κ_2 , respectively, of the model shown in the upper left. For example, setting the minimum curvature κ_2 to zero while keeping κ_1 fixed straightens out the curved regions, as shown in the top right. In the bottom right, the curvatures of a smaller scale are set to the curvatures estimated at a larger scale, as visualized in the curvature plots in the bottom left.

that are difficult to achieve by manipulating spatial 3D coordinates but trivial to formulate in the curvature domain.

Implementation. Following the pipeline depicted in Figure 3.19, we first compute discrete principal curvatures on the input mesh as described in Section 3.3.1. These curvature values can then be modified by the user by applying filtering operations or direct edits, and the resulting surface is reconstructed using our constraint based optimization framework. For Curvature-Domain Shape Processing, only a subset of the optimization energies described in Sections 3.2 and 3.3 are required: The user constraints are enforced by the principal curvature energy of Equation 3.12, where the curvature values $\kappa_{1,i}, \kappa_{2,i}$ of the original surface are replaced by the user specified target curvatures $\hat{\kappa}_{1,i}, \hat{\kappa}_{2,i}$. As with the modeling constraints discussed in the previous section, the resulting surface is not uniquely specified by the principal curvature constraints alone. Therefore we additionally preserve the metric properties of the original surface as well as possible.

The three surface energies (curvature and metric) are combined as described in Section 3.3 and can be weighted to control the deformation behavior. Our experiments have shown that for Curvature-Domain Shape Processing a dominant conformal energy usually gives the best deformation flexibility to achieve the prescribed principal curvatures. We therefore use a very small optimization weight k_{areal} for all our examples below. Since the surface energies are invariant under rigid transformations, either a number of vertices have to be fixed to their original position or soft position constraints have to be added to all vertices using a small weight, as described in Section 3.3.

For some applications, there may be a conflict between the desired curvatures and the original shape metric, since the requested curvatures necessitate a deviation in triangle shape. For this reason we apply the optional adaption of the optimization procedure, as proposed in Section 3.4, to allow the metric to evolve in an iterative fashion.

Results. Figure 3.21 shows a number of curvature-domain edits on a shape with a complex curvature distribution. For cross-scale smoothing, we set the target principal curvatures of the smallest scale (barycentric vertex area) to the values computed at a scale of four times

3 Constraint-Based Modeling

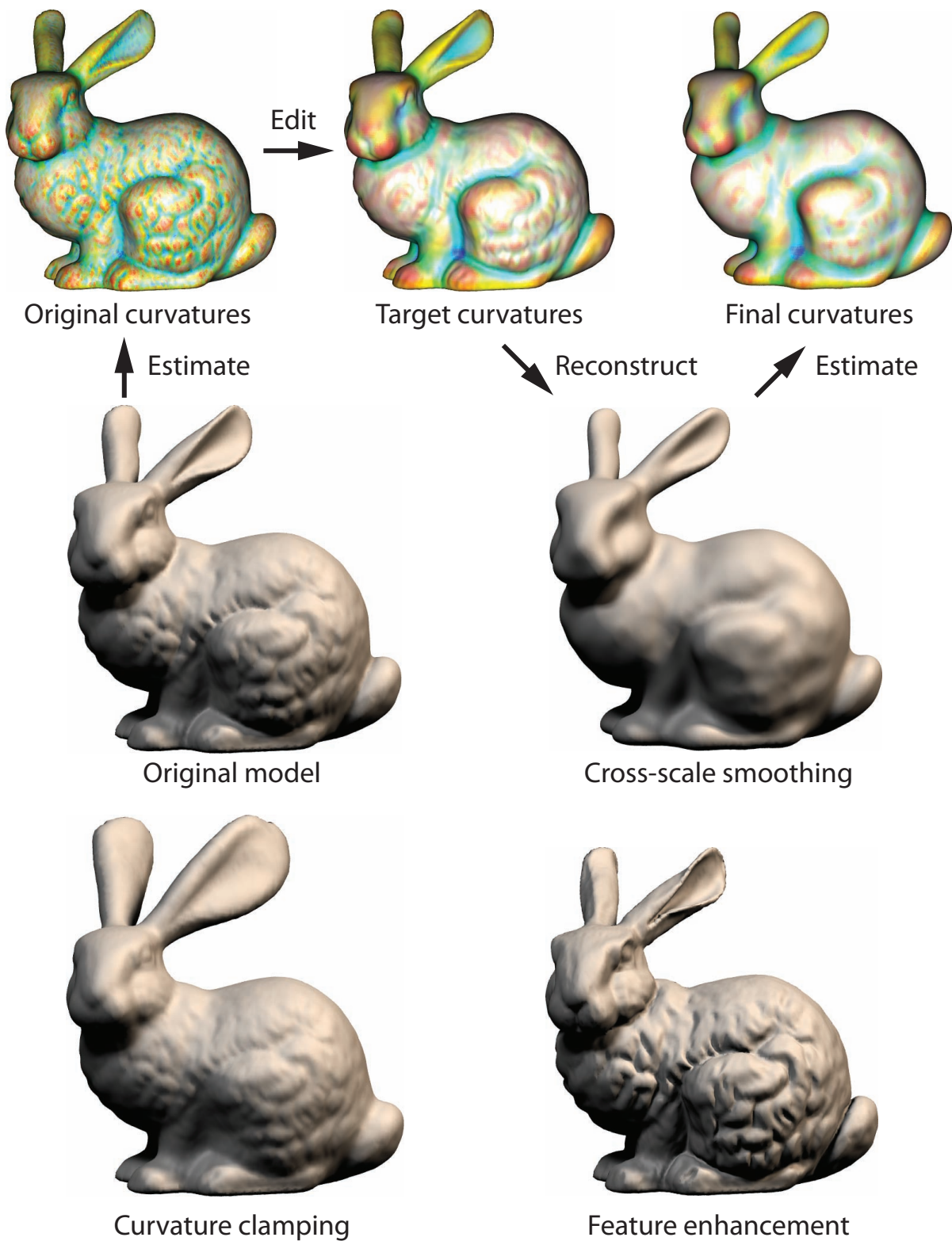


Figure 3.21 *Curvature-domain processing on the Stanford bunny. The top row shows the curvature plots of the original model, the prescribed target values, and the actual achieved curvatures on the final reconstructed model. Vertices on the base of the bunny are constrained to remain fixed in place in the optimization.*

the average one-ring radius. As the curvature plots indicate, the target curvature values are well approximated in the final model. The curvature plots visualize both principal curvatures according to the colormap in Figure 3.20. Note that the resulting smoothing avoids local shrinkage artifacts often observed with diffusion-based approaches that can increase mean curvature and eventually lead to pinch-off singularities.

Curvature clamping restricts both the minimum and maximum curvature to lie within a user-defined interval, thus removing the extreme curvatures from the surface. As a result, the bunny's ears inflate in order to avoid high curvatures. Note how surface detail that is characterized by curvatures within the clamping interval is preserved, while strong creases are smoothed and rounded.

Figure 3.23 shows one-sided curvature clamping on a machine part, where curvatures are restricted to lie in the interval $[-5, \infty]$. As a result, concave corners evolve into smooth fillets to meet the new curvature requirements. Note that normal discontinuities across feature edges are introduced purely for rendering purposes. Curvature processing is oblivious to these tagged edges, i.e. the entire mesh is considered a discrete approximation of a smooth surface.

Feature enhancement in Figure 3.21 is achieved by increasing the largest absolute principal curvature by an amount proportional to the difference of the absolute principal curvature values. This change enhances convex ridges as well as concave valleys. The same enhancement filter is applied in Figure 3.22 (c) on a digital elevation model. This figure also shows the effect of multi-scale editing (b), where target curvatures have been specified on two different scales: the curvatures of the original model are prescribed on the smallest scale, while on a coarser scale target curvatures are set to a constant in a circular region around the peak of the mountain. This "flattens out" the mountain flanks, while preserving the fine-scale structure. Multi-scale editing is easily incorporated into the optimization by including target curvatures at different scales in the energy E_c . Image (d) shows the result of a single-scale edit, where both principal curvatures for the center region have been set to the same constant, thus pushing the shape towards a spherical configuration.

3 Constraint-Based Modeling

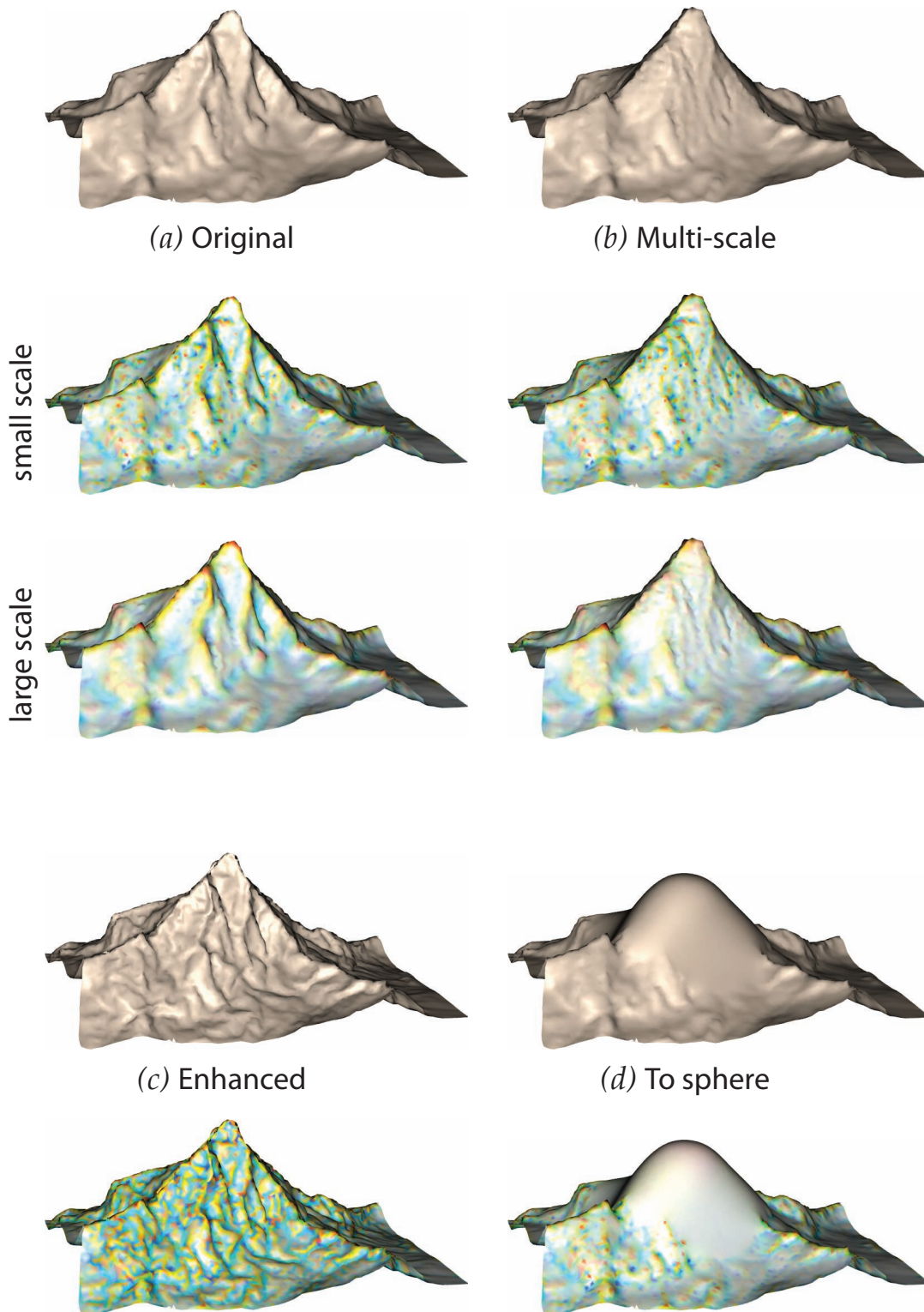


Figure 3.22 Different curvature-domain processing operations on the Matterhorn. Curvature plots show the resulting curvatures after the optimization. For (b) curvature optimization is performed at two scales.

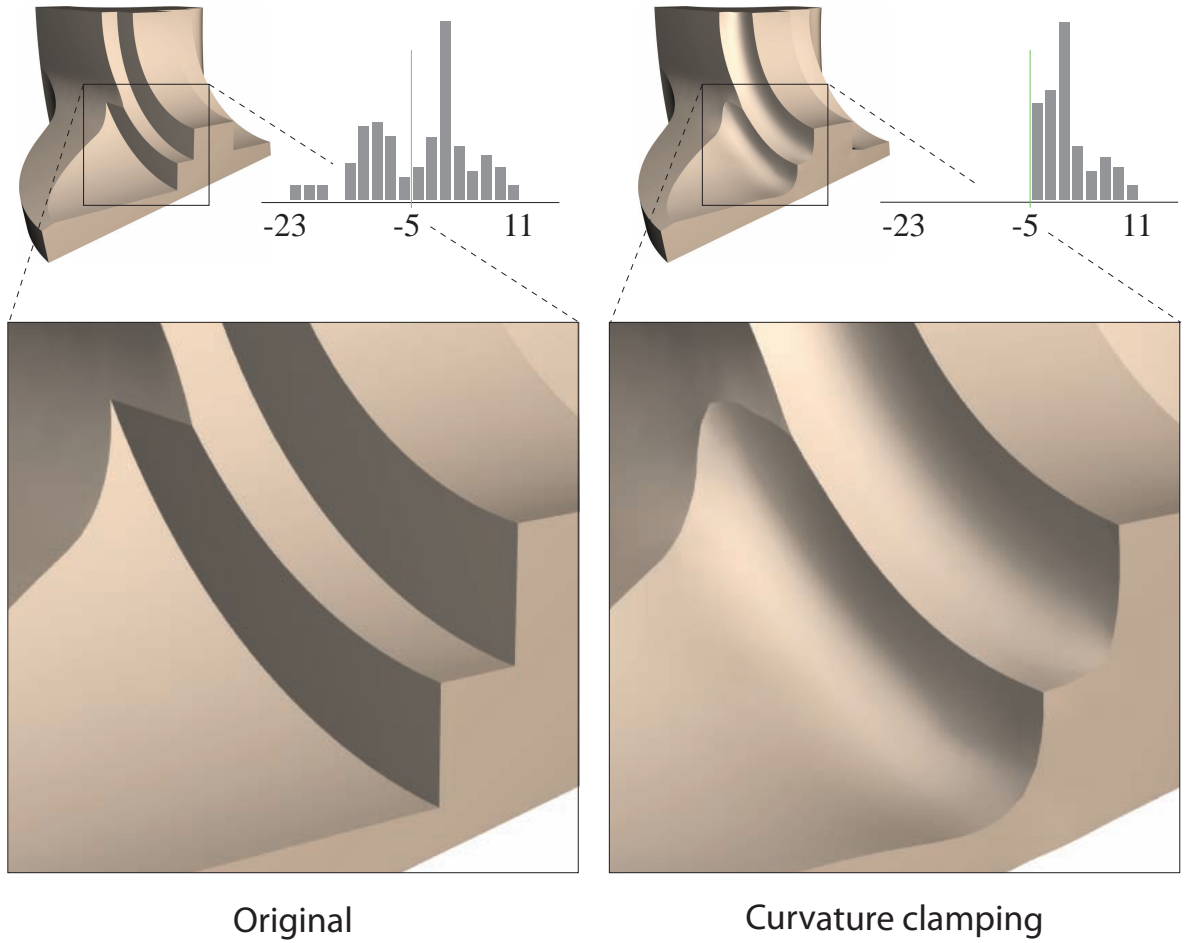


Figure 3.23 *Curvature clamping of a machine part. The histograms show the distribution of κ_2 on a logarithmic scale before and after the optimization.*

Figure 3.24 shows a curvature-domain bilateral filter applied to a noisy range scan. Target curvature values $\hat{\kappa}_{1,i}$ for vertex v_i (and analogously for $\hat{\kappa}_{2,i}$) are computed as local weighted averages that combine domain and range filtering

$$\hat{\kappa}_{1,i} = \frac{\sum_{v_j \in N_i} \phi_c(\|\mathbf{v}_i - \mathbf{v}_j\|) \phi_s(|\kappa_{1,i} - \kappa_{1,j}|) \kappa_{1,j}}{\sum_{v_j \in N_i} \phi_c(\|\mathbf{v}_i - \mathbf{v}_j\|) \phi_s(|\kappa_{1,i} - \kappa_{1,j}|)}, \quad (3.22)$$

where N_i is the local averaging region around vertex v_i , and ϕ_c and ϕ_s are two Gaussians measuring spatial closeness and curvature similarity, respectively (see [TM98] for details). Compared to isotropic Laplacian smoothing, the bilateral filter better preserves ridges and corners, while leading to an overall smoother curvature distribution. The curvature plots reveal the anisotropic behavior of the filter, i.e.,

3 Constraint-Based Modeling

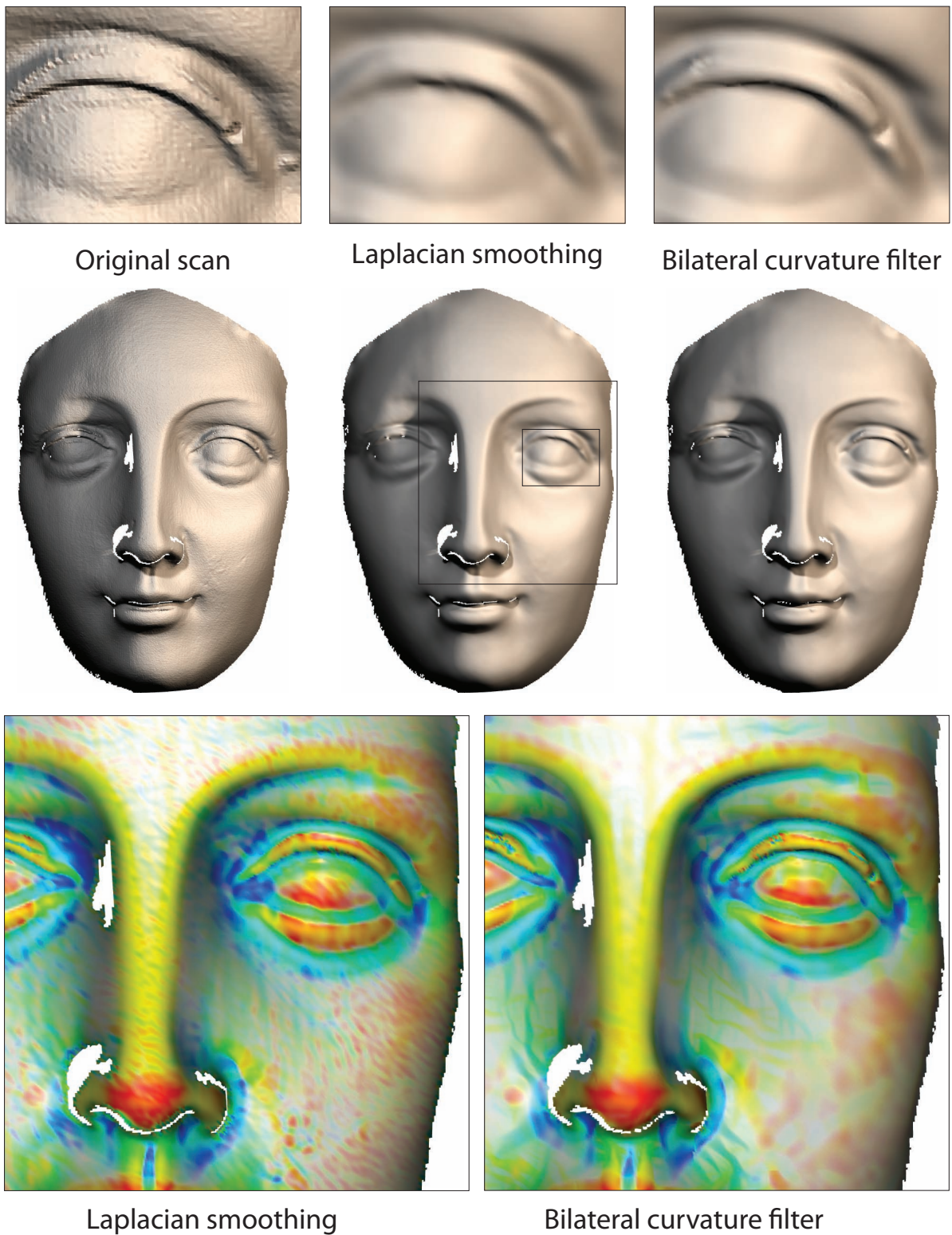


Figure 3.24 Comparison of isotropic Laplacian smoothing with bilateral filtering of curvatures on a noisy range scan.

illustrate how curvature variation is reduced along feature lines without blurring the surface across the features. For comparison we apply ten smoothing steps in both examples. Boundaries are handled without special treatment. Our formulation operates directly on discrete curvatures, i.e., scalar attributes defined on each mesh vertex, so that we avoid local height-field parameterizations of vertex positions, as for example used in [FDCO03], that can lead to distortions for larger neighborhoods or high curvature regions.

In general, not every set of target curvature values is realizable. For example, there is no 3D embedding of a genus-1 surface with constant principal curvatures everywhere. Therefore our algorithm tries to meet the specified curvature constraints as well as possible in a least squares sense. To evaluate the results of our algorithm we introduce a score function σ that measures the degree to which the prescribed curvatures are achieved by the optimization procedure. This function is designed to be a relative measure independent of both scale and sampling and is defined as

$$\sigma = 1 - \frac{\sum_{v_i \in \mathcal{V}} A_{v_i} [(\hat{\kappa}_{1,i} - \kappa'_{1,i})^2 + (\hat{\kappa}_{2,i} - \kappa'_{2,i})^2]}{\sum_{v_i \in \mathcal{V}} A_{v_i} [(\hat{\kappa}_{1,i} - \kappa_{1,i})^2 + (\hat{\kappa}_{2,i} - \kappa_{2,i})^2]}, \quad (3.23)$$

where $\kappa_{j,i}$ denotes the principal curvatures of the original model.

Higher values indicate that the optimization procedure was better able to match the desired curvatures. Table 3.1 lists the values of σ for the models shown in Figures 3.20 and 3.21. The differences in the achieved scores stem from the fact that the metric regularization prevents drastic changes of the shape and that target curvatures can be incompatible.

3.7 Discussion

Speed. One major challenge of our nonlinear optimization framework is speed. Ideally, the system should provide immediate feedback when the user modifies a constraint. Our initial prototype implementation was only able to achieve interactive response for such edits on relatively small meshes. The curve bending on the fandisk with

Fig. 3.20	σ	Bunny	σ
$\kappa_1 \rightarrow 2\kappa_1$.950	Clamp	.821
$\kappa_1 \rightarrow 2$.944	Cross-Scale	.963
$\kappa_2 \rightarrow 0$.858	Enhance	.721
$\kappa_{1,2} \rightarrow 1.5$.999		
$\kappa_2 \rightarrow -\kappa_2$.988		
$\kappa_{1,2} \rightarrow \kappa'_{1,2}$.994		

Table 3.1 The value $\sigma \in [0, 1]$ indicates to what extent the target curvatures are achieved by the final model.

6477 vertices took less than 10 seconds on a 2.4 GHz Intel[®] Core[™]2 Duo with 2GB of memory. The edits on the horse and the vase (both around 8000 vertices) took 8 to 150 seconds each. Since for 2D deformations smaller triangle meshes are usually sufficient and for every vertex there are only two instead of three variables, our initial prototype implementation already computed the constraint-based image deformations within a couple of seconds. Curvature-Domain Shape Processing is more costly but less time-critical, since most curvature filtering operations do not mandate a real-time response. For example, the curvature optimizations on the bunny model with 35,111 vertices, that is, more than 100k unknowns, took about 2.5 minutes.

