DIFFERENTIAL METHODS FOR INTUITIVE 3D SHAPE MODELING

by

HONGBO FU

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science and Engineering

July 2007, Hong Kong

Authorization

HONGBO FU

DIFFERENTIAL METHODS FOR INTUITIVE 3D SHAPE MODELING

by

HONGBO FU

This is to certify that I have examined the above Ph.D. thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.

DR. CHIEW-LAN TAI, THESIS SUPERVISOR

PROF. LIONEL NI, HEAD OF DEPARTMENT

Department of Computer Science and Engineering

20 July 2007

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge the great impact of my advisor and friend, Chiew-Lan Tai, on my research, on this thesis, and on me personally. Her near-infinite patience saw all manner of my wayward ideas, infeasible schemes, and frivolous diversions, yet her subtle guidance shaped them into a coherent path.

Many thanks to my thesis committee members, Beifang Chen, Pheng-Ann Heng, Huamin Qu, and Long Quan, for their patience in reading this thesis and the helpful comments. This thesis would not been possible in this form without the support and collaboration of several people, in particular Oscar Au, Yichen Wei, Daniel Cohen-Or, Shimin Hu, Ligang Liu, Hongxin Zhang, and Xiaohuang Huang. My thanks also go to the dear communication tutor, Shauna Dalton, for the great help all along in improving my research papers.

The team of the VisGraph Lab at the Hong Kong University of Science and Technology supported me a lot by providing valuable feedback in many fruitful discussions. Special thanks go to Pedro Sander, Chi-Keung Tang, Philip Fu, Albert Chung, Nelson Chu, Hui Zhao, Gang Zeng, Ruonan Pu, Hong Zhou, Ping Tan, Jia Chen, Kai Zhang, Jingdong Wang, Hongwei Li, Yingcai Wu, and Lu Yuan.

Finally, I am forever grateful to my parents and my elder sister, who have always been motivating and supporting me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DIFFERENTIAL METHODS FOR INTUITIVE 3D SHAPE MODELING

ABSTRACT

Modeling 3D digital shapes, such as curves and meshes, manually is challenging for three main reasons. First, realistic shapes are generally complex and involve many unknown degrees of freedom. Second, users usually only have 2D input devices, which are inadequate for manipulating 3D shapes. Third, unlike sculptors, ordinary people usually have no precise space perception, making precisely adjusting relative positions of 3D shapes even harder.

This thesis presents several novel intuitive techniques that make the modeling process significantly less labor-intensive. We achieve a balance between the control intuitiveness of tools and the geometric complexity of modeling output. Our techniques allow the user to intuitively control the modeling effect through only a small set of easy-to-use manipulators. The unknown vertex positions of the final models are then computed by solving a system of linear partial differential equations (i.e., the Laplace or Poisson equations) subject to the constraints derived from the manipulators. These differential-based techniques contribute to three important modeling applications: mesh deformation, mesh merging and hairstyle design. The differential mesh deformation technique allows the user to manipulate a small set of handles to deform highly detailed meshes interactively and to achieve physically plausible deformation effects. The differential mesh merging framework relieves the user's burden of both precise locating of 3D models and precise specification of merging boundaries over the meshes to be merged. For hairstyle modeling, we design a sketching interface and adapt an incremental solver to allow users to design compelling hairstyles by drawing only a small set of strokes. For all these algorithms, we pre-compute the most computationally expensive components to achieve interactive modeling with fast response. Extensive experiments demonstrate the robustness and usefulness of our techniques.

CHAPTER 1

INTRODUCTION

3D geometric models have been used extensively in the entertainment industry, particularly in the production of movies and video games. Due to their powerful ability to encode highly detailed geometric features, polygonal meshes and polygonal curves are by far the most common representations for discrete surfaces and curves, respectively. For example, to model a digital human character, polygonal curves are often used for hair curves, and polygonal meshes for the human body surface. Although a lot of techniques and devices, such as procedural modeling [5, 6], image-based modeling [7, 8, 9, 10] and 3D scanning device [11, 12, 13], have been proposed to construct 3D shapes, most 3D shapes have to be modeled via user input, either from scratch or by editing / modifying existing models.

Modeling 3D digital shapes manually, however, is a tedious and difficult task, primarily due to the following three reasons. First, many realistic shapes are generally complex in terms of both geometric and topology complexities. For example, there is a large volume of hair with typically over 100,000 hair curves on a human head. Clearly, it is impractical to model the geometry of individual hair curves. Second, most computer user interfaces are based on the keyboard and mouse, which are two dimensional input devices, thus making free manipulation of shapes in three dimensional spaces very cumbersome. Third, unlike sculptors, most people do not have precise space perception, causing both accurate locating of shapes and precise adjusting of relative shape locations in a 3D virtual world even harder. Although there exist many professional modeling packages (e.g., Maya [14], 3DS Max [15], LightWave [16], ZBrush [17], Rhino [18]), the above difficulties limit the complex modeling process to a small group of professional digital artists with extensive practical modeling experience. Even for these professionals, modeling a complex shape is still rather tedious and labor-intensive.

Our ultimate goal is to make 3D shape modeling more intuitive to control and easier to use even for non-professionals. To achieve this goal, we offer a balance between intuitive user control and modeling complexity: rather than letting the user bother with all the primitive elements which constitute a desirable shape, we provide a small set of manipulators, typically small portions of the primitive elements, to the user for controlling the modeling effect, and employ the underlying algorithms to optimize the positions of the rest of the primitive elements of the final shape. In other words, the modeling complexity is largely determined by the number of manipulators. With this basic idea in mind, we propose several novel intuitive techniques for surface modeling by editing existing meshes, including mesh deformation and mesh merging, and designing hairstyles from scratch. *Handles* (i.e., parts of a surface to be manipulated) and *strokes* are the respective manipulators for surface modeling and hairstyle design. The output of the optimizations constrained by the manipulators is easily predictable, making the whole modeling processing intuitive to control.

The main ingredient of our underlying algorithms is a system of linear partial differential equations:

$$\Delta(\mathbf{p}_i) = \delta_i, \tag{1.1}$$

subject to certain modeling constraints derived from the user-controlled manipulators, where $\Delta(\cdot)$ is the *discrete Laplace operator* (a second-order differential operator), also known as the *discrete Laplacian*. For the applications of mesh deformation and mesh merging, $\mathbf{p}_i$ are the (unknown) vertex positions of the desired deformed surface and δ_i are the so-called *differential coordinates* which are obtained by applying a differential operator, e.g., the discrete Laplace operator, to the vertex positions of the original unmodified surface. In this setting, δ_i are generally nonzero and thus the resulting system is essentially the Poisson equation in mathematics. For hairstyle modeling, $\mathbf{p_i}$ are the tangential directions of hair curves and $\delta_i \equiv \mathbf{0}$, making Equation 1.1 reduce to the Laplace equation.

The advantages of using Equation 1.1 to govern the modeling effect are manifold. First, solving this type of linear systems is fast since the associated system matrices are very sparse and either remain unchanged or undergo undergo little change during editing, thus making pre-computing the most computationally expensive components feasible at the initialization stage. The availability of highly-optimized sparse linear solvers makes the algorithms easy to implement and efficient. Second, the systems are robust and generally have unique global minimum since solving them is equivalent to minimizing well-defined quadratic energies (i.e., $\sum_i ||\Delta(\mathbf{p}_i) - \delta_i||^2$). Third, if the modeling constraints are carefully designed, the modeling effect is a smooth function of the modeling constraints, making the influence propagated from the constraints to the rest of the shape under modeling easily predictable. On the other hand, the smoothness of the modeling function of the constraints limits our methods to applications of low-frequency modeling of shape (i.e., low-frequency deformations of meshes possibly with highly detailed meshes and global shape design of complex hairstyles).

1.1 Contributions

The main contributions of this thesis are as follows:

- We present a novel linear differential mesh editing framework. To our best knowledge, our method provides the most effective linear approximation to the essentially nonlinear differential mesh deformation problem. Our deformation tool allows the user to intuitively design natural deformation effects through only a small set of handles which are directly defined over the manipulated mesh and are easy to control.

- We contribute to Poisson mesh merging in two aspects. We introduce the notion of *configuration-independent merging*, which produces the same merging result independent of the relative positions, orientations, scales of input meshes. Unlike existing techniques, configuration-independent merging does not involve the tedious adjustment of relative locations of input meshes. We also propose to optimize the merging boundaries such that the user only needs to *roughly* specify the merging boundaries on the meshes to be composed and our algorithm computes the final merging boundaries which lead to merging results with minimal distortions.

- We design an intuitive sketching interface for interactive hairstyle design. The user portrays the global shape of a desired hairstyle through several style primitives (e.g., stream curve, dividing curve, ponytail), manipulated by interactively sketching freeform strokes. We present an efficient incremental updating scheme of the numerical system, which gives rise to a sketching interface that provides interactive visual feedback to the user. Interesting hairstyles can be created in minutes.

1.2 Outline

The thesis is organized as follows:

Chapter 2 reviews the existing differential mesh deformation methods and presents our new linear differential mesh deformation framework.

Chapter 3 introduces two intuitive techniques for Poisson mesh merging.

Chapter 4 presents our sketching interface for interactive hairstyle design.

Chapter 5 concludes the thesis and gives several directions for future work.

CHAPTER 2

INTUITIVE MESH DEFORMATION

In this chapter, we introduce an intuitive mesh deformation framework which is based on a novel effective linear approximation solution to the nonlinear differential mesh deformation problem. With our deformation tool, the user is allowed to interactively deform existing highly-resolutions meshes through the manipulation of a small set of easy-to-use manipulators.

2.1 Introduction

The state-of-the-art 3D scanning devices allow fast and high-fidelity scanning of real-world objects into digital counterparts. However, it is often the case that there exists no real-world equivalent of a desired model. Then, modeling artists have to create digital models manually either from scratch or by modifying existing ones. This chapter focuses on polygonal modeling through deforming existing polygonal meshes, like those produced by scanning real-world objects. Polygonal meshes are basically a group of polygons which are connected together by shared vertices and they are by far the most common representation to represent or approximate object surfaces in 3D computer graphics. Complex polygonal meshes have a very large set of polygons as well as vertices. Hence, it is challenging and tedious to deform complex meshes by manually manipulating individual vertices or polygons.

Desirable shape deformation techniques for digital artists generally should possess three properties: easy-to-control, interactive, and detail-preserving. The first property means that a desirable deformation technique should provide simple means to let the user intuitively control deformation effects and make deformation effects easily predictable. This property is usually achieved by striking a balance between intuitive user control and modeling complexity. Unlike surgical applications which demand precise results, tools for artists prefer a satisfactory interactivity to precision and enable editing of shapes in interactive sessions. The detail-preserving property sets the criteria for physically plausible and visually pleasing deformations: local features should be changed as rigidly as possible; the differences of changes (deformations) of neighboring local features should be as small as possible (in other words, local deformations should be locally as similar as possible).

Since desirable deformation techniques generally allow only smooth deformations, that is, low-frequency (global) geometry to be changed during deformation, this motivates existing deformation techniques to constrain deformations of a model with large degree of freedom either by a subspace with small degree of freedom or by a smooth function (see the details in Section 2.2). The user is allowed to manipulate the subspace or the smooth function to directly or indirectly control deformations of the original shape. Due to the great reduction of DOF, the deformation process becomes significantly less lab-intensive. Clearly, deformation quality is highly dependent on the choices of subspace or function.

Recently, several differential mesh editing frameworks have been proposed for interactive mesh deformation. They formulate the mesh deformation problem as a global variational minimization problem whose solution is the desirable deformed surface. The rationale of differential mesh editing is to use the so-called differential coordinates (e.g., the Laplacian coordinates [3, 19] and the gradient fields [20]) to represent local surface details and to achieve smooth deformation by solving the Poisson equations which constrain the deformation smoothness for neighboring features. The resulting editing tools [19, 20] allow the user to simply manipulate parts of a mesh, called the handles, and the rest of the surface is reconstructed by minimizing the differential coordinates before and after editing. However, as the differential coordinates are defined in the global coordinate system, they are neither scale invariant nor rotation invariant [3]. Therefore the main challenge of differential editing frameworks is to find appropriate local transformations such that the transformed differential coordinates fit the orientations of the details in the deformed surface. To preserve local details, appropriate transformations are required as rigid as possible or as close to similarity transformations as possible if uniform scaling is allowed. As 3D rigid or similarity transformations are nonlinearly dependent on (unknown) vertex positions, differential mesh editing is thus essentially a nonlinear problem.

For fast computation, most existing solutions approximately linearize the original nonlinear problem. They either explicitly define the transformations at all the unconstrained vertices by interpolating the user-specified transformations at the constrained vertices [20, 4, 21] or implicitly define the transformations with respect to the (unknown) deformed surface [3, 19, 22]. However, these methods only partially solved the problem: explicit methods suffer from the serious translation-insensitivity problem (i.e., the inability of inferring the rotation information if the handles are only translated), while implicit methods cannot tackle large angle deformation.

In this thesis, we present a novel implicit differential editing framework [23], which is linear and able to effectively capture the local rotation transformations for the local features during editing. Like [19], the ultimate goal of our framework is to find a similarity transformation consisting of a rigid transformation and a uniform scaling. To achieve interactive editing, like most previous Laplacian editing frameworks, we aim to formulate a linear reconstruction problem. Since rigid transformations are nonlinearly dependent on vertex positions in 3D space, we resort to affine transformations, which linearly depend on vertex positions. By enforcing neighborhood coherence and user-specified boundary constraints, we solve for the affine transformations in a least-squares sense. The downside of using affine transformations is the introduction of shearing distortion. To remove shearing distortion, we perform polar decomposition to extract from each of the solved affine transformations a similarity transformation consisting of only rotation and uniform scaling.

By binding each differential coordinate with the corresponding similarity transformation, our method appropriately orients and scales the local features during editing. In other words, the changes of the local features are implicitly captured by the similarity transformations. Our framework produces more visually pleasing editing results than previous related work for deformation with large angle rotations and/or big-scale translations of handles. To make comparable deformation results, our deformation tool needs fewer handles than existing methods and thus is easier to use.

2.2 Related Work

Mesh deformation has been an active area of research in digital geometry processing, due to its wide applications in industrial and artistic design. A lot of mesh deformation techniques have been proposed to directly deform highly detailed meshes which are often irregular in terms of both sampling and connectivity. In this review, according to whether the domain of deformation about the original mesh is defined / constrained through a subspace or a smooth function, we divide the existing techniques into two categories: subspace-based shape deformation and function-based shape deformation, and discuss the strengths and weaknesses of each technique.

Our review here focuses on geometrically-based deformations methods rather than physically-based deformable models (see [24] and references therein). We also do not discuss techniques which are designed mainly for intuitive surface modeling from scratch. The inclined reader is therefore encouraged to browse more recent literature on, for example, space-warp operator [25], vector field based shape deformations [26, 27], sweep-based modeling metaphor [28], sketch-based freeform surface design [29, 30, 31, 32].

2.2.1 Subspace-Based Shape Deformation

Subspace-based shape deformation employs a reduced domain (subspace) to model the low-frequency deformation of the original mesh. The user controls deformation by manipulating the subspace and the final deformation is reconstructed from the deformed subspace. There are a variety of ways to construct such a deformation space. For example, it can be control lattices used in free-form deformation, multilevel base meshes in multiresolution mesh editing or skeletal bones in skeletal subspace deformations.

Free-Form Deformations (FFDs). FFDs [33] and their variants [34, 35, 36, 37, 38] embed a model to be deformed into a lattice and define a mapping which transfers deformations from the lattice to the model. The user is then allowed to control the deformation of the model by manipulating the control lattice. The deformation complexity is determined by the density of the control lattice. FFDs only needs a very coarse regular control lattice to create coarse-scale deformations of a model. However, for finer-scale deformations, a very dense control lattice is usually required. It is often easier to deform the underlying geometry directly than to manipulate a dense control lattice. In addition, since it is cumbersome to construct a control lattice which reflects the geometry of the embedded model, the mapping from the lattice to the model is often defined without considering the embedded model geometry. Therefore, like other space-based deformation methods, FFDs do not generate intuitive results for certain

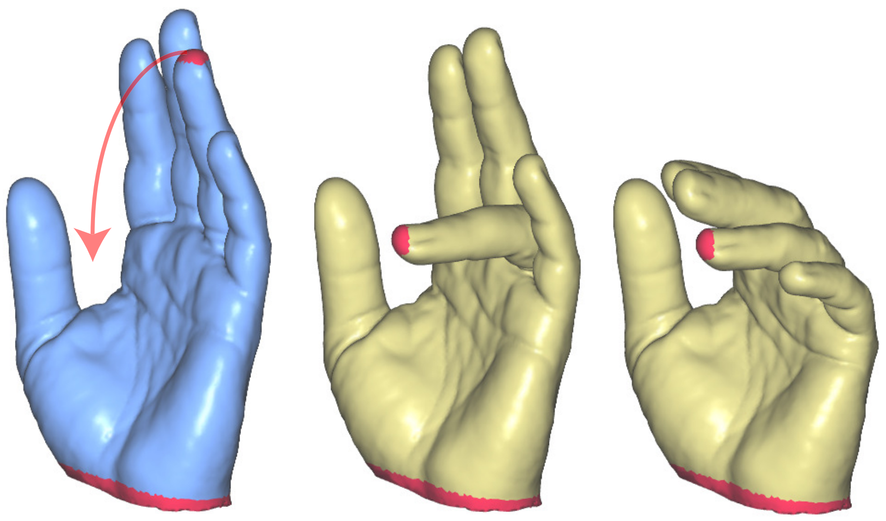


Figure 2.1: Manipulating the ring finger (**Left**). surface-based deformation [1] (**Middle**) allows more intuitive editing results than space-based deformation [2] (**Right**).

editing scenarios (see an example in Figure 2.1): FFDs may incorrectly assign heavy influence to regions that are spatially close to, but geodesically far from, a manipulated control lattice vertex. Despite their deficiencies, FFDs are often equipped with professional modeling tools (e.g., 3ds Max [15], Maya [14]), as they are sufficiently fast and do not put any constraints on the representations of the embedded models (e.g., irregular meshes, point clouds, parametric surfaces).

Singh [39] proposes to use domain curves, called wires, to define the domain of deformation about an object. Unlike the control lattices in FFDs, wires tracks deformable features of an object as such they provide a coarse geometric representation of an object and a means to deform it. Like FFDs which let the user employ the control lattice to drive the deformation of the object being deformed, Singh's method only allows the user to interact solely with the wires rather than directly the object. Recently, Sumner et al. [40] propose to use the so-called deformation graph, a more general deformation domain, for intuitive deformation for a wide range of shape representations and editing scenarios. Their method supports direct manipulation of a mesh being deformed and makes the deformation graph transparent to the user.

Skeletal Subspace Deformations (SSDs). SSDs [41, 42, 43, 44, 45] are by far the most popular methods for real-time animation and interactive posing of articulated models. They can also be applied to a wide range of soft objects, for example, cloth simulation [46]. Inspired by real-world skeletons of models, SSDs bind skeletal structures which are usually represented as tree structures whose nodes are identified with the joints (given by their positions and orientations) and edges with the bones, with polygonal meshes, called skins. A skeleton provides the domain of deformation about a skin mesh. Instead of animating each skin vertex individually, it is sufficient to manipulate the skeleton, and the skin deforms automatically. Automatic skeleton extraction purely according to the geometric information of a given mesh is rather difficult. The topology of the extracted skeleton is often not satisfactory since the extraction process is sensitive to the shape perturbation of the skin surface. Instead, Baran and Popović [47] present a method to automatically fit a given template skeleton with the fixed topology to a mesh. Schaefer and Yuksel [48] propose to extract a skeleton from a set of example shapes using the hidden motion information.

The most popular way to build the skeleton-to-skin binding is to bind each skin vertex to several neighboring / closest joints, a.k.a., single weight enveloping. The vertex weights which denote the amount of influence of individual joints must be specified during the so-called skinning process. The deformation of each vertex is then defined as a weighted blending of the transformations of its associated joints. Specifying appropriate skinning weights manually could be a very tedious task, which is often achieved by a painting interface in professional packages. Several approaches [45, 43, 48] propose to compute skinning weights from examples. Based on the observation that each bone only influence nearby skin vertices, the locality of skinning weights, Baran and Popović [47] and Weber et al. [49] compute the weights by solving the Laplace equation with carefully designed boundary constraints. Directly blending transformation matrices linearly causes serious artifacts (e.g., collapsing artifacts) for certain postures. Later, several advanced blending methods, such as, log-matrix blending [50], spherical blending skinning [51] and dual quaternion blending [52], have been proposed to alleviate the artifact problems. Most of these blending methods can be accelerated by graphics hardware [53].

James and Twigg [54] propose to extract generalized bones from given mesh animation sequences for the main purpose of fast GPU-based rendering. In their method, the extracted bones are *quasi-rigid* and thus are not suitable for arbitrary deformations. Kavan et al. [55] present a method of skinning *arbitrary* deformations for existing animations. Motivated by [54], Der et al. [56] employ generalized bones to construct a reduced domain for mesh-based inverse kinematics [57].

Multiresolution Mesh Deformation. Multi-resolution frameworks were first designed for spline surfaces [58], and later applied to subdivision surfaces and irregular meshes [59, 60, 61, 62]. Unlike the FFD and SSD techniques, which are essentially two-level models (one level is the original polygonal mesh and the other is the lattice/skeleton), multiresolution-based techniques decompose the mesh into multiple levels—coarser or smoother versions of the original mesh. The differences between successive levels are defined as local features. They are encoded with respect to the local frames of the lower level mesh. The details are transformed according to the changes in the local frames of the lower level mesh during editing. Compared with two-level deformation models which are mainly for global deformations of shapes, multiresolution methods are capable of both global and local deformations, which are achieved by allowing the user to choose an appropriate editing level of detail.

However, as details are defined and reconstructed locally, careful design of detail encoding and reconstruction is crucial to avoid artifacts (e.g., self-intersection, shearing) which often appear when the base mesh endures large deformations (especially bending). Several detail encoding methods, for example, normal displacement vectors [61], displacement prisms [63], local affine transformations [64], have been proposed. It is noteworthy that the transformation-based encoding scheme proposed by Xu et al. [64] incorporate the advantages of multiresolution frameworks and differential mesh editing frameworks into a unified framework. Botsch et al. [65] adopt a similar strategy by employing deformation gradients [66] to represent local details between two levels of details.

2.2.2 Function-Based Shape Deformation

The techniques belonging to this category either use certain functions (e.g., radial basis functions [67], moving least squares [68]) to model shape deformations explicitly or formulate deformation functional as a solution to variational problems (e.g., the differential-based shape deformation techniques (see [69, 70] and references therein)).

Deformations of a shape can be explicitly modeled by a function f that maps vertices in the undeformed shape to the deformed shape. Applying the function f to each point in the undeformed shape creates the deformed shape. The user selects parts of the shape to control the deformation. To get intuitive deformation, the desirable function should satisfy the following properties. First, the function should not influence the handles which are fully controlled by the user. Second, the function should be a smooth function, ensuring smooth deformation. Third, the function must remain the shape unchanged when the handles are not moved. Radial basis functions [67] and moving least squares [68] are two examples of functions which satisfy the above properties and have been used to define shape deformations. However, as these functions are chosen without respecting the geometry of the shape to be deformed, these techniques suffer from the same problem as other space-based deformation techniques.

Differential Mesh Editing. From the perspective that deformations of neighboring features should be as similar as possible, several variational deformation methods have been proposed. The rationale of these methods is to use the so-called differential coordinates (e.g., the Laplacian coordinates [19], the gradient fields [20], pyramid coordinates [71, 72]) to encode local details and to formulate the final deformed mesh as a solution to a variational problem in which the differences of the differential coordinates before and after editing, are minimized. The variational problem is subject to the constraints derived from the handles through which the user controls the deformation effects. As the differential coordinates are nonlinearly dependent on the unknown vertex positions, the resulting variational problem is nonlinear. How to solve the nonlinear global optimization efficiently is challenging.

For fast computation, most of earlier work linearized the problem by replacing the nonlinear dependence with an explicit process of transformations propagation from handles [20, 3, 4, 21, 73, 74, 65]. For each vertex, the relative degree of propagation from a handle is measured by either uniform weights (defined by geodesic distances [20] or handle-driven harmonic fields [4]) or non-uniform weights [75, 76]. The linearized global optimization is equivalent to solving a sparse linear system of equations, which can be solved efficiently using advanced sparse linear solvers (e.g., TAUCS [77], CHOLMOD [78]). Botsch et al. [79] give the detailed discussions on efficient linear system solvers for mesh processing. These methods can be further accelerated by using multiresolution frameworks [72] or multigrid methods [74].

Linear approximation methods in general suffer from serious artifacts under large-scale deformation. More recently, several iterative frameworks [80, 81, 2, 82, 83] to solve the original nonlinear deformation problem have been proposed, which are carefully designed to achieve interactive editing. Solving the linear or nonlinear deformation problem directly over the original mesh domain is not scalable to large-scale models due to memory bottleneck, expensive per-iteration cost, and slow convergence. To allow scalability, reduced models [2, 1] which formulate the original deformation problem with a small set of control variables have been proposed. In such reduced deformable models, the deformation of the original mesh is limited by the degree of freedom of the reduced domain.

Differential mesh editing techniques are designed without considering material properties of the mesh being deformed. Hence, in a sense, differential mesh editing itself is not sufficient to model physically plausible deformation of objects with materials. Popa et al. [76] present a material-aware differential mesh editing method by letting the user provide certain material information through painting. Combing differential mesh editing with skeleton-based deformation has proved very effective to characterize deformations of character animation in recent literature [84, 49, 85].

Nealen et al. [22] design a sketching interface for differential mesh editing [19]. Instead, we are more interested on the design of effective differential mesh editing algorithms themselves. Our method presented in Section 2.4 allows more intuitive deformation of models with the same user interface as the existing differential mesh editing techniques [20, 3, 4, 21, 73, 19], which are reviewed systematically in the next section.

2.3 Review on Differential Mesh Editing

Recently, intrinsic differential representations (the Laplacian coordinates [3, 19, 4], the gradient field [20] and the first/second fundamental surface forms [73]) have been adopted to mesh editing because the resulting editing tools are intuitive and detail-preserving. Using these tools, the user can interactively edit a region of interest (ROI) by manipulating a small set of handles. As our framework is based on the Laplacian coordinates, this discussion focuses on previous Laplacian-based work.

