2D-Centric Interfaces and Algorithms for 3D Modeling

by

Yotam Gingold

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2009

Denis Zorin

Acknowledgments

First of all, I must thank my advisor Denis Zorin under whose guidance I have learned how to conduct rigorous computer science research. Denis's knowledge is vast, and his opinions are measured. His curriculum extended beyond computer science and mathematics into life, public policy, and common sense.

During my time at NYU, I have been lucky to have had charming and intelligent labmates, friends, and collaborators: Ilya Rosenberg, Adrian Secord, Chris Wu, Jason Reisman, Elif Tosun, Denis Kovacs, Ian Spiro, Casey Muller, Philip Davidson, Jeff Han, Eitan Grinspun, Harper Langston, Robb Bifano, Ayse Erkan, Raia Hadsell, Matt Grimes, and Lyuba Chumakova. I collaborated with Phil and Jeff on the contents of Chapter 1. I collaborated with Takeo Igarashi on the contents of Chapter 3; that research was initiated during a stay at the Japan Science and Technology Agency (JST) ERATO User Interfaces for Design Project.

I am indebted to Mike Khoury for a long interview that helped us understand how artists view texturing (Chapter 1) and to Zack Shukan for discussions and artwork found in Chapter 2.

I wouldn't be who I am, nor would I have made it to and through graduate school, if it weren't for my family: my parents Monique and Harry Gingold and my siblings Chaim and Naomi. My father provided me with my earliest math education and served as my earliest positive example for the path I have taken. The phrase, "The pen is smarter than I am," (which he attributed to Euler) is no less true in computer science. My beloved aunt Rina, z"l, provided me with

wisdom at a crucial point in my career.

Finally, I would like to thank Shiro Oishi and David Williams, whose motivating words I hear in their own voices.

Abstract

The creation of 3D models is a fundamental task in computer graphics. The task is required by professional artists working on movies, television, and games, and desired by casual users who wish to make their own models for use in virtual worlds or as a hobby.

In this thesis, we consider approaches to creating and editing 3D models that minimize the user's thinking in 3D. In particular, our approaches do not require the user to manipulate 3D positions in space or mentally invert complex 3D-to-2D mappings. We present interfaces and algorithms for the creation of 3D surfaces, for texturing, and for adding small-to-medium scale geometric detail.

First, we present a novel approach for texture placement and editing based on direct manipulation of textures on the surface. Compared to conventional tools for surface texturing, our system combines UV-coordinate specification and texture editing into one seamless process, reducing the need for careful initial design of parameterization and providing a natural interface for working with textures directly on 3D surfaces.

Second, we present a system for free-form surface modeling that allows a user to modify a shape by changing its rendered, shaded image using stroke-based drawing tools. A new shape, whose rendered image closely approximates user input, is computed using an efficient and stable surface optimization procedure. We demonstrate how several types of free-form surface edits which may be difficult to cast in terms of standard deformation approaches can be easily performed using

our system.

Third, we present a single-view 2D interface for 3D modeling based on the idea of placing 2D primitives and annotations on an existing, pre-made sketch or image. Our interface frees users to create 2D sketches from arbitrary angles using their preferred tool—including pencil and paper—which they then "describe" using our tool to create a 3D model. Our primitives are manipulated with persistent, dynamic handles, and our annotations take the form of markings commonly used in geometry textbooks.

Table of Contents

TABLE OF CONTENTS

List of Figures

LIST OF FIGURES

LIST OF FIGURES

Introduction

A large portion of computer graphics is concerned with the display, representation, and motion of 3D geometry. This thesis is concerned with authoring free-form 3D models, a process akin to virtual sculpting. We introduce tools for authoring 3D models which are designed to simplify the process, leverage users' 2D drawing experience, and be accessible to novices.

The 3D modeling task is required by professional artists working on movies, television, and games, and desired by casual users who wish to make their own models for use in virtual worlds or as a hobby.

The task, in general, is open-ended. A model can be refined with fine geometric features and detailed coloration almost indefinitely. For applications such as animation, models may be further annotated with information about how the model should change as it moves. In this thesis, we introduce interfaces for applying surface colors (texturing), for adding small-to-medium scale geometric features, and for creating initial, undetailed geometry.

0.1 Motivation

Tools for creating and editing free-form geometry take several approaches. Users may describe models mathematically. This approach may be suitable for computer-aided design of models which must be machined, but it is unfeasible for complex, free-form models, where equations describing the surface are unknown.