In the Master Thesis of Rolf Weilenmann [Wei09] we investigated the potential of parallelism for our optimization framework. Since practically all involved terms in the optimization can be computed independently, we can heavily exploit the multicore architecture of our Intel[®] CPU using the OpenMP[™]API (www.openmp.org) and GPU parallelism using CUDA (www.nvidia.com/object/cuda_home.html, [Nvi05, BGo8]) to compute the optimization energies and their derivatives in parallel. Figure 3.25 compares the performance of this advanced implementation compared to the initial sequential implementation for the manipulation of curve length shown in Figure 3.3. To test the scale dependency of the implementations we run the optimization for an increasingly denser triangle mesh of the sphere. As shown in Figure 3.25 the parallel implementation achieves a significant speedup

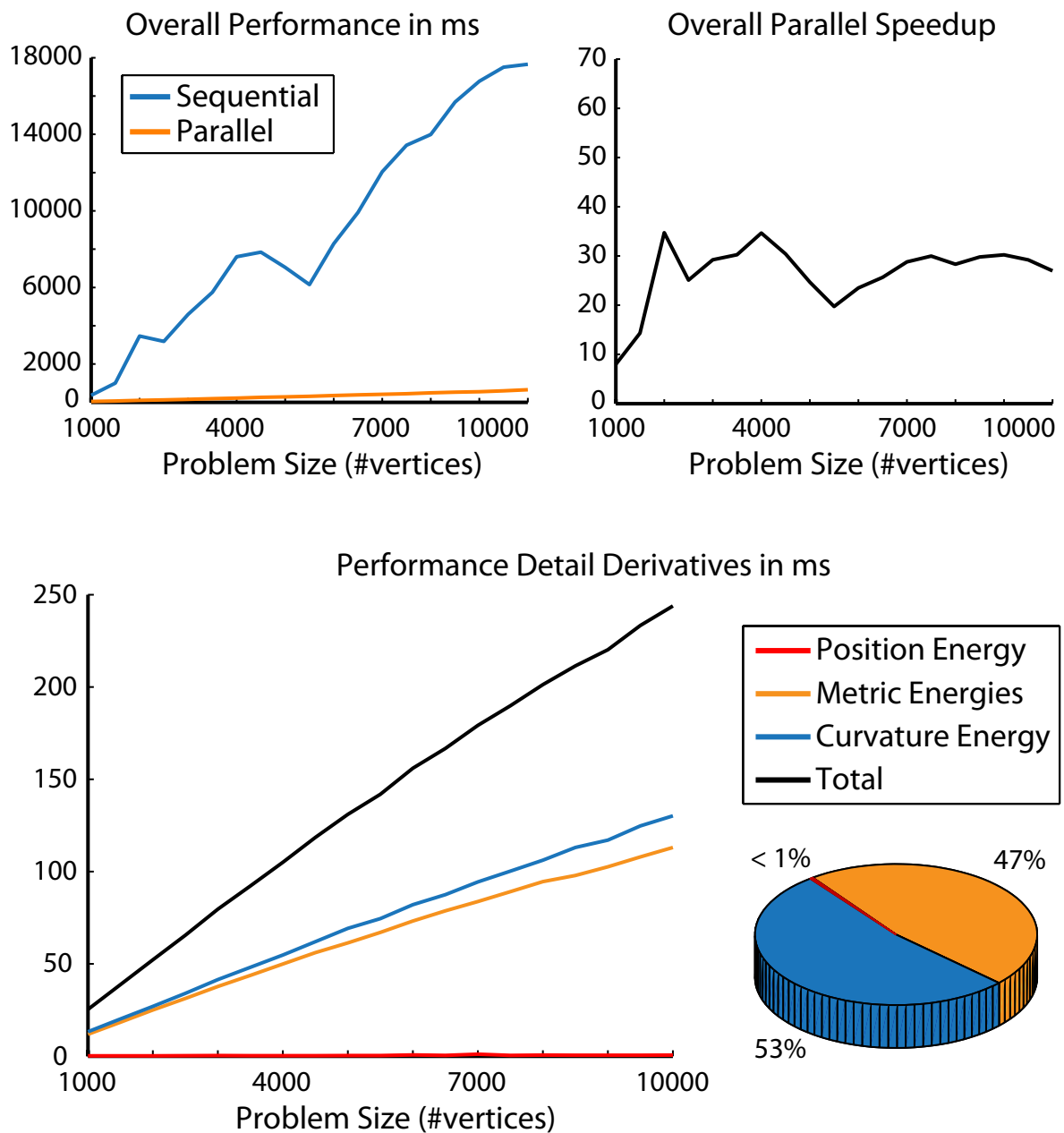


Figure 3.25 Performance evaluation of our constraint-based modeling framework for different problem sizes of the curve-length edit shown in Figure 3.3. Top: Comparison of total running times per iteration between sequential and parallel implementation. Bottom: detailed analysis of the parallel computation of derivatives for the involved energy terms.

3 Constraint-Based Modeling

and is 20 to 30 times faster than the sequential implementation, which enables interactive constraint-based editing for larger meshes. We believe that an additional performance improvement could be achieved by a hierarchical representation of the constraints and the surface to implement multi-level solvers in the spirit of multi-grid methods, which offers potential for future research.

Memory. For global constraints, such as the integrated area term, the Jacobian of the objective function might become too dense to fit into memory if the patch size is too large (see also Section 3.4). On our 3GB RAM desktop PC this occurred when selecting more than around one third of a model with 20'000 faces. This could be avoided by trading off space with speed and recomputing the entries of the Jacobian for each matrix access.

Advanced Metric Control. The local metric energies employed in our framework are isotropic measures. For specific edits, however, the user might want to penalize metric changes in certain directions more than in others. A challenging extension would be to develop anisotropic distortion measures and a proper interface that enables an intuitive control for these energies.

Curvature Domain-Shape Processing – Relation to Existing Techniques. The results shown in the previous section demonstrate the flexibility of curvature-domain shape processing. Although specialized techniques based only on mean curvature exist for some of the applications presented, our approach offers a different perspective since the specific geometric operations employed by our method are often quite different than the ones already known. Investigating these similarities and differences for specific applications may lead to new insights about shape processing and suggest avenues of future work in this area.

The example of Figure 3.24 demonstrates how a standard image processing filter can be applied in curvature-domain to implement an advanced geometry processing tool. We believe that our curvature-based processing metaphor offers a versatile framework for exploring a variety of such filtering methods. Another interesting direction of future

research is therefore to evaluate different filters for their potential use in curvature-based geometry processing.

An interesting connection of our approach to continuum mechanics becomes apparent when comparing the energy function of our combined curvature and metric energies (Equation 3.14) with the elastic thin-shell energy

$$E = \int_{\Omega} k_s \|\mathbf{I} - \mathbf{I}'\|_F^2 + k_b \|\mathbf{\Pi} - \mathbf{\Pi}'\|_F^2 \, dudv, \quad (3.24)$$

where \mathbf{I} denotes the matrix representation of the first and $\mathbf{\Pi}$ the matrix representation of the second fundamental form (see Chapter 2), and k_s and k_b are stiffness parameters that determine the stretching and bending resistance of the material [TPBF87]. In our formulation we replace the bending term by principal curvature differences, thus avoiding the directional component of the second fundamental form. The key difference, however, is that in Equation 3.24 the second fundamental form $\mathbf{\Pi}$ is computed from a given, initial rest state, whereas we prescribe arbitrary target curvatures values. Effectively, minimizing the objective function 3.14 pulls the surface to a “fictitious” rest state defined by the target curvatures.

Rusinkiewicz observed [Rus04] that the curvature tensor proposed by Cohen-Steiner and Morvan can yield a poor approximation of the true curvatures for small local neighborhoods at low-valence vertices. In our optimization this can lead to local distortions that diminish the quality of the final result. In general, we observed that the performance degrades with poor input mesh quality, since the estimation of curvatures and corresponding derivatives becomes numerically unstable. Our experiments indicate that these artifacts can be reduced by applying appropriate local remeshing operations, e.g., [SG03], to improve the stability of the curvature estimation.

3 Constraint-Based Modeling

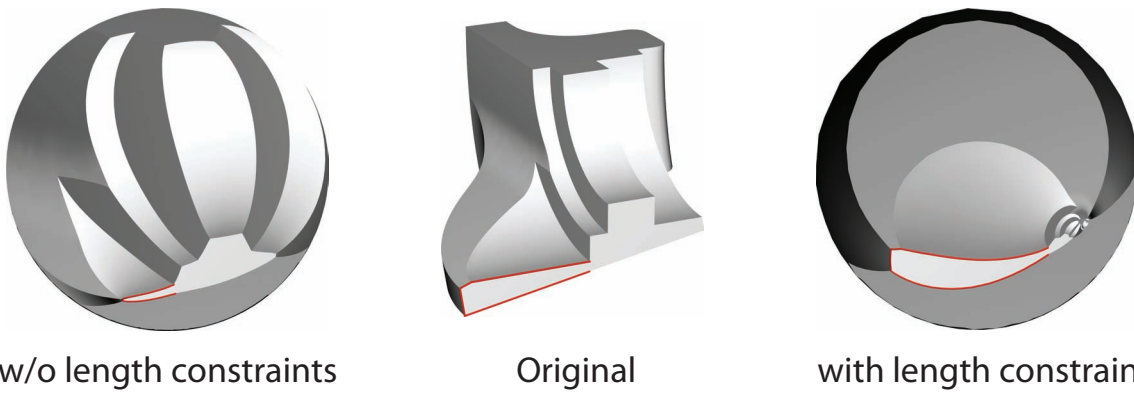


Figure 3.26 *Spherical conformal parameterization computed by prescribing constant curvature. On the right the marked curves are constrained to keep their original length.*

Combined Constraints Figure 3.26 illustrates a potential application that offers potential for future work. We can evolve the fan disk towards a sphere by setting all principal curvatures to a constant. With a high conformal energy term, this yields a quasi-conformal spherical parameterization of the model that can be customized with the use of additional metric constraints. Conceptually, this approach is similar to recent methods for planar conformal mappings [BCGBo8, SSPo8].

Paneling Architectural Freeform Surfaces

4.1 Introduction

Freeform shapes play an increasingly important role in contemporary architecture. With the emergence of large-scale architectural freeform surfaces the essential question arises of how to proceed from a geometrically complex design towards a feasible and affordable way of production. This fundamental problem, in the architectural community referred to as *rationalization*, is largely related to the issue of *paneling*, i.e., the segmentation of a shape into simpler surface patches, so-called panels, that can be fabricated at reasonable cost with a selected manufacturing process (see Figure 4.1). The paneling problem can arise both for the exterior and interior skin of a building, and plays a central role in the design specification phase of any architectural project involving freeform geometry.

Recent technological advances enable the production of single- and double-curved panels that allow a faithful approximation of curved

4 Paneling Architectural Freeform Surfaces

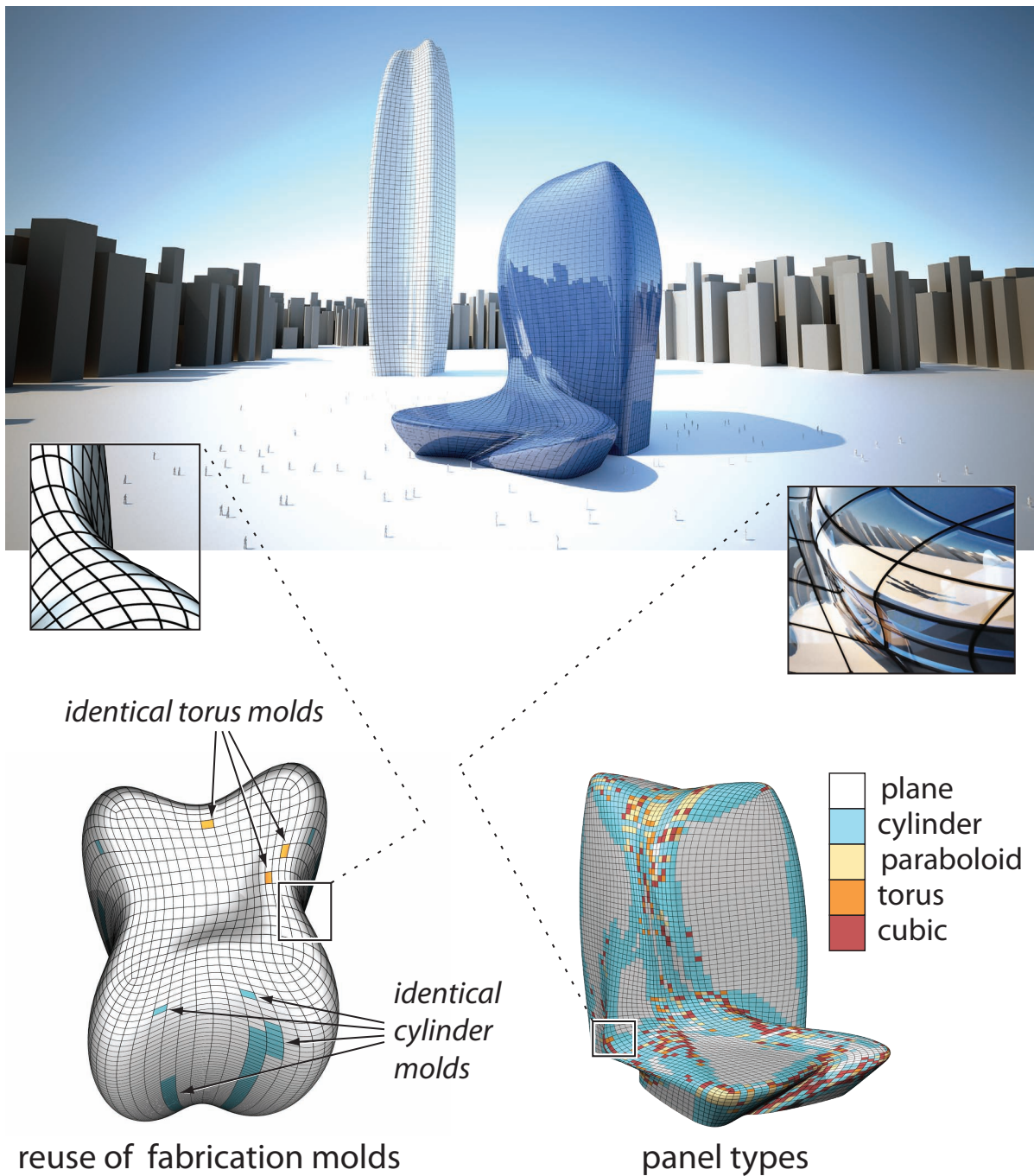


Figure 4.1 Rationalization of large-scale architectural freeform surfaces with planar, single-, and double-curved panels. Our algorithm computes a paneling solution that meets prescribed thresholds on positional and normal continuity, while minimizing total production cost. Reuse of molds (left) and predominant use of simple panels (right) are important drivers of the optimization. (Left: Zaha Hadid Architects, Liliun Tower, Warsaw. Right: Zaha Hadid Architects, National Holding Headquarters, Abu Dhabi.)

surfaces. While planar panels are always the most cost-effective, the progression towards the more expensive general freeform panels depends on the panel material and manufacturing process. Table 4.1 provides an overview of the state-of-the-art in architectural panel production. Most commonly, curved panels are produced using molds with the cost of mold fabrication often dominating the panel cost (see Figure 4.9). There is thus a strong incentive to reuse the same mold for the production of multiple panels to reduce the overall cost.

Our goal is to find a paneling solution for a given freeform design that achieves prescribed quality requirements, while minimizing production cost. The *quality* of the paneling is mainly determined by the geometric closeness to the input surface, the positional and normal continuity between neighboring panels, and the fairness of corresponding panel boundary curves. The *cost* mostly depends on the size and number of panels, the complexity of the panel geometry, and the degree of reuse of molds that need to be custom-built to fabricate the panels. A key objective of our work is to solve instances of the paneling problem on large-scale architectural freeform designs that often consist of thousands of panels. Due to the high complexity and global coupling of optimization objectives and constraints, manual layout of panels for these freeform surfaces is infeasible, mandating the use of advanced computational tools.

The paneling problem is a constraint-based surface approximation problem involving highly complex geometric constraints:

- The objective to minimize cost imposes constraints on the geometry of the panel shapes such that the panels can be efficiently produced and that mold reuse is computationally feasible.
- The quality requirements directly impose geometric constraints on the distances and angles between neighboring panels and the distances of the panels to the input design surface.

The paneling problem is also an engineering problem and the results of our paneling algorithm are eventually used as an input to actually manufacture architectural freeform structures. The constraint-based surface paneling problem is therefore of a different nature than the constraint-based surface modeling problem discussed in Chapter 3

4 Paneling Architectural Freeform Surfaces

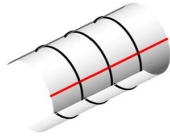

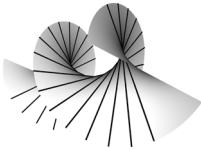

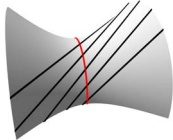
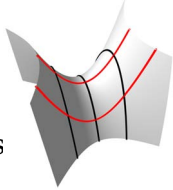
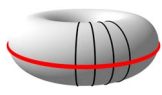
surface types		manufacturing possibilities		
		<i>glass</i>	<i>metal</i>	<i>fibre reinforced concrete/plastic</i>
single curved				
isometric to the plane, no or little plastic deformation of material				
<i>cylindrical</i> parts of right circular cylinders		machine for bending and thermal tempering	roll bending machine	configurable mold or custom hot-wire cut foam mold
<i>conical</i> parts of right circular cones		configurable or custom mold, no thermal tempering	machine or re-configurable mold	configurable mold or custom hot-wire cut foam mold
<i>general single curved</i> developable surfaces		custom mold, no thermal tempering		custom hot-wire cut foam mold
double curved				
usually plastic deformation of material is involved				
<i>general double curved</i>		custom molds, no thermal tempering of glass	machine or re-configurable mold	custom molds commonly made of EPS foam
<i>general ruled</i> generated by a moving straight line		straight lines can be exploited	see above	foam molds can be hot-wire cut
<i>translational</i> carries two families of congruent profiles		congruent profiles can be exploited		congruent profiles can be exploited
<i>rotational</i> carries one family of congr. profiles				

Table 4.1 Classification of panel types and state-of-the-art production processes for common materials in architecture. This table is for a rough guideline and do not cover all the relevant production processes. Planar panels have been left out. (Classification created by Evolute, www.evolute.at)

and requires different algorithmic strategies. Certain constraints like the distance between neighboring panels have to be met exactly and not only in a least squares sense. Additionally, the problem involves not only continuous but also discrete variables like how many and which types of molds should be used, and which molds should produce which panel. While a least squares optimization similar to the one introduced in the previous Chapter is an essential ingredient of our paneling algorithm, we present new computational tools to solve the particular challenges of the paneling problem.

4.1.1 Contributions

We introduce a computational approach to freeform surface paneling. As our main contributions, we

- identify the key aspects of the paneling problem that are amenable to computation and derive a mathematical framework that captures the essential design goals,
- present an algorithmic solution based on a novel global optimization method that alternates between discrete and continuous optimization steps to meet paneling quality constraints while reducing cost through mold reuse,
- introduce a novel 6-dimensional metric space to allow fast computation of approximate inter-panel distances, which dramatically improves the performance of the optimization and enables the handling of complex arrangements with thousands of panels, and
- demonstrate the practical relevance of our system by computing paneling solutions for real, cutting-edge designs that currently cannot be realized at the desired aesthetic quality.

4.1.2 Related Work

Early contributions to the field of freeform architecture come from research at Gehry Technologies (see e.g. [Sheo2]). These are mostly dedicated to developable or nearly developable surfaces, as a result of the specific design process that is based on digital reconstruction of models made from material that assumes (nearly) developable shapes.

Research on freeform architecture is promoted by the Smart Geometry group (www.smartgeometry.com), whose interest so far mostly focussed on parametric design tools. These can be helpful for shape generation processes that have panel properties built into them. However, such a forward approach makes it very difficult to achieve the desired shapes and obtain a satisfactory paneling solution for sufficiently complex geometries.

Most previous work on the paneling problem deals with planar panels. Initial research in this direction dealt with special surface classes [GSC⁺02]. Covering general freeform surfaces with planar quad panels could be approached with methods of discrete differential geometry [BS09] and led to new ways of supporting beam layout and the related computation of multi-layer structures [LPW⁺06, PLW⁺07]. This approach was extended to the covering of freeform surfaces by single-curved panels arranged along surfaces strips [PSB⁺08]. Additional results in this direction, e.g., hexagonal meshes with planar faces, have been presented at “Advances in Architectural Geometry” [PHK08]. The idea of optimizing for repeated elements by altering the vertex positions of a given mesh is explored by Fu et al. [FLHCO10] in the context of quad meshes and by Singh and Schaefer [SS10] in the context of triangle meshes, in order to create a set of reusable pre-fabricated tiles.

Our approach bears some similarity to variational methods for approximating a surface with simple geometric primitives. Originally introduced by Cohen-Steiner et al. [CAD04] for surface approximation by planes, various extensions have been proposed for additional surface types, e.g., spheres and cylinders [WK05], quadrics [YLW06], or developable surfaces [JKS05]. In [LSSK09], an optimization has been proposed to simultaneously partition the input surface, as well as de-



Figure 4.2 *Projects involving double-curved panels where a separate mold has been built for each panel. These examples illustrate the importance of the curve network and the challenges in producing architectural freeform structures. (Left: Peter Cook and Colin Fournier, Kunsthaus, Graz. Right: Zaha Hadid Architects, Hungerburgbahn, Innsbruck.)*

termine the types and number of shape proxies required. These methods optimize for a surface segmentation to reduce the approximation error. In our setting, the segmentation is part of the design specification and we optimize for position and tangent continuity across panel boundaries, allowing systematic deviations from the reference surface to improve the paneling quality and reduce cost. Enabling mold reuse and aesthetic control, which are key requirements of architectural rationalization, necessitates a substantially different approach both in the underlying formulation of the optimization as well as its implementation. Similarly, state-of-the-art methods in surface fitting and local registration (see e.g. [VMo2, Shao8]), while an integral component of our system, are insufficient to solve large-scale freeform paneling problems.

In shape analysis, the problems of symmetry detection [MGP06, PSG⁺06] and regularity detection [PMW⁺08] involve identification and extraction of repeated elements, exact and approximate, in 3D geometry. Subsequently, detected repetitions can be made exact by

symmetrization using subtle modifications of the underlying meshing structure [MGP07, GPF09] thus deforming a surface towards an exact symmetric configuration. These methods, designed to enhance detected symmetries, are unsuited for handling architectural freeform designs where large repeated sections are exceptions rather than the norm. Our optimization has a similar symmetrizing effect in enabling trading approximation error for a stronger degree of mold reuse, leading to significant savings in terms of manufacturing cost.

4.2 Problem Specification

In this section we introduce a specification of the paneling problem and the corresponding terminology common in architectural design (see Figure 4.3). The specification is the result of extensive consultations of our collaborator Evolute with architects and Evolute’s experience with real-world freeform design projects, some of which we highlight in Section 4.4.

4.2.1 Terminology

Panels and Molds. Let F be the given input freeform surface describing the shape of the design. Our goal is to find a collection $\mathcal{P} = \{P_1, \dots, P_n\}$ of *panels* P_i , such that their union approximates F . The quality of the approximation strongly depends on the position and tangent continuity across panel boundaries: *Divergence* quantifies the spatial gap between adjacent panels, while the *kink angle* measures the jump in normal vectors across the panel intersection curves.

Curved panels are commonly produced using a manufacturing *mold* M_k . We call the collection $\mathcal{M} = \{M_1, \dots, M_m\}$ with $m \leq n$ the *mold depot*. To specify which mold is used to produce which panel(s), we define a panel-mold *assignment function* $A : [1, n] \rightarrow [1, m]$ that assigns to each panel index the corresponding mold index. The arrangement of panels in world coordinates is established by rigid transformations T_i that align each panel P_i to the reference surface F .