The Laplacian coordinate (LC) δ_i at vertex $\mathbf{v}_i$ ($1 \leq i \leq n$) is defined as follows [3, 19]:

$$\delta_i = D(\mathbf{v}_i) = \sum_{j \in N(i)} w_{ij}(\mathbf{v}_j - \mathbf{v}_i),$$

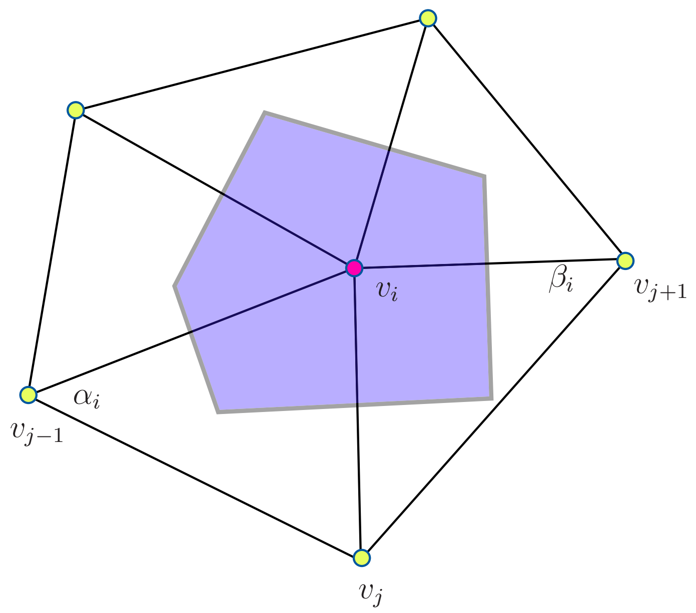


Figure 2.2: 1-ring neighborhood of vertex v_i is used to define the discrete Laplacian coordinate δ_i at v_i.

where $N(i)$ is the index set of the 1-ring neighboring vertices of $\mathbf{v}_i$ and w_{ij} is the weight of the edge (i, j). The resulting LC is essentially a 3D vector in the global coordinate system. Two common weighting schemes are uniform weighting [3, 19]

$$w_{ij} = \frac{1}{|N(i)|}$$

and cotangent weighting [86, 87]

$$w_{ij} = \frac{1}{|\Omega_i|}(\cot \alpha_i + \cot \beta_i),$$

where $|\Omega_i|$ is the area of the Voronoi cell associated with $\mathbf{v}_i$ (the shaded area in Figure 2.2), and α_i, β_i are the two angles opposite to the edge (i, j).

The Laplacian coordinates have nice mathematical properties: their directions approximates the vertex normals and their lengths approximate the mean curvature at the corresponding vertices. Therefore, the most straightforward way to formulate mesh deformation using the Laplacian coordinates is to minimize the change of the Laplacian coordinates before and after editing, subject to given modeling constraints:

$$D(\tilde{\mathbf{v}}_i) = \delta_i, \quad 1 \le i \le n, \tag{2.1}$$

where $\tilde{\mathbf{v}}_i$ are unconstrained vertices (to be solved) and δ_i are the Laplacian coordinates constructed over the original undeformed mesh. However, since the Laplacian coordinates are essentially vectors encoded in a global coordinate system and they are only translation-invariant rather than rotation-invariant, the above Naïve Laplacian editing attempts to preserve the original global orientations of the local details, thus easily suffering from serious artifacts. Desirable transformations are needed to transform the Laplacian coordinates before reconstructing the deformed surface from them.

In general, Laplacian mesh editing aims to solve the above transformation problem can be formulated as the following system

$$D(\tilde{\mathbf{v}}_i) = \mathbf{M}_i \delta_i, \quad 1 \leq i \leq n, \tag{2.2}$$

where $\mathbf{M}_i$ is a 3×3 transformation matrix. To avoid having the LCs deviate from the normal directions, $\mathbf{M}_i$ is mostly required to be a combination of rotation and uniform scaling [19]. The method of Poisson mesh editing [20] also results in a linear system similar to Equation 2.2, since in theory the gradient operator followed by the divergence operator is equivalent to the Laplace operator.

Depending on whether $\mathbf{M}_i$ is defined with respect to the (unknown) deformed surface, existing differential based mesh editing can be classified as implicit methods [3, 19] or explicit methods [20, 4, 21]. Explicit methods define $\mathbf{M}_i$ without considering the deformed surface. Instead, $\mathbf{M}_i$ is defined by propagating the transformations at the handles to all the unconstrained vertices, weighted by geodesic distances [20, 21] or a set of harmonic fields [4]. These methods can produce good deformation when the handles undergo large-angle rotations, including rotation angles greater than 2π. However, if the handles are only translated, there is no change of orientation to be propagated; thus these approaches cannot avoid shearing distortion caused by handle translation (see an example in Figure 2.8).

Implicit methods define $\mathbf{M}_i$ with respect to the deformed surface. This is essentially a chicken-and-egg problem. On the one hand, the deformed surface is to be reconstructed from the transformed LCs; on the other hand, the transformations are dependent on the resulting deformed surface. Lipman et al. [3] proposed a heuristic method: they first reconstruct a rough surface using the original LCs and use the reconstructed surface to estimate a local rotation for each vertex. Sorkine et al. [19] approximately represent $\mathbf{M}_i$ as a function of the unknown vertex positions. Existing implicit methods work well for small translations and/or small rotation angles of handles. However, they do not produce visually pleasing results when handles undergo large angle rotations (Figure 2.7) or big-scale translations (Figure 2.8). Our implicit method can handle these types of large deformation. Our framework has some similarity to Lipman et al. [73], which appeared after the initial submission of our method [88] to the journal of Computer Graphics Forum (see more detailed comparison in Section 2.5).

2.4 Implicit Laplacian Mesh Editing Framework

This section introduces our implicit Laplacian editing framework. We implicitly define local transformations in terms of the (known) original vertex positions and (unknown) deformed vertex positions. To avoid having the LCs deviate from the normal directions, thus suppressing shearing distortion, these transformations are required to be rigid [19]. However, rigid transformations in 3D space nonlinearly depend on the vertex positions. To attain linearity in the reconstruction in Equation 2.2, we therefore adopt affine transformations. Since simply representing affine transformations in terms of the unknown vertex positions makes the resulting system (Equation 2.2) under-constrained, we enforce neighborhood coherence to make the reconstruction problem well-posed. The shearing distortion accompanying affine transformations will be removed in the second step described in the next section.

2.4.1 Implicitly Defined Local Deformation Gradients

Like [19, 66], we define an affine transformation for each vertex using that vertex and its neighbors as follows:

$$\mathbf{M}_i \mathbf{v}_k + \mathbf{d}_i = \tilde{\mathbf{v}}_k, \qquad k \in \{i\} \cup N(i), \tag{2.3}$$

where $\mathbf{M}_i$ is a 3×3 matrix, $\mathbf{d}_i$ is the translation vector, and $\tilde{\mathbf{v}}_k$ is the unknown vertex. $\mathbf{M}_i$ and $\mathbf{d}_i$ together define an affine transformation at $\mathbf{v}_i$. Since the LCs to which the defined affine transformations will be applied are local difference of vertex positions, what we really care for is only $\mathbf{M}_i$, called the *deformation gradient* [57].

By eliminating the translation vector $\mathbf{d}_i$, we rewrite Equation 2.3 in matrix form as

$$\mathbf{M}_i \mathbf{V}_i = \tilde{\mathbf{V}}_i \tag{2.4}$$

where

$$\begin{array}{l} \mathbf{V}_i = [\mathbf{v}_i - \mathbf{v}_{i_0} \quad \mathbf{v}_{j_1} - \mathbf{v}_{i_0} \quad \cdots \quad \mathbf{v}_{j_{|N(i)|}} - \mathbf{v}_{i_0}] \\ \tilde{\mathbf{V}}_i = [\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_{i_0} \quad \tilde{\mathbf{v}}_{j_1} - \tilde{\mathbf{v}}_{i_0} \quad \cdots \quad \tilde{\mathbf{v}}_{j_{|N(i)|}} - \tilde{\mathbf{v}}_{i_0}] \end{array}, \quad j_* \in N(i)$$

with

$$\mathbf{v}_{i_0} = \frac{1}{|N(i)|} \sum_{j \in N(i)} \mathbf{v}_j \quad \text{and} \quad \tilde{\mathbf{v}}_{i_0} = \frac{1}{|N(i)|} \sum_{j \in N(i)} \tilde{\mathbf{v}}_j.$$

To get a least-squares solution for $\mathbf{M}_i$ in Equation 2.4, we consider the following equations

$$\mathbf{M}_i \mathbf{V}_i \mathbf{V}_i^{\mathbf{T}} = \tilde{\mathbf{V}}_i \mathbf{V}_i^{\mathbf{T}}.$$

If $\mathbf{V}_i \mathbf{V}_i^{\mathbf{T}}$ is invertible, we can directly derive $\mathbf{M}_i$ as

$$\mathbf{M}_i = \tilde{\mathbf{V}}_i \mathbf{V}_i^{\mathbf{T}} (\mathbf{V}_i \mathbf{V}_i^{\mathbf{T}})^{-1}. \tag{2.5}$$

The above expression is similar to the derived deformation gradient in [66]. The deformation gradient in their work always has a closed form expression because the vertices used to define the deformation gradient form a basis of 3D space, whereas we need to handle the special case when $\mathbf{V}_i \mathbf{V}_i^{\mathbf{T}}$ degenerates to a singular matrix (Section 2.4.3).

2.4.2 Neighborhood Coherence

We first assume that $\mathbf{M}_i$ is well defined for each unconstrained vertex; the degenerate case will be discussed in the next subsection. By applying $\mathbf{M}_i$ to the corresponding LC δ_i, we wish to reconstruct the editing vertex positions by minimizing the following error functional

$$E_L = \sum_{i=1}^{n} \| D(\tilde{\mathbf{v}}_i) - \mathbf{M}_i \delta_i \|^2, \tag{2.6}$$

However, this reconstruction problem by itself is under-constrained [19].

To make the reconstruction problem well-posed, we introduce a neighborhood coherence term to regularize the implicitly defined deformation gradients. Similar to [89, 66], we require the deformation gradients applied within a surface region to be as similar as possible. Specifically, besides E_L, we also minimize the following error functional

$$E_R = \sum_{i=1}^{n} \sum_{(i,j) \in SF} \| \mathbf{M}_i - \mathbf{M}_j \|_F^2, \tag{2.7}$$

where SF is the set of pairs of neighboring vertex indices and $\| \cdot \|_F$ is the Frobenius norm. Since the deformation gradient defined at each vertex is used to deform the

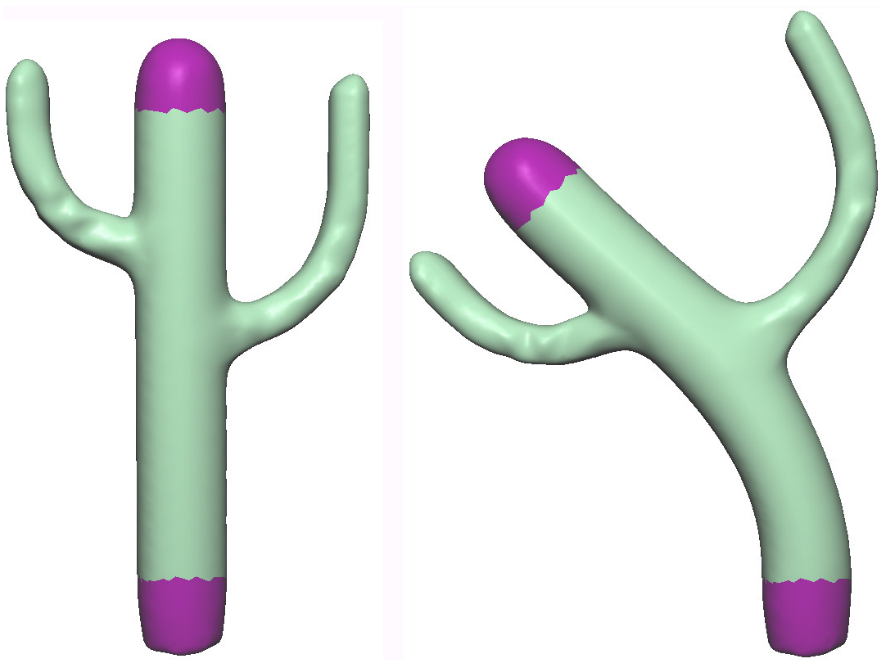


Figure 2.3: **Left**: the original cactus model. **Right**: the deformed cactus by rotating and translating the top handle. We render the ROI in blue and the handle(s) in purple in all the examples.

features locally (by transforming the corresponding LC), the neighborhood coherence term essentially minimizes the difference in deformation at neighboring vertices.

We now prove that given the position and the deformation gradient at an arbitrary mesh vertex $\mathbf{v}_0$, the minimization problem in Equation 2.6 with the neighborhood coherence constraint $\|\mathbf{M}_i - \mathbf{M}_j\|_F = 0$ is well-posed. First, we prove that with the neighborhood coherence constraint and the given deformation gradient, all the deformation gradients at the unconstrained vertices can be uniquely determined. As $\|\mathbf{M}_i - \mathbf{M}_j\|_F = 0$ implies $\mathbf{M}_i = \mathbf{M}_j$, the transformations at vertices adjacent to $\mathbf{v}_0$ must be the same as the given deformation gradient at $\mathbf{v}_0$. With the same argument, all the deformation gradients at the unconstrained vertices can be determined, and are equal to the given deformation gradient at $\mathbf{v}_0$. Second, we prove that, with all the computed deformation gradients and the given position constraint at $\mathbf{v}_0$, the positions of all the unconstrained vertices can be uniquely determined. After $\mathbf{M}_i$ is computed, the optimization in Equation 2.6 is equivalent to solving a simple Poisson system, which has a unique and exact solution given one position constraint [19].

When more than one position and more than one deformation gradient are constrained, exact solution is not guaranteed. The unique solution is solved by minimizing

$E_L + E_R$ in the least squares sense. Formally, the final optimization is formulated as follows

$$\begin{aligned} \arg\min\ E(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n) = E_L + E_R \\ \text{subject to} \qquad \tilde{\mathbf{v}}_{b_j} = \mathbf{u}_j, \qquad j \in \{1 \ldots s\} \\ \mathbf{M}_{c_k} = \mathbf{W}_k, \qquad k \in \{1 \ldots t\} \end{aligned} \tag{2.8}$$

where $\mathbf{u}_j$ is the position of the vertex with index b_j on the boundary condition, $\mathbf{W}_k$ is the local deformation gradient at the vertex with index c_k on the boundary condition, and s and t are the number of position constraints and the number of transformation constraints, respectively. Usually, we specify both the positions and the deformation gradients for the same set of boundary vertices (i.e. $\{\tilde{\mathbf{v}}_{b_j}\} = \{\tilde{\mathbf{v}}_{c_k}\}$). Nevertheless, it is possible to specify only the deformation gradients for the boundary vertices, and let their positions be free, or vice versa.

As $\mathbf{M}_i$ is linearly dependent on the unknown vertex positions, $\mathbf{M}_i$ and $\tilde{\mathbf{v}}_i$ are simultaneously solved from the optimization in Equation 2.8. Therefore, the formulated optimization can be used for mesh deformation: like previous Laplacian surface editing work [19, 20, 73], we allow the user to interactively change the boundary condition through manipulating the handles and reconstruct the unconstrained region of interest from the optimization. During editing, the locally defined deformation gradients accommodate the local changes of details (see an example in Figure **??**).

2.4.3 Handling Degenerate Vertices

This section explains how we handle the special case when $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is singular in Equation 2.5. We refer to the corresponding vertex $\mathbf{v}_i$ as a *degenerate vertex*. For a degenerate vertex $\mathbf{v}_i$, the rank of $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is equal to 2, which means that $\mathbf{v}_i$ and its neighbors are coplanar. In practice, meshes with high resolution are locally smooth. Therefore, degenerate vertices are common. As the uniqueness of the solution to the optimization in Equation 2.8 is under the condition that the neighborhood coherence term is well defined for every pair of neighboring vertices, we have to implicitly define the transformations for all the vertices, including the degenerate vertices.

Our basic idea of handling degenerate vertices is to move $\mathbf{v}_i$ out of the plane defined by its neighboring vertices. Let $\mathbf{v}'_i$ denote the modified vertex of $\mathbf{v}_i$. Since the direction

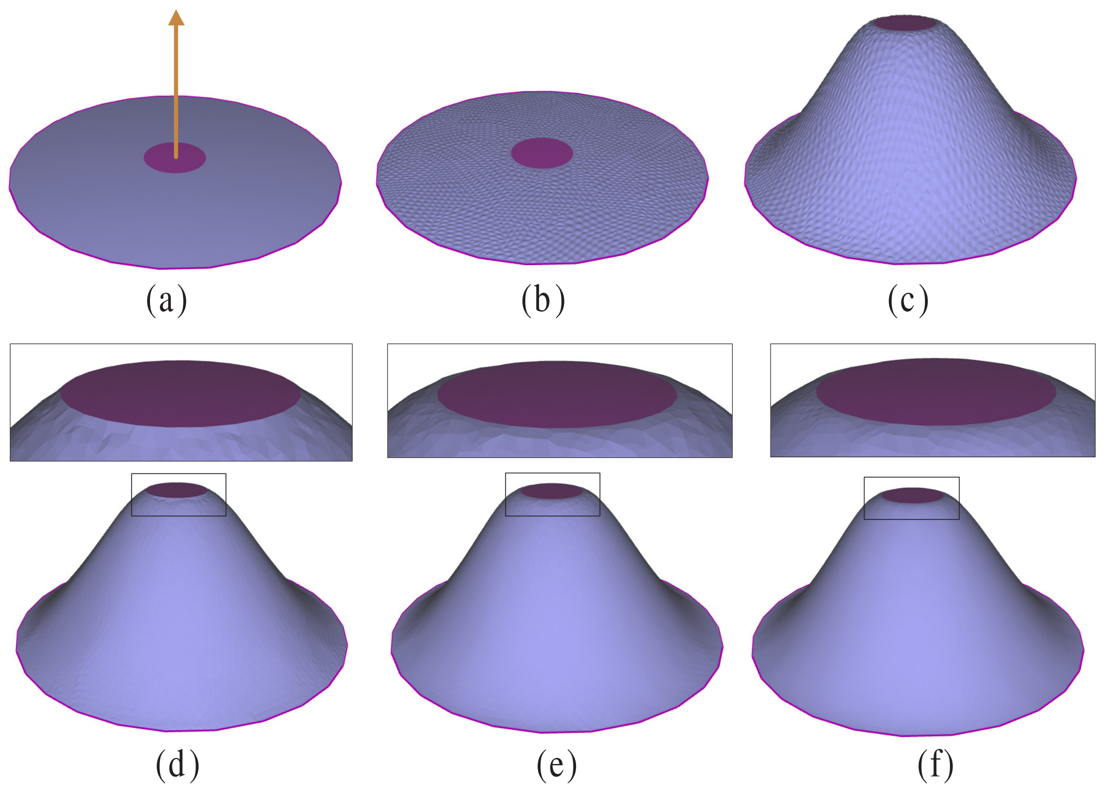


Figure 2.4: Handling degenerate vertices. (a) A planar irregular triangular mesh with the boundary vertices and the vertices in the central region as the boundary constraints. (b) After displacing every degenerate vertex. (c) Editing without removing the displacement. (d) and (e): Editing with the displacement removed, using uniform weighting and cotangent weighting, respectively. (f) Reconstruction errors are distributed over the ROI. All the images in this chapter are flat-shaded to better demonstrate that the errors introduced by the handling of degenerate vertices are unnoticeable.

of a nondegenerate LC approximates the normal, we place $\mathbf{v}'_i$ in the normal direction of $\mathbf{v}_i$ (see Figure 2.4 (b)). Typically, we set

$$\mathbf{v}'_i = \mathbf{v}_i + \frac{s_{disp}}{|N(i)|} \sum_{j \in N(i)} \|\mathbf{v}_i - \mathbf{v}_j\| \cdot \mathbf{n}_i, \tag{2.9}$$

where $\mathbf{n}_i$ is the unit normal vector at $\mathbf{v}_i$ and s_{disp} is a scaling factor. We traverse the entire vertex list to handle all the degenerate vertices. We classify $\mathbf{v}_i$ as a degenerate vertex if and only if the condition number of $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is greater than a given threshold (e.g., 2.0×10^5). If a degenerate vertex is on an open boundary, we adopt a small variation: if its valence is less than 3, we insert a new vertex at the midpoint of its opposite edge; if its neighboring vertices are collinear, we displace one of its neighboring non-boundary vertices instead of $\mathbf{v}_i$ itself.

After defining the deformation gradients for all the unconstrained vertices, we solve for the positions of the unconstrained vertices using Equation 2.8. Since the optimized

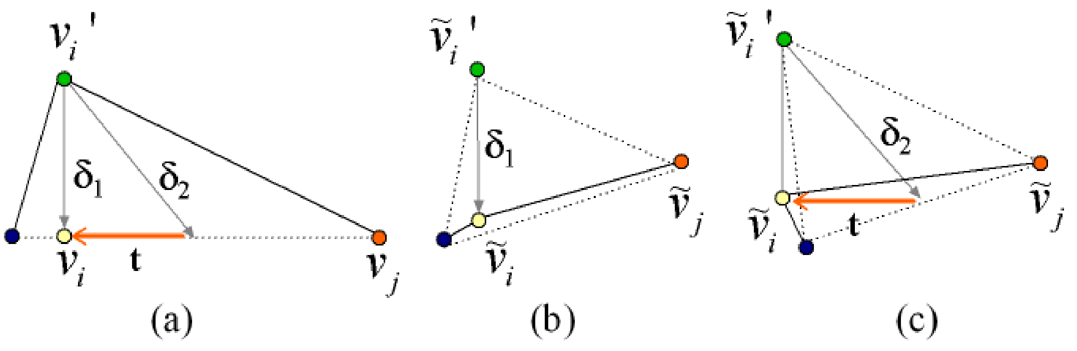


Figure 2.5: Difference between using cotangent weights (b) and uniform weights (c).

position of $\tilde{\mathbf{v}}_i'$, corresponding to the modified vertex $\mathbf{v}_i'$ of a degenerate vertex $\mathbf{v}_i$, is influenced by our deliberate displacement (Figure 2.4 (c)), we need to pull $\tilde{\mathbf{v}}_i'$ back to the correct position $\tilde{\mathbf{v}}_i$ (Figure 2.4 (d) and (e)). As $\mathbf{M}_i$ is already computed at this stage, we obtain the new positions of the degenerate vertices by applying the associated affine transformations to the original vertex positions (before the deliberate displacements):

$$\tilde{\mathbf{v}}_i = \mathbf{M}_i(\mathbf{v}_i - \mathbf{v}'_i) + \tilde{\mathbf{v}}_i'.$$

There is no stability problem as long as all the deformation gradients for the degenerate vertices are defined prior to the optimization (cf. Section 2.4.2). Nevertheless, the handling of degenerate vertices does introduce two types of errors in the editing results, albeit unnoticeable. Computing the edited position $\tilde{\mathbf{v}}_i'$ instead of $\tilde{\mathbf{v}}_i$ through optimization introduces the first type of error, which is an increasing function of s_{disp}. We experimented with various scaling factors (e.g., $s_{disp} = 0.2$ and $s_{disp} = 10$) and found that the resulting shapes are of little visual difference (see an example in Figure 2.7). Thus we use a small fixed scaling factor (e.g., $s_{disp} = 0.2$) in all our experiments.

The second type of error arises because the edited positions of the degenerate vertices are computed through *linear* transformations ($\tilde{\mathbf{v}}_i = \mathbf{M}_i(\mathbf{v}_i - \mathbf{v}'_i) + \tilde{\mathbf{v}}_i'$) whereas the edited positions of the nondegenerate vertices are solved in the *least-squares* sense. We found that discretizing the Laplacian using cotangent weights gives much better results (Figure 2.4 (e)) than using uniform weights (Figure 2.4 (d)). We illustrate the reason using a 2D example in Figure 2.5. Vertex $\mathbf{v}_i$ is an unconstrained degenerate vertex, which is displaced to $\mathbf{v}_i'$ before editing. The LC at $\mathbf{v}_i'$ is δ_1 (close to the normal direction) if cotangent weights are used or δ_2 (with a tangential component $\mathbf{t}$) if uniform weights are used. After editing, the tangential component $\mathbf{t}$ in δ_2 would cause tangen-

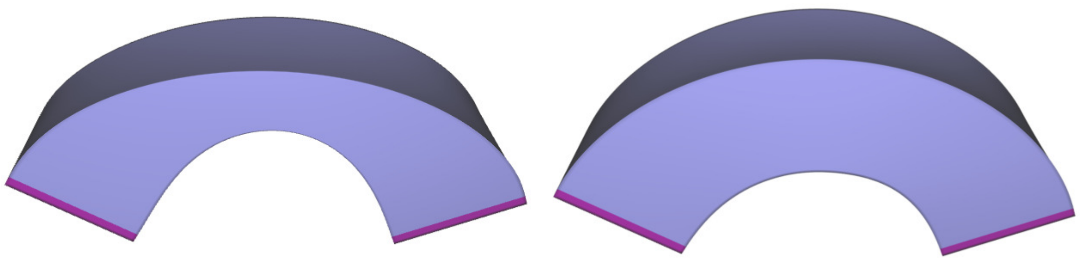


Figure 2.6: An deformation example without shearing removal (**Left**) and with shearing removal (**Right**).

tial drifting, bringing the vertex ”away” from the surface (Figure 2.5 (c)), especially at vertices in curved regions after editing (Figure 2.4 (d)), producing visual artifact and a bigger reconstruction error.

It is noteworthy that in the shearing removal step (described in the next section), solving the system (Equation 2.10 or 2.11) in the least-squares sense makes the second type of errors uniformly distributed to the whole ROI, making the errors unnoticeable (Figure 2.4 (f)).

2.4.4 Shearing Removal by Polar Decomposition

The deformation gradients may contain shearing transformations, making the LCs deviate from the normal directions and thus causing shearing distortion.

The neighborhood coherence term only partially solves the shearing problem. When different rigid transformations are applied to different handles, shearing problem may occur. The neighborhood coherence term only guarantees that features close to the handle boundaries deform almost rigidly[1]. Shearing distortion may occur at regions far away from the constrained regions since the rigid transformations applied to the handles has less influence on the transformations defined over the far-away regions (Figure 2.6 left).

To overcome the shearing problem, we introduce an additional step after solving the optimization in Equation 2.8. Since what we really care for in the information

[1]According to the formulation of the neighborhood coherence term, the more neighboring vertices we use to define the coherence term, the wider will be the rigid region (close to the handle boundaries). This is a desirable property. However, there are two side effects of using more vertices. First, the *local* deformation of features will be suppressed. An extreme example is to define the coherence term using the whole region of interest. Second, the introduction of more vertices makes the system matrix denser. We choose to use 2-ring neighbors (cf. SF in Equation 2.7) as it is a good trade-off.

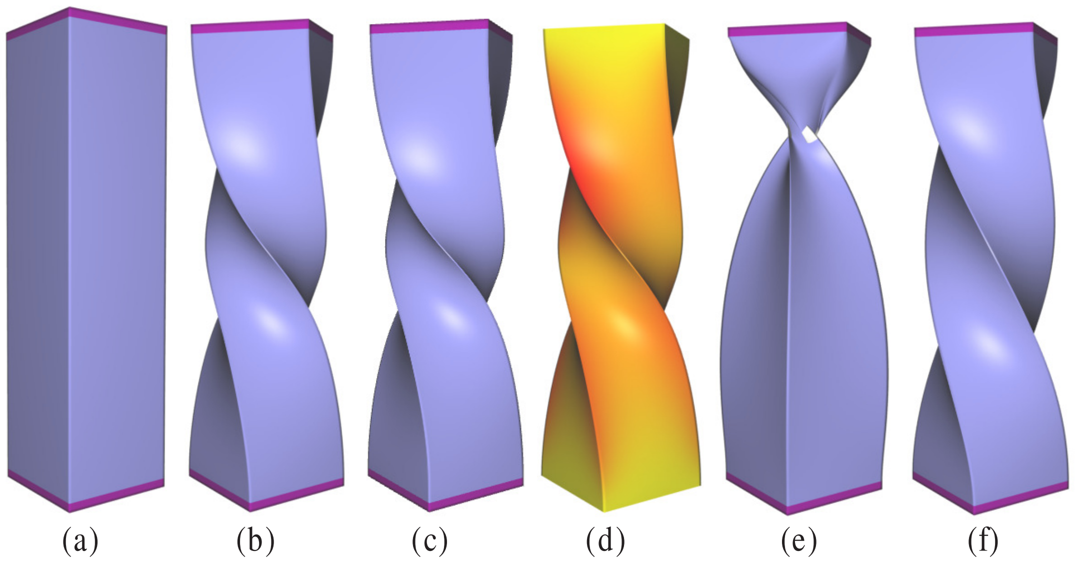


Figure 2.7: (a) The original bar model with an ROI (blue) and two handles (purple). After the top handle undergoes a large rotation, (b) and (c) are our results by setting $s_{disp} = 0.2$ and $s_{disp} = 10$, respectively, which have little visual difference. The difference between corresponding vertices is shown in (d) with red color denoting relatively big difference and yellow color denoting no difference. (e) and (f) are the deformation results with methods in [3] and [4], respectively.

encoded in $\mathbf{M}_i$ is the rotation part and the uniform scaling part, similar to [90], our solution is to extract the rotation and uniform scaling information from $\mathbf{M}_i$ by polar decomposition.

We first perform the singular value decomposition (SVD) on $\mathbf{M}_i$: $\mathbf{M}_i = \mathbf{U}_i\mathbf{W}_i\mathbf{V}_i^{\mathbf{T}}$, where $\mathbf{U}_i$ and $\mathbf{V}_i$ are two 3×3 orthogonal matrices and $\mathbf{W}_i$ is a 3×3 diagonal matrix whose elements are the eigenvalues $(\lambda_j, 1 \leq j \leq 3)$ of $\mathbf{M}_i$. Next we define the modified the deformation gradient without shearing as

$$\mathbf{M}_i' = \mathbf{R}_i\mathbf{S}_i,$$

where $\mathbf{R}_i = \mathbf{U}_i\mathbf{V}_i^{\mathbf{T}}$ and $\mathbf{S}_i = \text{diag}(s_i, s_i, s_i)$ with $s_i = \sqrt{\frac{\lambda_1^2+\lambda_2^2+\lambda_3^2}{3}}$.

We provide the user two options: to retain the sizes of the local features or automatically scale the geometric features during deformation. The respective underlying systems of the former and latter choices are

$$D(\tilde{\mathbf{v}}_i) = \mathbf{R}_i\delta_i \tag{2.10}$$

or

$$D(\tilde{\mathbf{v}}_i) = \mathbf{R}_i\mathbf{S}_i\delta_i. \tag{2.11}$$

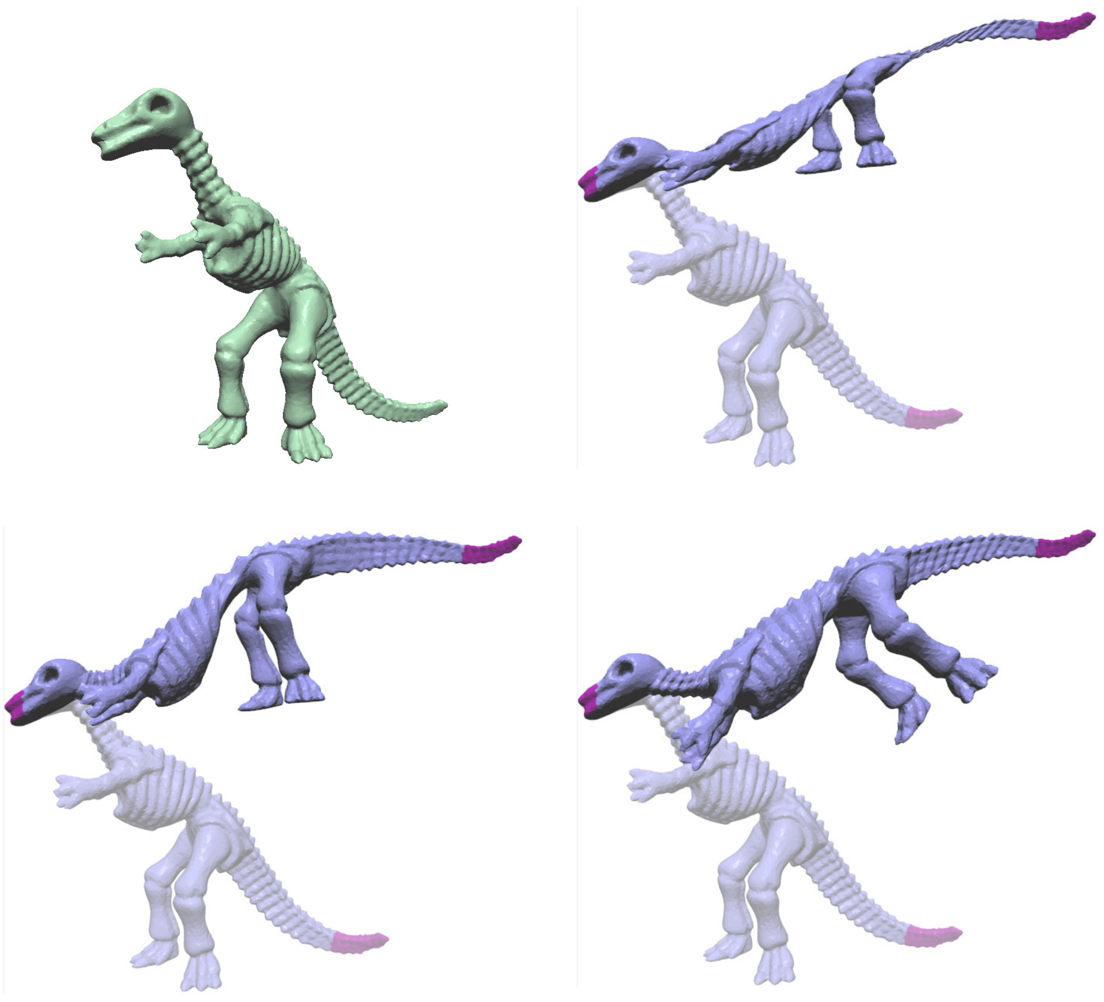


Figure 2.8: An example of rotation of local features resulting from translation of handles. **Top left**: The input dinosaur model. The deformation results by [4] (**Top right**), [3] (**Bottom left**) and ours (**Bottom right**) when the handle at the tail of the dinosaur undergoes a big-scale translation.

The first choice is mainly used when the user applies rigid transformations (including rotation and translation and excluding scaling) to the handles. The second choice is often adopted when the transformations applied to the handles include scaling. Note that the second choice is necessary for configuration-independent merging (Section **??**), as we want the scales of the features to be automatically determined.

With this shearing removal step, our deformation tool always produces more visually pleasing results than previous methods for large angle rotation or big-scale translation. Figure 2.7 shows an example when the top handle of the vertical bar undergoes a large rotation. Our method outperforms existing implicit methods [3, 19] [2]. Note

[2]Here we only compare with [3] and not with [19] as the latter only includes a small refinement on the results of the former when the approximation error is large.

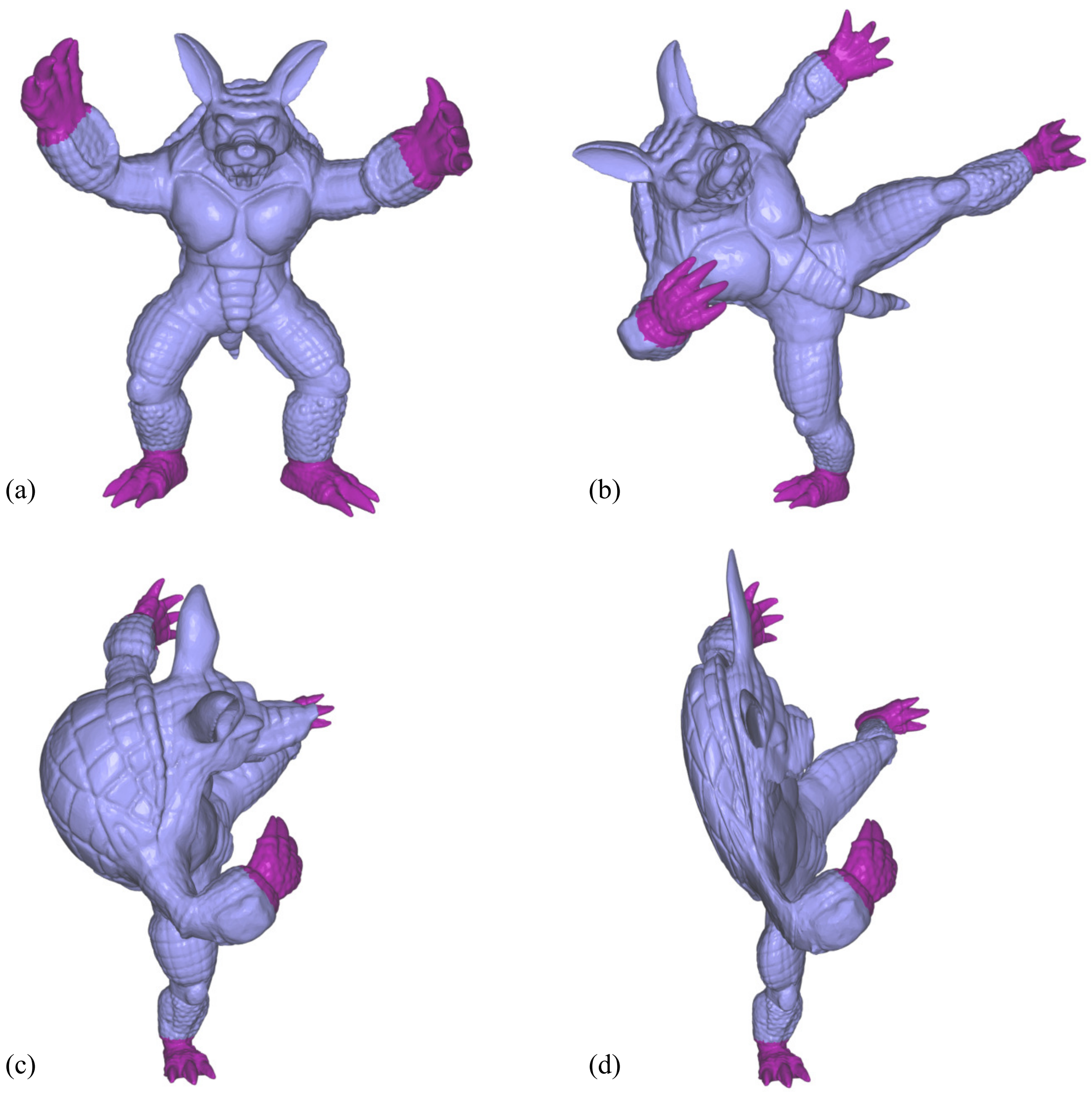


Figure 2.9: (a) The original Armadillo model with four handles specified. (b) and (c): Two views of the deformed model by applying several rigid transformations to the handles. (d) The deformation result with the same view as (c) but without shearing removal.

that, like ours, explicit methods [4, 20] also produce natural deformation results as transformations are explicitly propagated (see more discussions in Section 2.5).

Figure 2.8 demonstrates the effectiveness of our method for deformation under big-scale translation of the handle. Explicit methods [20, 21, 4] cannot infer any rotation information from translations of handles (Figure 2.8 left). Implicit methods [3, 19] can capture rotation under small-scale translation but do not work for big-scale translation (Figure 2.8 middle). With our method, the global shapes in the deformed model (e.g., the whole arms and legs) are appropriately rotated and the local details are well preserved. This example also demonstrates that most rotation information is successfully

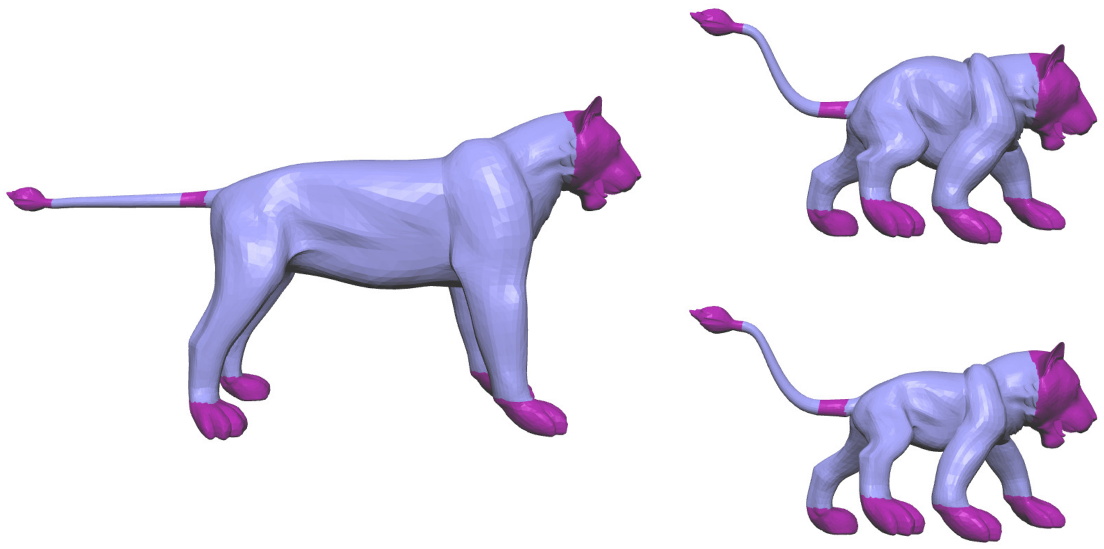


Figure 2.10: An example of scaling local features edited by translating handles. **Left**: the input lion model. **Right top**: the deformed model without scaling the LCs. **Right bottom**: the deformed model with scaling the LCs.

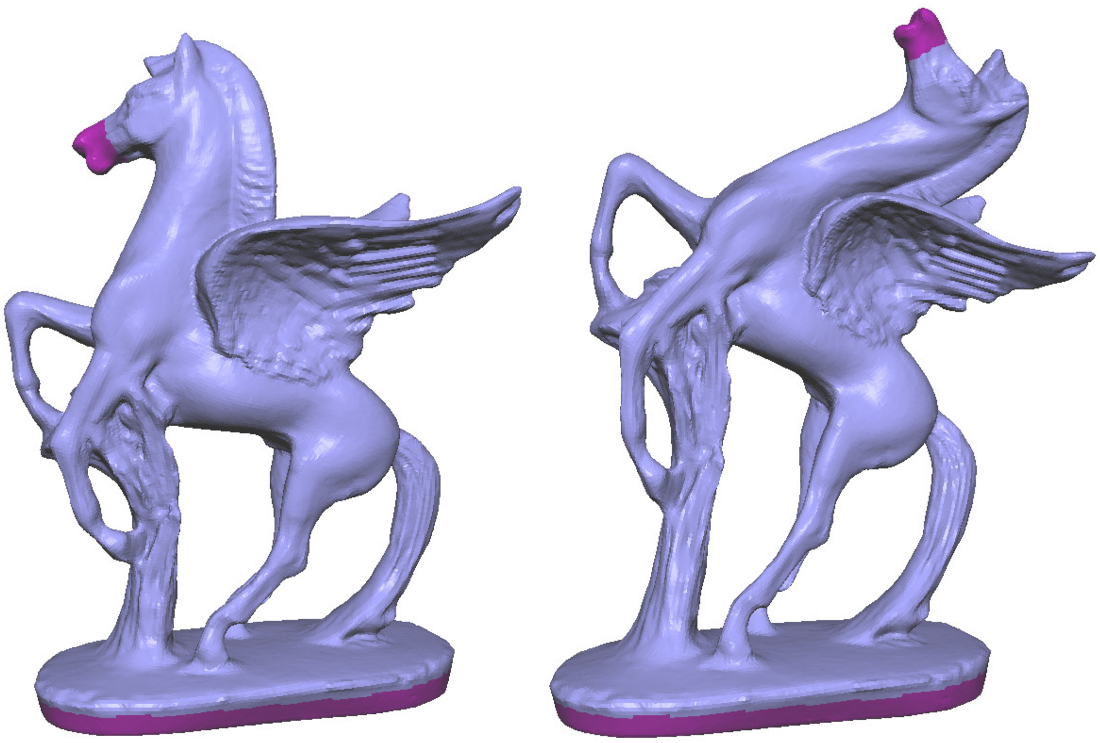


Figure 2.11: Our method works well for high-genus models.

captured by the affine transformations and effectively extracted in the shearing removal step. Figure 2.9 illustrates that it is easy to use our deformation tool to produce complicated but visually pleasing deformation results by simply applying several rigid transformations to the handles. The contrast between (c) and (d) clearly demonstrates the effectiveness of our shearing removal procedure.

The above examples are obtained by retaining the sizes of the geometric features (cf. Equation 2.10). Sometimes letting the system automatically scales the LCs is useful (cf. Equation 2.11). In Figure 2.10, the handles are moved closer to each other, thus the space between them becomes smaller and cannot accommodate the big global features of the original body. Automatically scaling the LCs (using Equation 2.11) gives a better visual result.

Our method requires the input model to be 2-manifold and allows deformation of models with nonzero genus. Figure 2.11 shows that our method works well for models with high genus.

2.5 Implementation Details and Discussions

The solution of the optimization in Equation 2.8 and the solution of the linear system in Equation 2.10/2.11 are obtained by solving the following form of normal equations

$$\mathbf{A}^{\mathbf{T}}\mathbf{A}[\mathbf{V}_x\ \mathbf{V}_y\ \mathbf{V}_z] = \mathbf{A}^{\mathbf{T}}[\mathbf{b}_x\ \mathbf{b}_y\ \mathbf{b}_z],$$

where $\mathbf{V}_x$, $\mathbf{V}_y$ and $\mathbf{V}_z$ are corresponding components (x, y, and z) of the unconstrained vertex positions, $\mathbf{b}_x$, $\mathbf{b}_y$ and $\mathbf{b}_z$ are three known vectors constructed from the position and transformation constraints, and $\mathbf{A}$ is a large sparse matrix. As matrix $\mathbf{A}$ is only dependent on the original mesh and each row of the deformation gradients (Equation 2.5) is only dependent on one dimension (e.g. x coordinates), we can pre-factorize $\mathbf{A}^{\mathbf{T}}\mathbf{A}$ using Choleskey factorization and solve $\mathbf{V}_x$, $\mathbf{V}_y$ and $\mathbf{V}_z$ separately by back substitution. In practice, we use an efficient sparse linear solver [77]. Table 3.1 lists the number of unconstrained vertices, the number of degenerate vertices, the factorization time and back-substitution time in Equations 2.8 and 2.10/2.11 as well as the time for computing SVD for all deformation gradients.

Given the boundary condition, the unknown surface is reconstructed through optimization, guaranteeing that any errors are distributed over the entire unknown surface. However, the resulting deformation is still sensitive to the resolution of the boundary condition. For example, if a complex ROI is subject to a boundary condition consisting of only 3 vertices, then the reconstructed surface would be extremely sensitive to the

Mesh	Lion	Cactus	Dinosaur	Bar	Armadillo	Pegasus
Free Vertices	7,903	8,978	13,252	24,480	48,027	54,517
Degenerate Vertices	77	41	3	5,852	61	788
Factor (Eqn 2.8)	0.313s	0.359s	0.625s	2.516s	4.813s	6.594s
Solve (Eqn 2.8)	0.015s	0.031s	0.031s	0.109s	0.188s	0.297s
Factor (Eqn 2.10/2.11)	0.063s	0.078s	0.110s	0.453s	0.875s	1.016s
Solve (Eqn 2.10/2.11)	0.016s	0.015s	0.015s	0.032s	0.078s	0.125s
SVD	0.156s	0.172s	0.266s	0.469s	0.985s	1.156s

Table 2.1: The timing results for our deformation examples on a 3.2GHz Pentium IV machine with 1G RAM.

changes in the positions or in the local deformation gradients of the boundary vertices. Therefore, for boundaries with considerably fewer vertices relative to the ROI, we first perform a local refinement. This is particularly useful in configuration-independent merging (without overlapped transition regions), since the boundary condition is composed of only one boundary curve.

We note that, due to the fact that transformations are explicitly propagated rather than determined via optimization, explicit methods [20, 4] can handle rotation angles of handles larger than π. Being an implicit method, our system does not support editing with rotation angles larger than π. The transformations are implicitly defined and solved simultaneously with the vertex positions by minimizing distortion. Since a larger angle would give greater distortion, our system always chooses the transformation with the smallest rotation angle ($< \pi$). To perform deformation with large rotation angles ($> \pi$), one possible way is to use more handles such that rotation angles between successive handles is smaller than π.

Although performing SVD on a 3×3 matrix is fast, performing SVD on a large set of deformation gradients still becomes the bottleneck of our system. Since solving the optimization in Equation 2.8 is fast (only involving the back-substitution), the screen is updated using the solved vertex positions when the user is manipulating a handle. SVD and the back-substitution of the second linear system are performed only when the user stops moving the handle. This allows our system to remain at interactive rate for editing large scale meshes.

The parallel work of Lipman et al. [73] is similar to ours. Their method binds triple vectors defining a discrete frame to each vertex. The discrete frames before and after editing essentially determine a deformation gradient (like $\mathbf{M}_i$ in our framework).

Essentially, in their method, the local details (rigid-invariant coordinates) at each vertex are encoded in the corresponding discrete frame, while in ours the local details (the LCs) are defined with respect to the corresponding deformation gradients. Since the discrete frames in [73] are nonlinearly dependent on vertex positions, the discrete frames and vertex positions cannot be formulated and solved as a single linear system. To make the reconstruction problem linear, their method uses two sparse linear systems: one for solving the discrete frames (using only the transformation constraints) and the other for solving the vertex positions given the discrete frames obtained from the first system (using the position constraints only). In our method, the deformation gradients are solved using only one linear system (considering both the position and transformation constraints). Our solution avoids the incompatibility between the position and transformation constraints encountered in [73] at the cost of introducing more shearing distortion. Consequently, we introduce an additional step to remove the shearing distortion.

2.6 Summary

This chapter presents an implicit differential mesh editing framework that is resistant to the non-rigid-invariant property of the differential coordinates. Due to the implicitly defined local deformation gradients (with shearing removed) introduced at each vertex, our deformation tool is automatically detail-preserving and produce visually pleasing deformed results. The deformed results are obtained by solving two sparse linear systems at interactive rate. We will demonstrate the the advantage of our framework further in a new application, called configuration-independent merging, which is presented in the next chapter.

Although to our best knowledge, our method provides the most effective linear approximation to the nonlinear differential mesh deformation, it still suffers from serious linear approximation errors under very large deformations. To solve the problem, we have proposed two efficient nonlinear Laplacian mesh editing frameworks [81, 1] very recently.

CHAPTER 3

POISSON MESH MERGING

In this chapter, we present a new Poisson mesh editing framework. Our contributions are two-fold. First, we present a new merging paradigm, called *configuration-independent merging*, based on our new Laplacian editing framework presented in the previous chapter. Our method produces the same merging result independent of the relative position, orientation, scale of input meshes. Second, we propose to optimize the boundary condition (the merging boundary) for Poisson mesh merging. The user needs only to casually mark a source region and a target region. Our algorithm automatically searches for an optimal boundary condition within the marked regions such that the change of the found boundary during merging is minimal in terms of similarity transformation. Experimental results demonstrate that our merging tool is easy to use and produces visually better merging results than unoptimized techniques.

3.1 Introduction

Directly modeling 3D geometric objects from scratch is often difficult and time-consuming. Instead, mesh editing techniques aim to create models by modifying existing ones, usually obtained from 3D scanners. Mesh merging, as one of the most popular mesh editing tools, produces new meshes by composing parts of existing models. For example, user can merge the body of a women model with the tail of a fish to create an interesting mermaid model. Mesh merging is achieved either by blending details of meshes through an intermediate surface or by deforming the merging boundaries of meshes as well as the meshes themselves and stitching the merging boundaries together. Our merging framework falls into the second category.

In recent years, several differential mesh editing techniques have been proposed (see the related literature review in the previous chapter). Besides their easy implementation, these techniques support intuitive user interface: they allow the user to simply manipulate parts of a surface, called handles, and the deformation of the rest surface

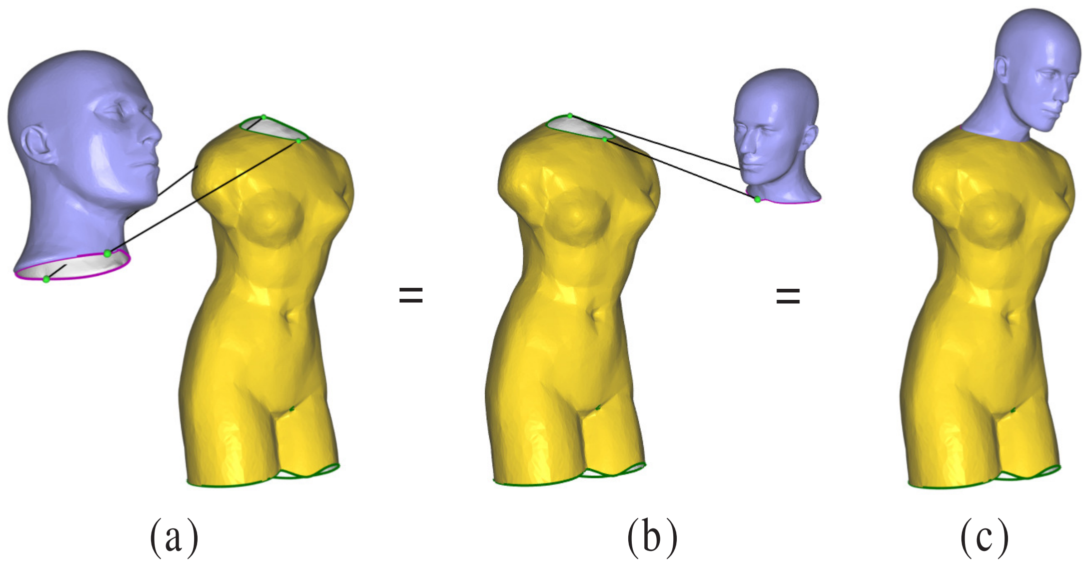


Figure 3.1: Configuration-independent merging. The goal is to merge the Mannequin head model (source) to the Venus model (target). The user only specifies the correspondence between the merging boundaries. In (a) and (b), the Mannequin head model has different positions, orientations and scales. Our configuration-independent merging method produces the same result (c), given the same boundary correspondence. The lines indicate two user-specified key correspondences.

is computed by solving the Poisson equation subject to boundary condition from the handles. By regarding the merging boundaries as the boundary condition, differential techniques are directly applicable to mesh merging [19, 20]. Without loss of generality, in this thesis we consider the merging problem as the deformation problem of a source mesh when the source merging boundary is deformed to the corresponding merging boundary on a target mesh, which is never deformed. Regardless of the specific differential representation (e.g. the Laplacian coordinates [19, 3] or gradient field [20]) used in these techniques, we call all the merging tools based on these techniques *Poisson mesh merging*, as all of them need to solve a set of Poisson equations subject to the Dirichlet boundary condition.