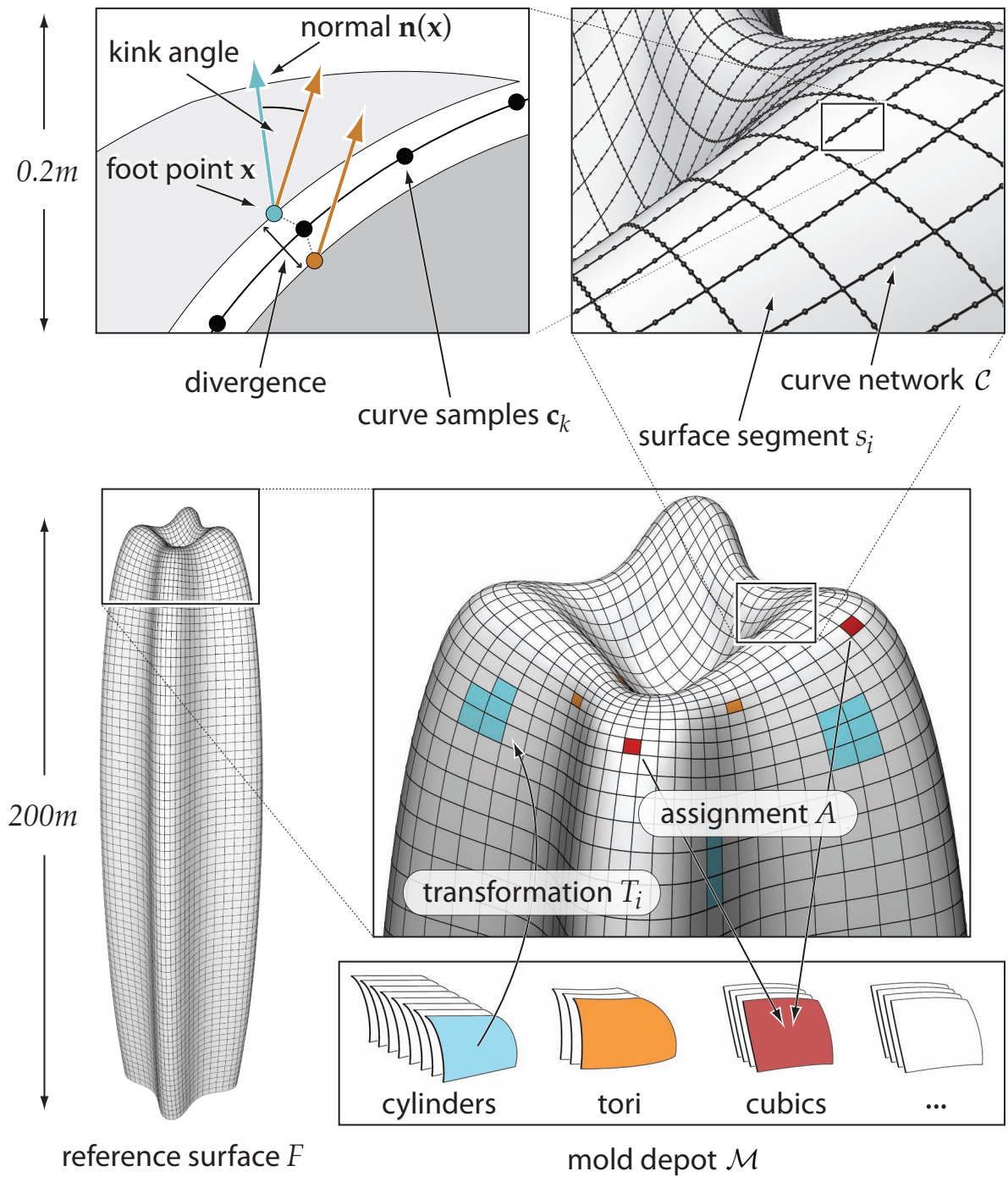


Figure 4.3 Terminology and variables used in our algorithm. The reference surface F and the initial curve network C are given as part of the design specification. The optimization solves for the mold depot M , the panel-mold assignment function A , the shape parameters of the molds, the alignment transformations T_i , and the curve network samples c_k .

4 Paneling Architectural Freeform Surfaces

Panels produced from the same mold are sub-patches of the mold surface and need not be congruent.

Let $c(M_k)$ be the fabrication cost of mold M_k and $c(M_k, P_i)$ the cost of producing panel P_i using mold M_k (see also Figure 4.9). The total cost of panel production can then be written as

$$\text{cost}(F, \mathcal{P}, \mathcal{M}, A) = \sum_{k=1}^m c(M_k) + \sum_{i=1}^n c(M_{A(i)}, P_i). \quad (4.1)$$

Ideally, the same mold will be used for the fabrication of multiple panels to reduce cost. The choice of panel types depends on the desired material and on the available manufacturing technology. Materials may be transparent or opaque, and include glass, glass-fibre reinforced concrete or gypsum, various types of metal, and wood.

Currently we support five panel types: planes, cylinders, paraboloids, torus patches, and general cubic patches. Planar panels are easiest to produce, but result in a faceted appearance when approximating curved freeform surfaces, which may not satisfy the aesthetic criteria of the design. A simple class of curved panels are cylinders, a special case of single-curved (developable) panels. Naturally, such panels can lead to a smooth appearance only if the given reference surface exhibits one low principal curvature. General freeform surfaces often require double-curved panels to achieve the desired tolerances in fitting error, divergence, and kink angles. We consider three instances of such panels: paraboloids, torus patches, and cubic patches. The former two carry families of congruent profiles (parabolaes and circles, respectively), which typically simplifies mold production. Cubic panels are most expensive to manufacture, but offer the highest flexibility and approximation power. Thus a small number of such molds are often indispensable to achieve a reasonable quality-cost tradeoff.

Curve Network. The intersection curves between adjacent panels are essential for the visual appearance of many designs (see Figure 4.2) and typically affect the structural integrity of the building, as they often directly relate to the underlying support structure. An initial layout of these curves is usually provided by the architect as an integral part of the design. While small deviations to improve the paneling quality

are typically acceptable, the final solution should stay faithful to the initial curve layout and reproduce the given pattern as well as possible by the intersection lines of adjacent panels. Our paneling algorithm supports arbitrary curve network topology and is not restricted to predefined patterns. The collection of all panel boundary curves forms the *curve network* that we denote with \mathcal{C} . Projecting \mathcal{C} onto the input freeform surface F yields a partitioning of F into a collection $\mathcal{S} = \{s_1, \dots, s_n\}$ of *segments* s_i . The panel P_i associated with segment s_i can be created by trimming the aligned mold surface $M_i^* := T_i(M_{A(i)})$. The trim curves are obtained by projecting the network curves associated with segment s_i onto M_i^* .

4.2.2 The Paneling Problem

We formulate the paneling problem as follows: Approximate a given freeform surface by a collection of panels of preferred types such that the total production cost is minimized, while the paneling respects pre-defined thresholds on divergence and kink angle between adjacent panels, and reproduces the initial curve network as well as possible. A closer look at this specification reveals that any solution of the paneling problem has to address the following central aspects:

- **Mold Depot:** determine the number and types of molds that should be fabricated.
- **Assignments:** find the optimal assignment function to establish which panel is best produced by which mold.
- **Registration:** compute the optimal shape parameters for each mold and the optimal alignment of each panel such that the reference surface is faithfully approximated, thresholds on kink angles and divergence are met, and the panel intersections curves respect the design intent.

Mold depot and assignment function determine the total cost of fabrication, while registration affects the quality of the rationalization. Minimizing fabrication cost calls for a maximum amount of mold reuse and the wider use of panels that are geometrically simple and thus

cheaper to manufacture (see Equation 4.1). On the other hand, achieving the design constraints on the paneling quality pushes the solution towards more complex panel shapes with less potential for mold reuse.

The high intricacy of the paneling problem arises both from the large scale of typical projects (1k – 10k panels) and the tight global coupling of objectives. Neighboring panels are strongly linked locally through kink angle and divergence measures, but also subject to a highly non-local coupling through the assignment function that facilitates mold reuse. The cost-quality tradeoffs involved in using different mold types add additional complexity: It is obvious that we want to use as many cheap, simple molds as possible. However, adding one expensive mold might save us from having to add many cheap molds whose total cost might be higher. Also, using a complex mold at certain places may enable the use of simpler panels in its surroundings.

4.3 Paneling Algorithm

A key design decision in our paneling algorithm is to represent the curve network explicitly as a set of polygonal curves, rather than computing the boundary curves from the intersection of neighboring panels (Figure 4.3). By integrating the corresponding curve vertices \mathbf{c}_k as free variables in our system, we gain several important advantages. In particular, we

- avoid numerical instabilities when explicitly computing intersections of neighboring, nearly tangent-continuous patches,
- simplify the specification of surface fitting and continuity constraints across neighboring panels (see Section 4.3.1),
- achieve better quality at lower cost by allowing the curves to move away from the reference surface as part of the optimization (see Figure 4.4), and
- provide essential means of control for the designer, who can explicitly specify where neighboring panels should intersect.

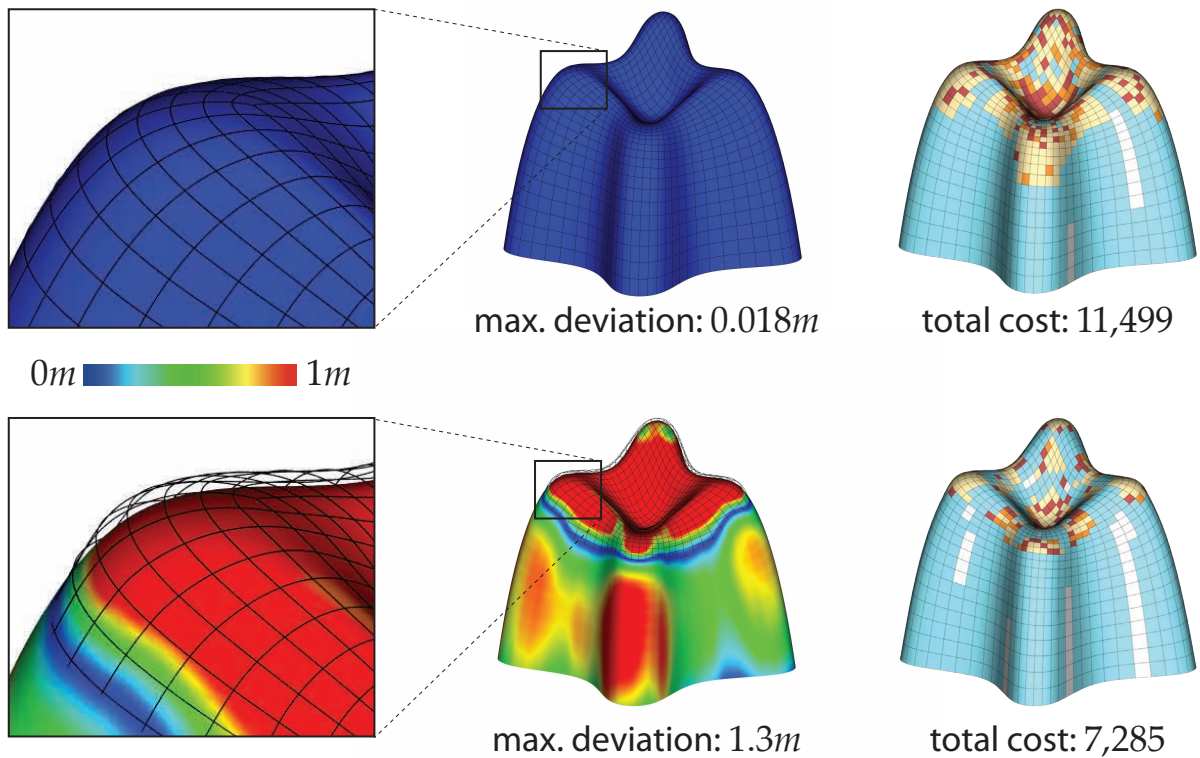


Figure 4.4 Our algorithm allows controlling the amount of deviation from the reference surface, shown here for the example of the Liliun tower. Larger deviations enable a more cost-effective solution using cheaper panels, while still satisfying the thresholds on kink angle and divergence.

An important aspect of the paneling problem is the need to simultaneously solve for both discrete and continuous variables. The discrete variables are the number and type of molds constituting the mold depot \mathcal{M} , and the panel-mold assignment function A . The continuous variables are the shape parameters of the molds, the transformations T_i that align mold $M_{A(i)}$ to segment s_i , and the positions of the curve network samples \mathbf{c}_k .

Given a mold depot \mathcal{M} and assignments A , we use a continuous non-linear least-squares optimization (Section 4.3.1) to globally improve the curve network and mold alignments. Similar to the least squares optimization presented in Chapter 3 the optimization minimizes divergence, kink angles, and the deviation of the paneling solution from the reference surface. Such a least-squares optimization alone, however, does not ensure that the user-specified thresholds on kink angles and divergence are met, nor can it be used to determine a mold depot or as-

signments. The core challenge and main contribution of our algorithm is to find a mold depot and an assignment function that minimize cost and at the same time meet the specified thresholds. Minimizing cost in geometric terms means approximating the design with as many simple and repetitive elements as possible. In Section 4.3.2 we show how this difficult objective can be mapped to a generalized set cover problem, a classical problem in computer science. Figure 4.5 demonstrates the effectiveness of the set cover optimization compared to alternative techniques to enable mold reuse on an illustrative example. We present an efficient approximation algorithm that on its own provides a solution to the paneling problem. Due to the exponential complexity of possible mold-panel assignments this algorithm can only employ local registrations and therefore neither supports globally coupled continuous registration nor optimization of the curve network to allow deviations from the reference surface. Since the algorithm described in Section 4.3.2 enables solving for all the discrete variables we call it *Discrete Optimization* in contrast to the global *Continuous Optimization* of Section 4.3.1. To combine the strengths of the discrete and continuous optimization we present an iterative scheme in Section 4.3.3 to interleave the two to obtain a powerful global paneling algorithm. We demonstrate in Figure 4.11 how this interleaved iteration significantly improves the paneling compared to only applying the algorithm described in 3.2.

4.3.1 Continuous Optimization Step

The continuous step aims at reducing the deviation to the reference surface, the divergence, and the kink angles by optimizing the continuous variables, i.e., the shape parameters of the molds, the rigid alignments of mold surfaces to segments, and the positions of the curve network samples. During this optimization, the panel-mold assignments are kept fixed.

A mold surface M_k is specified in a canonical coordinate system as a function of a set of parameters that we store in a vector \mathbf{m}_k (see Appendix B). Depending on the type of mold surface, \mathbf{m}_k contains zero (plane) to six entries (cubic). The rigid transformations T_i that align

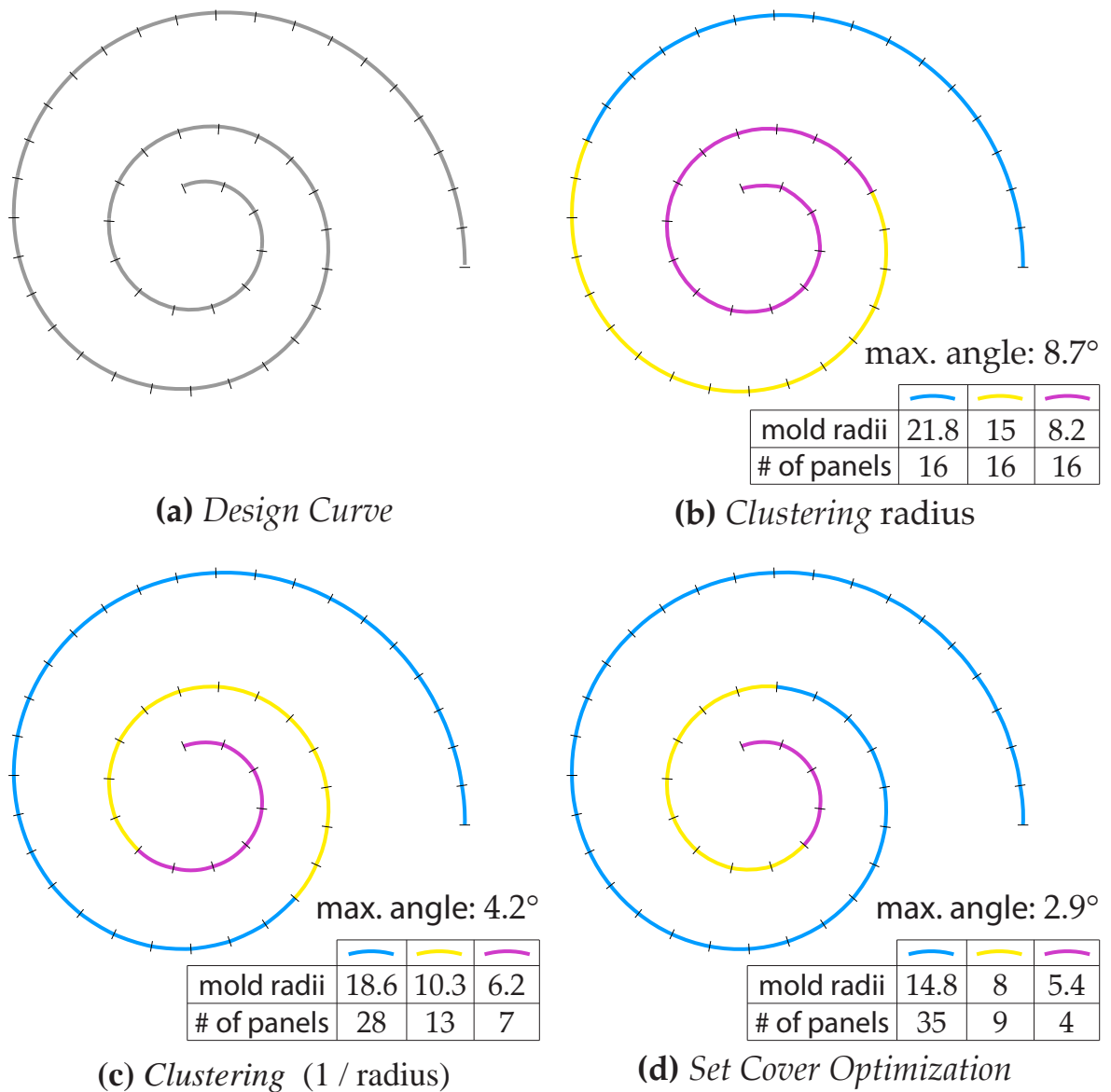


Figure 4.5 Illustrative comparison of different techniques for mold reuse. The input design curve curve shown in (a) consists of nicely aligned circle arcs with decreasing radii from 25 to 5. The method shown in (b) clusters these radii (using k -means clustering) to obtain 3 molds and assigns the best mold to each segment. The colors indicate the segments sharing the same mold. The method shown in (c) does the same, but performs a clustering of ($1/\text{radius}$) instead of clustering the radius itself, which is a much better distance approximation for cylinders as shown in Appendix B and therefore the maximal kink angle is already much lower. The method shown in (d) performs the full set cover optimization described in Section 4.3.2 and leads to an even better mold depot that enables a paneling with very low kink angles. The differences presented on this schematic example become even more prominent if more complex surfaces and/or panel types are involved.

4 Paneling Architectural Freeform Surfaces

mold surface $M_{A(i)}$ to the corresponding surface segment s_i are initialized by the the local alignments computed in the discrete optimization. The curve network \mathcal{C} is represented by samples $\mathbf{c}_l \in \mathbb{R}^3, l = 1, \dots, L$, where $L \approx 9|\mathcal{S}|$ in all our examples. We denote with $i(l)$ and $j(l)$ the indices of the two surface segments adjacent to the boundary curve sample \mathbf{c}_l . For notational brevity of the subsequent formulae, we omit the more complex adjacencies at corner points of the curve network. These are handled analogously by considering all combinations of neighbors.

The explicit representation of the curve network enables direct control of the panel intersection curves and allows formulating objective functions for inter-panel position and tangent continuity using simple closest-points projections. Effectively, the curve network serves as a “glue” that binds the panels to the surface at the curve samples. Since the manufacturing process mandates that the panel surfaces are smooth and exhibit a low variation of curvature, we found that additional samples in the interior of the segments are not required.

Surface Fitting. The deviation of the curve network from the underlying reference surface F is measured as

$$E_{\text{fit}} = \sum_{l=1}^L \|\mathbf{c}_l - \mathbf{f}_l\|^2, \quad (4.2)$$

where \mathbf{f}_l is the closest point on F from \mathbf{c}_l .

Divergence. We measure the divergence indirectly using the distances between the curve network samples and the closest points of the adjacent aligned mold surfaces as

$$E_{\text{div}} = \sum_{l=1}^L \|\mathbf{c}_l - \mathbf{x}_{i(l)}\|^2 + \|\mathbf{c}_l - \mathbf{x}_{j(l)}\|^2, \quad (4.3)$$

where $\mathbf{x}_{i(l)}$ is the point closest to \mathbf{c}_l on the aligned mold surface $M_{i(l)}^*$. The point $\mathbf{x}_{j(l)}$ is defined analogously. Minimizing E_{div} ensures that neighboring panels fit together nicely without leaving undesirable gaps and that the curve network stays spatially close to the aligned mold surfaces.

Kink Angles. Tangent continuity is achieved by minimizing the kink angle energy

$$E_{\text{kink}} = \sum_{l=1}^L \|\mathbf{n}(\mathbf{x}_{i(l)}) - \mathbf{n}(\mathbf{x}_{j(l)})\|^2, \quad (4.4)$$

where $\mathbf{n}(\mathbf{x}_{i(l)})$ is the normal vector of the aligned mold $M_{i(l)}^*$ at $\mathbf{x}_{i(l)}$, and analogously for $\mathbf{n}(\mathbf{x}_{j(l)})$ (Figure 4.3).

Curve Fairness. The curve network is coupled to the surface through the surface fitting energy, but can move freely along the surface. To maintain the design intent encoded in the initial curve layout, we need to prevent strong undulations and large tangential motions of the boundary curves. Fortunately, the explicit representation of the curve network allows us to directly control its motion. We avoid tangential drift entirely by restricting the displacements of the curve network to the normal direction of the underlying reference surface. This simplification has the additional benefit of reducing the number of optimization variables for the curve network to one third and thus significantly improves performance. The shape of the curves is controlled using an additional fairness term on the curve deformation. Let $\mathbf{c}_l = \mathbf{c}'_l + d_l \mathbf{n}'_l$, where \mathbf{c}'_l is the initial position of the curve network sample on the reference surface, \mathbf{n}'_l is the corresponding surface normal, and d_l is the displacement magnitude. A fairness term (see also Section 2.1) can then be defined directly on the normal displacements as

$$E_{\text{fair}} = \sum_{(j_1, j_2) \in \mathcal{C}} (d_{j_1} - d_{j_2})^2, \quad (4.5)$$

where (j_1, j_2) is an index pair denoting an edge of the polygonal representation of the curve network \mathcal{C} .

Panel Centering. With the exception of plane and cylinder molds, we need an additional energy term to facilitate mold reuse. Panels that are fabricated from the same mold should cover a similar region of the mold surface to reduce wastage in fabrication. For this purpose we add a centering energy that minimizes tangential drifting of the mold center away from the segment center:

$$E_{\text{cen}} = \sum_{i=1}^n \|\mathbf{b}_i - \mathbf{p}_i\|^2. \quad (4.6)$$

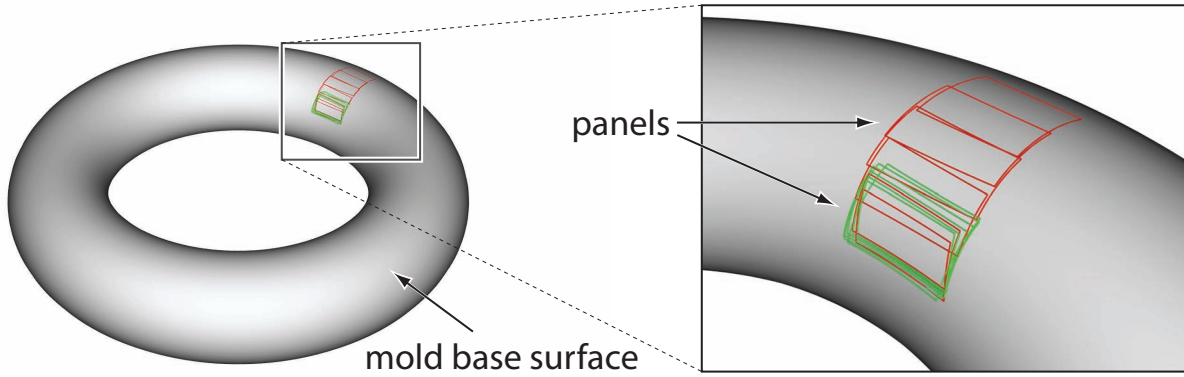


Figure 4.6 A low weight for E_{cen} may result in scattered molds (red) over a base primitive leading to wastage and increased cost. A higher weight results in tightly clustered molds (green) enabling cost-effective production of a significantly smaller mold.

Here \mathbf{b}_i is an approximation of the barycenter of segment s_i computed as the average of all adjacent curve segment samples, and \mathbf{p}_i is the projection of \mathbf{b}_i onto the mold-surface normal at the center of the aligned mold M_i^* . For torus molds we include an additional variable that positions the center along the defining circle (see Appendix B). Figure 4.6 illustrates the effect of the centering energy.

Global Optimization. The above continuous energy terms are combined into a global energy

$$E = \alpha_{\text{fit}}E_{\text{fit}} + \alpha_{\text{div}}E_{\text{div}} + \alpha_{\text{kink}}E_{\text{kink}} + \alpha_{\text{fair}}E_{\text{fair}} + \alpha_{\text{cen}}E_{\text{cen}}, \quad (4.7)$$

where the weights α allow additional control of the optimization. The unknown variables in E that we solve for are: the mold parameters \mathbf{m}_k , the rigid transformations T_i , and the positions of the curve network samples \mathbf{c}_j . For the T_i we use the formulation based on instantaneous velocities that ensures rigidity of the transformation [PW01]. The global energy E is minimized using a Gauss-Newton solver. See Section 2.2 for an extensive discussion on nonlinear least squares problems and solvers.

Parameters. As detailed in Section 4.3.2, our system allows the user to provide thresholds ϵ for divergence and δ for kink angles. To balance the corresponding optimization objectives, we set $\alpha_{\text{kink}} = (\epsilon/\delta)^2\alpha_{\text{div}}$ in Equation 4.7. This ensures that a divergence

of ϵ receives the same penalty as a kink angle of δ . The free parameters have an intuitive meaning and offer direct control of the paneling quality. The specific values for these parameters depend on the user preferences. Figure 4.4 illustrates the effect of α_{fit} to control the deviation from the reference surface using values of 0.0001 and 1000, respectively. Figure 4.6 demonstrates the influence of α_{cen} . For all other examples we use the same set of parameters: $\alpha_{\text{fit}} = 1$, $\alpha_{\text{div}} = 1000$, $\alpha_{\text{fair}} = 1$, $\alpha_{\text{cen}} = 10$.

4.3.2 Discrete Optimization Step

The discrete optimization finds a mold depot and a corresponding panel-mold assignment function that minimize cost while respecting the specified divergence and kink angle thresholds. During the discrete optimization, the curve network is fixed. The essential geometric information required in this step is stored in the curve network samples and corresponding normal vectors that we obtain by averaging for each curve network sample the normals of all neighboring panels of the current solution. We therefore represent each surface segment by the associated curve network samples and normals, and use these to evaluate divergence and kink angles. This segment representation allows an efficient fitting of molds to segments based on local registration independent of the neighboring panels, using local versions of the divergence, kink angle, and centering energies (Equations 4.3, 4.4, 4.6).

To enrich the mold depot, we first create new candidate molds by locally aligning each of the available panel types to each of the segments. These $t|\mathcal{S}|$ new molds, where t is the number of panel types, together with the existing mold depot of the current solution form the set \mathcal{M}' of candidate molds. The algorithm optimizes for a new mold depot as a subset $\mathcal{M} \subseteq \mathcal{M}'$ of all candidate molds that enables paneling at minimal cost, while satisfying the current thresholds on divergence and kink angle.

Set Cover. Assume that we have determined for each mold $M_k \in \mathcal{M}'$ the set $\mathcal{S}_k = \{s_{k_1}, \dots, s_{k_l}\}$ of surface segments that can be approximated by M_k within the prescribed tolerances. We can com-

4 Paneling Architectural Freeform Surfaces

pute the potential fabrication costs attributed to set \mathcal{S}_k as $c(\mathcal{S}_k) = c(M_k) + |\mathcal{S}_k| c(M_k, P_*)$, i.e., the cost of the mold plus for all assigned panels the cost $c(M_k, P_*)$ of producing a panel with mold M_k .

Finding the optimal mold depot and mold-segment assignments now amounts to covering the set of segments $\mathcal{S} = \{s_1, \dots, s_n\}$ with sets \mathcal{S}_k of minimal total cost. This optimization is reminiscent of the classical *weighted set cover* problem [Joh74], where the weights correspond to production costs. While this problem is known to be NP-hard, a polynomial-time approximation strategy can be used to find an approximate solution whose cost is guaranteed to be within $\log n$ times the cost of the optimal solution. It has been shown that $\log n$ is the best possible approximation ratio of any polynomial-time algorithm [Fei98].

What distinguishes our setting from the classical weighted set cover problem is that each segment is eventually assigned to only one set. Thus our weights (costs) depend on which segment is assigned to which set: Once a mold has been chosen, all segments covered by the corresponding set have to be removed from the sets of the remaining candidate molds. Nevertheless, as shown in Appendix A.2, the proof for the $\log n$ approximation ratio of the polynomial-time weighted set cover algorithm directly translates to our more general setup.

Algorithm. Let $\mathcal{S}' \subseteq \mathcal{S}$ be the set of all uncovered segments at any point of the algorithm. We define the *efficiency* of a set \mathcal{S}_k relative to \mathcal{S}' as the function $\phi(\mathcal{S}_k, \mathcal{S}') = |\mathcal{S}_k|/c(\mathcal{S}_k)$, where both the size of the set \mathcal{S}_k and its cost are adapted to only consider elements in \mathcal{S}' . Efficiency measures the number of panels for segments in \mathcal{S}' that can be produced by mold M_k relative to the corresponding cost. Let σ be the unknown collection of covering sets and σ' the collection of all sets \mathcal{S}_k that have not yet been chosen for σ .

Our algorithm to determine the covering sets starts with an empty collection σ . Thus the collection σ' contains the sets \mathcal{S}_k of all candidate molds in \mathcal{M}' and the set \mathcal{S}' contains all segments that can be covered with the sets in σ' . We then successively add the set \mathcal{S}_i with highest efficiency to the current solution. The segments of \mathcal{S}_i are removed from

\mathcal{S}' , the efficiency of all remaining sets is updated, and the algorithm is iterated until all segments are covered.

```

 $\sigma \leftarrow \emptyset, \sigma' \leftarrow \{\mathcal{S}_1, \dots, \mathcal{S}_{|\mathcal{M}'|}\}, \mathcal{S}' \leftarrow \mathcal{S}_1 \cup \dots \cup \mathcal{S}_{|\mathcal{M}'|}$ 
while  $\mathcal{S}' \neq \emptyset$ 
  eval.  $\phi(\mathcal{S}_k, \mathcal{S}') \forall \mathcal{S}_k \in \sigma'$            update efficiencies
   $\mathcal{S}_i \leftarrow \arg \max_{\mathcal{S}_k \in \sigma'} \phi(\mathcal{S}_k, \mathcal{S}')$    set with max. efficiency
   $\sigma \leftarrow \sigma \cup \{\mathcal{S}_i\}$                    add to covering sets
   $\mathcal{S}' \leftarrow \mathcal{S}' - \mathcal{S}_i, \sigma' \leftarrow \sigma' - \{\mathcal{S}_i\}$  remove covered segments
end

```

The covering sets in the solution σ define the mold depot \mathcal{M} . For all segments that cannot be covered by any of the initial sets \mathcal{S}_k we add and assign the best-fit cubic mold. Once the mold depot has been selected, a gather step re-computes the panel-mold assignment function by selecting for each segment the cheapest valid mold from the depot \mathcal{M} . In case a segment is covered by multiple molds of the same cost, we pick the one with the smallest maximal distance to the curve network.

Sets Initialization. A critical bottleneck in the set cover optimization is the estimation of the initial sets \mathcal{S}_k . To avoid the exhaustive computation of aligning every mold with every segment using nonlinear registration, we implement a pruning step that discards mold-segment pairs based on a conservative estimate of the corresponding registration distance. For this purpose, we introduce a 6D metric space that facilitates efficient approximate distance computations between molds and surface segments without the need for an explicit alignment.

We first select for each segment s_i its least-squares optimal cubic mold $C_i \in \mathcal{M}'$ as the segment's representative. The cubic mold C_i provides the best local approximation possible for this segment in the current configuration. To estimate whether a given mold surface $M_k \in \mathcal{M}'$ is a suitable candidate for segment s_i , we compute an approximate alignment distance between M_k and C_i . By mapping both patches to points in a 6D space as described in Appendix B, this computation amounts to simple Euclidean distance evaluations. Finding potential molds to

4 Paneling Architectural Freeform Surfaces

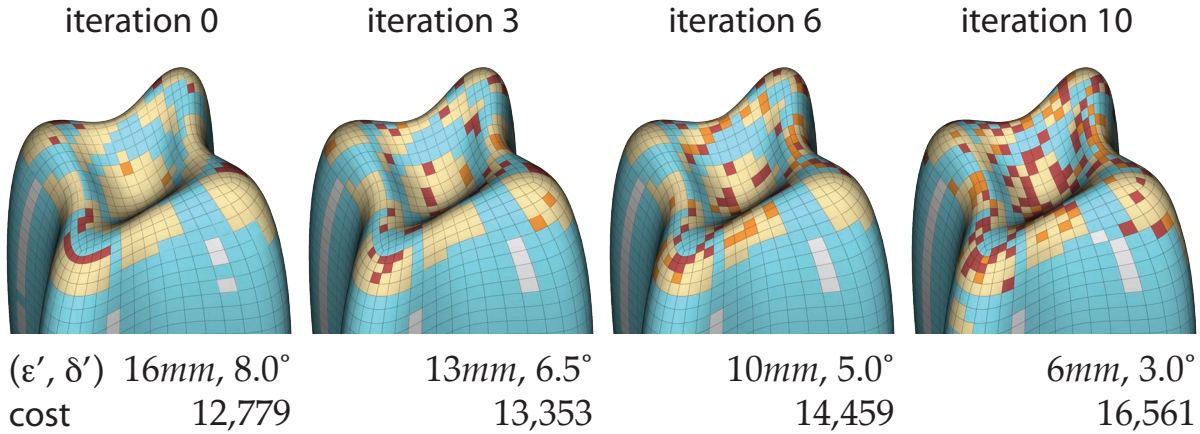


Figure 4.7 Progressive panel assignments for different iterations (decreasing thresholds) of the paneling algorithm (Lilium tower).

fit a given segment then only requires a range query in the 6D space that is performed efficiently using spatial data structures such as a kd-tree [AM93]. This pruning step typically leads to a reduction in explicit mold-segment registrations to about 1% – 5%. For this substantially smaller set, we then perform the full nonlinear registration as described above to build the sets \mathcal{S}_k .

4.3.3 Interleaved Iteration

Our algorithm iteratively executes the discrete and continuous optimization steps in an interleaved fashion. The exponential number of possible panel-mold assignments mandates the use of local registration methods in the discrete optimization. As a result, the set cover algorithm produces a mold depot and assignment function that is too costly for the given thresholds. Therefore, we start the iterations with thresholds well above the specified target values ϵ for divergence and δ for kink angles. Thresholds are then successively reduced after each iteration until the target values are reached.

We use the following fixed scheduling scheme for all our examples: We start the optimization with $\epsilon' = \epsilon + 10mm$ and $\delta' = \delta + 5^\circ$ and apply 10 iterations of alternating discrete/continuous optimization steps, reducing the thresholds in each iteration by 1mm and 0.5°.

The algorithm starts with a mold depot consisting of a single plane. We perform an initialization step that first determines all panels that do not meet the target thresholds (ϵ, δ) . These panels are replaced by locally fitting a separate mold of the cheapest type that satisfies the thresholds (ϵ', δ') . Subsequently, we perform a continuous optimization step. This initialization is applied before every discrete optimization to reverse unsuccessful assignments of the set cover and avoid a premature fixing of panel-mold assignments.

The inner loop of the optimization consists of the following steps:

- discrete optimization
- continuous optimization
- re-initialization
- reduce ϵ', δ'

At the end of the optimization we apply a discrete step using the target thresholds to generate the final mold depot and assignment function. Figure 4.7 illustrates the iterations of the paneling algorithm. Panels for which the target thresholds cannot be met mandate the fabrication of custom freeform molds. An important benefit of our approach is that the global discrete/continuous optimization leads to a small number of these custom molds (see also Figure 4.11).

4.4 Evaluation and Discussion

Means of Control. We allow the designer to explore the space of paneling solutions using two main modes of control: (i) specifying the quality using thresholds for divergence and kink angles (see Figure 4.8), and (ii) allowing the paneling solution to deviate from the reference surface in order to achieve a more cost-effective solution while maintaining the original design intent (see Figure 4.4). Due to our choice of α_{fit} (Section 4.3.1) the maximal deviation from the reference surface is within 10-20cm for all our results (average panel size $4m^2$). Unless stated otherwise, we use the cost set of glass panels as denoted in Figure 4.9 and the target thresholds $(\epsilon, \delta) = (6mm, 3^\circ)$.

4 Paneling Architectural Freeform Surfaces

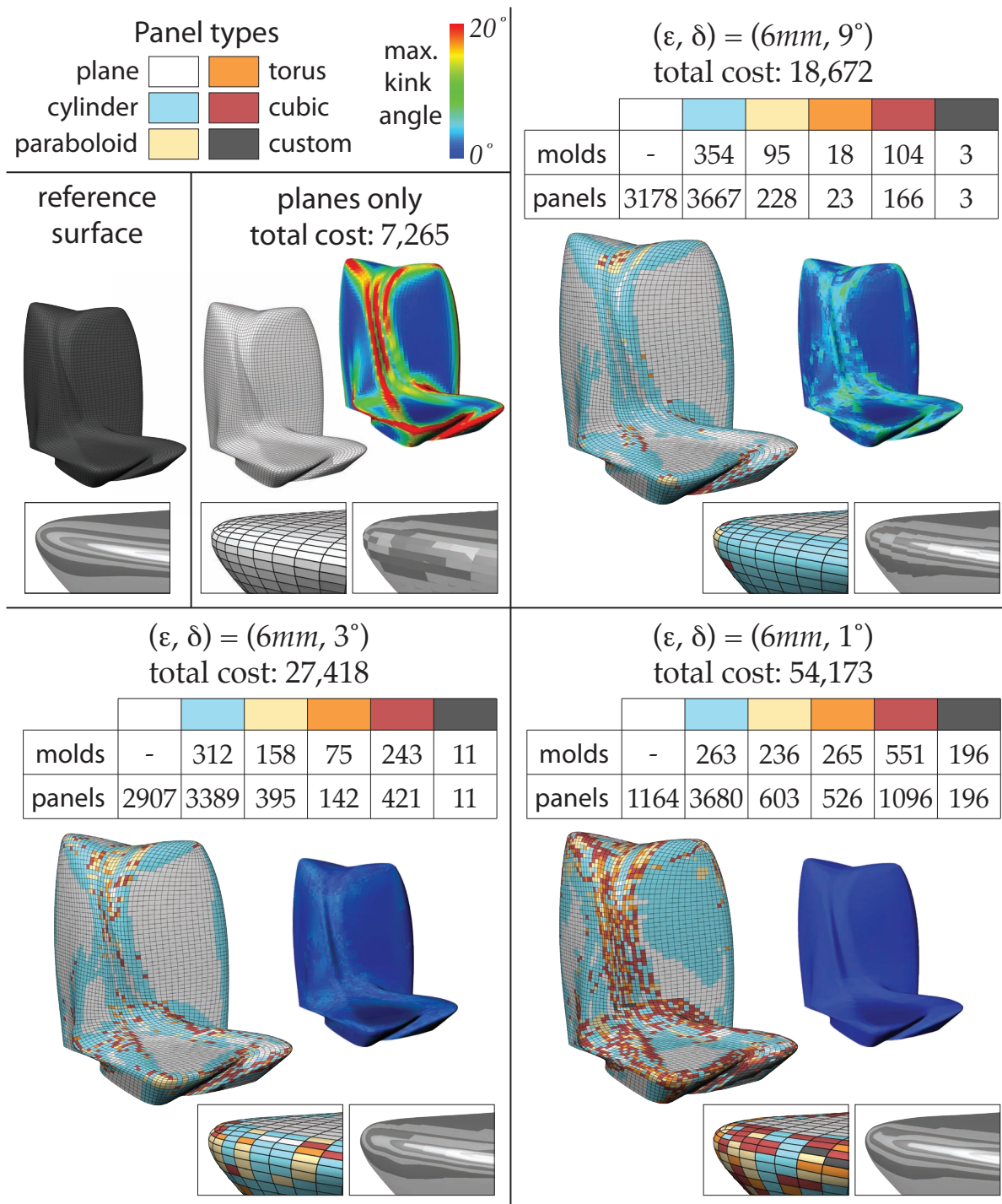


Figure 4.8 Paneling results with varying kink angle thresholds δ and fixed divergence thresholds $\epsilon = 6mm$ for the design of the National Holding Headquarters. The images in the top left corner show a solution using only planar panels of which 3,796 do not meet the prescribed divergence threshold. The zooms show reflection lines to illustrate inter-panel continuity which successively improves with lower kink angle thresholds.

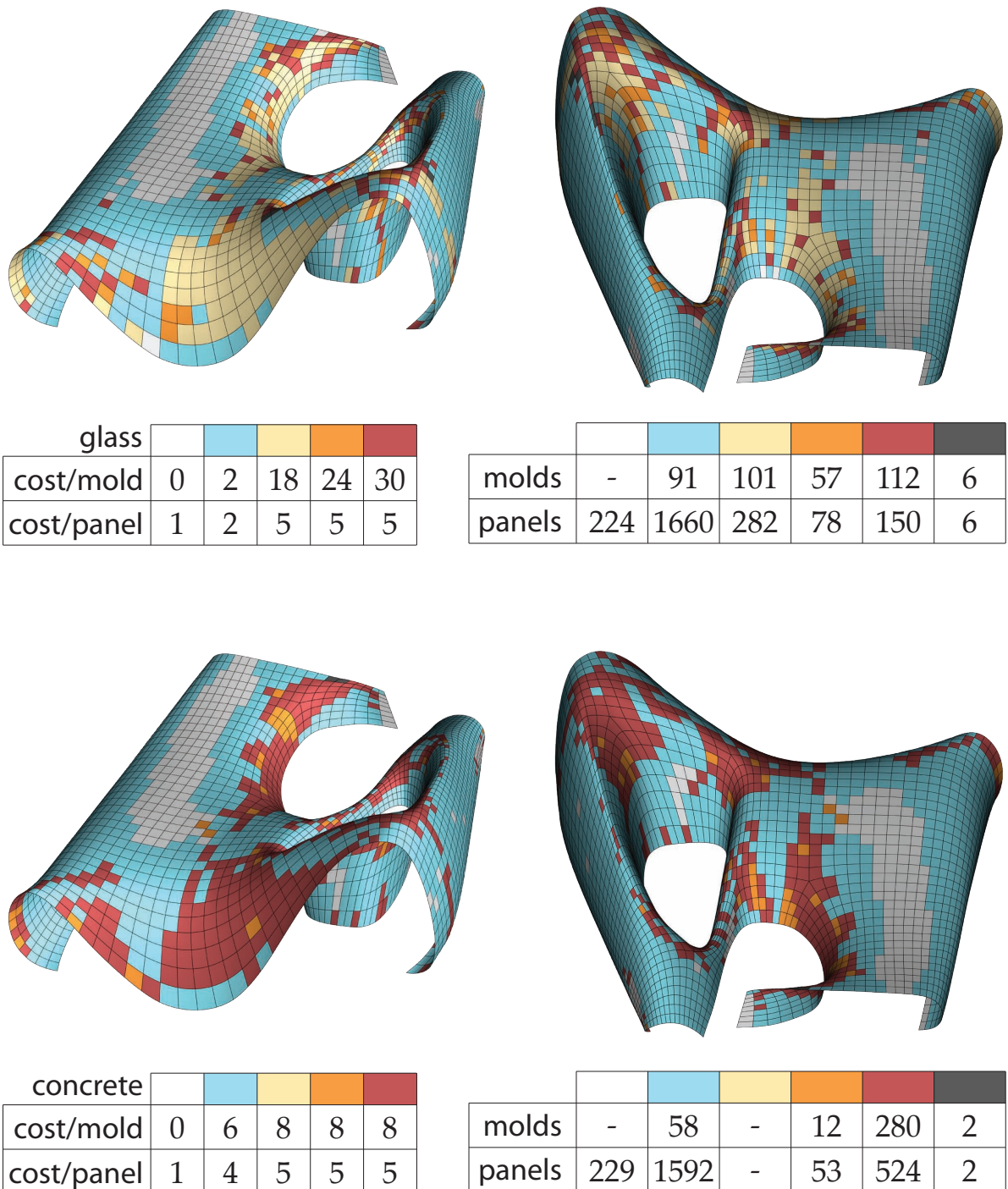


Figure 4.9 *Glass vs. concrete. Different relative costs for mold fabrication and panel production for two different materials affect the distribution of panel types. For glass, costs for producing double curved molds are significantly higher than for concrete, resulting in a solution with more cylindrical panels. (Zaha Hadid Architects, interior skin of Heydar Aliyev Merkezi Cultural Center, Baku.)*

4 Paneling Architectural Freeform Surfaces

Case Studies. Paneling solutions strongly depend on the design (reference surface and curve network), the choice of material, and the manufacturing technology. We explore the implications of these design decisions on various contemporary architectural freeform projects from leading architects in Figures 4.9, 4.12, and 4.10. Figure 4.13 compares our paneling solution to the original, perfectly smooth reference design. Figure 4.11 demonstrates the effectiveness of the core components of our paneling algorithm. The most complex case study is the Dongdaemun Design Plaza and Park project (Figure 4.11) consisting of 8,385 panels. For this example, one step of discrete optimization takes on average 30 minutes, while one step of continuous optimization requires about 10 minutes, leading to a total computation time of 430 minutes. All computations are performed on a 3GHz Intel[®] Core[™]2 Duo with 3Gb of memory.

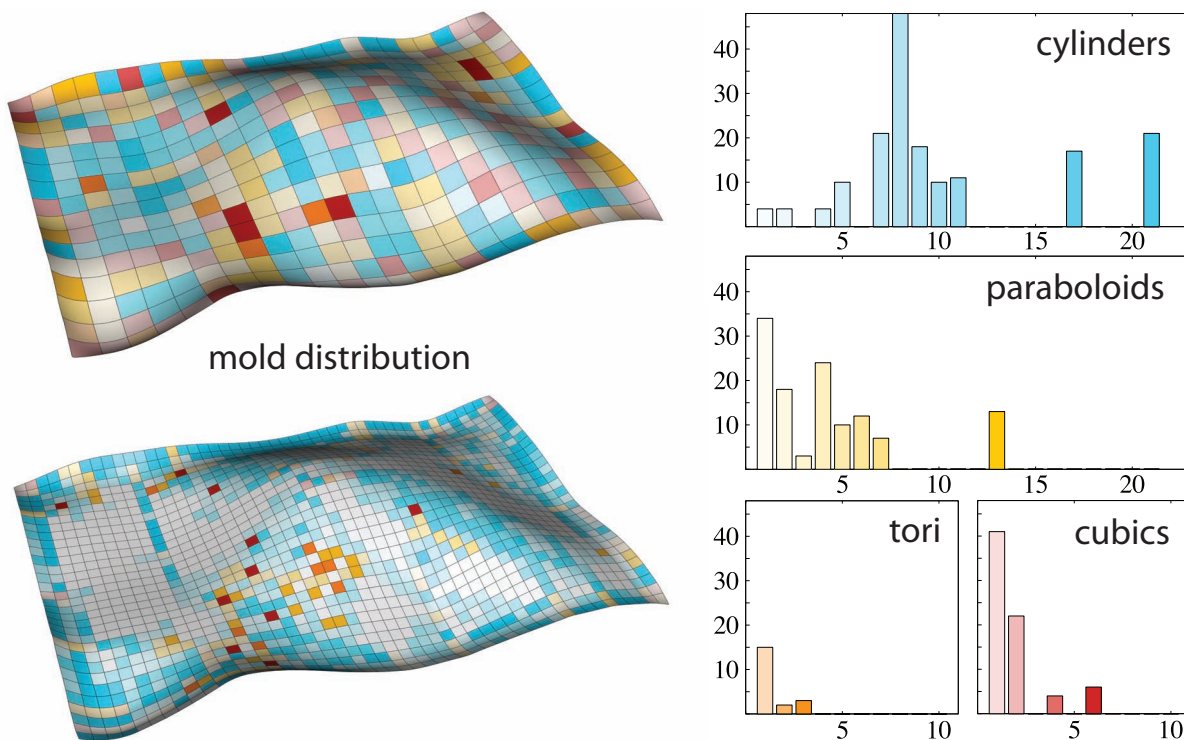


Figure 4.10 *Distribution of panel types and mold reuse for two different refinement levels of the curve network. The histograms on the right illustrate mold reuse for the top image, where the x-axis denotes how many panels can be produced by a mold and the y-axis indicates the number of panels corresponding to these molds. (Mario Bellini Architects, Rudy Ricciotti, Museum of Islamic Arts at Louvre Museum, Paris.)*