We present a new merging paradigm, called *configuration-independent merging* [23], based on our new Laplacian editing framework presented in Chapter 2.4. By configuration we refer to the relative position, orientation and scale of meshes. Most related previous merging techniques require the user to adjust the configuration of input meshes prior to merging [19, 91, 92]. Thanks to the implicitly defined transformations, our merging approach can eliminate this user interaction requirement. Our method computes the transformations (corresponding to configuration adjustment in existing

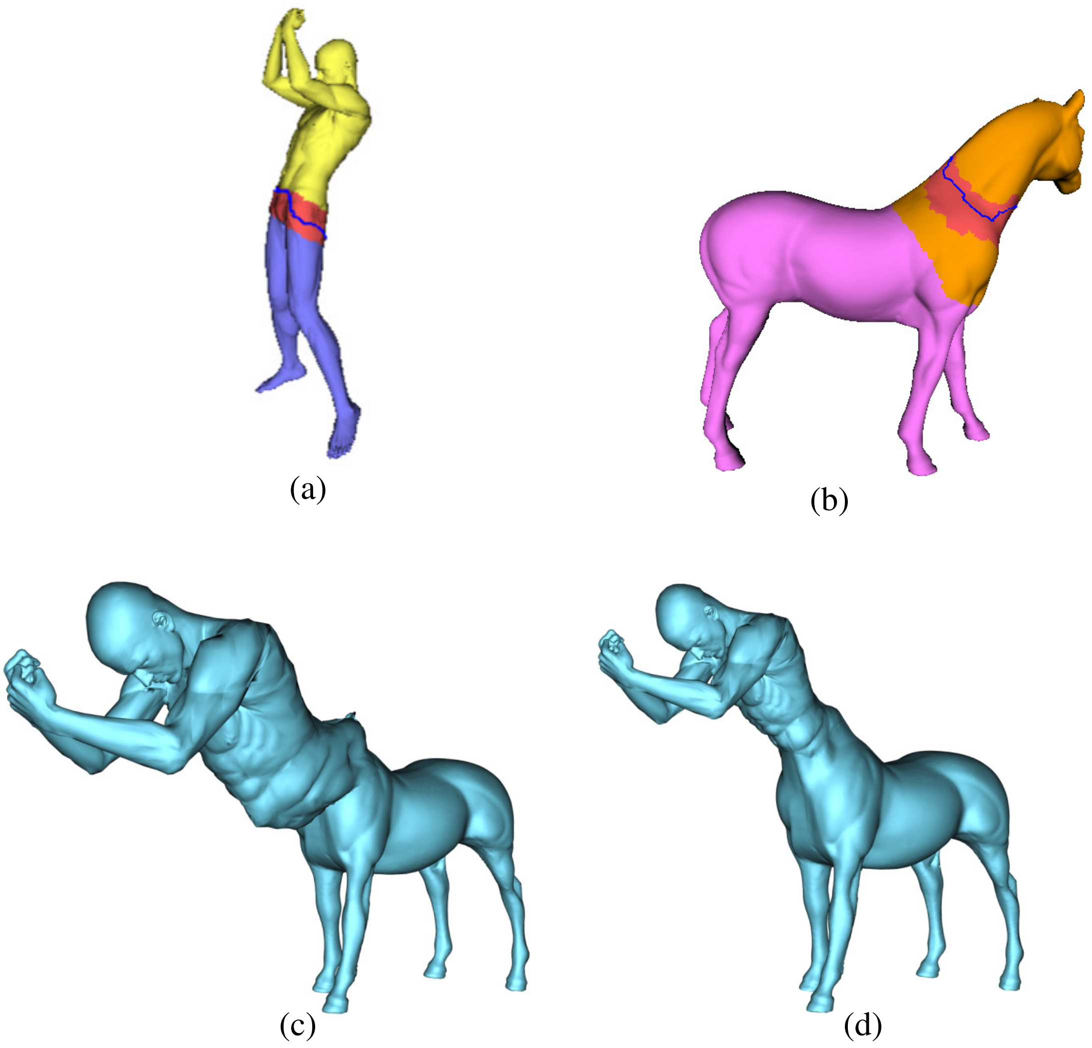


Figure 3.2: The effectiveness of Poisson mesh merging is highly dependent on the choice of the boundary conditions. (a) Source mesh. (b) Target mesh. (c) and (d) are the merging results with the user-specified (unoptimized) boundary condition (i.e. the boundary separating the region in red and the region in blue) and the optimal one found by our algorithm (in blue), respectively.

methods) and solves for the merged mesh simultaneously (Figure 3.1). The user only specifies the merging boundaries and indicates several key vertex correspondences between the boundaries.

Existing Poisson mesh editing techniques, including our configuration-independent merging, mainly focus on designing schemes to propagate deformation from a given boundary condition to a region of interest. Although solving the Poisson system in the least-squares sense distributes the distortion errors over the entire region of interest, large deformation in the boundary condition might still lead to severely distorted results. The distortion is especially large near the boundary condition (Figure 3.2c), as it provides soft or hard constraints to the deformation optimization [19]. Therefore the effectiveness of the existing Poisson mesh merging techniques is highly dependent on how carefully the user specifies the merging boundaries (the boundary condition).

A similar problem exists in Poisson image editing. Poisson image editing [93] may generate bad image composition results, especially when the boundary conditions on the source and target images severely conflict with each other. To address the problem, Jia et al. [94] propose to compute an optimized boundary condition for Poisson image editing: a boundary condition is optimal if it undergoes only a translation transformation in $\{r, g, b\}$ color spaces during composition.

Motivated by [94], we present an algorithm for easy Poisson mesh merging. It finds an optimal merging boundary within the regions casually marked by the user. A new objective function is proposed to find a boundary condition under an (unknown) similarity transformation during merging in the least-squares sense. Unlike images, meshes often have irregular sampling. We incorporate an edge-based weighting scheme to alleviate the influence of irregular sampling. Similar to [94], we use an alternating optimization method to solve the resulting complicated nonlinear optimization problem. After obtaining the optimized boundary condition, we apply the differential mesh editing technique presented in Chapter 2.4 to deform and merge the source mesh to the target mesh.

Compared with existing Poisson mesh merging techniques, our system supports much easier user interface: the user only needs to *casually* mark the region to be cut on the source mesh and the desirable region to be pasted on the target mesh. Without the user's fine tuning of the merging boundaries, the optimal boundary condition leads to visually good merging results (Figure 3.2d).

3.2 Related Work

Mesh Merging and Surface Pasting. Cut-and-paste editing is ubiquitous in text and image processing applications. It has been extended to 3D mesh domain to compose new models from parts of existing models.

Kanai et al. [95] present a mesh merging technique based on local 3D metamorphosis. This method allows details from the source and target meshes to be smoothly blended together in the final merging result. Later, the idea of transferring details is extended to multiresolution framework [96] and differential framework [19]. These methods need to build one-to-one correspondence between the whole source and target regions of interest, thus requiring their topologies the same. To remove this requirement of topology, Fu et al. [97] uses the base surfaces of the source and target regions of interest for correspondence building.

Recently, merging techniques directly based on Poisson mesh editing have been proved effective. The merging methods of Yu et al. [20] and Sorkine et al. [19] connect two meshes at their open boundaries without 2D parameterization. These methods consider the merging boundaries as the handles used in Poisson mesh editing. Therefore, these methods only require the merging boundaries to have the same topology. In [19], the merging operation first fills the gap between two boundaries and then mixes the details by surface reconstruction. In [20], the two boundaries are first deformed to an intermediate boundary, and the deformation is propagated from the deformed boundaries to the interior of the meshes. The smoothness along the merging boundary is improved by Poisson normal smoothing. Both methods require the user to adjust the configuration of the meshes to be merged. In addition, if the merging boundary is not well chosen, the merging result might still be bad. In this thesis, we present an automatic algorithm to find an optimal merging boundary within the merging regions that are roughly specified by the user [98].

The merging results with most existing merging techniques [19] largely depend on the well-adjusted relative positions of source and target meshes. However, precisely adjusting relative positions of models in 3D space is a difficult task, even for experienced users. To ease user's effort, Sharf et al. [99] present an intuitive mesh merging technique, with which the user only needs to roughly adjust the relative positions of models until there is a significant overlap between them, then the source mesh is automatically snapped and merged to the target mesh. Hassner et al. [100] introduce a part-in-whole model alignment method to aid the user in positioning the models. After the model alignment, they find a minimal cut on the graph respecting both the source and target models to simultaneously cut and stitch the models.

Poisson Image Editing. Poisson image editing is a powerful tool for image composition [93]. This technique seamlessly blends two images by solving the Poisson equations with guidance fields from the source image and a boundary condition from the target image. However, the effectiveness of Poisson image editing is dependent on how the user carefully specifies the boundary condition. To make Poisson mesh editing easier to use, Jia et al. [94] propose to optimize the boundary condition within a region roughly marked by the user.

3.3 Configuration-Independent Merging

In this section, we present two methods of mesh merging: configuration-independent merging and configuration-dependent merging. Although some existing techniques [20, 4, 73] could be adopted to implement configuration-independent merging, we believe that this is the first time such an application is proposed.

3.3.1 Configuration-Independent Merging

The configuration of the objects to be merged refers to their relative position, orientation and scale. For simplicity, we assume that the *target mesh* is fixed during merging, and let the relative position, orientation and scale of the *source mesh* be free. The merging is accomplished by deforming the merging boundary of the source mesh (and the entire source mesh as the ROI) to the merging boundary of the target mesh[1].

Having the user adjust the configuration of input meshes is equivalent to applying a global similarity transformation to the source mesh. Recall that differential-based editing frameworks deform an ROI by modifying the boundary condition. Therefore, the basic requirement of performing configuration-independent merging using such frameworks is that when a global similarity transformation is applied to the boundary condition, the positions of the vertices in the ROI, obtained through solving the optimization with the modified boundary condition, must reflect the same transformation. Figure 3.3 (right) demonstrates that our method satisfies this requirement.

To perform configuration-independent merging, the user only needs to establish the correspondence between the source merging boundary and the target merging boundary. We implement a vertex correspondence tool similar to [20]. From a set of user-

[1]An alternative approach, which we did not implement, is to merge the meshes along a user-specified intermediate boundary [20].

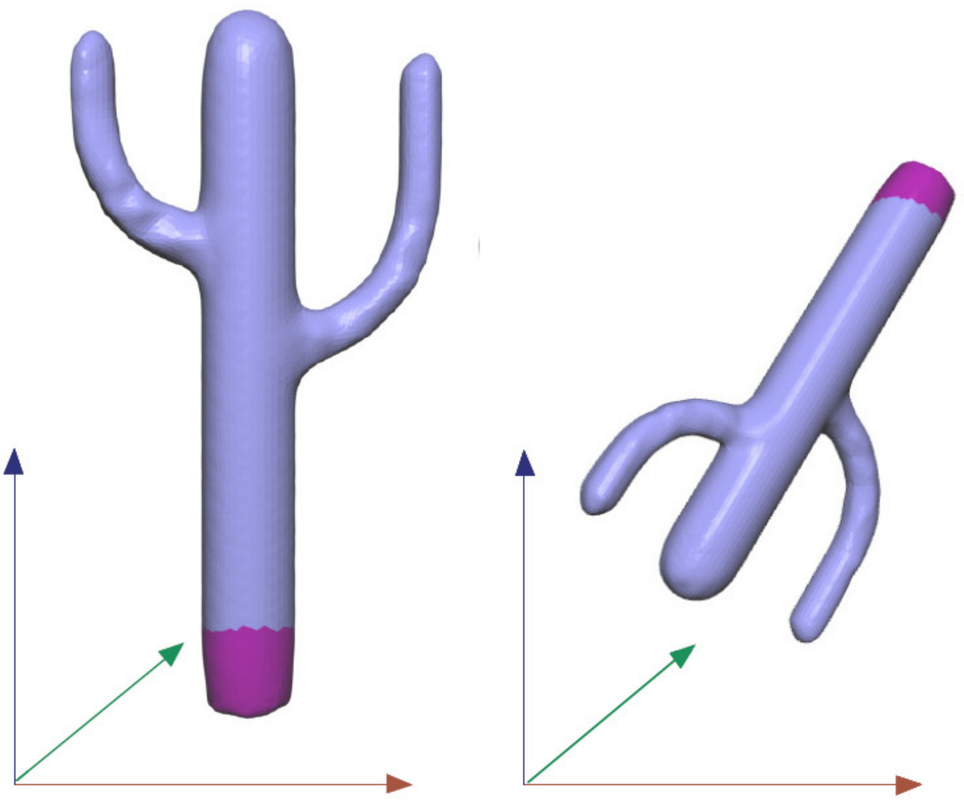


Figure 3.3: **Left**: the original cactus model. **Right**: the reconstructed model after a global transformation (including translation, rotation and uniform scaling) is applied to the handle of the original cactus model (**Left**).

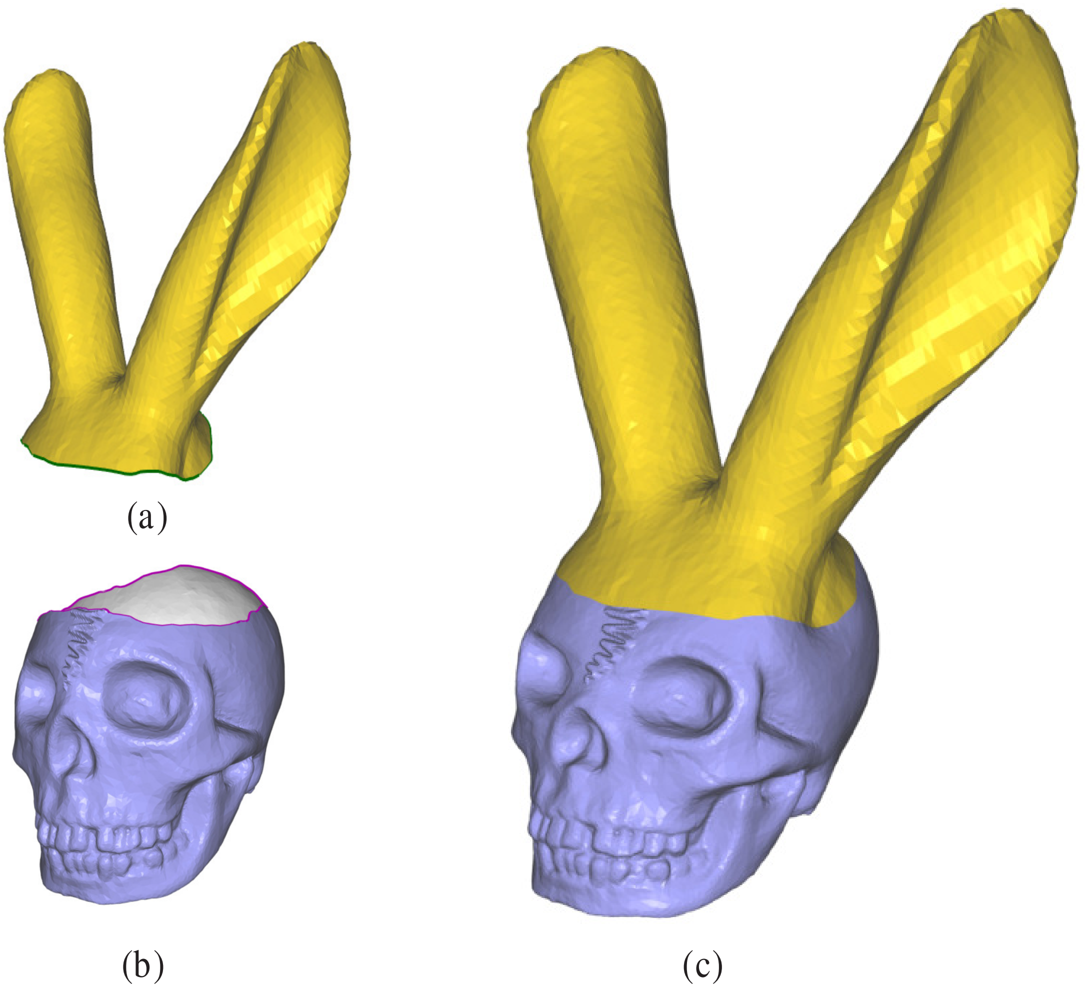


Figure 3.4: Configuration-independent merging: the Headus skull model (b) is deformed and merged to the ears of the Stanford Bunny (a). The two merging boundaries have undulations and are of different shapes.

specified key vertex correspondences, our system finds the corresponding position on the target merging boundary for each remaining vertex on the source merging boundary by curve parameterization. These positions will be used as position constraints of the optimization.

Next we need to compute a rotation transformation $\mathbf{R}_i$ and a scaling factor s_i for each pair of corresponding vertices ($\mathbf{v}_i^s$,$\mathbf{v}_i^t$), where $\mathbf{v}_i^s$ is a vertex on the source merging boundary and $\mathbf{v}_i^t$ is its corresponding position on the target merging boundary. By defining two local frames at $\mathbf{v}_i^s$ and $\mathbf{v}_i^t$ (composed of the associated tangent vector along the boundary curve, unit normal vector and the cross product of the previous two vectors), we compute $\mathbf{R}_i$ as the transformation from the local frame at $\mathbf{v}_i^s$ to the local frame at $\mathbf{v}_i^t$. The scaling factor s_i is computed as the ratio of the tangent magnitude at $\mathbf{v}_i^t$ to the tangent magnitude at $\mathbf{v}_i^s$. The deformation gradient $\mathbf{M}_i$ is then defined as $\mathbf{R}_i\mathbf{S}_i$ ($\mathbf{S}_i = \text{diag}(s_i, s_i, s_i)$) and used as the transformation constraint of the optimization.

With the position and transformation constraints, we solve the optimization in Equation 2.8 to deform the source mesh and merge it to the target mesh. The shearing distortion is removed in the same way as in mesh deformation application. To get a watertight seam, we trivially zip the target merging boundary and the deformed source merging boundary [19] by removing one band of triangles adjacent to the target merging boundary and re-triangulating the resulting gap. To improve smoothness at the seam, we apply several iterations of the umbrella operator [101]. Detail-preserving merging is discussed in Section 3.3.3.

Figure 3.1 shows the result of merging a source mesh to a target mesh in two different configurations. This example demonstrates that, given the same boundary correspondence, our configuration-independent merging method produces the same result. Figure 3.4 demonstrates that our method also works well for merging boundaries of different shapes and with undulations. Figure 3.5 shows an example with multiple pairs of merging boundaries. The center CAD model (source mesh to be deformed) is to be merged to the four cylinders (fixed target meshes). The CAD model is symmetrical, but the four cylinders have different scales. With configuration-dependent merging [19], the user would have great difficulty adjusting the configuration of the CAD model. Our configuration-independent merging method creates the same result independent of the configuration. This example also demonstrates that the merging method is applicable to meshes with non-zero genus. The close-up shows that the smoothness across the merging boundary is not satisfactory. Section 3.3.3 addresses this problem by using overlapping transition merging regions.

3.3.2 Configuration-Dependent Merging

In configuration-independent merging, the entire source mesh is deformed and merged to the target mesh. If the size of the source mesh is very large, computation would be expensive. Allowing only the region near the source merging boundary to be deformed can accelerate the merging progress. Moreover, the user may want to specify handles so as to fix certain features on the source mesh. Such tasks can be achieved using configuration-dependent merging.

For configuration-dependent merging, the user specifies the configuration of the source and the target mesh as well as a set of handles on the source mesh. The remaining algorithm is the same as the configuration-independent merging except the following. We modify only the positions of the source boundary vertices and their scaling factors, keeping their local frames unchanged. The deformed source merging boundary and the fixed handles together provide the boundary condition to the optimization problem. This is similar to the transplanting method in [19].

Figure 3.6 compares some results of the two merging methods. With configuration-independent merging, the legs of the Feline model are not in harmony with the legs of the Dinosaur model. With configuration-dependent merging, we adjust the position, orientation and scale of the Feline model and fix the two feet, and the merging result is better.

3.3.3 Merging with Overlapped Transition Region

We have assumed so far that no transition region is specified for merging. The target mesh is never deformed. In order to coincide with the target merging boundary, the source merging boundary is deformed (consequently deforming the whole source ROI too). Since the geometry near the target merging boundary and the geometry near the deformed source boundary are usually different, smoothing needs to be performed [20]. This smoothing step however also filters out geometry details.

Mesh	Feline Fig. 3.6 (c)	CAD Fig. 3.5	Feline Fig. 3.6 (a)	Mannequin head	Headus skull
Free Vertices	7,733	9,050	9,434	16,219	16,982
Degenerate Vertices	186	1,839	206	534	77
Factor (Eqn 2.8)	0.593s	0.563s	0.718s	1.437s	1.390s
Solve (Eqn 2.8)	0.032s	0.031s	0.032s	0.062s	0.062s
Factor (Eqn 2.10/2.11)	0.203s	0.110s	0.125s	0.234s	0.203s
Solve (Eqn 2.10/2.11)	0.015s	0.015s	0.016s	0.032s	0.031s
SVD	0.125s	0.156s	0.187s	0.328s	0.329s

Table 3.1: The timing results for our configuration-independent and configuration-dependent merging examples on a 3.2GHz Pentium IV machine with 1G RAM.

To produce merging region with smoothly transited details, the user could specify a transition region on both the source mesh and the target mesh [19]. The ROIs of both meshes are deformed and the details are mixed by interpolating the LCs in the transition regions. For configuration-independent merging, we first apply the implicitly defined local deformation gradients onto the LCs before interpolating them.

We let the user specify a width for each mesh and define the transition region as the region encompassing all vertices whose shortest path to the merging boundary has a length less than a specified width. Correspondence between the merging boundaries is established by specifying several key vertex correspondences (Figure 3.7 (a)). For correspondence between the transition regions, we cut each transition region along the shortest path between two key vertices and parameterize the region over the unit square domain. A vertex $\mathbf{v}_i$ in one transition region is mapped to a position $\overline{\mathbf{v}}_i$ within a triangle τ in the other transition region,

$$\overline{\mathbf{v}}_i = b_{i_1}\mathbf{v}_{i_1} + b_{i_2}\mathbf{v}_{i_2} + b_{i_3}\mathbf{v}_{i_3},$$

where $\mathbf{v}_{i_*}$ are the vertices of triangle τ and b_{i_*} are the corresponding barycentric coordinates.

We add an additional error term E_T to the optimization problem in Equation 2.8.

$$E_T = \sum_{i \in TR} \|M_i\delta_i - \sum_{j=1}^{3} b_{i_j} M_{i_j} \delta_{i_j}\|^2,$$

where TR is the index set of vertices in the transition regions, and M_{i_*} and δ_{i_*} are the corresponding affine transformations and Laplacian coordinates of $\mathbf{v}_{i_*}$, respectively.

Adding the error term to the objective function of the optimization problem, we get

$$\text{arg min } E(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n) = E_L + E_R + w_T E_T,$$

where w_T is a weight (e.g. $w_T = 0.1$ in our experiments). By minimizing the objective function with the same boundary condition as before, we obtain a merged mesh. The optimal solution, however, generally does not overlap two transition regions exactly. Therefore, like the method in [19], we reconstruct a smooth transition by using the connectivity information of one transition region and the linearly interpolated LCs (Figure 3.7 (b)).

3.4 Optimal Boundary for Poisson Mesh Merging

Our boundary condition optimization for Poisson mesh merging is inspired by [94]. However, extending the boundary condition optimization algorithm from Poisson image pasting [94] to Poisson mesh merging is not straightforward.

First, the optimization problem in mesh merging becomes more complicated, with more unknowns introduced. In 2D image editing [94], the resulting composite has the best quality when the difference between the source and target boundary conditions is a constant (i.e., corresponding to a pure translation of the boundary condition in color spaces). In Poisson mesh merging, the desirable scenario is when there exists a similarity transformation (consisting of a rigid transformation and uniform scaling) between the source and target merging boundaries, as there would be no shearing or stretching distortion in the reconstructed meshes [23]. However, such desirable transformation does not exist in most merging scenarios, thus we search for a least-squares solution: the change of the boundary condition in the source mesh during merging is minimal in terms of an unknown similarity transformation.

Second, irregular sampling in meshes makes the objective function in [94] inapplicable to Poisson mesh merging. For images, whether or not sampling factor is considered in the objective function is insignificant due to the regular structure. However, sampling consideration is crucial when designing the objective function to optimize the boundary condition in 3D.

3.4.1 System Overview

We give a system overview in this section. Our goal is to find an optimal merging boundary on the target mesh to be used as the boundary condition to deform the part of the source mesh containing the features to be pasted onto the target mesh. The target mesh remains undeformed. Our system contains the following main steps (Figure 3.8):

1. The user casually marks a region of interest Ω_0 on the source mesh. This region should be large enough to cover the features $\Omega_{feature}$ (i.e. $\Omega_{feature} \subset \Omega_0$) that the user really wants to paste onto the target mesh. Intuitive cutting techniques, e.g. easy mesh cutting [102], can be used to identify $\Omega_{feature}$. To avoid having the optimal boundary $\partial\Omega$ cutting into $\Omega_{feature}$, we constrain $\partial\Omega$ to be within the region $\Omega_0 \setminus \Omega_{feature}$. On the target mesh, the user roughly chooses a region Ω_1 onto which the features from the source mesh are to be pasted. We assume that Ω_1 does not contain complex features; otherwise we simply remove the features before the pasting so as to reduce distortion in the next parameterization step.