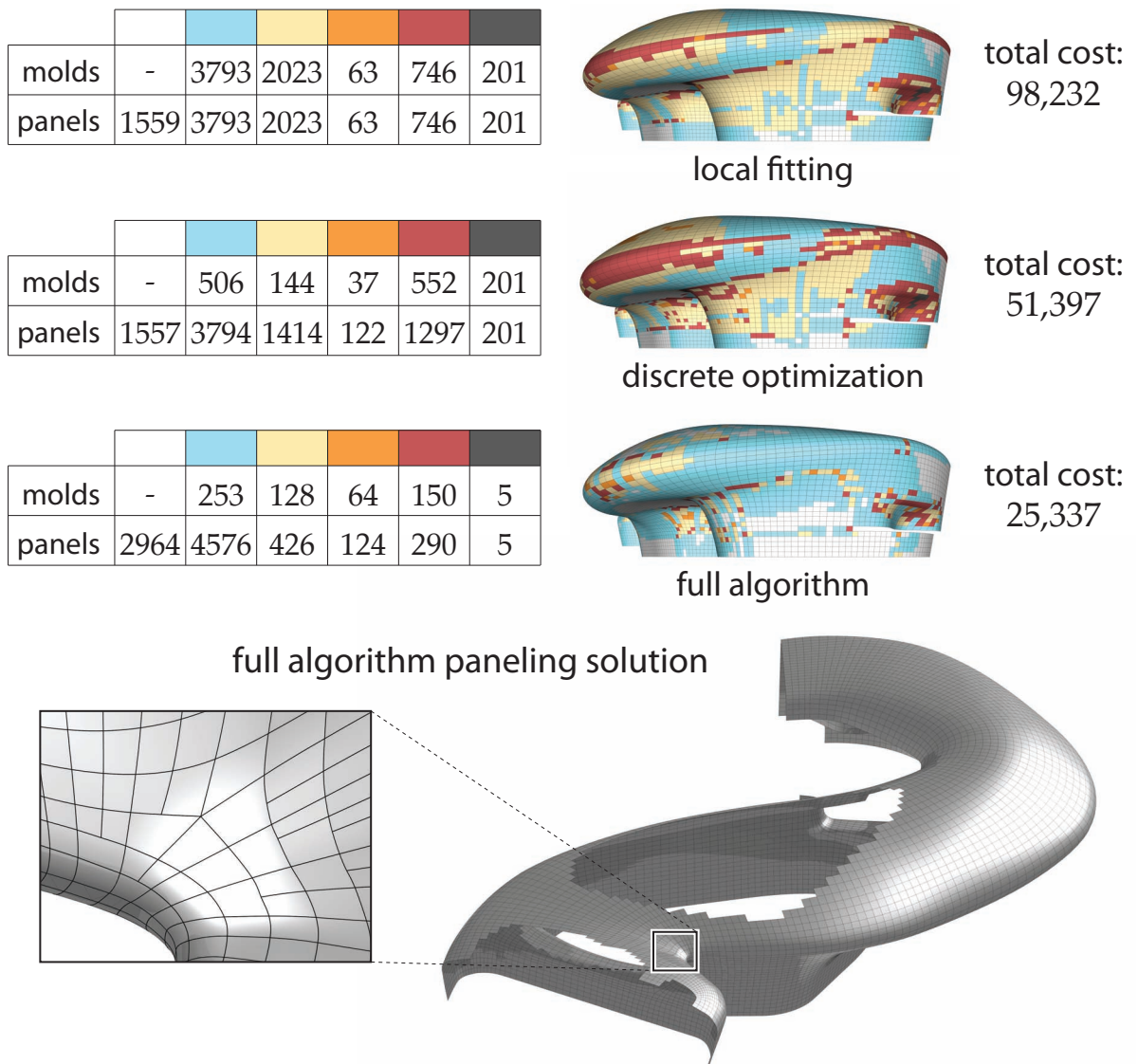
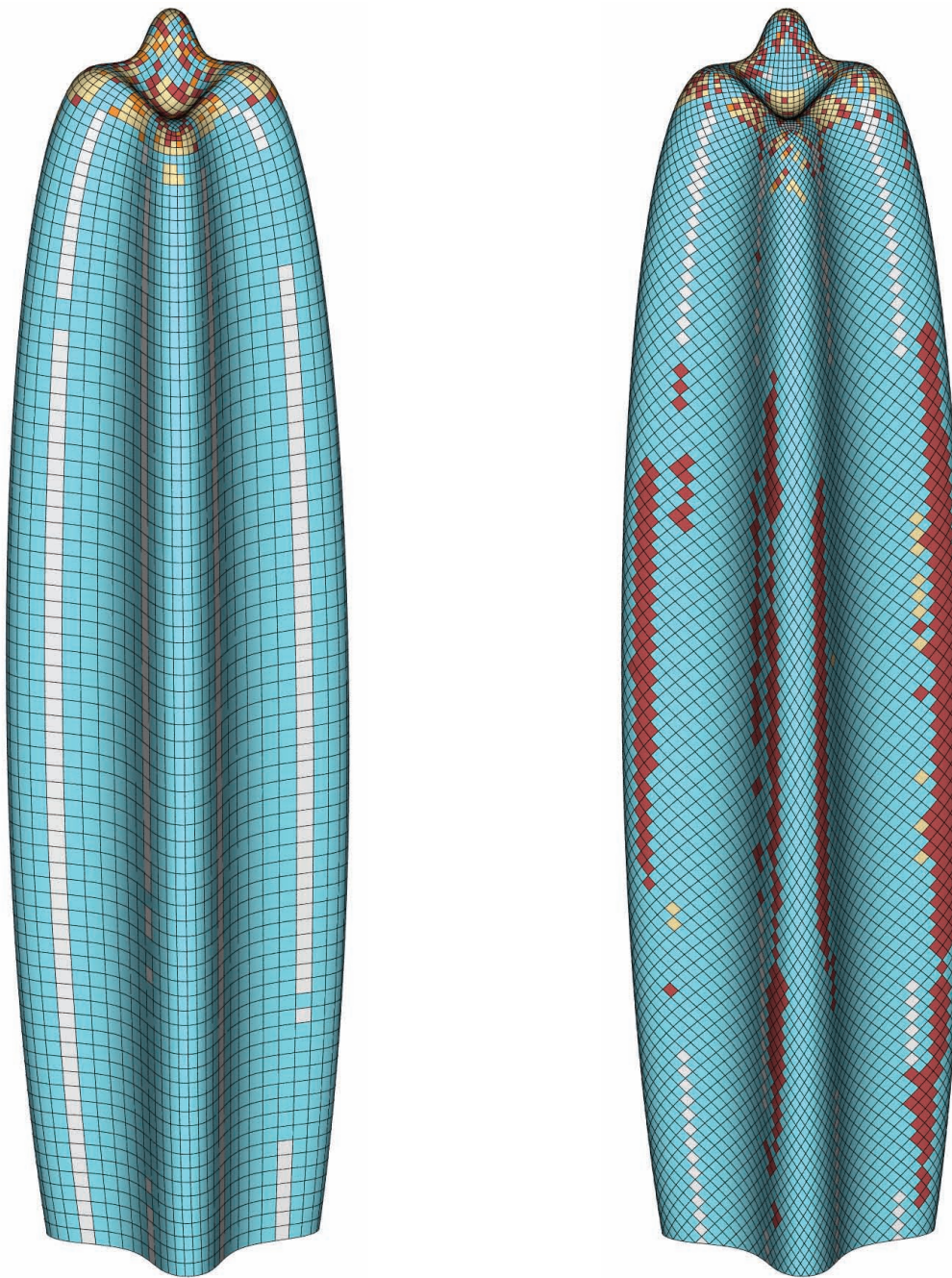


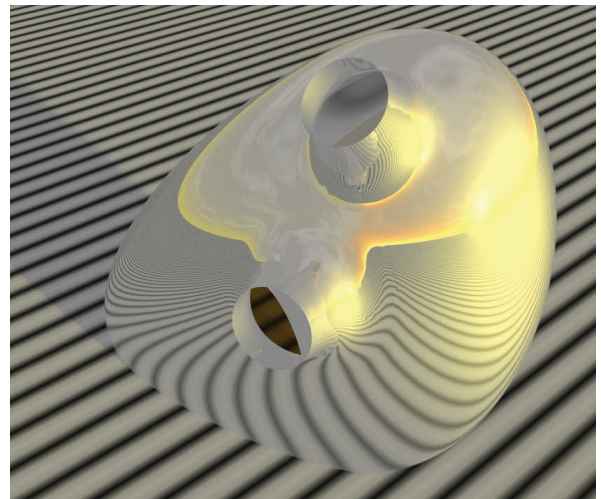
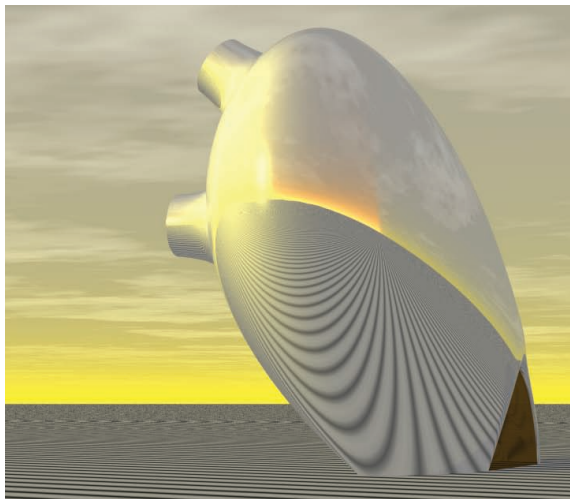
Figure 4.11 Comparison of different methods for the same quality thresholds. State-of-the-art commercial tools only support a greedy panel assignment based on local fitting (top). Just one single application of our discrete optimization greatly reduces cost without loss in surface quality (middle). The full paneling algorithm interleaving discrete optimization with global continuous registration produces a high quality paneling (bottom). This solution contains 90% single curved panels and a very small number of custom molds, leading to a significantly reduced cost compared to greedy and local methods. The zoom on the right shows that our algorithm supports arbitrary curve network topology, including t-junctions. (Zaha Hadid Architects, Dongdaemun Design Plaza and Park, Seoul.)

4 Paneling Architectural Freeform Surfaces

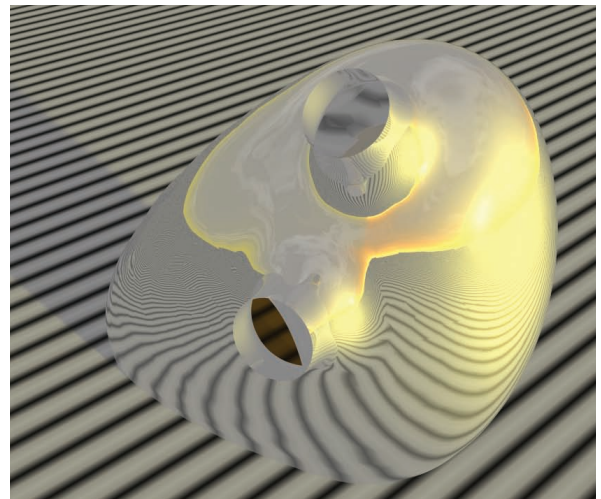
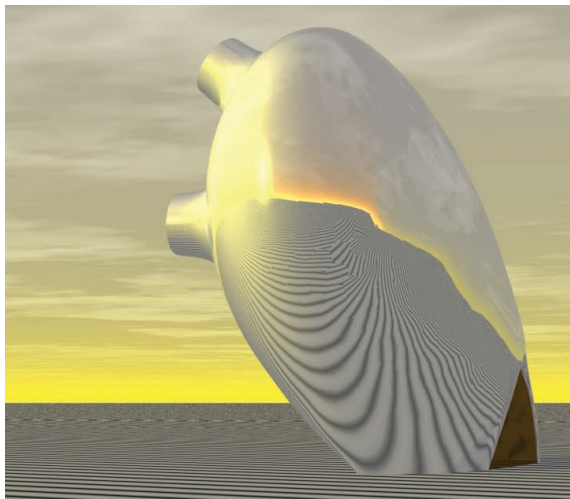


-	115	107	38	65	-	molds	-	2005	133	21	139	-
442	4523	264	59	88	-	panels	414	8569	417	33	1351	-

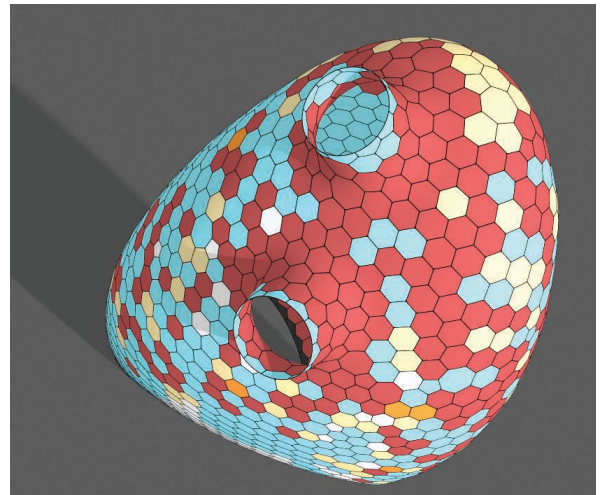
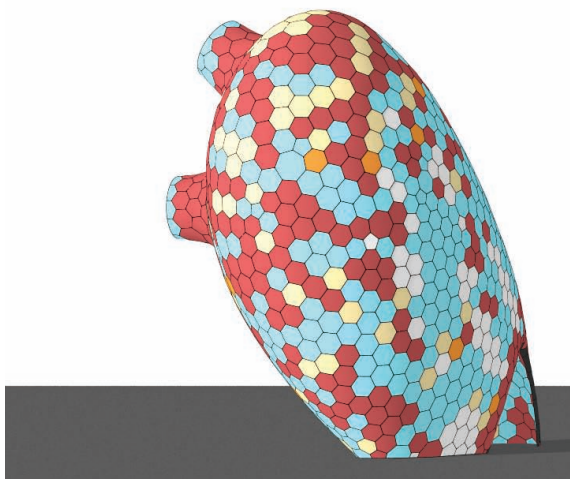
Figure 4.12 Paneling solutions for different curve networks on the Lilium tower. Significant changes in the curve network layout result in different paneling solutions. The left curve network, being closer to a conjugate network [PLW⁺07], is better suited for rationalization with planar and single curved panels.



reference surface



panelized surface



mold types

Figure 4.13 The kink angle constraints of our paneling algorithm ensure a smooth approximation of the design surface. The paneling solution shown in the middle and bottom row was produced using a kink angle threshold of 1° . (Texxus www.texxus.com, Skipper Library Design Study.)

4 Paneling Architectural Freeform Surfaces

Extensibility. Our framework for paneling architectural freeform surfaces takes important practical requirements into consideration. It is extensible in various ways to account for further specific needs and constraints.

In practice, the quality requirements on the paneling solution might vary for different regions of the design. In Figures 4.14-4.17 we show an extension that uses adaptive kink angle thresholds. Instead of a single global threshold, a separate value is specified for every curve network sample point. In the examples shown, these local thresholds are determined by a visibility function shown in Figure 4.14 that computes the visibility for every point on the design surface, assuming the surface is viewed from two different access paths. Figures 4.15-4.17 demonstrate how this adaptive quality control directs the use of expensive panels towards regions where they are needed most, leading to an improved paneling quality at similar or lower costs compared to globally specifying thresholds. The same technique can be used to adaptively control the divergence or the deviation from the original design surface.

The basic paneling algorithm described above assumes that the input reference surface is smooth everywhere. Sharp crease lines, however, are used in architectural freeform designs to highlight strong characteristic features and to enhance the visual appeal of a design. To support sharp features the algorithm can easily be adapted such that kink angle thresholds are not applied across the curves describing sharp features and the tangent continuity between two panels on opposite sides of a sharp feature is not optimized. Figure 4.18 demonstrates how this extension enables paneling solutions with sharp crease lines.

Further possible extensions include restrictions on mold reuse (restricting the use of the same mold to only parts of the surface determined by the assembling schedule; restricting the number of panels to be produced by the same mold) or explicit specification of panel types allowed for selected segments. Currently our system supports five important panel types. Incorporating further types is not difficult, but may require some thought to integrate them into a metric space for fast panel-segment distance computations (see Appendix B). Since all our architectural designs are segmented rather uniformly, our current

cost model does not take panel sizes into account. Panel sizes could be easily included, however, by specifying a cost per square meter for each panel type. In Appendix A.2 we show how to extend the discrete set cover optimization to this more general cost model and proof that the logarithmic approximation ratio still holds.

Limitations. Although we achieve reasonable computation times for our examples, very large models ($> 50k$ segments) may only be effectively handled by splitting them into several parts that are treated separately. Even if splitting of a large model is not necessary, the performance may not allow the user to fully exploit the potential of the available control mechanisms (thresholds, allowed deviation from reference surface).

In our current problem formulation we expect an initial curve network as part of the design. Certain materials like gypsum are suitable for producing freeform surfaces without visible seams. Hence the aesthetics of the curve network plays a minor role. While our framework could be extended to allow stronger movement of the curve network, the additional flexibility of freely optimizing over the curve layout is currently not exploited in our algorithm.

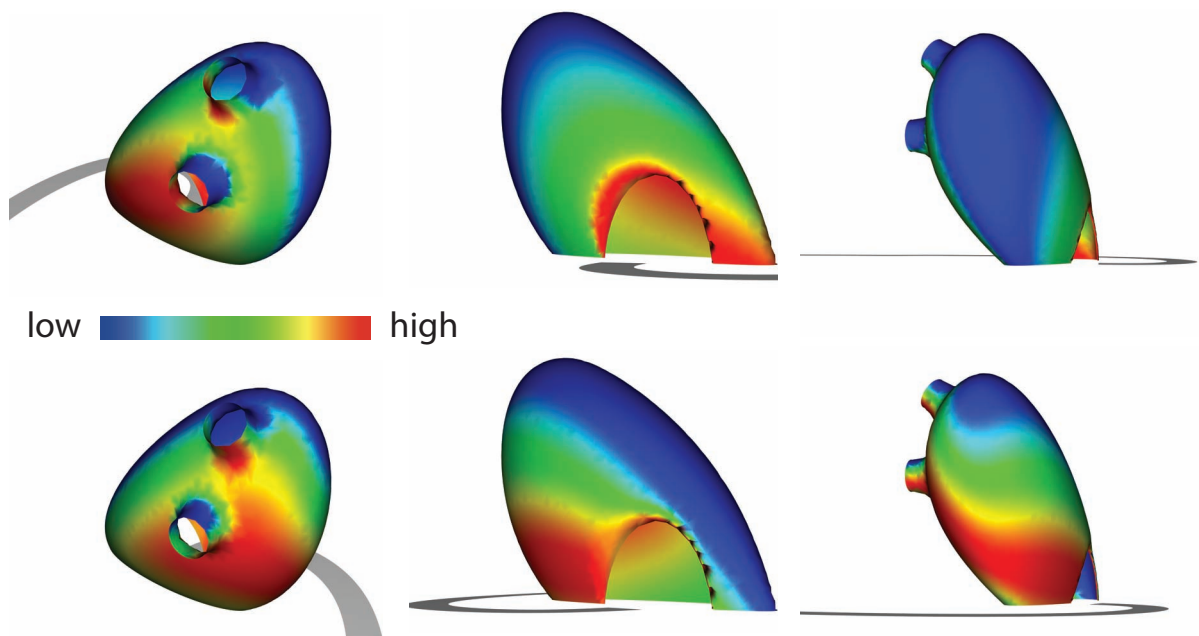
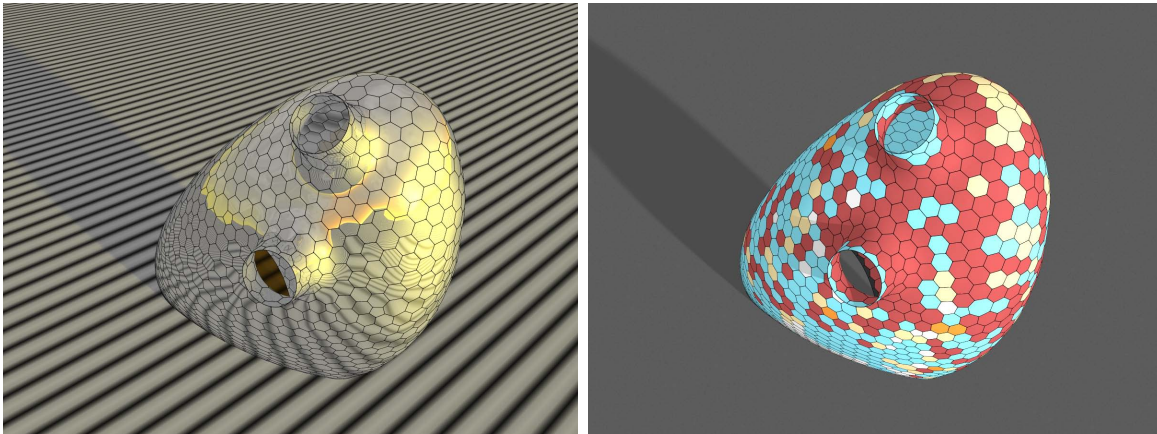
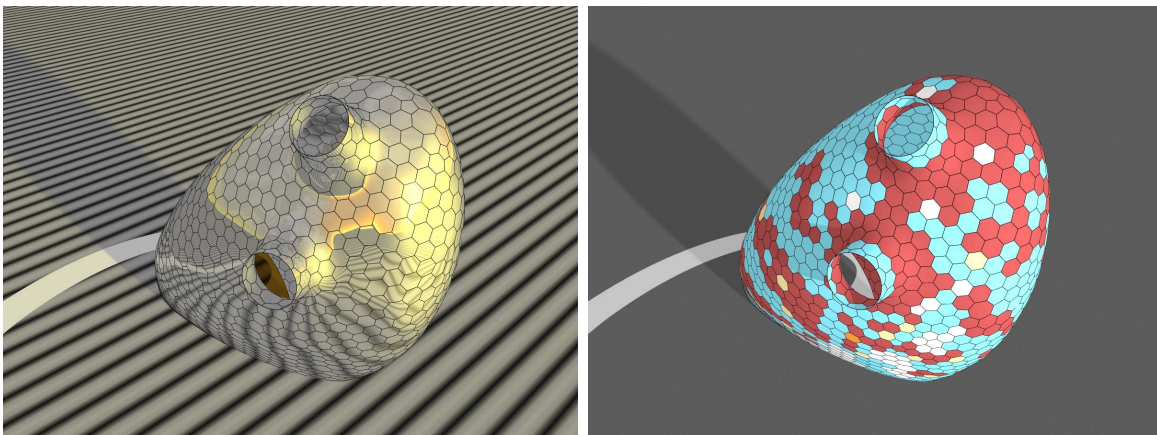


Figure 4.14 Spatially adaptive kink angle weights computed based on visibility from path 1 (top row) and path 2 (bottom row). These weights are used for paneling solutions as shown in Figures 4.15-4.17 (b) and (c).

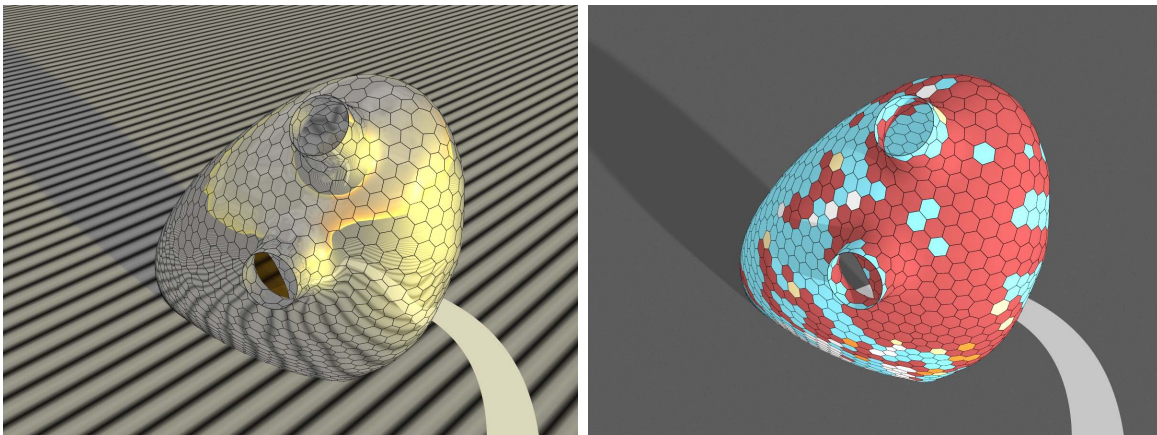
4 Paneling Architectural Freeform Surfaces



(a) Paneling solution with kink angle thresholds specified globally over the surface.

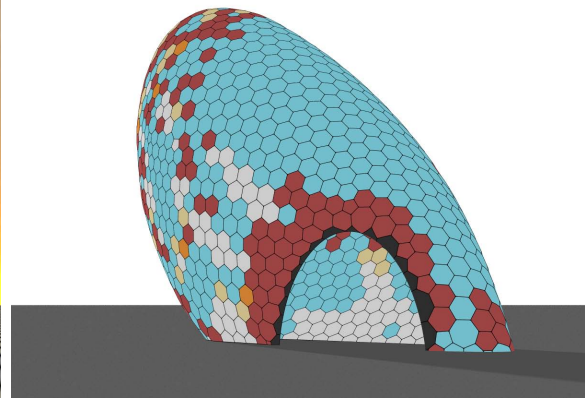
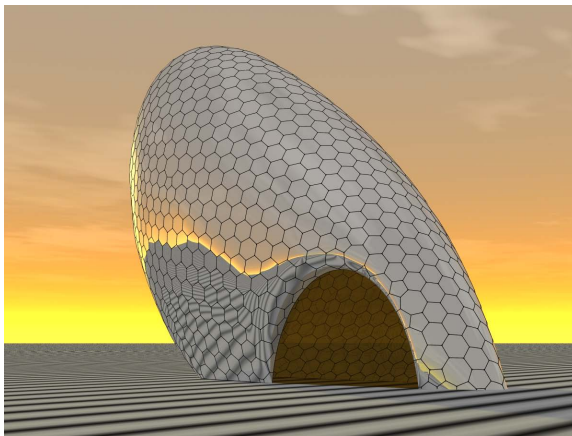


(b) Paneling solution with spatially adaptive kink angle thresholds.

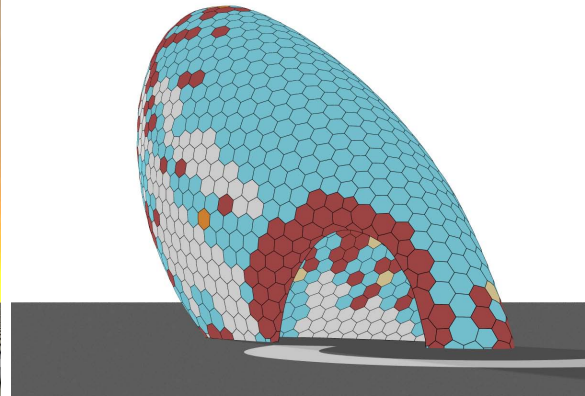
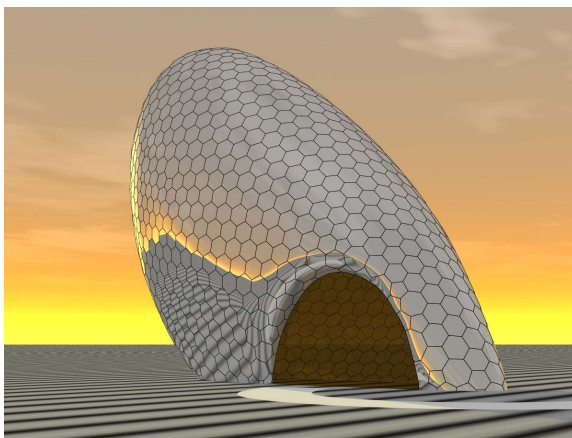


(c) Paneling solution with another set of spatially adaptive kink angle thresholds.

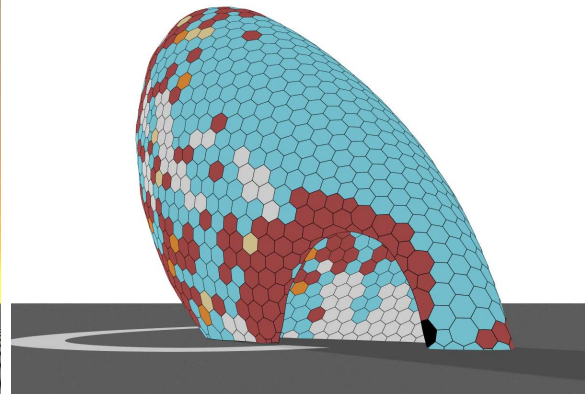
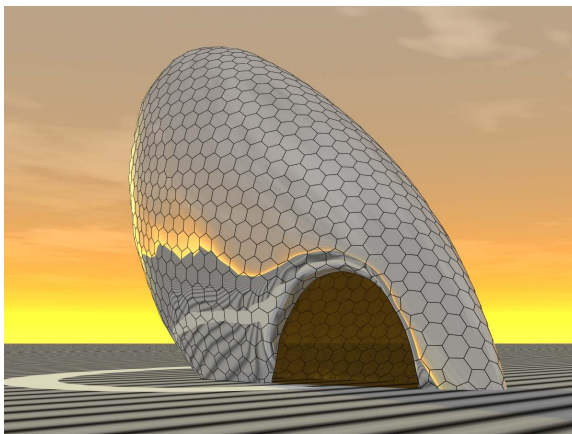
Figure 4.15 Effect of global vs adaptive kink angle constraints for the Skipper Library. Paneling solutions using a global kink angle specification (a) and using adaptive kink angle thresholds (b, c) computed based on the extent of visibility while moving along the indicated ground paths (Figure 4.14). Left: reflection lines. Right: panel assignments. See Figures 4.16 and 4.17 for statistics and different views.



(a) Paneling solution with kink angle thresholds specified globally over the surface.



(b) Paneling solution with spatially adaptive kink angle thresholds.



(c) Paneling solution with another set of spatially adaptive kink angle thresholds.

(a) global cost: 5946

	-	38	15	2	119	32
molds	-	38	15	2	119	32
panels	102	622	84	11	349	32

divergence: 6mm
max angle: 3°

(b) path 1 cost: 5810

	-	73	8	1	169	22
molds	-	73	8	1	169	22
panels	152	683	17	5	321	22

divergence: 6mm
max angle: 1°-6° (adaptive)

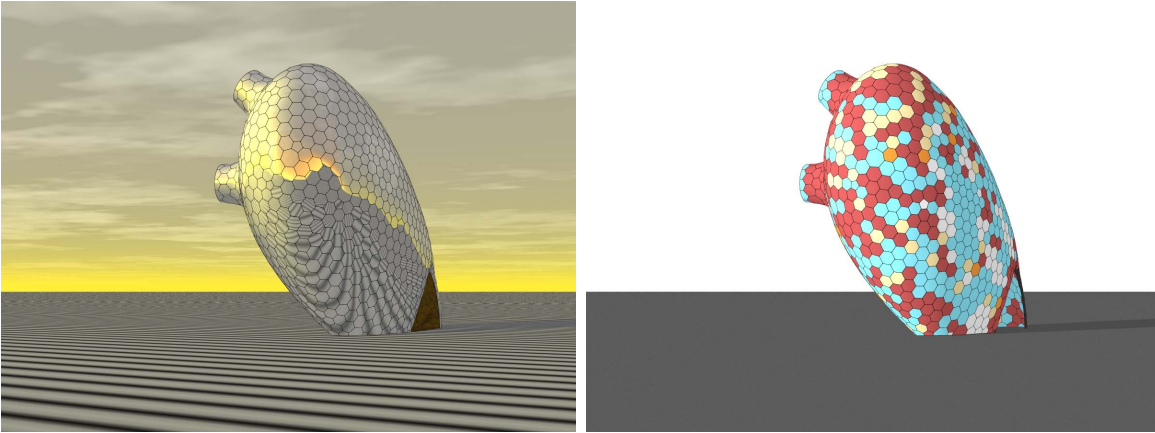
(c) path 2 cost: 6265

	-	45	7	5	191	15
molds	-	45	7	5	191	15
panels	97	631	17	13	427	15

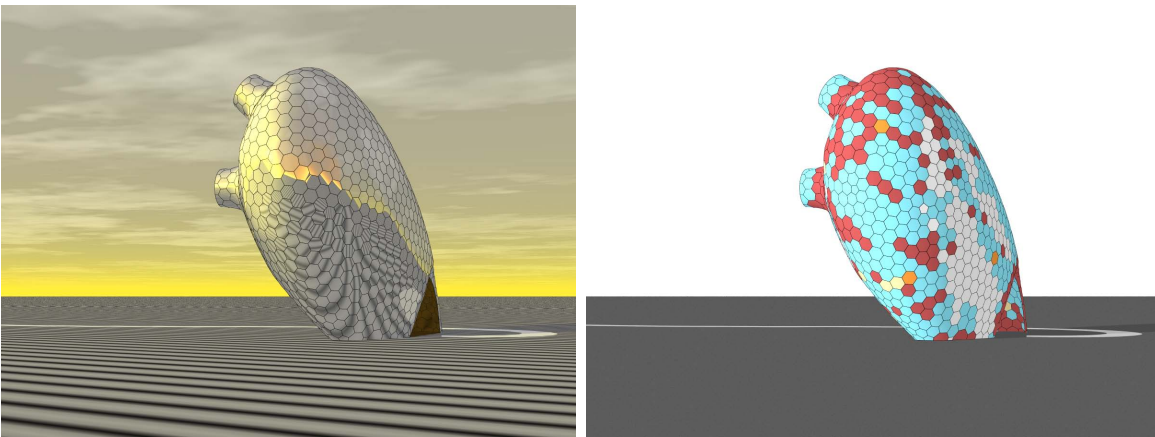
divergence: 6mm
max angle: 1°-6° (adaptive)

Figure 4.16 Global vs adaptive kink angle constraints for the Skipper Library and corresponding mold depot statistics (see also Figure 4.15).

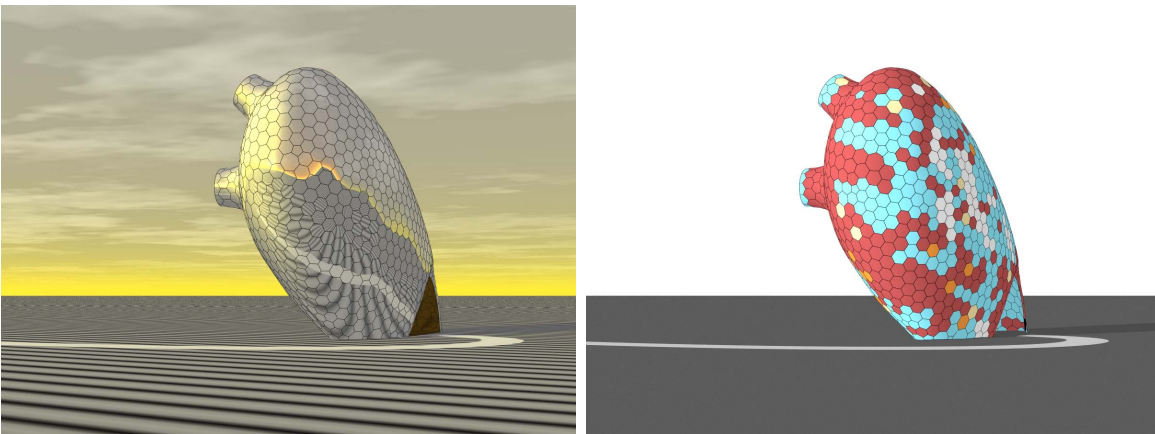
4 Paneling Architectural Freeform Surfaces



(a) Paneling solution with kink angle thresholds specified globally over the surface.



(b) Paneling solution with spatially adaptive kink angle thresholds.



(c) Paneling solution with another set of spatially adaptive kink angle thresholds.

Figure 4.17 Global vs spatially varying kink angle specifications on the Skipper Library. Please refer to Figure 4.15 and 4.16 for details.

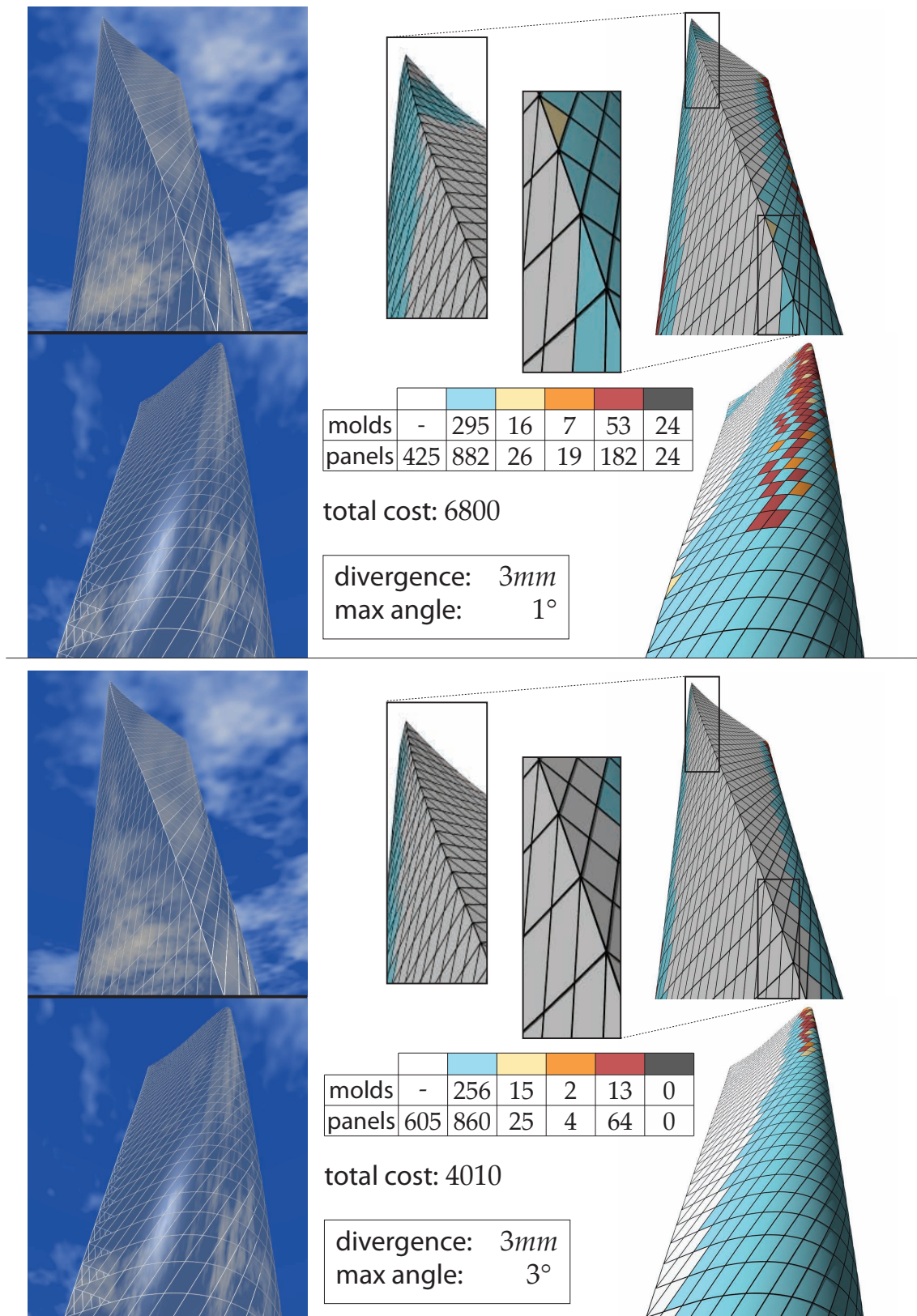


Figure 4.18 Paneling solutions for two different kink angle thresholds. The characteristic sharp feature line of the Lissajous Tower is preserved in our paneling solution. (Evolute www.evolute.at, Lissajous Tower Feasibility Study.)

Future Work. There are a number of desirable extensions to our method that constitute challenging problems for future research. One direction is to incorporate further important aspects into the optimization, e.g., structural feasibility and efficiency of the underlying support structure, especially if it is aligned with the network of panel boundaries. This may be done using properly simplified mechanical models, in extension of work by Whiting et al. [WOD09], and should lead to a new powerful tool for form-finding. The optimization could also include energy performance values or try to optimally integrate solar panels. These and related extensions towards the optimized use of the building would be in the spirit of the Building Information Modeling paradigm. To our knowledge the paneling problem is also of interest to the ship-hull construction industry (e.g., Figure 4.19). We plan to investigate this application in future work.

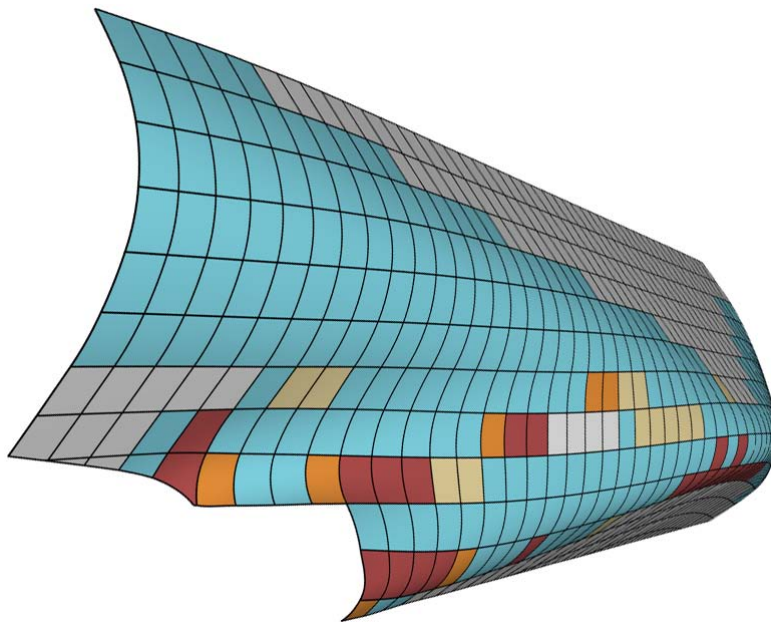


Figure 4.19 *Potential for future work: paneling study of a ship hull. (Data provided by Evolute, www.evolute.at)*

Conclusion and Outlook

In this thesis we have investigated various aspects of constraint-based surface processing for shape modeling and freeform architecture. For modeling, we have introduced an optimization framework that allows direct manipulation of derived surface properties. The user can modify lengths, areas and bending of the surface by drawing curves and patches on the surface and manipulating their metric or curvature properties. Curvature-Domain Shape Processing allows arbitrary curvature values to be prescribed on a given surface and enables advanced geometry processing techniques as simple operations in the curvature domain. Local surface measures provide additional control on the deformation semantics. Our approach complements existing methods for shape manipulation and leads to versatile surface deformations for shape design. For architecture, we have cast the paneling of freeform surfaces into a global optimization problem that takes into account panel production cost, reuse of manufacturing molds, and various constraints on surface quality and fabrication. The solution is based on a combination of discrete and continuous optimization, as well as a new inter-panel distance approximation for complexity reduction. We have demonstrated the performance of our system on several complex freeform designs by leading contemporary architects.

In both application scenarios, surface modeling and freeform architecture, advanced geometric constraints lead to challenging optimization problems with a complex solution space and a high number of variables. In this thesis we introduce various concepts on how to address such problems from specific implementation details to important design decisions, novel approximation strategies, and algorithmic schemes. These collected insights form a broad set of new tools for constraint-based surface processing to utilize, build upon, and extend in future research.

5.1 Impact

Although the research of this thesis is evaluated on specific application scenarios, many concepts introduced in this thesis are of relevance in the broader context of digital geometry processing. As indicated in Chapter 3 our constraint-based modeling framework has potential applications in parameterization and could be extended to serve more specialized purposes in industry, for example supporting surface processing using milling machines or analyzing and controlling the deformation behavior of stamped sheet metal. In Curvature-Domain Shape Processing we push the complexity of geometry processing tasks toward computation, thus reducing the burden on the user, who can apply simple and intuitive modifications and filtering operations in the curvature domain to edit and design geometric models. Such a mapping of a geometric problem into a domain that is more accessible to the user offers a new perspective on shape optimization and provides a platform for further development in 3D geometry processing.

Our paneling algorithm described in Chapter 4 finds an approximation of a general shape by a nearly smooth union of aesthetically arranged panels of a limited number of types and is a significant extension of the state-of-the-art in surface rationalization. Simultaneously deciding on the number of different panel types required, while optimizing their parameters, leads to an interesting and unusual mixture of discrete and continuous global optimization with potential applications in other domains. Finally, mold reuse by making relations explicit across

different object parts and removing redundancies in the shape information content, goes beyond symmetry detection and symmetrization towards global shape understanding.

5.2 Outlook

In the context of constraint-based surface processing, challenging open problems give rise to future research. We have discussed the extensibility of our methods and suggested application-specific directions of future work in the corresponding Chapters 3 and 4.

Efficiency. One of the major universal challenges of constraint-based surface processing is efficiency. As we have seen in this thesis, implementing advanced geometric constraints means solving complex non-linear optimization problems for which it is hard to find efficient solutions. Unfortunately, in many applications interactive performance is the ultimate goal.

In modeling, the user wishes to manipulate a surface in realtime by dragging the mouse or moving a slider to have optimal feedback and control. Our constraint-based surface processing framework introduced in Chapter 3 achieves interactive rates only for small problem instances.

In freeform architecture efficiency is less critical, since the construction of architectural freeform surfaces, as the main source of open geometric problems, is often decoupled from the design of such structures: the architect (user) designs the surface that is then given to geometry experts to compute, for example, a paneling. Design and construction, however, are rarely independent. Changing the design might improve the rationalization quality or cost and in order to be able to produce a surface with a given technology, significant changes in the design might be required. This dependancy leads to an expensive iterative production cycle where the architect adapts the design to observe changes on the construction layout. Ideally, the construction information would be directly integrated into the design process. In

5 Conclusion and Outlook

Architecture in the digital age: Design and Manufacturing [Kolo3], Branko Kolarevic writes:

"With the use of digital technologies, the design information is the construction information. (...) It is the digitally-based convergence of representation and production processes that represents the most important opportunity for a profound transformation of the profession..."

In freeform architecture the design information is not the construction information. The main reason is the high complexity of the involved optimization problems and the lack of efficient solutions. Even for comparably simple problem instances, such as paneling with planar quadrilateral panels, it is currently not possible to provide interactive feedback on the construction information during surface design.

We propose three main directions to address efficiency in future work:

1. **Implementation.** A non-neglectable efficiency gain can and should be obtained by careful implementation adapting to new software and hardware technologies. For example, the constraint-based surface processing frameworks presented in this thesis have a high potential for parallelism. In Chapter 3 we have demonstrated how multi-core and GPU parallelism can significantly boost optimization time. Our paneling algorithm (Chapter 4) makes use of parallelism as well, when building the sets for the discrete optimization. Additionally, as shown in this thesis, specific implementation details of the optimization solver (from stopping conditions to energy reformulations and multiresolution techniques) can make a big difference.
2. **Algorithms & Approximation.** While a careful implementation can significantly improve running times, algorithmic strategies can make the difference between feasible and infeasible. For example, the approximative distance space introduced for our paneling algorithm (Chapter 4) brings down computing time from months to hours. The key to successful performance improvements for constraint-based surface processing thus lies in the research of new algorithms and approximation techniques. A promising strategy is to search for relations to known problems in computer science and draw from decades of research on ef-

efficient algorithms with guarantees on complexity or approximation quality (see for example the set-cover analogy established in Chapter 4).

3. **Problem Formulation.** Sometimes even the best algorithm and the most efficient implementation will not be able to achieve the desired computation time. For example, paneling architectural freeform surfaces with double-curved panels, mold reuse, and ten-thousands of panels, as described in Chapter 4, is not likely to be solvable at interactive rates in the near future. We believe that the solution lies in adapting the problem formulation: Divide the problem into parts that should be solved at interactive rates and less critical parts for which more computation time can be afforded. This mandates a careful evaluation of what aspects of the problem allow and most critically demand interactive performance. In the application of constraint-based surface modeling for example, an interesting direction of future research is the investigation of fast preview techniques: Approximative tools based on simplifications of the surface and simplifications of the involved optimization terms can give the user the possibility to interactively explore the modeling space. Once the user is satisfied with the edits she/he can actuate the full optimization, observe the result, and continue editing. Similar techniques can be investigated to integrate construction information into freeform surface design: Form finding is guided by interactive feedback about the resulting paneling layout, without restricting the designer's expressiveness. This allows the architect to make more informed design decisions and significantly improves the efficiency of the production process. The main challenge of this direction is to formulate the constraint-based surface processing problem such that the user obtains as interactive and as accurate control on the important aspects of the solution as possible.

Useability. The methods for constraint-based surface processing investigated in this thesis are directly motivated by applications in art and industry. Our constraint-based shape deformation framework introduced in Chapter 3 provides new tools for modelers and artists to manipulate the geometry of a surface. The paneling algorithm pre-

5 Conclusion and Outlook

sented in Chapter 4 enables architects and architectural geometry consultants to automatically compute and control freeform surface rationalization.