2. For each vertex in $\Omega_0 \setminus \Omega_{feature}$, we find the corresponding position on Ω_1 (Figure 3.10). Unlike 2D image editing, there is no explicit correspondence between the source and target meshes in 3D. We use one of state-of-the-art surface parameterization methods, least squares conformal maps [103], to build the correspondence.
   Unfortunately, the band shape of $\Omega_0 \setminus \Omega_{feature}$ often leads to large parameterization distortion, which might defeat the gain from having an optimal boundary condition. To reduce parameterization distortion, we first fill the hole induced by boundary $\partial\Omega_{feature}$ through an optimal triangulation that minimizes the total triangle area [104]. We then parameterize the surface $(\Omega_0 \setminus \Omega_{feature}) \cup \Omega_{filled}$, where Ω_{filled} is the region resulting from the boundary triangulation. Replacing $\Omega_{feature}$ with Ω_{filled} for the purpose of correspondence building has the following advantages. First, additional distortion would not be introduced from parameterizing $\Omega_{feature}$, which could be of complicated geometric shape. Second, as $\Omega_{feature}$ is not used in surface parameterization, this region can be of complex topology (e.g. with nonzero genus).
   To obtain a meaningful correspondence, the user needs to manually translate,

scale and rotate the parameterization of $\Omega_0 \setminus \Omega_{feature}$ with respect to the parameterization of Ω_1. The relative positions of the source and target models are roughly fixed once the correspondence is determined. The subsequent algorithm only fine tunes the final orientations and scalings. Therefore, the user can anticipate the composition effect when specifying the correspondence.

3. We search for a closed path $\partial\Omega$ within region $\Omega_0 \setminus \Omega_{feature}$ as the optimal boundary condition through an iterative optimization algorithm (Section 3.4). To guarantee that $\partial\Omega$ encloses $\Omega_{feature}$, we cut across the ring-like region $\Omega_0 \setminus \Omega_{feature}$ and search for a boundary that begins and ends at this cut.

4. We move the vertices on $\partial\Omega$ to their corresponding target positions on Ω_1 and perform Poisson mesh merging to deform the region enclosed by $\partial\Omega$ (containing $\Omega_{feature}$). We choose to use the Laplacian editing framework presented in Chapter 2.4.

3.4.2 Boundary Energy Minimization

Following the above discussion, an optimal boundary for Poisson mesh merging is a boundary $\partial\Omega$ in $\Omega_0 \setminus \Omega_{feature}$ such that the transformation between $\partial\Omega$ and its corresponding boundary $\partial\Omega^*$ on Ω_1 is as close as possible to an unknown similarity transformation $\mathbf{T}$. We formulate the objective function to be minimized as follows:

$$E(\partial\Omega, \mathbf{T}) = \sum_{e \in \partial\Omega} \|\mathbf{T}e - e^*\| \cdot length(e), \quad \partial\Omega \subset \Omega_0 \setminus \Omega_{feature}, \tag{3.1}$$

where e is any edge on $\partial\Omega$, i.e., a vector with its endpoint positions as the starting and ending points, e^* is the corresponding edge of e on Ω_1, and $\|\cdot\|$ denotes L2 vector norm. As meshes often have irregular sampling of geometry, we add the term $length(e)$ to prevent the optimal boundary from bypassing regions with dense sampling. Another desirable effect of adding the weighting factor is that the length of the optimal boundary will be as short as possible, pushing it to approach $\partial\Omega_{feature}$.

3.4.3 Iterative Optimization

Since the optimal boundary might contain all the vertices in $\Omega_0 \setminus \Omega_{feature}$, minimizing $E(\partial\Omega, \mathbf{T})$ to solve for the optimal merging boundary and the transformation simultaneously is intractable. Similar to [94], we use an alternating method to solve the minimization problem iteratively. Mainly, it contains the following steps:

1. Initialize $\partial\Omega$ as $\partial\Omega_0$.

2. Given the current boundary $\partial\Omega$ on the source mesh and its corresponding boundary $\partial\Omega^*$ in Ω_1 on the target mesh, we compute the optimal similarity transformation $\mathbf{T}$. Specifically, given the corresponding sets of points on $\partial\Omega$ and $\partial\Omega^*$, we use the algorithm in [105] to compute a rigid motion. The uniform scaling factor is defined as the ratio of the average edge length of $\partial\Omega^*$ to that of $\partial\Omega$.

3. Given the current transformation $\mathbf{T}$, we optimize the boundary $\partial\Omega$.

4. Repeat steps 2 and 3 until the change of the energy $E(\partial\Omega, \mathbf{T})$ converges or it reaches a prescribed maximum number of iterations.

Given $\mathbf{T}$, solving for $\partial\Omega$ by minimizing the boundary energy $E(\partial\Omega, \mathbf{T})$ is equivalent to finding a shortest path in $\Omega_0 \setminus \Omega_{feature}$. However, we have an additional requirement here: $\partial\Omega$ should enclose $\Omega_{feature}$. The path found by a standard shortest path problem (e.g. Dijkstra algorithm [106]) is very likely not the one we need. To fulfill the requirement, we first break the ring-like region $\Omega_0 \setminus \Omega_{feature}$ by adding a cut C, as shown in Figure 3.8, and then find a shortest path that starts and ends at C. After cutting, each original vertex on the cut C is split into two vertices, on different sides of C.

To achieve better performance, we want a cutting path C with minimal number of vertices. A zigzag cut C may make the found shortest path $\partial\Omega$ intersects C more than once, thus leading to a non-optimal boundary condition [94]. Straightening the cut can greatly reduce the possibility of multiple intersections. Therefore we find the shortest path with a source vertex on $\partial\Omega_0$ and a sink vertex on $\partial\Omega_{feature}$ as the cut C. The shortest path is computed using Dijkstra algorithm [106] with edge lengths as the weighting costs.

Given a cut C, we show how to compute a closest shortest path that begins and ends at a vertex on C as the boundary $\partial\Omega$. We associate each edge e with cost

$$||\mathbf{T}e - e^*|| \cdot length(e)$$

. The accumulated cost of a path is defined as the summation of the costs of all edges on the path. For each vertex u on one side of the cut C, we use Dijkstra algorithm to compute the shortest path $path(u)$ with minimal cost to the vertex v which is originally split from the same vertex as u. The optimal boundary $\partial\Omega$ is set as the one with the minimum cost from the set $\{path(u) \mid u \in C\}$.

Like ours, the algorithm proposed by Hassner et al. [100] finds a merging boundary respecting both the source and target models. Their solution consists of two main steps: model alignment (to find an appropriate transformation) followed by a minimal cut. In a sense, their solution only corresponds to one iteration of ours and thus is not optimal.

3.4.4 Examples and Discussion

In this section, we demonstrate that optimal boundary conditions lead to merging results with less distortion (i.e., less shearing and stretching) than those reconstructed using user-specified unoptimized boundary conditions Ω_0. We do not compare the change of the global shapes of the source features when using the two different types of boundary conditions, as it is dependent on the scale factor. Instead, we compare the local distortions of the merged source meshes. For the optimal or unoptimized boundary condition, we use the ratio of the average length of the source and target merging boundaries to uniformly scale the Laplacian coordinates of the source mesh to account for the difference in the sizes between the source and target boundaries.

When the source and target merging boundaries are of very different shapes, the deformed source mesh inevitably exhibits local distortion. The distortion is more noticeable near the merging boundary, as the merging boundary serves as soft or hard constraint to the deformation optimization. For the source and target models in Figure 3.2, the region $\Omega_0 \setminus \Omega_{feature}$ on the source mesh and its corresponding region on the target mesh are of very different shapes (Figure 3.10). Using the user-specified boundary condition, the local distortion, especially near the boundary (i.e. the waist region), is large. In contrast, the optimal boundary condition leads to a much better merging result. Better merging results are also demonstrated by the examples in Figures 3.9, 3.11, 3.12 and 3.13, when optimal merging boundaries are used.

We need a planar surface parameterization to build the correspondence of $\Omega_0 \setminus \Omega_{feature}$ between the source and target meshes. However, as $\Omega_{feature}$ itself is not involved in the parameterization step, our system does not require the topology of $\Omega_{feature}$ to be homeomorphic to a disk. This is demonstrated by the example in Figure 3.14.

It is hard to theoretically prove the convergence of the proposed iterative method. However, experiments show that the iterative process has no convergence problem. Although the iterative process may fall into a local minimum, the resulting boundaries always lead to better merging results than those with unoptimized boundaries. As the time complexity of the Dijkstra algorithm is $O(N \log N)$, for each iteration, the overall computational complexity of finding an optimal boundary is $O(MN \log N)$, where M and N are the number of vertices on the cut C and in $\Omega_0 \setminus \Omega_{feature}$, respectively. For example, given a region $\Omega_0 \setminus \Omega_{feature}$ with about 10K vertices, it takes about 1 minutes to compute the final optimal boundary condition. Performing the Laplacian deformation is very efficient, as demonstrated in the previous chapter.

3.5 Summary

We present an easy-to-use Poisson mesh merging tool without needing adjusting of relative configuration of input meshes to be merged and without requiring careful user-specified merging boundaries. Our algorithm automatically finds an optimal boundary based on the information casually provided by the user and produces visually better merging results.

We have shown that, in configuration-dependent merging, the user can choose to fix specific features during merging. This capability can be extended to configuration-independent merging. For each feature to be fixed, we implicitly define a common deformation gradient for all the Laplacian coordinates associated with the feature. In other words, the feature is subject to a transformation rather than being completely fixed like in configuration-dependent merging.

The distortion introduced in the correspondence building step using surface parameterization definitely influences the final merging results. For models with complex shapes at $\Omega_0 \setminus \Omega_{feature}$, the parameterization distortion might defeat the gain from the optimal boundary. As a future work, we will explore other registration methods, e.g. iterative closest point (ICP) algorithm [107] or its variants, instead of using direct surface parameterization techniques.

The change of the local frames at each vertex of the boundary condition before and after merging can be used to increase the smoothness across the merging boundary [20]. We plan to incorporate the local rotations or similarity transformations into the boundary condition optimization formulation to find optimal boundaries that will lead to better smoothness across the merging boundary.

Currently, we search for the shortest paths on the graph of the original mesh, which restricts the found optimal boundary to be composed of the mesh edges. Getting rid of this constraint may further improve the quality of merging results.

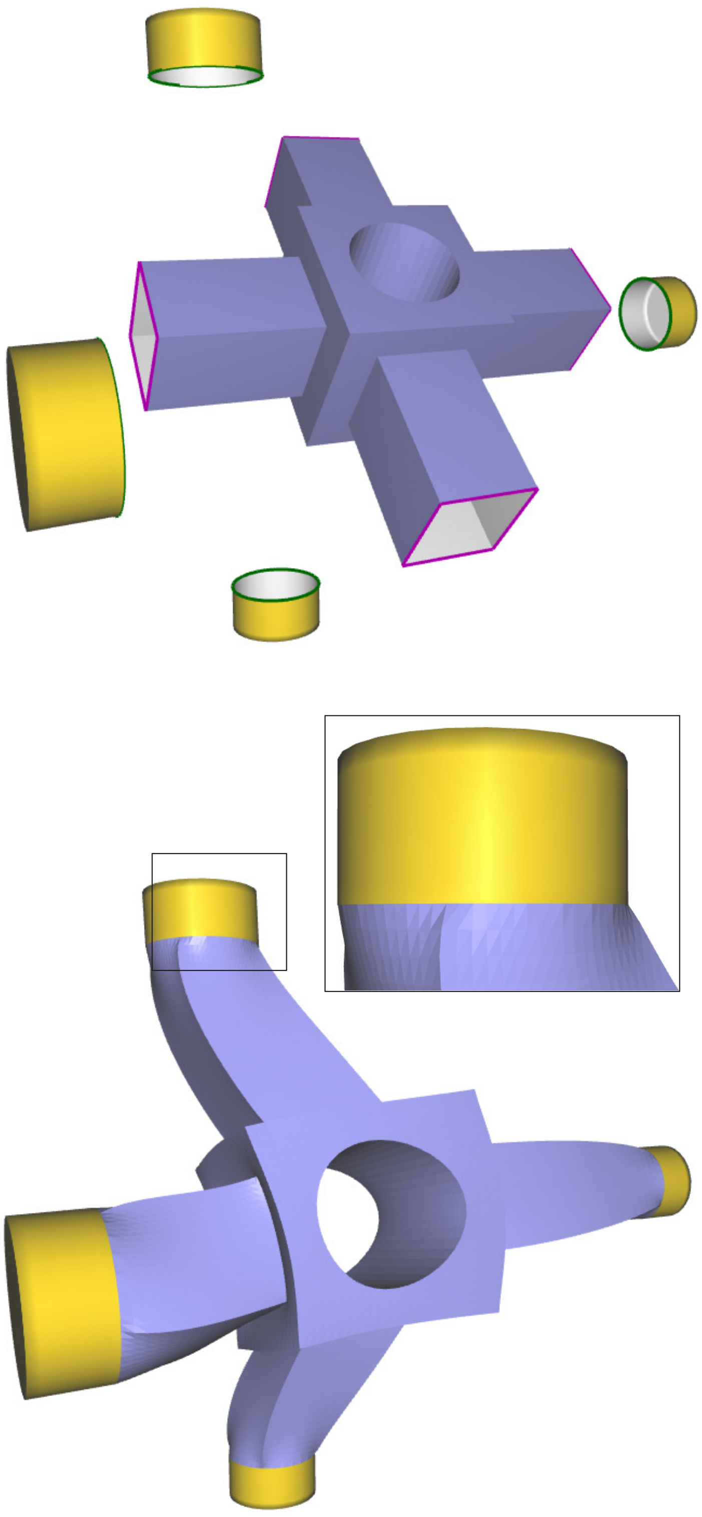


Figure 3.5: A configuration-independent merging example with multiple pairs of merging boundaries.

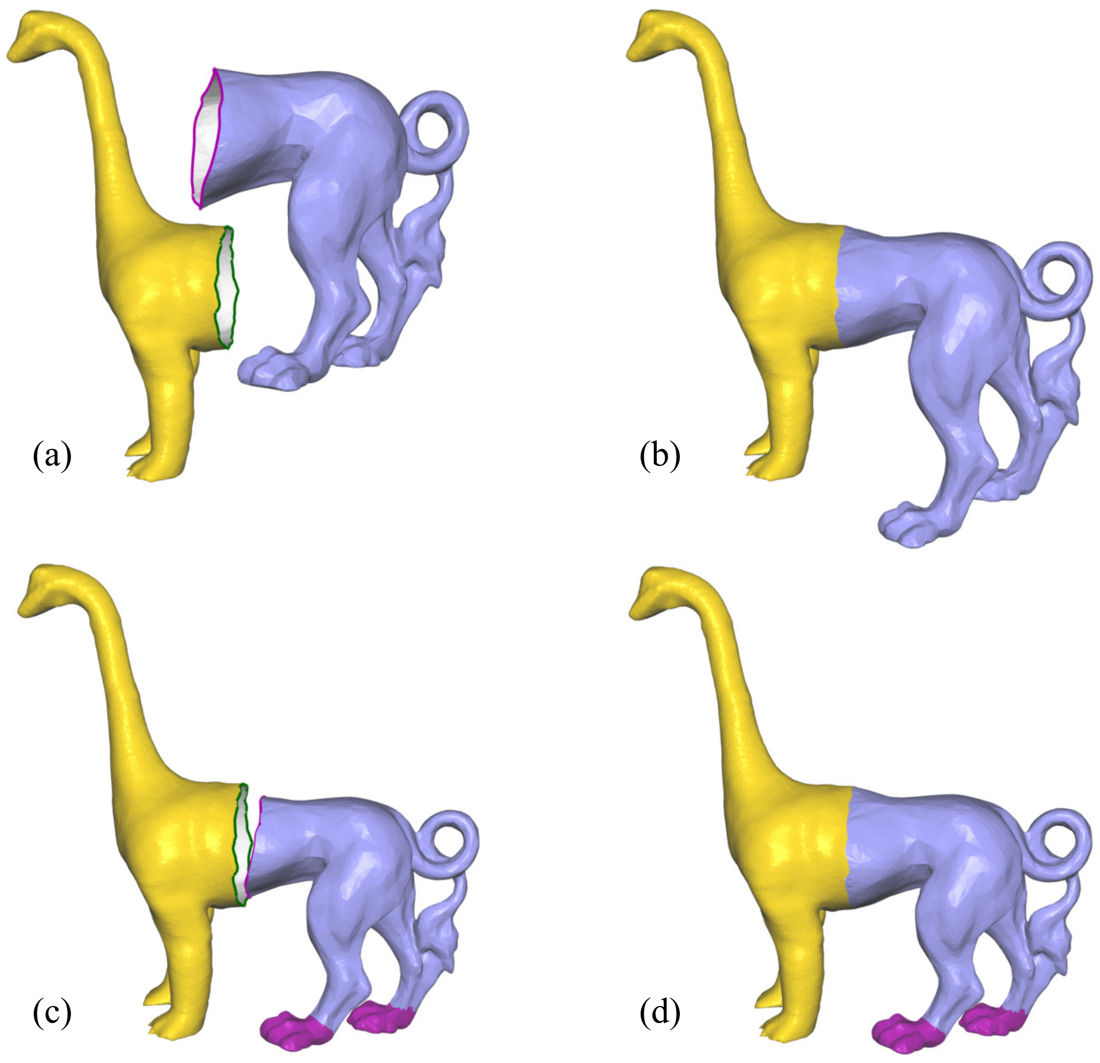


Figure 3.6: The hind part of the Feline model is deformed and merged to the fore part of the Dinosaur model. (b) is the result of configuration-independent merging with the configuration in (a), and (d) is the result of configuration-dependent merging with the configuration in (c). In the configuration-dependent merging, the two feet are specified as handles (purple), thus remain fixed. The same boundary correspondence is used for both merging.

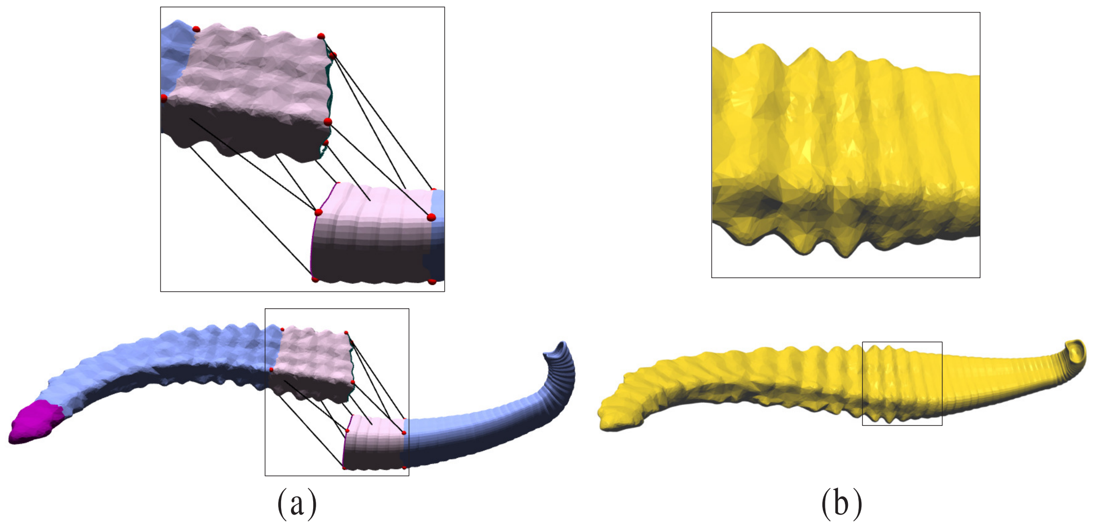


Figure 3.7: Configuration-independent merging with overlapped transition regions. (a) Two transition regions with key correspondences specified. (b) The merging result.

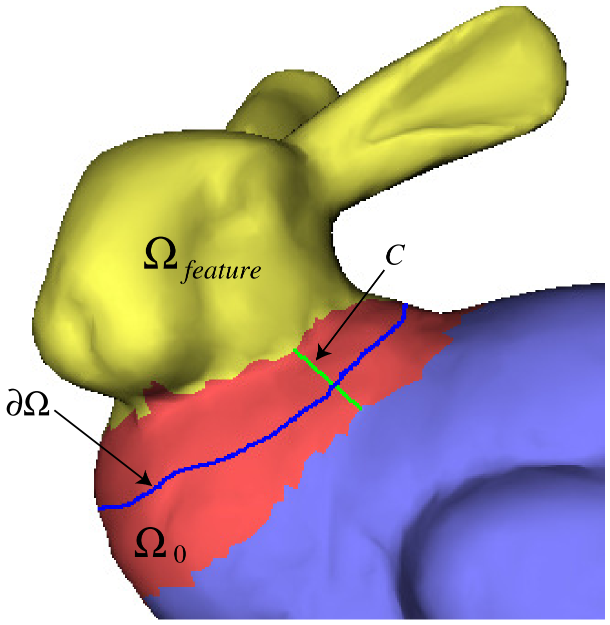


Figure 3.8: An illustration of different types of boundaries and regions on the source mesh. Ω_0 is the region of interest (in red) casually marked by the user. $\Omega_{feature}$ contains the features to be merged (in yellow). The optimal boundary $\partial\Omega$ (in blue) lies in the region $\Omega_0 \setminus \Omega_{feature}$. Cut C (in green) breaks the ring of $\Omega_0 \setminus \Omega_{feature}$.

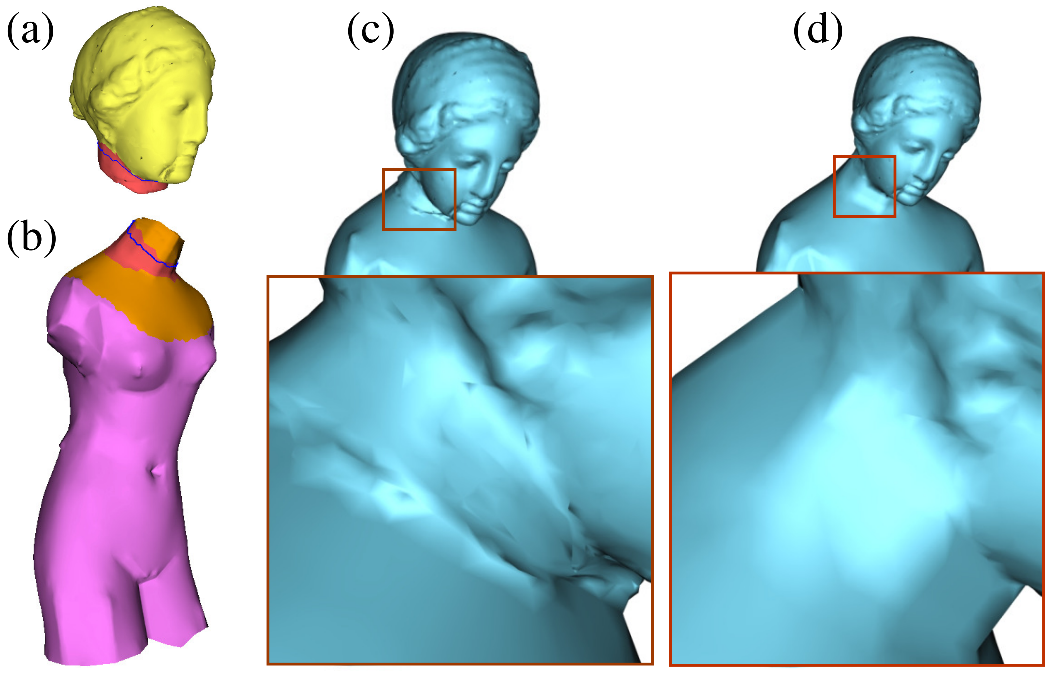


Figure 3.9: (a) Source. (b) Target. (c) and (d) are the merging results using the unoptimized and optimal boundary conditions, respectively.

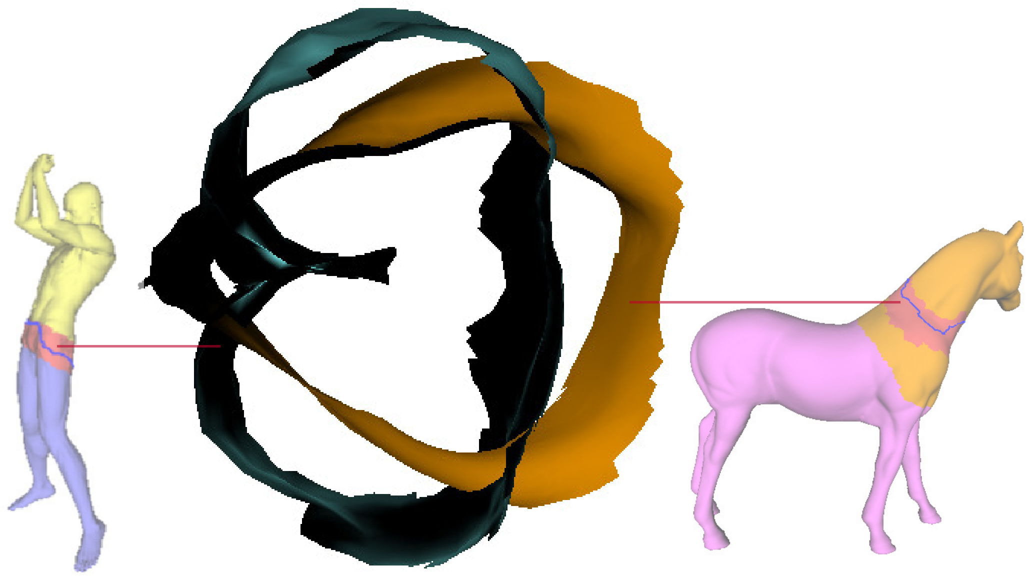


Figure 3.10: Region $\Omega_0 \setminus \Omega_{feature}$ on the source mesh and its corresponding region on the target mesh.