Applied research can get great inspiration from a careful evaluation and understanding of the application specific needs and requirements and of how people will eventually use the tools the research enables. For our research on paneling architectural freeform surfaces we had the pleasure of closely collaborating with our colleagues from the geometry consulting company Evolute. Through Evolute we had direct access to invaluable knowledge about state-of-the-art working processes and its deficiencies and to cutting edge architectural designs from leading architects. This unique opportunity enabled well motivated design decisions and made the research results stronger as they could have ever been without these insights on the application domain and direct feedback on the usability of our tools.

A comprehensive analysis of the application of the presented tools in practice, however, is beyond the scope of this thesis and provides potential for future work. Future research in constraint-based surface processing would certainly benefit from scientific work

- presenting means and studies to measure and compare the effectiveness of constraints for surface processing and
- providing insights on further application specific needs and open challenges.

Interdisciplinary Problems. In practice, problems are rarely purely geometric but often simultaneously involve a combination of problems from other domains. For example, surface modeling is used in car and airplane design, biology, medicine, and many other fields with their own application-specific demands. Paneling architectural freeform surfaces involves requirements from structural engineering, material technology, and environment concerns such as lighting conditions. An important future challenge is therefore the interdisciplinary collaboration combining constraint-based surface processing with research in other fields to develop comprehensive solutions for modeling, architecture, and beyond.

New Constraints, New Applications. Constraint-based surface processing is an unbounded research topic with a large number of fascinating applications that cannot nearly be covered by one PhD thesis alone. New constraints and new applications will pose new challenges for future work and will continue to inspire exiting research in digital geometry processing.

A P P E N D I X



Optimization

A.1 Derivatives of Geometric Properties Evaluated on Triangle Meshes

Building the Jacobian matrix for the nonlinear least squares optimization employed in our constraint-based modeling framework (Chapter 3) requires the computation of partial derivatives of all involved optimization terms with respect to the vertex positions. In this section we derive analytic expressions for these derivatives.

A.1.1 Derivatives of Area, Length, and Angle

By applying the extended chain rule, the derivatives of all energies introduced in Chapter 3 can be broken down to weighted sums of derivatives of triangle areas, lengths of path/mesh edges, and angles between two incident edges (or vectors in general). Since the extended chain rule is a standard operator we concentrate on providing expressions for these three basic components using the notation shown in Figure A.1. The only derivative requiring special treatment is the

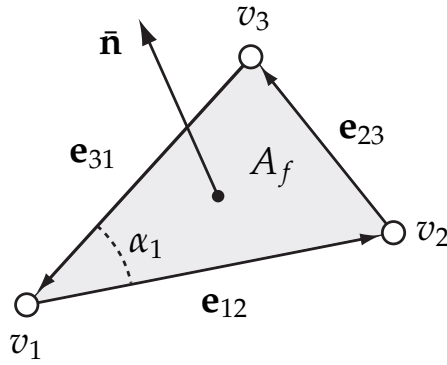


Figure A.1 Notation used in the derivation of derivatives.

derivative of principal curvatures which we will discuss in detail in Section A.1.2.

The gradient of the face area can be computed as

$$\frac{\partial}{\partial \mathbf{v}_1} A_f = \frac{1}{2} (\bar{\mathbf{n}} \times \mathbf{e}_{23}), \quad (\text{A.1})$$

where \$\bar{\mathbf{n}}\$ is the normalized face normal. The gradients of the edge lengths are

$$\frac{\partial}{\partial \mathbf{v}_1} \|\mathbf{e}_{12}\| = -\bar{\mathbf{e}}_{12} \quad \frac{\partial}{\partial \mathbf{v}_2} \|\mathbf{e}_{12}\| = \bar{\mathbf{e}}_{12} \quad (\text{A.2})$$

where \$\bar{\mathbf{e}}_{12}\$ is the normalized edge. For an angle between two edges, the derivatives can be computed as

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_2} \alpha_1 &= \frac{\mathbf{e}_{12} \times \bar{\mathbf{n}}}{\|\mathbf{e}_{12}\|^2} & \frac{\partial}{\partial \mathbf{v}_3} \alpha_1 &= \frac{\mathbf{e}_{31} \times \bar{\mathbf{n}}}{\|\mathbf{e}_{31}\|^2} \\ \frac{\partial}{\partial \mathbf{v}_1} \alpha_1 &= -\frac{\mathbf{e}_{12} \times \bar{\mathbf{n}}}{\|\mathbf{e}_{12}\|^2} - \frac{\mathbf{e}_{31} \times \bar{\mathbf{n}}}{\|\mathbf{e}_{31}\|^2} \end{aligned} \quad (\text{A.3})$$

A.1.2 Curvature Derivatives

The optimization of principal curvatures requires derivatives of per-vertex principal curvatures with respect to the unknown vertex positions. We present the complete derivative formulas in this section. In the following equations, \$\mathbf{v}_{i,c}\$ refers to the component \$c\$ of vertex \$i\$, \$c \in \{x, y, z\}\$.

The principal curvatures κ_1 and κ_2 are eigenvalues of the curvature tensor \mathcal{K} defined in Equation 3.13. Umbilic points occur whenever the curvature tensor field is isotropic, indicating that the underlying surface locally approximates a sphere or plane. At such points, the principal curvatures are equal and their derivatives are undefined. However, this problem falls into the class of composite nonsmooth optimization [WF86] since, although the derivatives are discontinuous at umbilic points, they can be approximated from information available at the discontinuities. Following Kim and colleagues [KCH02], we average the derivative computation at nonsmooth points to achieve a viable approximation. With these observations in place, we compute the curvature derivatives according to three cases.

Case 1: The most common case occurs when all three eigenvalues are distinct. In this situation, we have

$$\frac{\partial}{\partial \mathbf{v}_{i,c}} \lambda_j = \mathbf{u}_j^\top \left(\frac{\partial}{\partial \mathbf{v}_{i,c}} \mathcal{K} \right) \mathbf{u}_j, \quad (\text{A.4})$$

where λ_j and \mathbf{u}_j are the j^{th} eigenvalue and normalized eigenvector, respectively, of the curvature tensor $\mathcal{K}(v_i)$ and $\lambda_1 < \lambda_2 < \lambda_3$ [HUY95]. The tensor derivative is given below in Equation A.7.

Case 2: A cylindric point occurs when the two smallest eigenvalues, in an absolute sense, are nearly equal and distinct from the remaining eigenvalue. The derivative of this remaining eigenvalue, which corresponds to either κ_1 or κ_2 , can be computed as described above. The derivative of the other principal curvature can be computed using the mean curvature $H = \frac{1}{2}(\kappa_1 + \kappa_2)$. Since $H = \frac{1}{2}\text{trace}(\mathcal{K})$, we have

$$\frac{\partial}{\partial \mathbf{v}_{i,c}} H = \frac{1}{2} \left(\frac{\partial}{\partial \mathbf{v}_{i,c}} t_{11} + \frac{\partial}{\partial \mathbf{v}_{i,c}} t_{22} + \frac{\partial}{\partial \mathbf{v}_{i,c}} t_{33} \right), \quad (\text{A.5})$$

where t_{jj} , $j \in 1 \dots 3$, are the diagonal entries of \mathcal{K} . The formula for their partial derivatives is presented in Equation A.7.

Case 3: When $\kappa_1 = \kappa_2$, we have an umbilic point and the derivative is not defined. However, a viable approximation to the derivative at

this point is given by averaging the derivatives that meet at the discontinuity [KCHo2], which leads to

$$\frac{\partial}{\partial \mathbf{v}_{i,c}} \kappa_1 = \frac{\partial}{\partial \mathbf{v}_{i,c}} \kappa_2 \approx \frac{\partial}{\partial \mathbf{v}_{i,c}} H, \quad (\text{A.6})$$

where the rightmost term is defined in Equation A.5.

As a side remark, the principal curvatures and their derivatives can also be estimated in terms of the mean and Gaussian curvatures. Under the assumption that the third eigenvalue is zero, this approach avoids eigenvalue decomposition and leads to simpler derivative expressions. However, we found this method to be less accurate near umbilic points.

Tensor Derivatives. Equation A.4 requires the partial derivative of the curvature tensor \mathcal{K} with respect to the unknown vertices:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_{i,c}} \mathcal{K} &= \frac{1}{|B|} \frac{\partial}{\partial \mathbf{v}_{i,c}} \hat{\mathcal{K}} - \frac{\hat{\mathcal{K}}}{|B|^2} \frac{\partial}{\partial \mathbf{v}_{i,c}} |B| \\ \text{where } \hat{\mathcal{K}} &= \sum_{e \in B} \beta(e) |e \cap B| \bar{\mathbf{e}} \bar{\mathbf{e}}^\top. \end{aligned} \quad (\text{A.7})$$

The derivative of $\hat{\mathcal{K}}$ is given by

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_{i,c}} \hat{\mathcal{K}} &= \sum_{e \in B} \left[\left(\frac{\partial}{\partial \mathbf{v}_{i,c}} \beta(e) \right) |e \cap B| \bar{\mathbf{e}} \bar{\mathbf{e}}^\top + \right. \\ &\quad \beta(e) \left(\frac{\partial}{\partial \mathbf{v}_{i,c}} |e \cap B| \right) \bar{\mathbf{e}} \bar{\mathbf{e}}^\top + \\ &\quad \left. \beta(e) |e \cap B| \left(\frac{\partial}{\partial \mathbf{v}_{i,c}} \bar{\mathbf{e}} \bar{\mathbf{e}}^\top \right) \right] \end{aligned} \quad (\text{A.8})$$

where

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_{i,c}} \bar{\mathbf{e}} \bar{\mathbf{e}}^\top &= \frac{\partial}{\partial \mathbf{v}_{i,c}} \frac{\mathbf{e} \mathbf{e}^\top}{\|\mathbf{e}\|^2} \\ &= \left(\frac{\partial}{\partial \mathbf{v}_{i,c}} \mathbf{e} \mathbf{e}^\top \right) \frac{1}{\|\mathbf{e}\|^2} - \frac{\bar{\mathbf{e}} \bar{\mathbf{e}}^\top}{\|\mathbf{e}\|^2} \frac{\partial}{\partial \mathbf{v}_{i,c}} \|\mathbf{e}\|^2. \end{aligned} \quad (\text{A.9})$$

The outer product derivative is given by

$$\frac{\partial}{\partial \mathbf{v}_{i,c}} \mathbf{e} \mathbf{e}^\top = \frac{\partial}{\partial \mathbf{v}_{i,c}} \begin{bmatrix} \mathbf{e}_x \mathbf{e}_x & \mathbf{e}_x \mathbf{e}_y & \mathbf{e}_x \mathbf{e}_z \\ \mathbf{e}_y \mathbf{e}_x & \mathbf{e}_y \mathbf{e}_y & \mathbf{e}_y \mathbf{e}_z \\ \mathbf{e}_z \mathbf{e}_x & \mathbf{e}_z \mathbf{e}_y & \mathbf{e}_z \mathbf{e}_z \end{bmatrix}, \quad (\text{A.10})$$

A.1 Derivatives of Geometric Properties Evaluated on Triangle Meshes

which is straightforward to evaluate entrywise.

The remaining terms in Equations A.7, A.8, and A.9 are more easily interpreted geometrically as derivatives with respect to vertex \mathbf{v}_i , rather than with respect to its individual components.

The area gradient for Equation A.7 is given by

$$\frac{\partial}{\partial \mathbf{v}_i} |B| = \sum_{f \in B} B_f \frac{\partial}{\partial \mathbf{v}_i} A_f \quad (\text{A.11})$$

where A_f is the face area and B_f is the fraction of A_f which is inside B . Since B is a union of barycentric regions, B_f is always equal to $\frac{1}{3}$, $\frac{2}{3}$, or 1. The derivative of the face area is derived in Equation A.1.

As shown by Bridson, Marino, and Fedkiw [BMF03], the dihedral angle gradients from Equation A.8 are given by

$$\frac{\partial}{\partial \mathbf{v}_1} \beta(e) = -\|\mathbf{e}\| \bar{\mathbf{n}}_1 \quad (\text{A.12})$$

$$\frac{\partial}{\partial \mathbf{v}_2} \beta(e) = -\|\mathbf{e}\| \bar{\mathbf{n}}_2 \quad (\text{A.13})$$

$$\frac{\partial}{\partial \mathbf{v}_3} \beta(e) = -\frac{(\mathbf{v}_1 - \mathbf{v}_4) \cdot e}{\|\mathbf{e}\|} \bar{\mathbf{n}}_1 - \frac{(\mathbf{v}_2 - \mathbf{v}_4) \cdot e}{\|\mathbf{e}\|} \bar{\mathbf{n}}_2 \quad (\text{A.14})$$

$$\frac{\partial}{\partial \mathbf{v}_4} \beta(e) = -\frac{(\mathbf{v}_1 - \mathbf{v}_3) \cdot e}{\|\mathbf{e}\|} \bar{\mathbf{n}}_1 + \frac{(\mathbf{v}_2 - \mathbf{v}_3) \cdot e}{\|\mathbf{e}\|} \bar{\mathbf{n}}_2. \quad (\text{A.15})$$

Similar to Equation A.2, the edge length derivatives from Equations A.8 and A.9 are

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_1} |e_{12} \cap B| &= -B_{e_{12}} \bar{\mathbf{e}}_{12} & \frac{\partial}{\partial \mathbf{v}_2} |e_{12} \cap B| &= B_{e_{12}} \bar{\mathbf{e}}_{12} \\ \frac{\partial}{\partial \mathbf{v}_1} \|\mathbf{e}_{12}\|^2 &= -2\mathbf{e}_{12} & \frac{\partial}{\partial \mathbf{v}_2} \|\mathbf{e}_{12}\|^2 &= 2\mathbf{e}_{12}, \end{aligned} \quad (\text{A.16})$$

where $B_{e_{12}}$ is the fraction of the edge which is inside B . Again because of the barycentric regions, $B_{e_{12}}$ is either $\frac{1}{2}$ or 1.

A.2 Element-Weighted Set Cover

An important contribution of our algorithm for paneling architectural freeform surfaces (Chapter 4) is the mapping of the constraint-based optimization problem to a set cover problem. This section provides a detailed analysis of the set cover problem and the algorithm we introduced in Section 4.3.2, including a proof of the approximation quality. As discussed in Chapter 4, the set cover algorithm can be extended to the more general setting where the panel cost is given as a per-unit-area cost and has to be multiplied with the panel area to obtain the total panel cost. In the following we reformulate the problem (Section A.2.1) and the algorithm (Section A.2.2) to support this more general cost model. We describe the algorithm in slightly more technical terms compared to Chapter 4 to enable a precise analysis (Section A.2.3).

A.2.1 Problem

Given a set of elements $\mathcal{S} = \{s_1, \dots, s_n\}$ and a set of m subsets of \mathcal{S} , $\sigma^* = \{\mathcal{S}_1, \dots, \mathcal{S}_m\}$, and a cost function $c : \sigma^* \rightarrow \mathbb{R}$, defined as

$$c(\mathcal{S}_k) = c_{set}^k + c_{element}^k \sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i} \quad (\text{A.17})$$

where c_{set}^k and $c_{element}^k$ are constant, non-negative base and element costs for set \mathcal{S}_k , $a : (\mathcal{S}, \sigma^*) \rightarrow \{0, 1\}$ is an assignment function, that determines for every element s_i in set \mathcal{S}_k whether s_i is assigned to \mathcal{S}_k (value 1) or not (value 0). If $s_i \notin \mathcal{S}_k$ the function $a(s_i, \mathcal{S}_k)$ is defined to be 0. A_{s_i} is a non-negative cost factor for every element s_i by which the element cost $c_{element}^k$ has to be multiplied. We therefore call $c_{element}^k$ the *unit element cost*, i.e., A_{s_i} defines how many units element s_i corresponds to.

Find a collection $\sigma \subseteq \sigma^*$ and an assignment function a , such that

1. every element s_i is assigned to exactly one set $\mathcal{S}_k \in \sigma$ that contains it, that is $\forall s_i : \sum_{k: s_i \in \mathcal{S}_k \in \sigma} a(s_i, \mathcal{S}_k) = 1$,
2. and the total cost $\sum_{k: \mathcal{S}_k \in \sigma} c(\mathcal{S}_k)$ is minimized.

Relation to Paneling. In the context of Paneling (Chapter 4), every set \mathcal{S}_k corresponds to a mold and denotes all segments $s_i \in \mathcal{S}_k$ that can be fitted by this mold within given error tolerance thresholds. σ^* contains the initial mold candidates. The result σ defines the final mold depot, whereas the assignment function a uniquely determines which mold produces which segment. The cost c_{set}^k corresponds to the cost of mold k , the unit cost $c_{element}^k$ denotes the per-unit-area cost of producing a panel with mold k , and the cost factor A_{s_i} is the area of segment s_i . The full cost of producing segment s_i with mold k is therefore $c_{element}^k A_{s_i}$. This special cost structure is important and the proof given in Section A.2.3 does not extend, for example, to the more general case that the cost to produce a panel with a given mold is an arbitrary non-constant cost function $c_{element}^k(s_i)$. The special case where the cost of a panel does not depend on the panel size (or all panels have the same size) is included in the above formulation by setting $A_{s_i} = 1$ for all segments. The formulation as a set cover problem defines a paneling solution with minimal cost that meets the specified error tolerance thresholds.

Relation to Classical Weighted Set Cover. From condition 1. follows that σ covers all elements in \mathcal{S} , that is, $\cup_{k:\mathcal{S}_k \in \sigma} \mathcal{S}_k = \mathcal{S}$. The problem described above is therefore related to the classical *weighted set cover* problem [Joh74], where the weights correspond to costs in our case. While this problem is known to be NP-hard, a polynomial-time approximation strategy can be used to find an approximate solution whose cost is guaranteed to be within $\log n$ times the cost of the optimal solution. It has been shown that $\log n$ is the best possible approximation ratio of any polynomial-time algorithms [Fei98]. What distinguishes our setting from the classical weighted set cover problem is that each element is eventually assigned to only one set. Thus our weights (costs) depend on which elements are assigned to which set. If the element costs $c_{element}^k$ are zero for all sets \mathcal{S}_k , the problem above is equal to the classical weighted set cover problem. Even if $c_{element}^k > 0$, however, the same polynomial-time approximation algorithm can be used to solve our problem (Section A.2.2) and the proof for the logarithmic approximation ratio of the polynomial-time weighted set cover algorithm directly translates to our more general setup (Section A.2.3).

A.2.2 Algorithm

We define the *efficiency* of a set \mathcal{S}_k with respect to an assignment a as the function

$$\phi(\mathcal{S}_k, a) = \frac{\sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i}}{c(\mathcal{S}_k)}, \quad (\text{A.18})$$

where the cost of \mathcal{S}_k is computed according to the current assignment function a . The efficiency determines how well a set covers elements in \mathcal{S} compared to the cost it requires to do so. During the execution of the algorithm, let $\sigma' \subseteq \sigma^*$ be the collection of all sets \mathcal{S}_k that have not yet been chosen for σ .

Our algorithm to determine the covering sets starts with an empty collection σ . Thus the collection σ' contains all the sets $\mathcal{S}_k \in \sigma^*$ and all elements s_i are initially assigned to all sets \mathcal{S}_k containing s_i . We then successively add the set \mathcal{S}_m with highest efficiency to the current solution σ . The elements currently assigned to \mathcal{S}_m are permanently assigned to only \mathcal{S}_m . The assignment function and consequently the efficiency of all remaining sets are updated, and the algorithm is iterated until all elements are covered.

$\sigma \leftarrow \emptyset, \sigma' \leftarrow \sigma^*$	
$\forall \mathcal{S}_k \in \sigma^*, s_i \in \mathcal{S}_k : a(s_i, \mathcal{S}_k) = 1$	
while $\cup_{\mathcal{S}_k \in \sigma} \mathcal{S}_k \neq \mathcal{S}$	<i>while not all elements are covered</i>
$\mathcal{S}_m \leftarrow \arg \max_{\mathcal{S}_k \in \sigma'} \phi(\mathcal{S}_k, a)$	<i>candidate set with max. efficiency</i>
$\sigma \leftarrow \sigma \cup \{\mathcal{S}_m\}$	<i>add to covering sets</i>
$\sigma' \leftarrow \sigma' - \{\mathcal{S}_m\}$	<i>remove from candidate sets</i>
$\forall s_i \in \mathcal{S}_m, \mathcal{S}_k \in \sigma' : a(s_i, \mathcal{S}_k) = 0$	<i>update assignments</i>
end	

A.2.3 Analysis

Theorem. The algorithm described in Section A.2.2 achieves an approximation ratio of

$$\frac{ALG}{OPT} \leq \ln \left(\frac{A_{tot}}{A_{min}} \right) + 1 \quad (\text{A.19})$$

to the optimal solution OPT of the problem described in Section A.2.1, where

$$A_{tot} = \sum_{i=1}^n A_{s_i} \quad (\text{A.20})$$

is the total number of units and

$$A_{min} = \min_{s_i \in \mathcal{S}} A_{s_i} \quad (\text{A.21})$$

is the minimum number of units for the elements in \mathcal{S} .

Proof. The **per unit cost** of a set is the average cost per unit for all elements assigned to the set and can be computed as the inverse of the set's efficiency $\phi(\mathcal{S}_k, a)$. Thus for each element s_i assigned to \mathcal{S}_k we define the per unit cost $c(s_i) = 1/\phi(\mathcal{S}_k, a)$. Note that $\sum_{s_i \in \mathcal{S}} c(s_i) A_{s_i} = ALG$, that is, the total cost of the algorithm.

Let us sort the elements in the order that they are covered (permanently assigned) by the algorithm, breaking ties arbitrarily. At the time that the i^{th} element (call it s_i) gets covered (directly before), the total number of uncovered units is at least $\sum_{j=i}^n A_{s_j}$. This holds by construction of the ordering of the elements. The cost OPT_{sub} of solving the element-weighted set cover for the subproblem with elements that are currently not covered and the current candidate sets σ' is at most OPT . This holds since one solution for the subproblem can easily be obtained by taking the optimal solution for the whole problem and removing all elements that were already covered. Removing elements can only decrease the cost. Therefore, the "per unit cost" of the subproblem is at most

$$\frac{OPT_{sub}}{A_{sub}} \leq \frac{OPT}{\sum_{j=i}^n A_{s_j}}, \quad (\text{A.22})$$

A Optimization

where A_{sub} denotes the total number of units present in the subproblem. Thus, in the final solution of the subproblem there must be at least one set \mathcal{S}_k with "per unit cost" $\leq \text{OPT} / \sum_{j=i}^n A_{s_j}$. At the time the element s_i gets covered, this set \mathcal{S}_k can only have a lower or equal "per unit cost". This holds, because the "per unit cost" of a set

$$\frac{1}{\phi(\mathcal{S}_k, a)} = \frac{c(\mathcal{S}_k)}{\sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i}} \quad (\text{A.23})$$

$$= \frac{c_{set}^k + c_{element}^k \sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i}}{\sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i}} \quad (\text{A.24})$$

$$= \frac{c_{set}^k}{\sum_{s_i \in \mathcal{S}_k} a(s_i, \mathcal{S}_k) A_{s_i}} + c_{element}^k \quad (\text{A.25})$$

can only decrease, if more elements are assigned to the set. Since the set we choose by the Algorithm in Section A.2.2 when we cover s_i is the set with the highest efficiency, that is, the lowest "per unit cost", it holds

$$c(s_i) \leq \frac{\text{OPT}}{\sum_{j=i}^n A_{s_j}}. \quad (\text{A.26})$$

Over the execution of the algorithm the value of i goes from 1 to n . Thus, the total cost is at most

$$\text{ALG} \leq \sum_{i=1}^n \frac{\text{OPT}}{\sum_{j=i}^n A_{s_j}} A_{s_i} = \text{OPT} \sum_{i=1}^n \frac{A_{s_i}}{\sum_{j=i}^n A_{s_j}}. \quad (\text{A.27})$$

Figure A.2 shows a graphical interpretation of Equation A.27. The element cost factors (unit counts) A_{s_i} are mapped to intervals on the x -axis in order of descending indices i . The i^{th} x -axis value counted from the right therefore corresponds to $\sum_{j=i}^n A_{s_j}$. The corresponding y -axis value is its inverse $1 / \sum_{j=i}^n A_{s_j}$. The sum

$$\sum_{i=1}^n \frac{A_{s_i}}{\sum_{j=i}^n A_{s_j}} \quad (\text{A.28})$$

therefore denotes the area of the vertical bars over each interval A_{s_i} (see Figure A.2). The area of the bar of the left-most interval is always

$$\frac{A_{s_n}}{\sum_{j=n}^n A_{s_j}} = 1 \quad (\text{A.29})$$

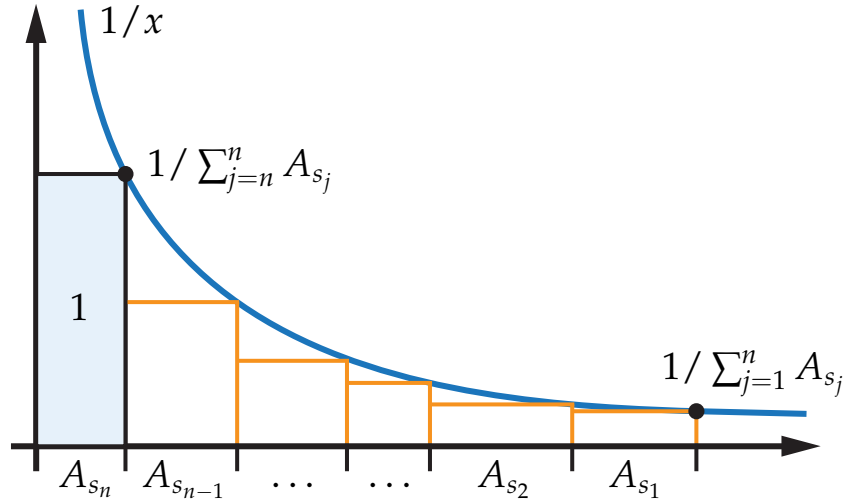


Figure A.2 Graphical interpretation of Equation A.27 (discussion see text).