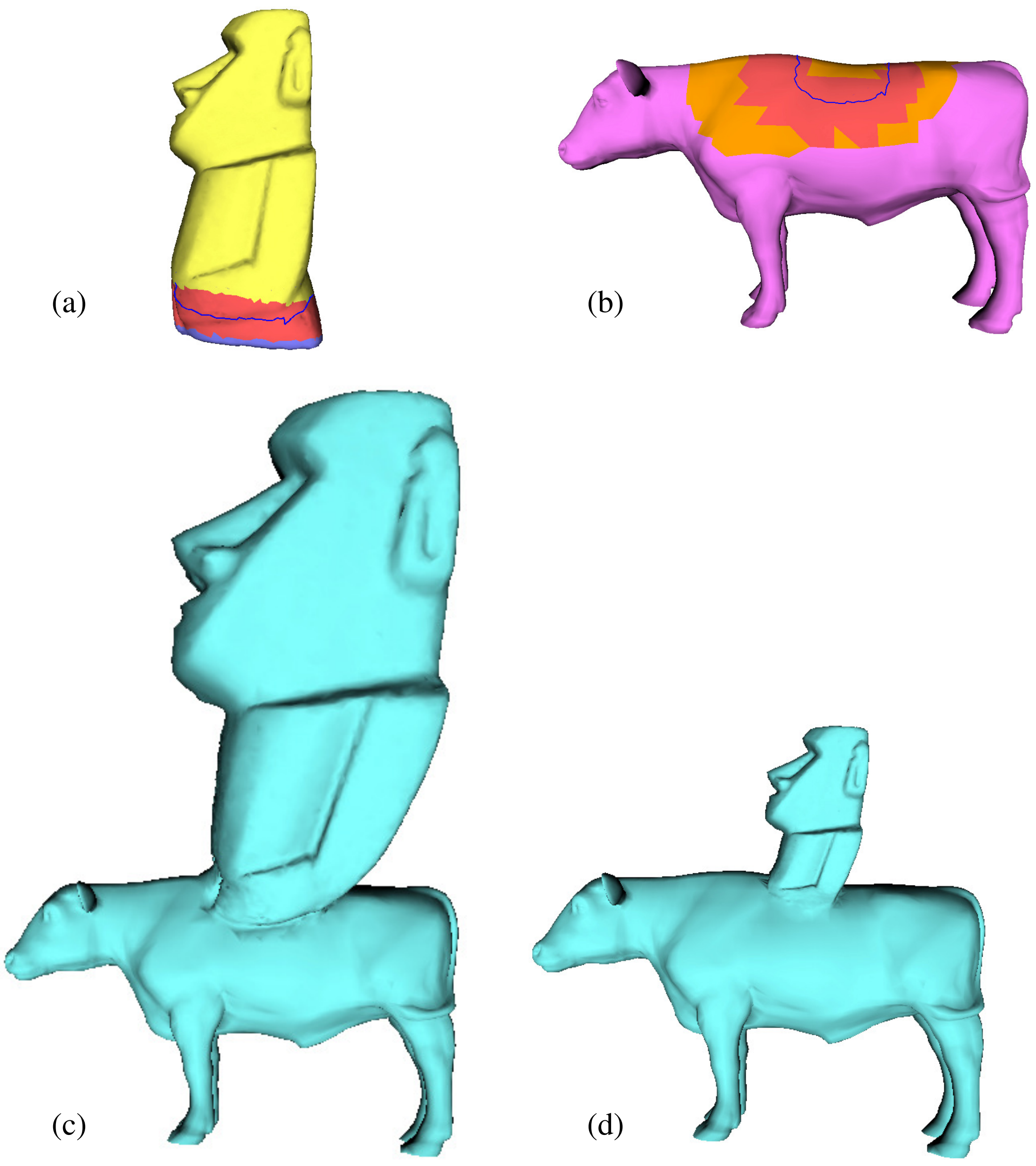
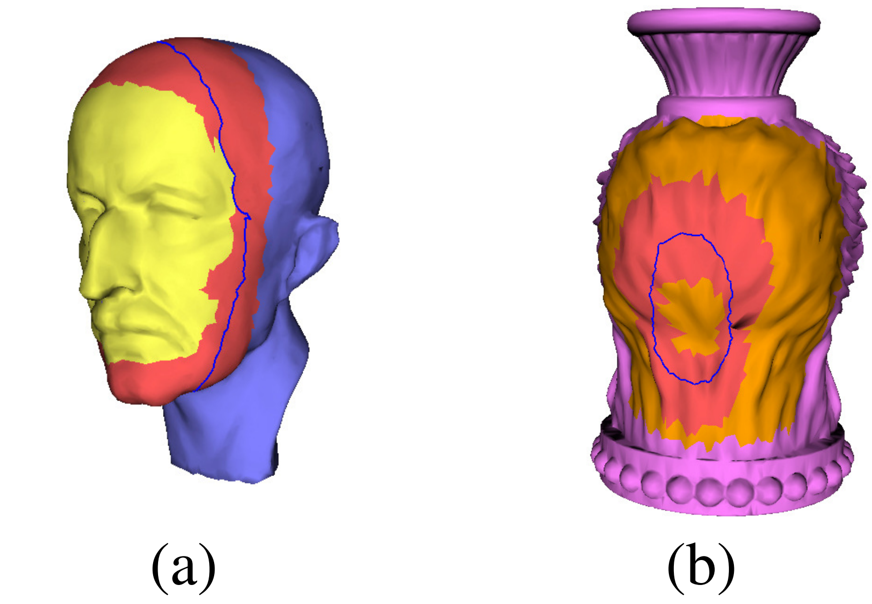


Figure 3.11: (a) Source. (b) Target. (c) and (d) are the merging results using the unoptimized and optimal boundary conditions, respectively.



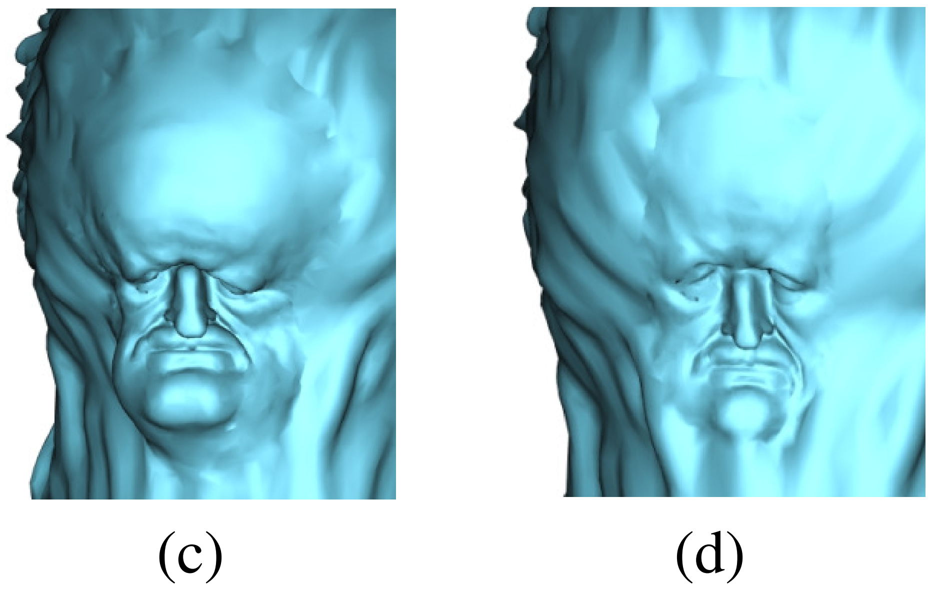


Figure 3.12: (a) Source. (b) Target. (c) and (d) are the merging results using the unoptimized and optimal boundary conditions, respectively.

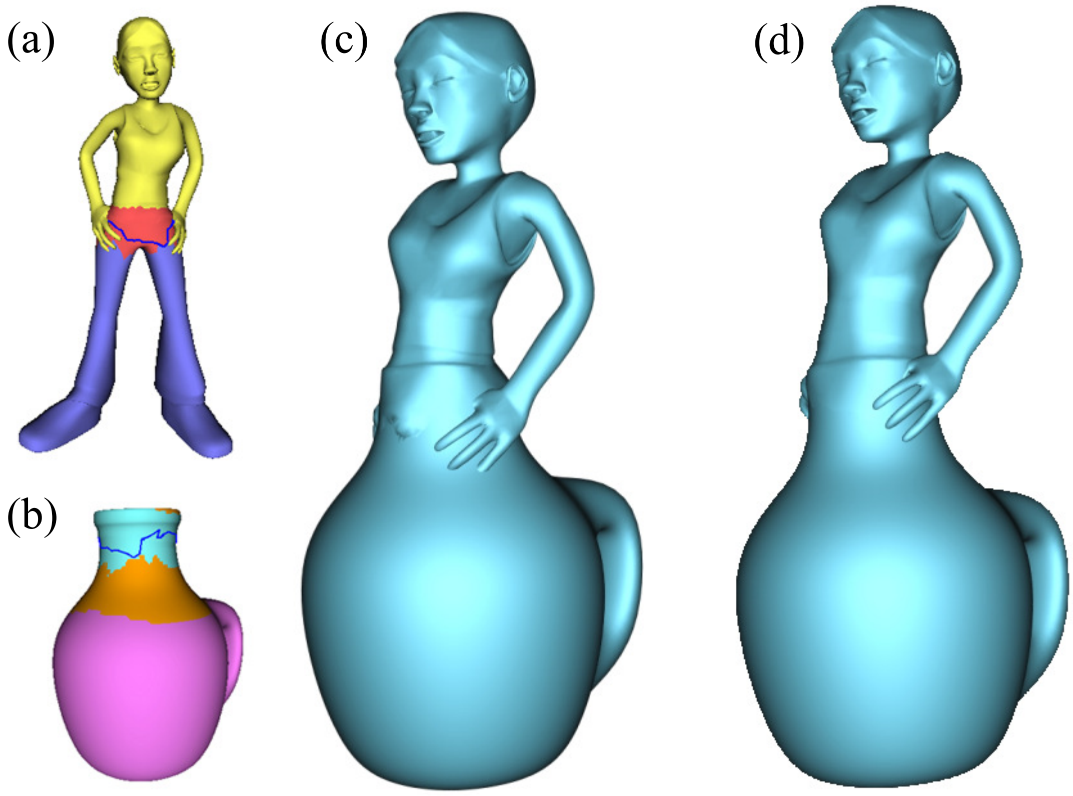


Figure 3.13: (a) Source. (b) Target. (c) and (d) are the merging results using the unoptimized and optimal boundary conditions, respectively.

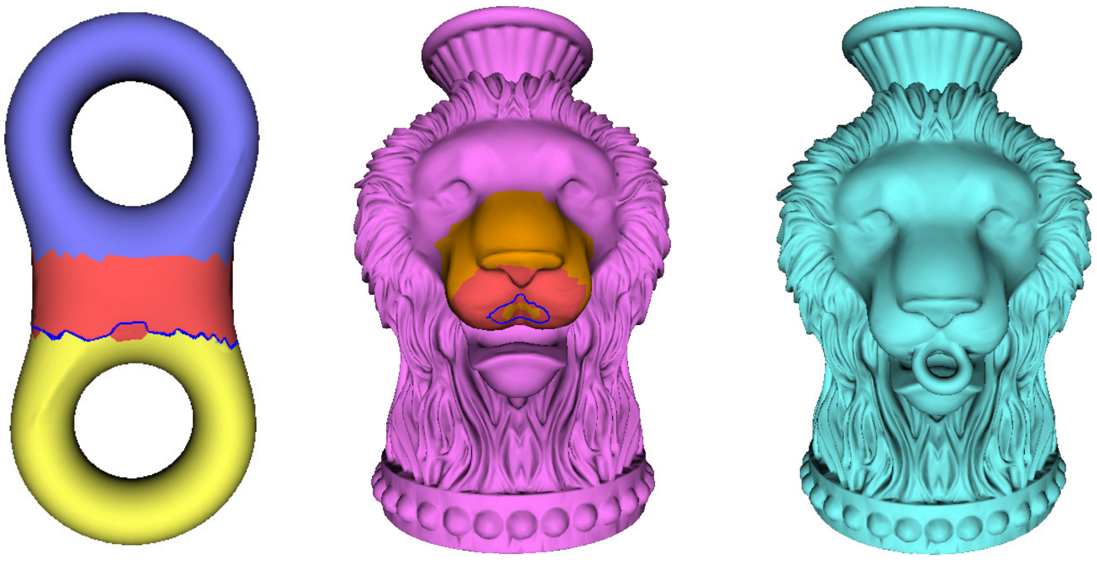


Figure 3.14: Our merging is applicable to a region of interest with nonzero genus.

CHAPTER 4

HAIRSTYLE SKETCHING

This chapter presents an intuitive sketching interface for interactive hairstyle design, made possible by an efficient numerical updating scheme. The user portrays the global shape of a desired hairstyle through a few 3D style curves which are manipulated by interactively sketching freeform strokes. Our approach is based on a vector field representation which is obtained by solving a sparse linear system with the style curves acting as boundary constraints. The key observation is that the specific sparseness pattern of the linear system enables an efficient incremental numerical updating scheme. This gives rise to a sketching interface that provides interactive visual feedback to the user. Interesting hairstyles can be easily created in minutes.

4.1 Introduction

Realistic looking hair is an important feature of virtual characters which appear in many applications, such as movies and games. While significant progress has been made on hair simulation [108] and rendering (see [109] and references therein), hair modeling still remains a difficult problem. This is due to the huge number of individual hair curves on a human head (typically more than 100K) and the large variance of hairstyles.

The key contribution of our work is a hairstyle design system equipped with a sketching interface and a fast vector field solver. The user draws freeform strokes to create and edit a few style curves which depict the global shape of the desired hairstyle. The hairstyle is then generated by growing along the flow lines in a vector field, which is transparent to the user. The vector field is formulated as the solution of a sparse linear system $\mathbf{Ax} = \mathbf{b}$ with the style curves acting as boundary constraints. Despite the high sparsity of $\mathbf{A}$, directly solving the system is still too slow (more than twenty seconds for 50K variables) for user interaction. Instead, we observe that modifying the style curves induces only changes of $\mathbf{b}$ and the diagonal elements of $\mathbf{A}$. Once initialized,

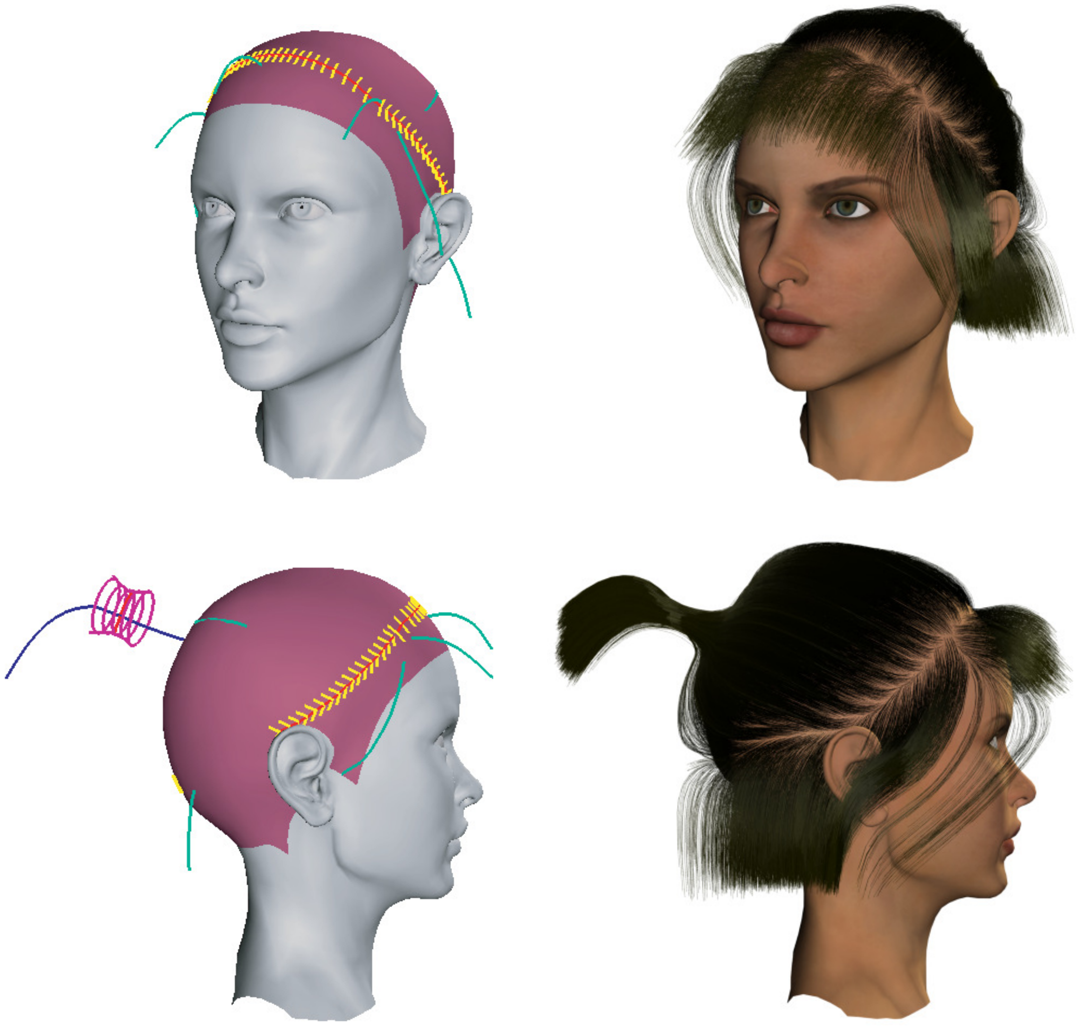


Figure 4.1: A realistic hairstyle created using our system **in five minutes**. The user is allowed to design interesting hairstyles by intuitively sketching three types of style primitives: streaming curve, dividing curve and ponytail.

the linear system can be efficiently re-solved incrementally due to the special pattern, usually taking only a few seconds.

The combination of the sketching interface and the efficient vector field solver gives rise to a user-friendly system. The user continuously draws strokes to modify the hairstyle, responding to the interactive feedback, until satisfied. Interesting hairstyles can be easily created (see Figure 4.1 for an example).

4.2 Related Work

A variety of hair modeling techniques have been proposed (see the latest survey in [110]). We review only the work most related to ours.

Direct Hair Modeling. Many previous interactive hair modeling techniques directly manipulate the geometry of hair curves, or a group of hair curves, called a hair cluster [111, 112, 113]. Modeling a complete hair model with such techniques could be tedious and time-consuming (usually several hours) since hundreds of hair clusters have to be created manually.

Vector Field-based Hair Modeling. Vector field-based techniques can effectively reduce manual work by automatically tracking the hair curve flow in a vector field. The idea was first explored by Hadap and Magnenat-Thalmann [114]. Yu [115] extended the idea by introducing more vector field primitives to create more complex hairstyles. A major limitation of these methods is that the global vector field is continuously represented as the superimposition of many local vector fields generated by those primitives. When the vector field is changed, several minutes are needed to re-evaluate the vector field and re-generate a hair model. The high computational cost makes user interaction inconvenient. Moreover, the vector field is modified via positioning and rotating primitives in space, whose effect on the hairstyle is not always intuitive [115]. Rather than using a single vector field, Choe et al.[116] proposed to apply individual vector fields each time to incrementally generate more complex hairstyles, such as braid hair. However, their styling vector fields are produced using a procedural approach, not allowing users to fully design hairstyles.

Sketching Interface for Hair Modeling. Sketching interface for 3D design has been proved intuitive [117, 29]. Mao et al. [118] were the first to apply sketching to hair modeling, but their approach only assumes symmetric smooth hairstyles. The recent approach of Malik [113] allows user to draw freeform strokes to mimic various hairstyling operations on individual hair clusters. Since the user directly manipulates the hair geometry and the influence of the editing operation is local, it is not easy to design a globally complex hairstyle. Wither et al. [119] proposed a sketch-based interface for controlling a physically-based hairstyle generator.

To let the user more easily control the global shape of a hairstyle, we constrain the hairstyle by a vector field which can be designed by sketching a small set of style primitives. Note that, parallel to our work, Takayama et al. [120] and Fisher et al. [121] have also proposed sketch-based interfaces for designing vector fields inside a volumetric 3D heart model and over arbitrary triangular meshes, respectively.

Image-based Hair Modeling. Recently, researchers have shown that automatic hair geometry reconstruction based on computer vision techniques is possible [122, 9]. Typically multiple images are captured about the subject head, and dense hair curve geometry is recovered from hair's photometric or stereoscopic properties. While current results are impressive, the use of these approaches are limited by data capturing conditions.

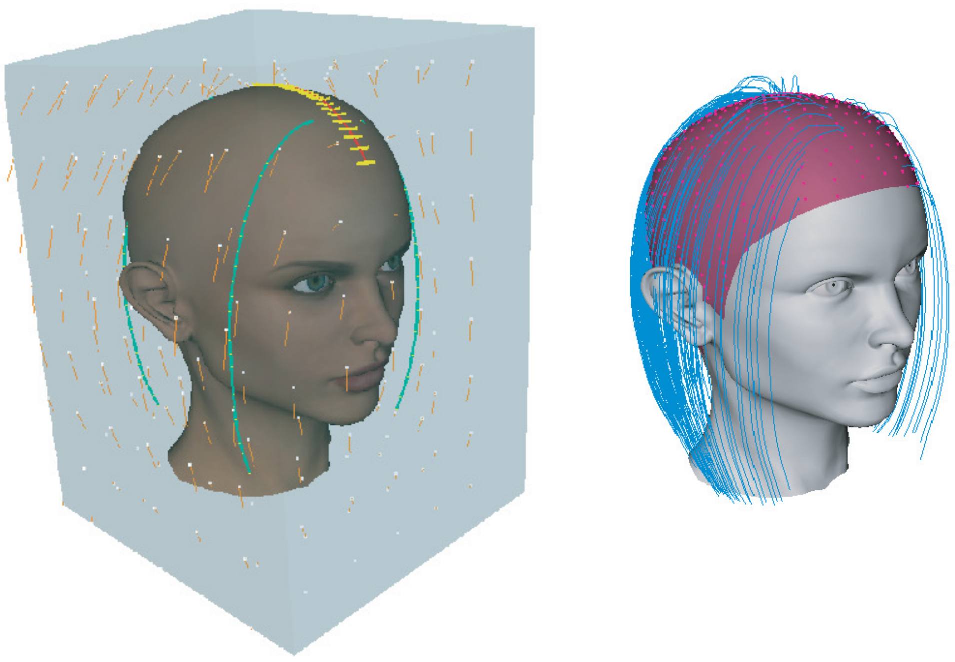


Figure 4.2: **Left**: bounding volume, vector field and style curves. **Right**: hair curves generated from scalp.

4.3 System Overview

Our system consists of four components (see Figure 4.2): a head mesh, a vector field defined in the bounding volume, a set of style curves, and a resulting hairstyle consisting of tens of thousands of hair curves. To design a specific hairstyle, the user first sketches a few style curves depicting the global hair shape. These style curves are created and modified via drawing freeform strokes. A discrete vector field is defined in a 3D uniform grid within the bounding volume of the head. This vector field is formulated as the solution of a linear system. Different boundary constraints are derived from the style curves to provide known directions for part of the vector field. For example, a stream curve (cyan curves in Figure 4.2) causes its neighboring grid points to have their directional vectors set along the curve's tangent, and a dividing curve (red curve in Figure 4.2) causes the neighboring grid points on its two sides to assume roughly opposite directions (indicated as short yellow lines).

Once solved, the vector field is used to automatically generate a hairstyle as follows. Each hair curve starts from a root point on the scalp and grows along the flow directions in the vector field. Equipped with an efficient incremental solver for the linear system, our system allows the user to modify the style curves, re-solve the linear system and generate the new hairstyle in several seconds. The interactive visual feedback greatly facilitates the design process.

4.4 Fast Vector Field Computation

This section introduces the linear system which produces a vector field for hair growth. We use a Laplacian system as a field interpolator, given the boundary constraints derived from the style curves. A fast solver based on incremental Cholesky factorization is presented.

4.4.1 Laplacian System as Field Interpolator

In recent years, the Laplace operator has been extensively adopted in mesh editing due to its ability to produce smooth deformation (see [69] and references therein). We adopt a similar formulation. For each vertex $\mathbf{v}_i$ of the grid, the discrete Laplace operator is defined as

$$\Delta(\mathbf{t}_i) = \sum_{j \in N(i)} \frac{1}{N(i)} (\mathbf{t}_j - \mathbf{t}_i),$$

where $\mathbf{t}_i$ is a directional vector defined at $\mathbf{v}_i$ and $N(i)$ is the index set of the 1-ring neighboring vertices of $\mathbf{v}_i$. In our hairstyling application, $\mathbf{t}_i$ indicates the tangent direction of a hair curve passing through $\mathbf{v}_i$.

We formulate the problem of field interpolation as a minimization problem with the cost function,

$$E(\mathbf{t}_1, \ldots, \mathbf{t}_n) = \sum_{i=1}^{n} ||\Delta(\mathbf{t}_i)||^2, \tag{4.1}$$

subject to

$$\mathbf{t}_i = \mathbf{c}_i, \quad i \in C$$

where $C = \{k_1, \ldots, k_m\}$ is the index set of the boundary constraints which specify the known directions $\mathbf{c}_i$ at certain vertices marked by the style curves (see the detailed specification in the next section). Unlike differential mesh editing, which allows using of hard constraints or soft constraints, the minimization here must use soft constraints to adapt incremental solver described in the next subsection. It is well known that the above minimization with the soft constraints applied is equivalent to solving the following linear system in a least-squares sense

$$\mathbf{A}\mathbf{t}^{(\mathbf{x})} = \begin{pmatrix} \mathbf{D} \\ \mathbf{W} \end{pmatrix} \mathbf{t}^{(\mathbf{x})} = \begin{pmatrix} \mathbf{0} \\ \omega\ \mathbf{c}^{(x)} \end{pmatrix} = \mathbf{b}^{(x)}, \tag{4.2}$$

where matrix $\mathbf{D}$ is an $n \times n$ matrix with the entries obtained from the discrete Laplacian, $\mathbf{W} = (w_{ij})_{m \times n}$, and w_{ij} is ω ($\omega = 100$ in our implementation) if $k_i = j$, and 0 otherwise. The column vectors $\mathbf{t}^{(x)}$ and $\mathbf{c}^{(x)}$ contain the x-component of $\mathbf{t}_i$ and $\mathbf{c}_i$, respectively. Similar systems are defined for the y and z components. Solving Equation (4.2) in the least-squares sense is equivalent to solving the normal equation below:

$$\mathbf{A}^{\mathbf{T}}\mathbf{A}\mathbf{t}^{(x)} = (\mathbf{D}^{\mathbf{T}}\mathbf{D} + \mathbf{W}^{\mathbf{T}}\mathbf{W})\mathbf{t}^{(x)} = \mathbf{A}^{\mathbf{T}}\mathbf{b}^{(x)}. \tag{4.3}$$

Note that $\mathbf{W}^{\mathbf{T}}\mathbf{W}$ is a diagonal matrix.

4.4.2 Incremental Cholesky Factorization

When the user changes the style curves, the set of boundary constraints is updated (see details in Section 4.5). This requires $\mathbf{W}^{\mathbf{T}}\mathbf{W}$ to be updated, the right hand side of Equation 4.3 to be changed ($\mathbf{D}^{\mathbf{T}}\mathbf{D}$ always remains unchanged), and the system to be re-solved. Although the system matrix is very sparse, solving this system with $50K$ unknowns still takes more than twenty seconds.

A key observation is that changing the boundary constraints (via modifying the style curves) only affects the diagonal elements of $\mathbf{W}^{\mathbf{T}}\mathbf{W}$. Specifically, the number of

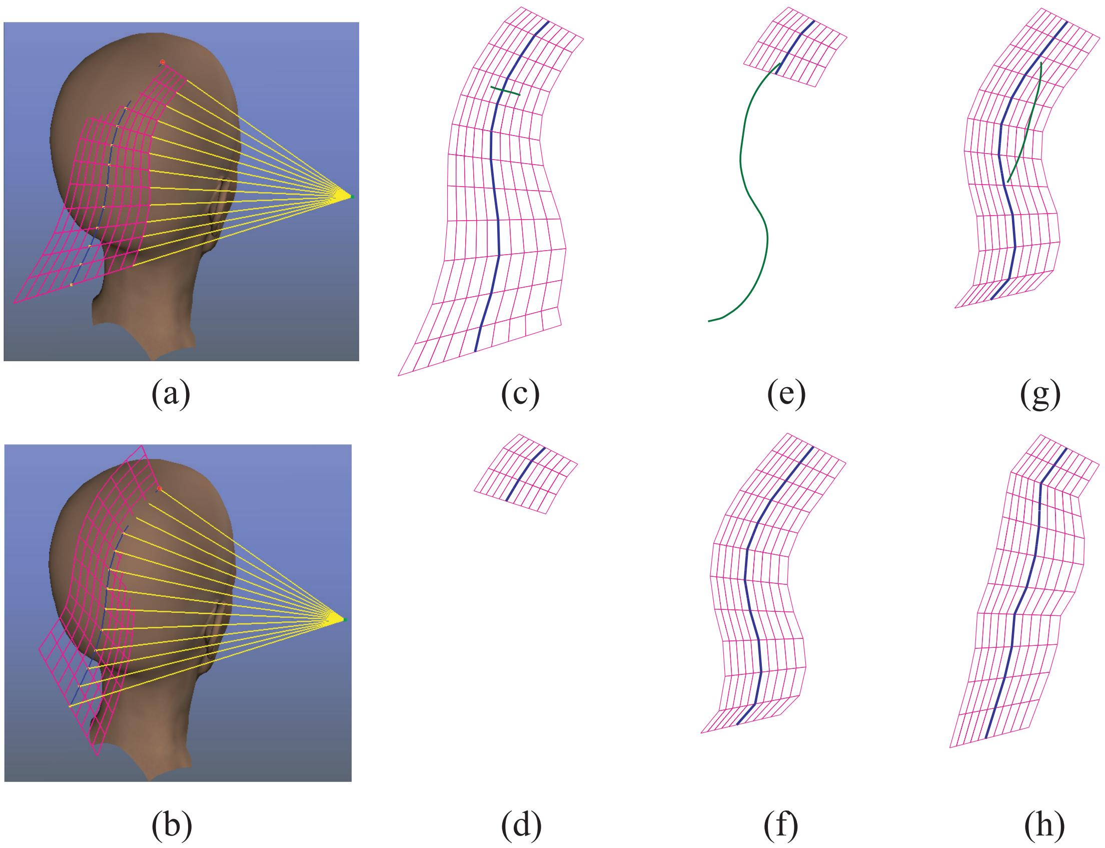


Figure 4.3: Editing a stream curve (blue). (a) and (b): two types of supporting surfaces (purple) and the viewpoint associated to a stream curve. (c) and (d): cut operation: the hint stroke (green) is long and runs across the stream curve on the supporting surface; (e) and (f): concatenation operation: the hint stroke starts near the stream curve and ends far away from it, possibly out of the supporting surface (in which case the depth is extrapolated from the last segment on the supporting surface); (g) and (h): insertion operation: the hint stroke starts and ends near the stream curve.

affected elements is $|\bar{C} - C| + |C - \bar{C}|$, where C and $\bar{C}$ are the old and new sets of boundary constraints, respectively. Consequently, for such special modifications to the system matrix, solving the normal equation by modifying the existing Cholesky factorization of $\mathbf{A}^\mathbf{T}\mathbf{A}$ is much more efficient than solving the system from scratch [123]. Efficient modification of sparse Cholesky factorization is only possible for some special cases. Specifically, given a sparse positive definite matrix $\mathbf{A}^\mathbf{T}\mathbf{A}$ and its associated Cholesky factorization, its modification is efficient only when $\mathbf{A}^\mathbf{T}\mathbf{A}$ changes in form of $\mathbf{A}^\mathbf{T}\mathbf{A} + \mathbf{R}^\mathbf{T}\mathbf{R}$ (called an *update*) or $\mathbf{A}^\mathbf{T}\mathbf{A} - \mathbf{R}^\mathbf{T}\mathbf{R}$ (called a *downdate*), where $\mathbf{R}$ is an arbitrary matrix [123].

We adopt the incremental Cholesky factorization to solve the system in Equation 4.3. First, we perform a general sparse Cholesky factorization

$$\mathbf{L}\mathbf{L}^{\mathbf{T}} = \mathbf{A}^{\mathbf{T}}\mathbf{A} = \mathbf{D}^{\mathbf{T}}\mathbf{D} + \mathbf{W}^{\mathbf{T}}\mathbf{W},$$

which is pre-computed only once. Given the new boundary constraints set $\bar{C}$, similar to the definition of $\mathbf{W}$, we let $\mathbf{W}_+$ and $\mathbf{W}_-$ denote the matrices corresponding to $\bar{C} - C$ and $C - \bar{C}$, respectively. The new system matrix is

$$\bar{\mathbf{A}}^{\mathbf{T}}\bar{\mathbf{A}} = \mathbf{D}^{\mathbf{T}}\mathbf{D} + \mathbf{W}^{\mathbf{T}}\mathbf{W} + \mathbf{W}_+^{\mathbf{T}}\mathbf{W}_+ - \mathbf{W}_-^{\mathbf{T}}\mathbf{W}_-.$$

The new Cholesky factor $\bar{\mathbf{L}}$ is computed by performing an update to $\mathbf{L}$:

$$\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{\mathbf{T}} = \mathbf{L}\mathbf{L}^{\mathbf{T}} + \mathbf{W}_+^{\mathbf{T}}\mathbf{W}_+,$$

followed by a downdate to $\widetilde{\mathbf{L}}$:

$$\bar{\mathbf{L}}\bar{\mathbf{L}}^{\mathbf{T}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{\mathbf{T}} - \mathbf{W}_-^{\mathbf{T}}\mathbf{W}_-.$$

Once the new Cholesky factor is computed, back-substitution is simply used to compute the vector field. We perform the incremental Cholesky factorization using an efficient sparse Cholesky factorization package [78]. The parallel work of Takayama et al. [120] and Fisher et al. [121] use numerical schemes similar to ours to incrementally solve for the vector fields.

4.5 Sketch-based Hairstyle Design System

Our current implementation supports three kinds of style primitives: stream curve, dividing curve and ponytail. The ponytail primitive is a composite style consisting of four style curves. We represent a 3D style curve as a sequence of connected line segments. Each segment is of the same length, set as the discretization size of the vector field for convenience. These style curves are created and modified by sketching freeform strokes (Figure 4.3).

Depth determination is the main difficulty in 3D editing using 2D input devices. Our system relies on the scalp surface and the *supporting surface* of a style curve for depth determination.

Stream Curve. This is the simplest style primitive. It indicates the general flow direction of the hairstyle. Every segment of a stream curve designates its neighboring

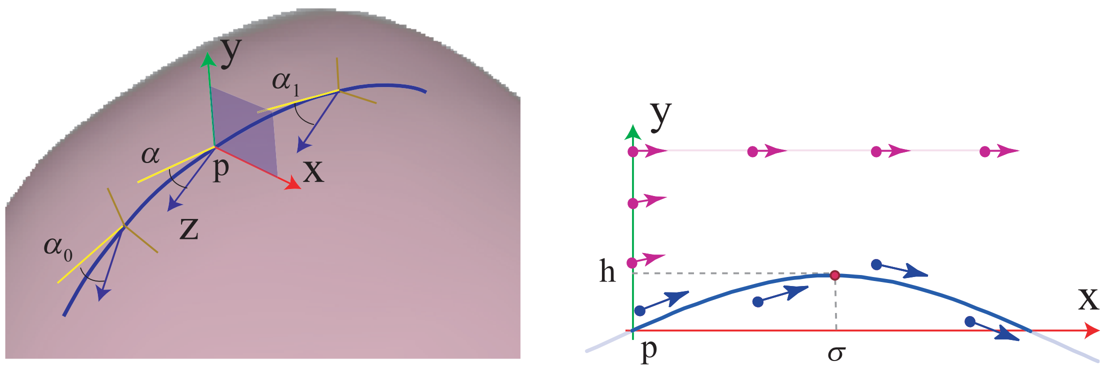


Figure 4.4: A dividing curve and its boundary constraints. **Left**: a local coordinate frame defined on the curve. Boundary constraints are defined in the local xy planes. The tilting angle α of a local plane is interpolated from two parameters α_0 and α_1. **Right**: boundary constraints on a local xy plane. Blue vectors mimic the local shape of the parting hairstyle and follow the tangents of a Gaussian function, which passes through the origin **p** at its inflexion point and with its local shape controlled by two parameters σ and h (set to 1 and 0.5 by default, with the unit as step size in the vector field). Red vectors explicitly represent the discontinuities around the parting line in the vector field and stop hair from growing across the parting line. Their magnitude is set smaller than 1 (we use 0.3) to reduce their global effect.

grid points as boundary constraints, each of which has its directional vector set as the direction of the segment's tangent (blue vectors in left of Figure 4.5). Optionally, to prevent hair growth beyond the end of the stream curve, we extrapolate and append a few extra segments. The magnitudes of the directional vectors associated to these extra segments vanish gradually, from unit length to zero (red vectors in left of Figure 4.5).

A stream curve is created by drawing a stroke starting from the scalp. The depth of the starting point is the depth of the intersecting point on the scalp (Figure 4.3(a)). For each subsequent stroke point, if it is still on the scalp, it takes the depth of the scalp point, otherwise its depth is set as the depth of the last point on the scalp. A stream curve is usually long and needs incremental refinement from different viewpoints. To facilitate editing, its *supporting surface* is defined via expanding the stream curve in its neighborhood, and with respect to the viewpoint. We provide two ways of building the supporting surface: the supporting surface is either a degenerate rule surface (Figure 4.3(a)) interpolating the stream curve and the associated viewpoint or a strip (Figure 4.3(b)) composed of lines locally orthogonal to each triangle formed by a stream curve segment and the viewpoint. The user can freely change the viewpoint of a stream curve and/or switch between the two supporting surface modes to rebuild the supporting surface that is convenient for editing.

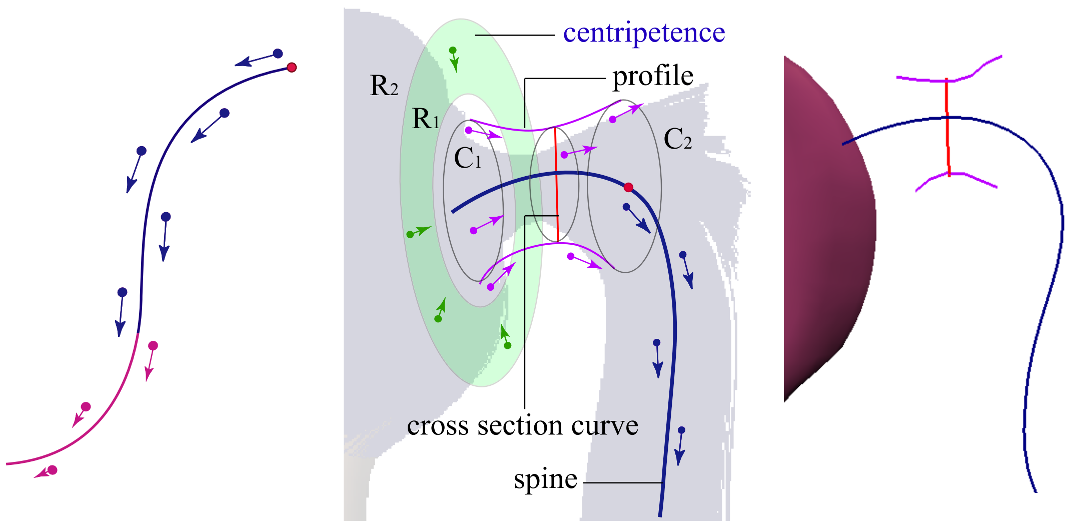


Figure 4.5: **Left**: a stream curve and its boundary constraints. **Middle**: a ponytail primitive and its boundary constraints. **Right**: four style curves of a ponytail primitive.

Once created, the user may edit a stream curve by sketching a *hint stroke* starting on its supporting surface. Depending on the general direction, starting and ending points, a hint stroke is interpreted as three different editing operations (see Figure 4.3(c)-(h)).

Dividing Curve. Many hairstyles have a clear dividing line on the head where hair strands part and flow in opposite directions. A dividing curve is drawn on the scalp to represent such a line (see Figure 4.4). To model the discontinuities in the vector field around the parting line, a local Cartesian coordinate frame $\mathbf{L} = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$ is defined at every unit point $\mathbf{p}$ on the dividing curve (Figure 4.4(a)), where $\mathbf{y}$ is the normal of the scalp at $\mathbf{p}$, and $\mathbf{z}$ is the tangent of the dividing curve at $\mathbf{p}$ rotated by a tilt angle α that can be easily adjusted by the user. Boundary constraints are defined in the local xy planes, on two opposite sides of $\mathbf{y}$ axis. The local shape of the boundary constraints can be easily adjusted via a few controlling parameters. See Figure 4.4 for details.

A dividing curve must be on the scalp and is relatively short, thus can usually be created by drawing one stroke on the scalp (inappropriate ones can be easily discarded). The user can then draw a hint stroke on each side of the dividing curve (in the strip spanned by the yellow lines in Figure 4.2) to adjust the tilting angle of each local plane (see Figure 4.4).

Ponytail. The ponytail primitive consists of four style curves: a spine curve, a cross section curve, and two profile curves (Figure 4.5). The creation and editing of the spine curve is the same as that of a stream curve. The user then changes the viewpoint, rebuilds the supporting surface, and sketches the cross section curve and profile curves on it. A set of cross sections (circles) is then generated along part of the spine curve between the profile curves. Let C_1 and C_2 denote the first (near scalp) and the last cross sections respectively.

Three types of boundary constraints are derived (Figure 4.5). The first type (purple) are defined at grid points near to the cross sections, with their directions along the longitudinal directions of the revolutional shape. The second type (blue) are at grid points near the spine curve. Their direction assignment is the same as for stream curves except that we start near the cross section C_2 (since fewer changes of boundary constraints means more efficient modification of Cholesky factorization). The third type (green), called centripetal constraints, are introduced to make the vector field flow into the first cross section C_1. They point to the center of C_1 and are defined at the region between two concentric circles R_1 and R_2 lying on the plane of C_1. The radii of R_1 and R_2 are proportional to the radius of C_1 (we use 1.3 and 1.9). The magnitude of the last set of constraints is a parameter to control the tightness of the ponytail (we use the default value of 0.6). Essentially, the introduction of the centripetal constraints transfers the complexity of creating a ponytail from the user to the design of the ponytail sketch tool. Without the centripetal constraints, the user would need to draw many more stream curves to guide the hair curves to pass through the cross sections.

4.6 Implementation and Results

System Initialization The bounding volume and resolution of the discrete vector field is fixed during the whole design process. The vector field resolution depends on the desired hairstyle. A low resolution (about 25K variables) is used for smooth hairstyles, and high resolution (about 50K variables) for complex hairstyles such as a ponytail. The initialization of the linear system takes tens of seconds.

Hair Growth. To generate hair curves from the vector field, root points are first uniformly sampled on the scalp, with a small amount of randomness added. Two spherical coordinates $\theta \in [0, 180)$ and $\phi \in [0, 360)$ are used for scalp surface parameterization, and the hair density is controlled via the sampling steps of these two angles. Each hair curve is a sequence of connected line segments. In hair growth, let $\mathbf{p}$ denote the current end point of a hair curve, a new segment $l \cdot \mathbf{v}(\mathbf{p})$ is appended, where

$\mathbf{v}(\mathbf{p})$ is the directional vector linearly interpolated from the vector field, and the scalar l takes a smaller value than the step size d in the vector field to make the growth smooth ($l = 0.5d$ in our implementation). Hair growth is terminated when $|\mathbf{v}(\mathbf{p})|$ is too small (< 0.05 in our implementation) or **p** is out of the bounding volume. The whole process of growing thousands of hair curves is extremely fast (e.g., less than 0.5 seconds for 100K hair curves).

Scalp Penetration Detection and Avoidance. It is not guaranteed that the flow lines in the vector field will not penetrate the scalp, which is undesirable and should be avoided. If necessary, the user can draw enough stream curves (usually about ten) around the scalp so that the flow lines in the vector field will not penetrate the scalp. We use a simple strategy to alleviate such non-essential user interaction. User first designs a hairstyle by drawing style curves without considering the scalp penetration problem. When a satisfactory hair shape is obtained, for each grid point near the scalp with direction pointing inwards[1], its direction is first replaced with its projection on the tangent plane of the nearest scalp point. All the grid points near the scalp are then added as boundary constraints. The magnitude of these directions are set small (we use 0.3) to reduce global influence. This step is done only once in the design process. This strategy is useful for saving user interactions when designing complex hairstyles such as ponytails.

Rendering. We use a free Renderman compliant software Aqsis [124] and the hair rendering algorithm proposed by Kajiya and Kay [125] to render all the results.

Results. Figure 4.6 shows a result created only using stream curves. Figure 4.7 demonstrates a typical hairstyle with a parting line. Figure 4.8 demonstrates a hairstyle with two ponytails. The hairstyle in Figure 4.1 uses eleven stream curves, four dividing curves and one ponytail primitive. All examples are created in a short time. The most complex hairstyle in Figure 4.1 takes five minutes.

4.7 Summary

We present a hairstyle design system [126] supported by an intuitive sketching interface and a fast vector field solver. User can modify the hairstyle with interactive visual

[1]This can be easily detected by computing the angle between the direction and its nearest scalp normal.

feedback. Interesting hairstyles can be easily designed. We plan to design more style primitives.

Since we only use one vector field to depict a hairstyle and each vertex has only one direction, our current method cannot handle hairstyles which require multiple tangent directions at a vertex, e.g., braids. Multiple vector fields coupled with physically guided tool proposed by Choe and Ko [116] may address this limitation.

Due to the smooth interpolation in the vector field, the output hairstyles of our system may lack local variations (e.g., curls) that are present in real human hair. One straightforward solution is to integrate previous hair modeling techniques as a post-processing step to add local details directly on the already-grown hair curves. Since the global shape and the positions of the hair curves are already satisfactory, post processing on such output is easier than designing hairstyles from scratch. Therefore, our system could be used as an independent design tool or a complementary pre-processor for other hair modeling techniques. Another possible solution is to introduce local details by differential coordinates as employed in differential mesh editing [69] by rewriting Equation 4.1 as

$$E(\mathbf{t}_1, \ldots, \mathbf{t}_n) = \sum_{i=1}^{n} ||\Delta(\mathbf{t}_i) - \delta_i||^2 + \omega^2 \sum_{i \in C} ||\mathbf{t}_i - \mathbf{c}_i||^2,$$

where δ_i are the differential coordinates which encode the local details built over the existing example hairstyles.

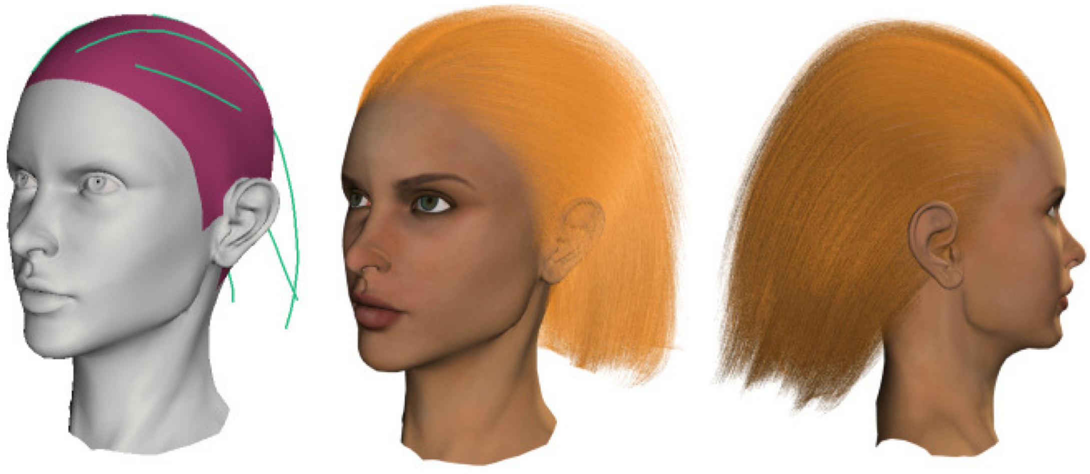


Figure 4.6: A smooth hairstyle created with ten stream curves.

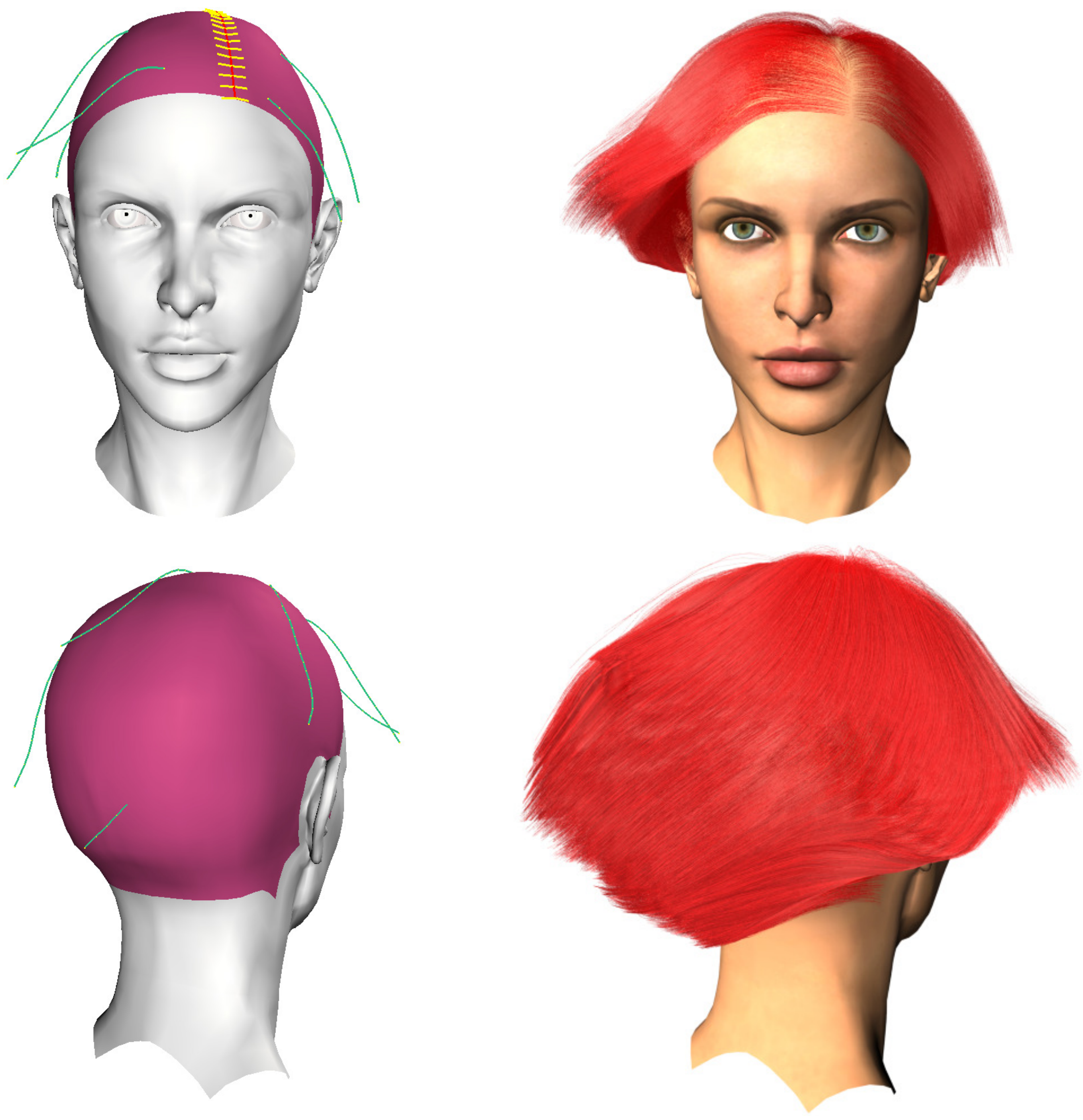


Figure 4.7: A hairstyle with a short parting line, created with seven stream curves and a dividing curve.

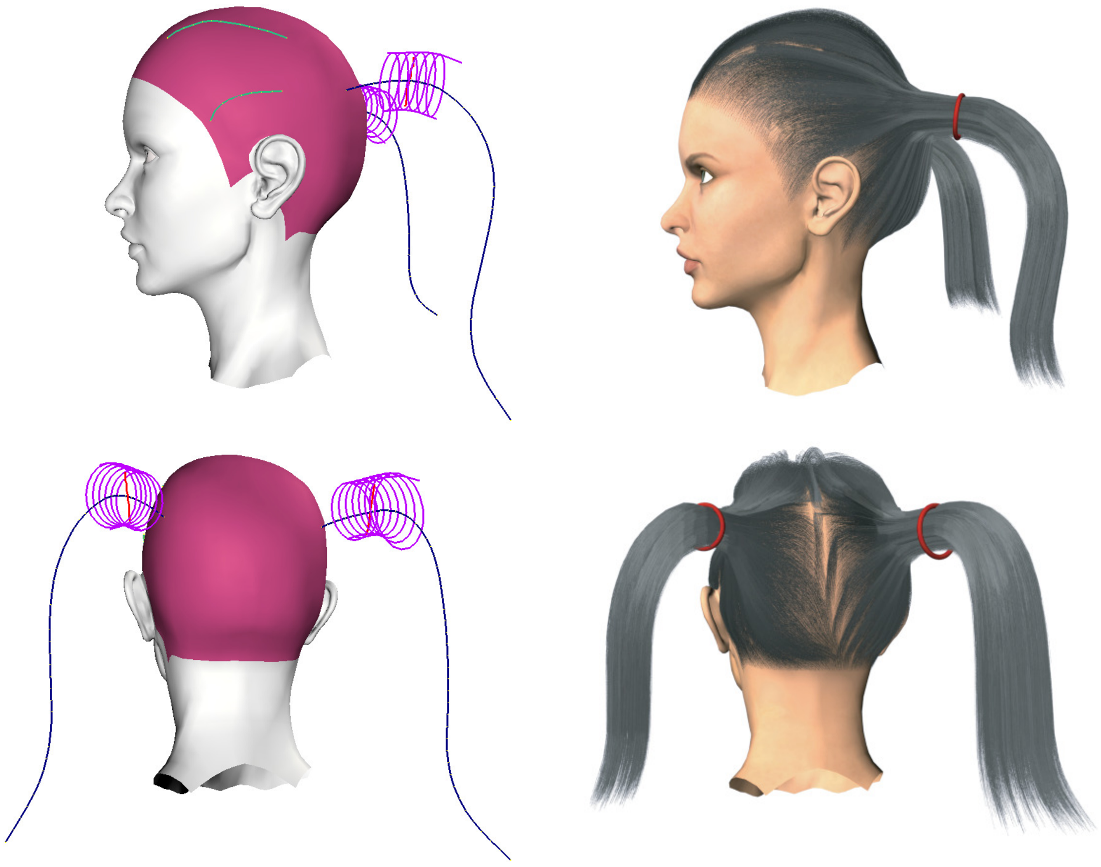


Figure 4.8: A hairstyle with two ponytails, created with four stream curves and two ponytail primitives. This is created in three minutes. See accompanying video for a demo.