Equation A.27 can thus be continued as

$$\text{ALG} \leq \text{OPT} \sum_{i=1}^n \frac{A_{s_i}}{\sum_{j=i}^n A_{s_j}} \quad (\text{A.30})$$

$$= \text{OPT} \left(1 + \sum_{i=1}^{n-1} \frac{A_{s_i}}{\sum_{j=i}^n A_{s_j}} \right) \quad (\text{A.31})$$

$$\leq \text{OPT} \left(1 + \int_{A_{\min}}^{A_{\text{tot}}} \frac{1}{x} dx \right) \quad (\text{A.32})$$

$$= \text{OPT} \left(1 + \ln \frac{A_{\text{tot}}}{A_{\min}} \right) \quad (\text{A.33})$$

where the step from Equation A.31 to Equation A.32 makes use of the fact that the former is a lower Riemann sum approximation of the latter (see Figure A.2). **q.e.d.**

Implication for the Paneling Algorithm. In the context of paneling (Chapter 4) this result means, that the approximation quality of the set cover algorithm depends on the ratio between the total surface area A_{tot} and the area of the smallest segment A_{\min} . If the panel cost does not depend on the panel size, or all panels have the same size, the approximation ratio depends on the number of segments, since in that case $A_{\text{tot}}/A_{\min} = n$.

Metric Space for Approximate Panel-Segment Distances

In our paneling algorithm (see Chapter 4) we efficiently estimate registration errors in a 6D metric space to avoid more than 95% of the costly nonlinear mold-segment alignments for the discrete optimization step of Section 4.3.2. We define a mapping of planes, paraboloids, and cubic patches into this space and show how cylinders and tori can be approximated with cubics. Representing segments by their best fitting cubic polynomial then allows calculating upper bounds on the registration error through simple 6D Euclidean distances computations. Figure B.1 evaluates the accuracy of this approximate registration error computation.

Distances Between Cubic Polynomials. We represent a cubic polynomial patch in the principal frame of the patch center as

$$P_i(x, y) = a_i x^2 + b_i y^2 + c_i x^3 + d_i x^2 y + e_i x y^2 + f_i y^3.$$

Our goal is to find potential mold candidates for a given segment. We thus consider the domain $D = [-L, L]^2$, where L is half the segment side length computed on the initial curve network. Since these

B Metric Space for Approximate Panel-Segment Distances

approximate distances are only used for pruning, it is sufficient to assume a quadratic shape even for non-quadratic segments. We define the L_2 -distance $E_{ij} = E(P_i, P_j)$ between two polynomials P_i and P_j by optimizing over the relative shift in z -direction

$$E_{ij} = \min_{\tau} \iint_D (ax^2 + by^2 + cx^3 + dx^2y + exy^2 + fy^3 + \tau)^2 dx dy,$$

where $a := a_i - a_j$, etc. Since the integral can be factored into the form $A\tau^2 + 2B\tau + C$, we obtain $E_{ij} = C - B^2/A$ for the optimal value of $\tau = -B/A$. Substituting $\bar{d}_i = d_i + f_i$ and $\bar{e}_i = c_i + e_i$ and reordering terms yields

$$E_{ij} = 4L^2 \left(\frac{4L^4}{45} a^2 + \frac{4L^4}{45} b^2 + \frac{L^6}{15} \bar{d}^2 + \frac{L^6}{15} \bar{e}^2 + \frac{8L^6}{105} c^2 + \frac{8L^6}{105} f^2 \right).$$

By setting $\bar{E} = \sqrt{E/(4L^2)}$, our distance measure between cubic patches corresponds to the canonical Euclidean distance where every cubic is represented as a point in the 6D Euclidean space as

$$\mathbf{P}_i = \left(\frac{2L^2}{3\sqrt{5}} a_i, \frac{2L^2}{3\sqrt{5}} b_i, \frac{L^3}{\sqrt{15}} \bar{d}_i, \frac{L^3}{\sqrt{15}} \bar{e}_i, \frac{\sqrt{8}L^3}{\sqrt{105}} c_i, \frac{\sqrt{8}L^3}{\sqrt{105}} f_i \right)^T.$$

Since we are interested in the shape of the cubic rather than its specific orientation, we consider for every cubic P_i all four right handed coordinate transformations and another four with a flipped normal direction z :

$$P_i^* = \{P_i(x, y), P_i(-y, x), P_i(-x, -y), P_i(y, -x), \\ -P_i(x, -y), -P_i(y, x), -P_i(-x, y), -P_i(-y, -x)\}$$

which correspond to eight points in the derived Euclidean space. We finally define our approximative distance measure as

$$d(P_i, P_j) = \min_{P'_j \in P_j^*} \bar{E}(P_i, P'_j),$$

which defines a metric on the space of cubic polynomials.

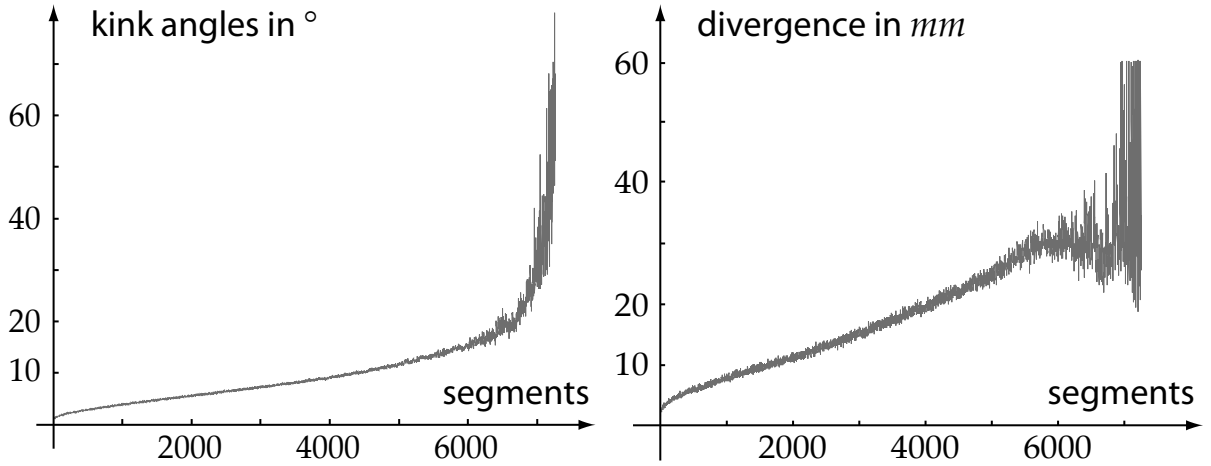
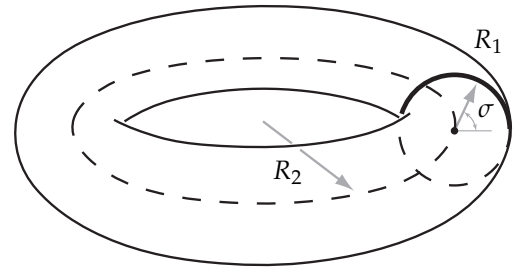


Figure B.1 Maximal kink angle and divergence averaged by fitting 400 randomly selected molds to all the 7265 segments of the National Head Quarters design (see Chapter 4). The values are sorted by our approximate panel-segment distances to demonstrate the strong correlation of the approximative and the exact fitting errors.

Cylinder and Torus. Planes and paraboloids can be directly described as cubic polynomials with zero higher order coefficients. For cylinders and tori, we use the Taylor expansion expressed in the coordinate system of the principal frame at a center point on the mold as an approximation. For cylinders with radius R the only nonzero coefficient is thus $a_i = 1/(2R)$.

Torus molds are defined by a meridian radius R_1 , a parallel radius R_2 , and angle σ that determines where on the meridian the mold center lies. Since this representation has three parameters, we use the third-order Taylor expansion



$$T_3(x, y) = -\frac{1}{2} \left(\frac{1}{R_1} x^2 + \frac{\cos \sigma}{R} y^2 + \frac{R_2 \sin \sigma}{R^2 R_1} x y^2 \right), \quad (\text{B.1})$$

where $R := R_2 + R_1 \cos \sigma$, and $-1/R_1$ and $-(\cos \sigma)/R$ are the principal curvatures at the mold center.

Bibliography

- [ACSD⁺03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 22(3), 2003.
- [AKM⁺06] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation – a comparative study. In *IEEE International Conference on Shape Modeling and Applications*, 2006.
- [AM93] Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280, 1993.
- [AS06] Alexis Angelidis and Karan Singh. Space deformations and their application to shape modeling. In *ACM SIGGRAPH Course Notes*, 2006.
- [BCGB08] Mirela Ben-Chen, Craig Gotsman, and Guy Bunin. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum (Proceedings of Eurographics)*, 27(2), 2008.
- [BGo8] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, December 2008.
- [Bjö96] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

Bibliography

- [BKo4] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 23(3), 2004.
- [BMFo3] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Symposium on Computer Animation*, 2003.
- [Boto5] Mario Botsch. *High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes*. PhD thesis, RWTH Aachen, 2005.
- [BPGKo6] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt. Primo: coupled prisms for intuitive surface modeling. In *Symposium on Geometry Processing*, 2006.
- [BPK⁺o8] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Rössl. Geometric modeling based on polygonal meshes. In *Eurographics Course Notes*, 2008.
- [BSo5] Alexander I. Bobenko and Peter Schröder. Discrete willmore flow. In *Symposium on Geometry Processing*, 2005.
- [BSo8] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1), 2008.
- [BSo9] Alexander Bobenko and Yuri Suris. *Discrete differential geometry: Integrable Structure*. Graduate Studies in Math. AMS, 2009.
- [CAAo9] Robert Carroll, Maneesh Agrawala, and Aseem Agarwala. Optimizing content-preserving projections for wide-angle images. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2009.
- [CADo4] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 23(3):905–914, 2004.

- [CDHR09] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Software*, 35(3), 2009.
- [CP05] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aided Geom. Des.*, 22(2), 2005.
- [CS92] Xin Chen and Francis Schmitt. Intrinsic surface properties from surface triangulation. In *European Conference on Computer Vision*, pages 739–743, 1992.
- [CSM03] David Cohen-Steiner and Jean-Marie Morvan. Restricted delaunay triangulations and normal cycle. In *Symposium on Computational Geometry*, 2003.
- [Dav] T. A. Davis. Algorithm 8xx: Suitesparseqr, a multifrontal multithreaded sparse qr factorization package. submitted to *ACM Trans. Math. Software*.
- [dC76] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
- [dC92] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser Boston, 1992.
- [DMK03] P. Degener, J. Meseth, and R. Klein. An adaptable surface parameterization method. In *Proceedings of 12th Int. Meshing Roundtable*, 2003.
- [DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 317–324, 1999.
- [DS87] J. E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial Mathematics, 1987.
- [EDS⁺10] Michael Eigensatz, Mario Deuss, Alexander Schiftner, Martin Kilian, Niloy J. Mitra, Helmut Pottmann, and Mark

Bibliography

- Pauly. Case studies in cost-optimized paneling of architectural freeform surfaces. In *Advances in Architectural Geometry*, 2010.
- [EKS⁺10] Michael Eigensatz, Martin Kilian, Alexander Schiftner, Niloy Mitra, Helmut Pottmann, and Mark Pauly. Paneling architectural freeform surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3), 2010.
- [EP09] Michael Eigensatz and Mark Pauly. Positional, metric, and curvature control for constraint-based surface deformation. *Computer Graphics Forum (Proceedings of Eurographics)*, 28(2), 2009.
- [EPT⁺07] I. Eckstein, J.-P. Pons, Y. Tong, C.-C. J. Kuo, and M. Desbrun. Generalized surface flows for mesh processing. In *Symposium on Geometry Processing*, pages 183–192, 2007.
- [ESP08] Michael Eigensatz, Robert W. Sumner, and Mark Pauly. Curvature-domain shape processing. *Computer Graphics Forum (Proceedings of Eurographics)*, 27(2), 2008.
- [FDCO03] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 22(3):950–953, 2003.
- [Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *JACM*, 45(4):634–652, 1998.
- [FH05] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In *In Advances in Multiresolution for Geometric Modelling*, pages 157–186, 2005.
- [Fle00] R. Fletcher. *Practical Methods of Optimization*. Wiley, 2000.
- [FLHCO10] Chi-Wing Fu, Chi-Fu Lai, Ying He, and Daniel Cohen-Or. K-set tilable surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3), 2010.
- [GGRZO6] Eitan Grinspun, Yotam Gingold, Jason Reisman, and Denis Zorin. Computing discrete shape operators on gen-

- eral meshes. *Computer Graphics Forum (Proceedings of Eurographics)*, 25(3), 2006.
- [GHDS03] Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of SCA*, 2003.
- [GI04] Jack Goldfeather and Victoria Interrante. A novel cubic-order algorithm for approximating principal direction vectors. *ACM Transactions on Graphics*, 23(1):45–63, 2004.
- [GL81] A. George and J. W. H. Liu. *Computer solution of large sparse positive definite matrices*. Prentice Hall, 1981.
- [GL96] G. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins Press, 3 edition, 1996.
- [GMW89] P. E. Gill, W. Murray, , and M. Wright. *Practical Optimization*. Academic Press, London, 1989.
- [GPF09] Aleksey Golovinskiy, Joshua Podolak, and Thomas Funkhouser. Symmetry-aware mesh processing. *Mathematics of Surfaces*, 2009.
- [GSC⁺02] James Glymph, Dennis Shelden, Cristiano Ceccato, Judith Mussel, and Hans Schober. A parametric strategy for freeform glass structures using quadrilateral planar facets. In *Acadia*, pages 303–321, 2002.
- [GZ08] Yotam Gingold and Denis Zorin. Shading-based surface editing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3), 2008.
- [HG00] K. Hormann and G. Greiner. MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*. Vanderbilt University Press, 2000.
- [HLS07] K. Hormann, B. Levy, and A. Sheffer. Mesh parameterization: theory and practice. In *ACM SIGGRAPH Course Notes*, 2007.
- [Hor01] K. Hormann. *Theory and Applications of Parameterizing Triangulations*. PhD thesis, Department of Computer Science, University of Erlangen, 2001.

Bibliography

- [HP04] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum*, 23(3), 2004.
- [HS02] Eyal Hameiri and Ilan Shimshoni. Estimating the principal curvatures and the darboux frame from real 3d range data. *Symposium on 3D Data Processing Visualization and Transmission*, page 258, 2002.
- [HUY95] J.-B. Hiriart-Urruty and D. Ye. Sensitivity analysis of all eigenvalues of a symmetric matrix. *Numer. Math.*, 70(1):45–72, 1995.
- [IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1999.
- [JKS05] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum (Proc. Eurographics)*, 24(3):581–590, 2005.
- [Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer a. System Sciences*, 9:256, 1974.
- [KCH02] M. S. Kim, D. H. Choi, and Y. Hwang. Composite non-smooth optimization using approximate generalized gradient vectors. *J. Optim. Theory Appl.*, 112(1):145–165, 2002.
- [KCVS98] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1998.
- [Kolo3] Branko Kolarevic. *Architecture in the digital age: design and manufacturing*. Taylor & Francis, 2003.
- [KSNS07] E. Kalogerakis, P. Simari, D. Nowrouzezahrai, and K. Singh. Robust statistical estimation of curvature on discretized surfaces. In *Symposium on Geometry Processing*, pages 13–22, 2007.

- [LBS05] Torsten Langer, Alexander G. Belyaev, and Hans-Peter Seidel. Asymptotic analysis of discrete normals and curvatures of polylines. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 229–232, 2005.
- [LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3):1–10, 2008.
- [LPW⁺06] Yang Liu, Helmut Pottmann, Johannes Wallner, Y.-L Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):681–689, 2006.
- [LSCO⁺04] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Differential coordinates for interactive mesh editing. In *Proceedings of SMI*, 2004.
- [LSLCO05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3), 2005.
- [LSSK09] Bao Li, Ruwen Schnabel, Jin Shiyao, and Reinhard Klein. Variational surface approximation and model selection. *Proceedings of Pacific Graphics*, 28(7):1985–1994, 2009.
- [Mar09] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall, 2009.
- [MCW01] K.T. Miura, Fuhua Cheng, and Lazhu Wang. Fine tuning: curve and surface deformation by scaling derivatives. In *Proceedings of Pacific Graphics*, 2001.
- [MDSB02] M. Meyer, M. Desbrun, P. Schroeder, and A. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *VisMath.*, 2002.
- [MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3D geometry.

Bibliography

- ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):560–568, 2006.
- [MGP07] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Symmetrization. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3):63, 2007.
- [MHIL02] Kwan-Liu Ma, Aaron Hertzmann, Victoria Interrante, and Eric B. Lum. Recent advances in non-photorealistic rendering for art and visualization. *ACM SIGGRAPH Course Notes*, 2002.
- [MNT04] K. Madsen, H.B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems. Technical report, Technical University of Denmark, 2004.
- [MS92] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1992.
- [Nie99] H. Nielsen. Damping parameter in marquardt’s method. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 1999.
- [NISA07] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3), 2007.
- [Nvio5] Nvidia. Nvidia - cuda programming guide, 2005.
- [OBS02] Y. Othake, A. Belyaev, and H-P. Seidel. Mesh smoothing by adaptive and anisotropic gaussian filter. *Vision, Modeling, and Visualization*, 2002.
- [PFT04] H.B. Nielsen P.E. Frandsen, K. Jonasson and O. Tingleff. Unconstrained optimization. Technical report, Technical University of Denmark, 2004.
- [PHKo8] Helmut Pottmann, Michael Hofer, and Axel Kilian, editors. *Advances in Architectural Geometry*. Vienna, 2008.

- [PLW⁺07] Helmut Pottmann, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. Geometry of multi-layer freeform structures for architecture. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3):65, 2007.
- [PMW⁺08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottman, and Leonidas J. Guibas. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3):43, 2008.
- [PP93] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1), 1993.
- [PS07] J. Pushkar and C. Sequin. Energy minimizers for curvature-based surface functionals. *Computer-Aided Design and Applications*, 2007.
- [PSB⁺08] Helmut Pottmann, Alexander Schiftner, Pengbo Bo, Heinz Schmiehofer, Wenping Wang, Niccolo Baldassini, and Johannes Wallner. Freeform surfaces from single curved panels. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3):76, 2008.
- [PSG⁺06] Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz, and Thomas Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):549–559, 2006.
- [PW01] Helmut Pottmann and Johannes Wallner. *Computational Line Geometry*. Springer-Verlag New York, Inc., 2001.
- [Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*, pages 486–493, 2004.
- [Rus06] Andrzej Ruszczyński. *Nonlinear Optimization*. Princeton University Press, 2006.

Bibliography

- [SCOL⁺04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Symposium on Geometry Processing*, 2004.
- [SF98] Karan Singh and Eugene Fiume. Wires: a geometric deformation technique. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1998.
- [SG03] V. Surazhsjy and C. Gotsman. Explicit surface remeshing. In *Symposium on Geometry Processing*, pages 20–30, 2003.
- [SG04] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Gener. Comput. Syst.*, 20(3):475–487, 2004.
- [Shao6] Ariel Shamir. Segmentation and shape extraction of 3d boundary meshes. *Eurographics State-of-the-Art Report*, 2006.
- [Shao8] Ariel Shamir. A survey on mesh segmentation techniques. In *Computer Graphics Forum*, volume 27, pages 1539–1556, 2008.
- [She02] Dennis Shelden. *Digital surface representation and the constructibility of Gehry's architecture*. PhD thesis, M.I.T., 2002.
- [SPR06] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis.*, 2(2):105–171, 2006.
- [SS10] M. Singh and S. Schaefer. Triangle surfaces with discrete equivalence classes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3), 2010.
- [SSPo8] B. Springborn, P. Schröder, and U. Pinkall. Conformal equivalence of triangle meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3), 2008.
- [Tau95a] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *International Conference on Computer Vision*, page 902, 1995.

- [Tau95b] Gabriel Taubin. A signal processing approach to fair surface design. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 351–358, 1995.
- [TGRZ07] E. Tosun, Y. I. Gingold, J. Reisman, and D. Zorin. Shape optimization using reflection lines. In *Symposium on Geometry Processing*, 2007.
- [TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, page 839, 1998.
- [TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 205–214, 1987.
- [VM02] Tamas Varady and Ralph Martin. Reverse engineering. In *Handbook of CAGD*, pages 651–681, 2002.
- [Wei09] Rolf Weilenmann. Interactive constraint-based modeling. Master’s thesis, ETH Zurich, 2009. Supervised by Michael Eigensatz and Mark Pauly.
- [WF86] R S Womersley and R Fletcher. An algorithm for composite nonsmooth optimization problems. *J. Optim. Theory Appl.*, 48(3):493–523, 1986.
- [WK05] Jianhua Wu and Leif Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum (Proceedings of Eurographics)*, 24(3):277–284, 2005.
- [WOD09] Emily Whiting, John Ochsendorf, and Frédo Durand. Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 28(5):112, 2009.
- [WW92] William Welch and Andrew Witkin. Variational surface modeling. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1992.

Bibliography

- [YLW06] Dong-Ming Yan, Yang Liu, and Wenping Wang. Quadric surface extraction by variational shape approximation. In *GMP*, pages 73–86, 2006.
- [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 Transactions on Graphics (Proceedings of SIGGRAPH)*, 23(3), 2004.
- [ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1997.

Curriculum Vitae



MSc CS Michael Eigensatz

Personal Data

16 December 1979 Born in Lucerne, Switzerland
Nationality Swiss

Education

2 July 2010 Ph.D. defense.
Mai 2006 – Jun. 2010 Research assistant and Ph.D. student at the Applied Geometry Group of ETH Zurich, Switzerland, under the supervision of Prof. Mark Pauly.
April 2006 Degree Master of Science ETH in Computer Science.
Oct. 2000 – Apr. 2006 Masters Studies of Computer Science, Major in Computational Sciences, Minor in Artificial Intelligence.
March 2000 Swiss Federal Matura. Type C: specialization in sciences.
1998 – 2000 Rudolf Steiner School Schloss Glarisegg, Switzerland
1986 – 1998 Rudolf Steiner School Baar, Switzerland

Scientific Publications

Michael Eigensatz, Mario Deuss, Alexander Schiffner, Martin Kilian, Niloy Mitra, Helmut Pottmann, and Mark Pauly. Case Studies in Cost-Optimized Paneling of Architectural Freeform Surfaces. *Advances in Architectural Geometry*, 2010.

Michael Eigensatz, Martin Kilian, Alexander Schiffner, Niloy Mitra, Helmut Pottmann, and Mark Pauly. Paneling Architectural Freeform Surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3), 2010.

Michael Eigensatz and Mark Pauly. Positional, Metric, and Curvature Control for Constraint-Based Surface Deformation. *Computer Graphics Forum (Proceedings of EUROGRAPHICS)*, 28(2), 2009.

Michael Eigensatz, Robert W. Sumner, and Mark Pauly. Curvature-Domain Shape Processing. *Computer Graphics Forum (Proceedings of EUROGRAPHICS)*, 27(2), 2008.

Michael Eigensatz, Joachim Giesen, and Madhusudan Manjunath. The Solution Path of the Slab Support Vector Machine. *Canadian Conf. on Computational Geometry*, 2008.