CHAPTER 5

CONCLUSION

Modeling complex polygonal meshes and polygonal curves is generally tedious and limited to a small group of professional digital artists. Our ultimate goal is to make 3D shape modeling easy to use even for novices. In this thesis, we achieve this goal by striking a balance between control intuitiveness of tools and output modeling complexity: the modeling process is governed by the pre-designed optimizations and a small set of easy-to-use manipulators by which the user controls the modeling effect. The user is allowed to use more manipulators to obtain more complex modeling results. Our optimization methods are based on systems of the Laplace or Poisson equations subject to the modeling constraints derived from the user-controlled manipulators. The resulting optimizations are easy to implement by using the available highly-optimized sparse linear system solvers and are efficient to solve by pre-computing the most time-consuming components.

We have contributed to three important modeling applications: mesh deformation, mesh merging and hairstyle design. More specifically, we present a new linear differential mesh deformation framework, two intuitive techniques for Poisson mesh merging, and a system for interactive hairstyle design. All these modeling techniques support intuitive user interface and interactive editing. The user is allowed to easily use only a small set of manipulators to interactively design compelling shapes.

In the future, we plan to explore more applications of the idea of controlling modeling effect through a small set of manipulators. Several possible directions are as follows:

- Interactive mesh segmentation. Mesh segmentation has a variety of applications in geometry modeling, such as skeleton extraction and object recognition. Whether the segmentation results are desirable or not is very subjective and application-oriented. Hence automatic mesh segmentation methods [127, 128] are definitely not omnipotent. Instead, we are more interested in interactive mesh segmentation, that is, letting the user to give hints for guidance of desirable segmentation. Ji et al. [102] proposed an intuitive user interface for interactive mesh segmentation which allows the user to roughly draw 2D foreground/background strokes to indicate which regions are of interest. It is possible to use an optimization method similar to the Laplacian system used for hairstyle modeling in Section 4.4 to propagate the guidance information at the user-drawn strokes to the unknown regions. However, how to design appropriate weighting schemes for discretizing the Laplace operator (cf. similarity metrics in other segmentation algorithms), is the most challenging, so this aspect of the problem requires some additional research.

- Intuitive correspondence specification for cross-parameterization of meshes. Cross-parameterization (also known as inter-surface mapping) is very useful for texture blending and shape morphing between meshes. Existing cross-parameterization techniques [129, 130] rely on dozens of pairs of user-specified point correspondences over meshes in order to align important corresponding features. However, in many cases, using points only can be cumbersome to specify semantically meaningful regional features (e.g., legs, arms of human character models). A possible solution is to use stroke-based guidance as adopted in interactive mesh segmentation. With the boundary condition derived from the user-drawn corresponding stroke pairs, it might be possible to apply the harmonic fields [4] which is obtained by solving the Laplacian system, to build a one-to-one correspondence between meshes.

- Sketch-based surface reconstruction from unorganized point clouds. Many of existing highly detailed models are obtained by scanning real-world objects. The raw output of 3D digital scanners is a set of unorganized point clouds, possibly with vertex normals. The problem of reconstructing polygonal meshes from unorganized point clouds is ill posed. The problem becomes even harder to address when the sampling of the point clouds is highly nonuniform. Very recently, Ju et al. [131] and Sharf et al. [132] have proposed to use minimal user guidance to get topologically correct reconstructed surfaces. It would be an interesting topic to employ minimum user intervention to improve the reconstruction of geometric details at uncertain regions (e.g., big missing areas).

REFERENCES

[1] O. K.-C. Au, H. Fu, C.-L. Tai, and D. Cohen-Or, “Handle-aware isolines for scalable shape editing,” *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. to appear, 2007.

[2] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum, “Subspace gradient domain mesh deformation,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1126–1134, 2006.

[3] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel, “Differential coordinates for interactive mesh editing,” in *Proceedings of Shape Modeling International*. IEEE Computer Society Press, 2004, pp. 181–190.

[4] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel, “Harmonic guidance for surface deformation,” *Computer Graphics Forum*, vol. 24, no. 3, pp. 601–609, 2005.

[5] Y. I. H. Parish and P. Müller, “Procedural modeling of cities,” in *SIGGRAPH ’01*. New York, NY, USA: ACM Press, 2001, pp. 301–308.

[6] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, “Procedural modeling of buildings,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 614–623, 2006.

[7] P. E. Debevec, C. J. Taylor, and J. Malik, “Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach,” in *SIGGRAPH ’96*. New York, NY, USA: ACM Press, 1996, pp. 11–20.

[8] ImageModeler, Realviz, 2007.

[9] Y. Wei, E. Ofek, L. Quan, and H.-Y. Shum, “Modeling hair from multiple views,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 816–820, 2005.

[10] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang, “Image-based plant modeling,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 599–604, 2006.

[11] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, “The digital

michelangelo project: 3d scanning of large statues," in *SIGGRAPH '00*, 2000, pp. 131–144.

[12] L. Zhang, N. Snavely, B. Curless, and S. M. Seitz, "Spacetime faces: high resolution capture for modeling and animation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 548–558, 2004.

[13] D. Scanner, Cyberware, 2007.

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

[15] 3ds Max, Autodesk, 2007, http://www.autodesk.com/3dsmax.

[16] LightWave, NewTek, 2007, http://www.newtek.com/lightwave/.

[17] ZBrush, Pixologic, Inc., 2007.

[18] Rhino, McNeel, 2007.

[19] O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel, "Laplacian surface editing," in *Symposium on Geometry Processing*, 2004, pp. 179–188.

[20] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, "Mesh editing with Poisson-based gradient field manipulation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 644–651, 2004.

[21] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, "Large mesh deformation using the volumetric graph Laplacian," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 496–503, 2005.

[22] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or, "A sketch-based interface for detail-preserving mesh editing," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1142–1147, 2005.

[23] H. Fu, O. K.-C. Au, and C.-L. Tai, "Effective derivation of similarity transformations for implicit Laplacian mesh editing," *Computer Graphics Forum*, vol. 26, no. 1, pp. 34–45, 2007.

[24] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, “Laplacian mesh optimization,” in *ACM GRAPHITE*, 2006.

[25] I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. D. Shaw, “Twister: a space-warp operator for the two-handed editing of 3D shapes,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 663–668, 2003.

[26] W. von Funck, H. Theisel, and H.-P. Seidel, “Vector field based shape deformations,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1118–1125, 2006.

[27] ——, “Elastic secondary deformations by vector field integration,” in *Symposium of Geometry Processing*, 2007.

[28] S.-H. Yoon and M.-S. Kim, “Sweep-based freeform deformations,” *Computer Graphics Forum*, vol. 25, no. 3, pp. 487–496, 2006.

[29] T. Igarashi, S. Matsuoka, and H. Tanaka, “Teddy: a sketching interface for 3D freeform design,” in *SIGGRAPH ’99*, 1999, pp. 409–416.

[30] C.-L. Tai, H. Zhang, and J. C.-K. Fong, “Prototype modeling from sketched silhouettes based on convolution surfaces,” *Computer Graphics Forum*, vol. 23, no. 1, pp. 71–83, 2004.

[31] O. A. Karpenko and J. F. Hughes, “Smoothsketch: 3d free-form shapes from complex sketches,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 589–598, 2006.

[32] A. Nealen, T. I. O. Sorkine, and M. Alexa, “FiberMesh: designing freeform surfaces with 3D curves,” *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

[33] T. W. Sederberg and S. R. Parry, “Free-form deformation of solid geometric models,” in *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, 1986, pp. 151–160.

[34] S. Coquillart, “Extended free-form deformation: a sculpturing tool for 3D geometric modeling,” in *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 1990, pp. 187–196.

[35] W. Hsu, J. Hughes, and H. Kaufman, “Direct manipulation of free-form deformations,” *Computer Graphics*, vol. 26, no. 2, pp. 177–184, 1992.

[36] Y. Chang and A. Rockwood, “A generalized de Casteljau approach to 3D free-form deformation,” *Computer Graphics*, vol. 28, no. 4, pp. 257–260, 1994.

[37] R. MacCracken and K. I. Joy, “Free-form deformations with lattices of arbitrary topology,” in *Proceedings of ACM SIGGRAPH 96*, 1996, pp. 181–188.

[38] T. Ju, S. Schaefer, and J. Warren, “Mean value coordinates for closed triangular meshes,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 561–566, 2005.

[39] K. Singh and E. Fiume, “Wires: a geometric deformation technique,” in *SIGGRAPH ’98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 405–414.

[40] R. Sumner, J. Schmid, and M. Pauly, “Embedded deformation for shape manipulation,” *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

[41] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping,” in *Proceedings on Graphics interface ’88*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 1988, pp. 26–33.

[42] X. C. Wang and C. Phillips, “Multi-weight enveloping: least-squares approximation techniques for skin animation,” in *ACM SIGGRAPH/Eurographics Symposium on Computer animation*, 2002, pp. 129–138.

[43] P. Kry, D. James, and D. Pai, “Eigenskin: real time large deformation character skinning in hardware,” in *Symposium on Computer Animation*, 2002.

[44] A. Mohr and M. Gleicher, “Building efficient, accurate character skins from examples,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 562–568, 2003.

[45] J. P. Lewis, M. Cordner, and N. Fong, “Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation,” in *Proceedings of ACM SIGGRAPH 2000*, 2000, pp. 165–172.

[46] F. Cordier and N. Magnenat-Thalmann, “A data-driven approach for real-time clothes simulation,” *Computer Graphics Forum*, pp. 173–183, 2005.

[47] I. Baran and J. Popović, “Automatic rigging and animation of 3D characters,” *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

[48] S. Schaefer and C. Yuksel, “Example-based skeleton extraction,” in *Symposium on Geometry Processing*, 2007.

[49] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman, “Context-aware skeletal shape deformation,” *Computer Graphics Forum*, vol. 26, no. 3, p. To appear, 2007.

[50] M. Alexa, “Linear combination of transformations,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 380–387, 2002.

[51] L. Kavan and J. Zara, “Spherical blend skinning: a real-time deformation of articulated models,” in *I3D*, April 2005, pp. 9–16.

[52] L. Kavan, S. Collins, J. Zara, and C. O’Sullivan, “Skinning with dual quaternions,” in *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM Press, April/May 2007.

[53] E. Lindholm, M. J. Kligard, and H. Moreton, “A user-programmable vertex engine,” in *SIGGRAPH 2001*, 2001, pp. 149–158.

[54] D. L. James and C. D. Twigg, “Skinning mesh animations,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 399–407, 2005.

[55] L. Kavan, R. McDonnell, S. Dobbyn, J. Zara, and C. O’Sullivan, “Skinning arbitrary deformations,” in *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM Press, April/May 2007.

[56] K. G. Der, R. W. Sumner, and J. Popović, “Inverse kinematics for reduced deformable models,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1174–1179, 2006.

[57] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, “Mesh-based inverse kinematics,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 488–495, 2005.

[58] D. Forsey and R. Bartels, “Hierarchical b-spline refinement,” in *Proceedings of ACM SIGGRAPH 88*, 1988, pp. 205–212.

[59] D. Zorin, P. Schröder, and W. Sweldens, “Interactive multiresolution mesh editing,” in *ACM SIGGRAPH*, 1997, pp. 259–268.

[60] I. Guskov, W. Sweldens, and P. Schröder, “Multiresolution signal processing for meshes,” in *Proceedings of ACM SIGGRAPH 99*, 1999, pp. 325–334.

[61] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, “Interactive multiresolution modeling on arbitrary meshes,” in *Proceedings of ACM SIGGRAPH 98*, 1998, pp. 105–114.

[62] M. Garland, “Multiresolution modeling: survey & future opportunities,” Eurographics: State of the Art Report, Tech. Rep., 1999.

[63] M. Botsch and L. Kobbelt, “Multiresolution surface representation based on displacement volumes,” in *Eurographics*, 2003, pp. 483–491.

[64] D. Xu, W. Chen, H. Zhang, and H. Bao, “Multi-level differential surface representation based on local transformations,” *The Visual Computer*, 2006.

[65] M. Botsch, R. Sumner, M. Pauly, and M. Gross, “Deformation transfer for detail-preserving surface editing,” in *Vision, Modeling & Visualization*, 2006.

[66] R. W. Sumner and J. Popović, “Deformation transfer for triangle meshes,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 399–405, 2004.

[67] M. Botsch and L. Kobbelt, “Real-time shape editing using radial basis functions,” in *Eurographics*, 2005, pp. 611–621.

[68] S. Schaefer, T. McPhail, and J. Warren, “Image deformation using moving least squares,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 533–540, 2006.

[69] O. Sorkine, “Differential representations for mesh processing,” *Computer Graphics Forum*, vol. 25, no. 4, pp. 789–807, 2006.

[70] M. Botsch and O. Sorkine, “On linear variational surface deformation methods,” *IEEE Transactions on Visualization and Computer Graphics*, 2007, to appear.

[71] A. Sheffer and V. Krayevoy, “Pyramid coordinates for morphing and deformation,” in *3D Data Processing, Visualization, and Transmission*, 2004, pp. 68–75.

[72] V. Kraevoy and A. Sheffer, “Mean-value geometry encoding,” *International Journal of Shape Modeling*, vol. 12, no. 1, 2006.

[73] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, “Linear rotation-invariant coordinates for meshes,” *ACM Trans. Graph.*, vol. 24, pp. 479–487, 2005.

[74] L. Shi, Y. Yu, N. Bell, and W.-W. Feng, “A fast multigrid algorithm for mesh deformation,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1108–1117, 2006.

[75] D. Xu, H. Zhang, and H. Bao, “Non-uniform differential mesh deformation,” in *Lecture Notes in Computer Science*, 2006.

[76] T. Popa, D. Julius, and A. Sheffer, “Material-aware mesh deformations,” in *SMI ’06*, 2006.

[77] S. Toledo, “TAUCS: a library of sparse linear solvers, version 2.2,” 2003, tel-Aviv University, Available online at http://www.tau.ac.il/ stoledo/taucs/.

[78] T. Davis, “Cholmod: a sparse supernodal cholesky factorization package., version 1.5,” 2007, university of Florida, Available online at http://www.cise.ufl.edu/research/sparse/cholmod/.

[79] M. Botsch, D. Bommes, and L. Kobbelt, “Efficient linear system solvers for mesh processing,” *Lecture Notes in Computer Science*, vol. 3604, pp. 62–83, 2005.

[80] O. K.-C. Au, C.-L. Tai, H. Fu, and L. Liu, “Mesh editing with curvature flow laplacian operator,” Hong Kong University of Science Technology, Computer Science Technical Report, HKUST-CS05-10, Tech. Rep., 2005.

[81] O. K.-C. Au, C.-L. Tai, L. Liu, and H. Fu, “Dual Laplacian editing for meshes,” *IEEE Transaction on Visualization and Computer Graphics*, vol. 12, no. 3, pp. 386–395, 2006.

[82] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt, “PriMo: coupled prisms for intuitive surface modeling,” in *Symposium on Geometry Processing*, 2006, pp. 11–20.

[83] M. Botsch, M. Pauly, M. Wicke, and M. Gross, “Adaptive space deformations based on rigid cells,” *Computer Graphics Forum (Proc. Eurographics 2007)*, vol. 26, no. 3, p. To appear, 2007.

[84] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo, “Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics,” *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

[85] S. Yoshizawa, A. Belyaev, and H.-P. Seidel, “Skeleton-based variational mesh deformations,” *Computer Graphics Forum*, vol. 26, no. 3, p. To appear, 2007.

[86] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, “Implicit fairing of irregular meshes using diffusion and curvature flow,” in *Proceedings of ACM SIGGRAPH 99*, 1999, pp. 317–324.

[87] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, “Discrete differential geometry operators for triangulated 2-manifolds,” in *VisMath*, 2002, pp. 35–57.

[88] H. Fu and C.-L. Tai, “Mesh editing with affine-invariant laplacian coordinates,” Hong Kong University of Science and Technology, Computer Science Technical Report, HKUST-CS05-01, Tech. Rep., 2005.

[89] B. Allen, B. Curless, and Z. Popović;, “The space of human body shapes: reconstruction and parameterization from range scans,” *ACM Transaction on Graphics*, vol. 22, no. 3, pp. 587–594, 2003.

[90] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, “Meshless deformations based on shape matching,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 471–478, 2005.

[91] H. Biermann, D. Kristjansson, and D. Zorin, “Approximate Boolean operations on free-form solids,” in *Proceedings of SIGGRAPH 2001*, 2001, pp. 185–194.

[92] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr, “Level set surface editing operators,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 330–338, 2002.

[93] P. Pérez, M. Gangnet, and A. Blake, “Poisson image editing,” *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 313–318, 2003.

[94] J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum, “Drag-and-drop pasting,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 631–637, 2006.

[95] T. Kanai, H. Suzuki, J. Mitani, and F. Kimura, “Interactive mesh fusion based on local 3D metamorphosis,” in *Graphics interface ’99*, 1999, pp. 148–156.

[96] H. Biermann, I. Martin, F. Bernardini, and D. Zorin, “Cut-and-paste editing of multiresolution surfaces,” *ACM Transaction on Graphics*, vol. 21, no. 3, pp. 312–321, 2002.

[97] H. Fu, C.-L. Tai, and H. Zhang, “Topology-free cut-and-paste editing over meshes,” in *Geometric Modeling and Processing 2004*, April 2004, pp. 173–182.

[98] X. Huang, H. Fu, O. K.-C. Au, and C.-L. Tai, “Optimal boundaries for Poisson mesh merging,” in *ACM Solid and Physical Modeling Symposium 2007*, 2007, pp. 35–40.

[99] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or, “SnapPaste: an interactive technique for easy mesh composition,” *The Visual Computer*, vol. 22, no. 9, pp. 835–844, 2006.

[100] T. Hassner, L. Zelnik-Manor, G. Leifman, and R. Basri, “Minimal-cut model composition,” in *SMI*, 2005, pp. 72–81.

[101] G. Taubin, “A signal processing approach to fair surface design,” in *Proceedings of ACM SIGGRAPH 95*, 1995, pp. 351–358.

[102] Z. Ji, L. Liu, Z. Chen, and G. Wang, “Easy mesh cutting,” *Computer Graphics Forum*, vol. 25, no. 3, pp. 283–291, 2006.

[103] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, “Least squares conformal maps for automatic texture atlas generation,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 362–371, 2002.

[104] G. Barequet and M. Sharir, “Filling gaps in the boundary of a polyhedron,” *Computer Aided Geometric Design*, vol. 12, no. 2, pp. 207–229, 1995.

[105] B. K. P. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *Journal of the Optical Society of America*, vol. 4, no. 4, pp. 629–642, 1987.

[106] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[107] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[108] F. Bertails, B. Audoly, M.-P. Cani, B. Querleux, F. Leroy, and J.-L. Lévêque, “Super-helices for predicting the dynamics of natural hair,” *ACM Transaction on Graphics*, vol. 26, no. 3, pp. 1180–1187, 2006.

[109] J. T. Moon and S. R. Marschner, “Simulating multiple scattering in hair using a photon mapping approach,” *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1067–1074, 2006.

[110] K. Ward, F. Bertails, T.-Y. Kim, S. Marschner, M.-P. Cani, and M. Lin, “A survey on hair modeling: styling, simulation, and rendering,” *IEEE Transaction on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 213–234, 2007.

[111] A. V. Gelder and J. Wilhelms, “An interactive fur modeling technique,” in *Proc. Graphics Interface*, 1997, pp. 181–188.

[112] T.-Y. Kim and U. Neumann, “Interactive multiresolution hair modeling and editing,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 620–629, 2002.

[113] S. Malik, “A sketching interface for modeling and editing hairstyles,” in *EG Workshop on Sketch Based Interfaces and Modeling*, 2005, pp. 185–194.

[114] S. Hadap and N. Magnenat-Thalmann, “Interactive hair styler based on fluid flow,” in *EG Workshop on Computer Animation and Simulation*, 2000, pp. 87–99.

[115] Y. Yu, “Modeling realistic virtual hairstyles,” in *Pacific Graphics*, 2001, pp. 295–304.

[116] B. Choe and H.-S. Ko, "A statistical wisp model and pseudophysical approaches for interactive hairstyle generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 2, pp. 160–170, 2005.

[117] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes, "SKETCH: An interface for sketching 3D scenes," in *Proceedings of ACM SIGGRAPH*, 1996, pp. 163–170.

[118] X. Mao, H. Kato, A. Imamiya, and K. Anjyo, "Sketch interface based expressive hairstyle modelling and rendering," in *CGI '04*, 2004, pp. 608–611.

[119] J. Wither, F. Bertails, and M.-P. Cani, "Realistic hair from a sketch," in *Shape Modeling International*, June 2007.

[120] K. Takayama, T. Igarashi, R. Haraguchi, and K. Nakazawa, "A sketch-based interface for modeling myocardial fiber orientation," in *Smart Graphics 2007*, 2007.

[121] M. Fisher, P. Schröder, M. Desbrun, and H. Hoppe, "Design of tangent vector fields," *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

[122] S. Paris, H. M. B. o, and F. X. Sillion, "Capture of hair geometry from multiple images," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 712–719, 2004.

[123] T. A. Davis and W. W. Hager, "Modifying a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 3, pp. 606–627, 1999.

[124] Aqsis, 2007, http://www.aqsis.org/.

[125] J. Kajiya and T. Kay, "Rendering fur with three dimensional textures," in *Proc. ACM SIGGRAPH '89*, 1989, pp. 271–280.

[126] H. Fu, Y. Wei, C.-L. Tai, and L. Quan, "Sketching hairstyles," in *Eurographics Workshop on Sketched-based Interfaces and Modeling*, 2007.

[127] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 954–961, 2003.

[128] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal, "Mesh segmentation - a comparative study," in *SMI*, 2006, p. 7.

[129] V. Kraevoy and A. Sheffer, “Cross-parameterization and compatible remeshing of 3D models,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 861–869, 2004.

[130] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, “Inter-surface mapping,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 870–877, 2004.

[131] T. Ju, Q.-Y. Zhou, and S.-M. Hu, “Editing the topology of 3d models by sketching,” *ACM Transaction on Graphics*, vol. 26, no. 3, 2007.

[132] A. Sharf, T. Lewiner, G. Shklarski, S. Toledo, and D. Cohen-Or, “Interactive topology-aware surface reconstruction,” *ACM Transaction on Graphics*, vol. 26, no. 3, p. To appear, 2007.

APPENDIX A

PUBLICATIONS

Journal

1 Oscar Kin-Chung Au, **Hongbo Fu**, Chiew-Lan Tai, Daniel Cohen-Or. Handle-aware isolines for scalable shape editing. *ACM Transaction on Graphics (TOG) special issue: Proceedings of ACM SIGGRAPH 2007*. 26(3). In press.

2 Kun Xu, Yuntao Jia, **Hongbo Fu**, Shimin Hu, Chiew-Lan Tai. Spherical piecewise constant functions for all-frequency precomputed radiance transfer, *IEEE Transaction on Visualization and Computer Graphics (TVCG)*, Under second round review.

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

4 Oscar Kin-Chung Au, Chiew-Lan Tai, Ligang Liu, **Hongbo Fu**. Dual Laplacian editing for meshes, *IEEE Transaction on Visualization and Computer Graphics (TVCG)*. 12(3): 386-395, MAY/JUNE 2006.

Conference

1 **Hongbo Fu**, Yichen Wei, Chiew-Lan Tai, Long Quan. Sketching hairstyles, *Eurographics Workshop on Sketch-based Interfaces and Modeling 2007*, To appear.

2 Xiaohuang Huang, **Hongbo Fu**, Oscar Kin-Chung Au, Chiew-Lan Tai. Optimal boundaries for Poisson mesh merging, *ACM Solid and Physical Modeling Symposium 2007 (SPM 2007)*, To appear.

3 Xiangye Xiao, Qiong Luo, Dan Hong, **Hongbo Fu**, Slicing*-tree based web page transformation for small displays. *ACM Fourteenth Conference on Information and Knowledge Management (CIKM 2005)*, Bremen, Germany, 2005.

4 **Hongbo Fu**, Chiew-Lan Tai, Oscar Kin-Chung Au. Morphing with Laplacian coordinates and spatial-temporal texture, In *Proceedings of Pacific Graphics 2005 (PG 2005)*, pages 100-102.

5 Oscar Kin-Chung Au, Chiew-Lan Tai, **Hongbo Fu**, Ligang Liu. Mesh editing with curvature flow Laplacian, *Symposium on Geometry Processing 2005 (SGP 2005)*, Vienna, 4-6 July, 2005 (poster).

6 **Hongbo Fu**, Chiew-Lan Tai, Hongxin Zhang. Topology-free cut-and-paste editing over meshes, *Geometric Modeling and Processing 2004 (GMP 2004)*, pages 173-182, Beijing, April, 2004.

Working Paper

1 Chunxia Xiao, **Hongbo Fu**, Chiew-Lan Tai. Multiscale aggregation for efficient shape extraction, 2007.

2 Shu Liu, Chunxia Xiao, Chengchun Lin, **Hongbo Fu**, Chengfang Song, Qunsheng Peng. Smart belief propagation for video completion, 2007.

Technical Report

1 Oscar Kin-Chung Au, Chiew-Lan Tai, **Hongbo Fu**, Ligang Liu. Mesh editing with curvature flow Laplacian operator, *Technical report, HKUST-CS05-10*, July 2005.

2 **Hongbo Fu**, Chiew-Lan Tai. Mesh editing with affine-invariant Laplacian coordinates, *Technical report, HKUST-CS05-01*, January 2005.

3 **Hongbo Fu**, A survey of editing techniques on surface models and point-based models, *PhD Qualifying Examination*, Department of Computer Science and Engineering, HKUST, 19 December 2003.