Users may create sculptures in the physical world, and then scan or trace them into the computer. Scanning a model requires expensive hardware and there may be quality issues; tracing a model is tedious.

Finally, users may learn to use software tools for creating 3D geometry. Software tools are free from artificial constraints present in the physical world—users can zoom in and out and undo and redo actions—and are free to re-imagine the modeling process. Some still take the approach of virtual sculpting. While these tools may be familiar for artists with a sculpting background, they do not leverage the skills of artists with a background in life drawing, where 2D shading is used to convey 3D information.

Other tools employ a process wholly different from sculpting in the physical world. In limited domains, such as human bodies, template-based approaches which parameterize models with a small number of degrees-of-freedom, and allow users to generate models by manipulating sliders ([Sof08]).

A variety of approaches induce smooth models from sparse input. One example of sparse information is points or curves with topological connectivity. Another example is the placement of deformed shape primitives (such as spheres, cylinders, and cubes). These systems are often interactive; in sketch-based approaches users sketch curves in 2D and the system infers depth information automatically. Maya [Aut09] is an example of the primitive shape-placement and deformation approach. Examples of the point and curve placement approach are spline and subdivision surfaces, curve networks [WW92, SWZ04, NISA07], and sketch-based modelers such as Teddy [IMT99]. Users typically interact with these tools using a 2D input

device; 3D input devices are expensive and more awkward than sculpting a physical material when held freely in space.

To apply color to a 3D model (or other surface attributes), traditionally, the model's surface is parameterized in two dimensions, and then unwrapped onto the image plane; artists paint into a 2D image, called the texture, which is mapped back onto the surface according to the parameterization. Artists must invert the parameterization in their heads. An alternative approach is virtual 3D painting. (3D painting approaches may rely on parameterizations internally, but this is transparent to the user.) In either approach, photographs or existing textures cannot be used in a straighforward manner.

2D drawing remains much easier than 3D modeling, for professionals and amateurs alike. Professionals continue to create 2D drawings before beginning the 3D modeling process, and desire to use them in that process [TS08, TBSR04, EHBE97]. (In fact, one workflow employed by professional 3D modelers is placing axis-aligned sketches or photographs in the 3D scene for reference.) At the same time, amateurs are familiar with drawing in 2D but unfamiliar with modeling in 3D. Part of this familiarity and ease may be that tools for working with 2D images are more commonly available—these include pencil and paper as well as computer mice and drawing tablets. But 2D drawing is inherently a one-dimensionally less complex process.

An additional reason for preferring 2D drawing to 3D modeling is that there are virtually no requirements imposed by the interface on user input. Any mark made with a pencil is valid; the only requirement is that the drawing is understandable

by humans, which is trivially satisfied since the designer is himself a human. As humans, we have a powerful recognition system for interpreting drawings; if computers were similarly capable, the design process could simply be drawing concept sketches. However, we are many years away from having such computers. Instead, we strive to augment or complement the pencil-and-paper workflow.

0.2 Thesis Organization

We introduce three modeling systems, based on pushing more of the 3D editing operations into 2D. These systems are designed to make it possible for novices to create 3D models, and to make modeling easier for experts, as well. Our first system (Chapter 1) is concerned with texturing—for applying colors or other surface properties—and allows users to stretch 2D images over 3D models. Users need not mentally invert complex 3D-to-2D mappings, and can even use casual photographs. The system employs direct manipulation and exploits multi-touch hardware.

In our second system (Chapter 2), we turn to the problem of geometry modification. This system allows users to add geometric features to 3D models by drawing over them as if with a pencil. Users can leverage their existing 2D shading skills. The class of edits easy to perform using our tool are difficult to achieve with 3D sculpting tools. Our interface presents drawing tools similar to those found in 2D painting programs. Our algorithm can be viewed as solving a special case of the shape-from-shading problem.

Our third system (Chapter 3) introduces a solution for the creation of 3D models from guide images. This system allows users to sketch in 2D and then

place primitives and annotations over the image to generate a 3D model. Because it is easier to draw in 2D than to create 3D models, artists typically draw concept artwork before beginning the 3D modeling process. Our system allows artists to create 3D models from concept artwork, whereas concept artwork is not directly usable in existing modeling tools. Our interface can even be used by amateurs who cannot draw well in 2D, but would nevertheless like to create 3D models from found drawings.

1
A Direct Texture Placement and Editing Interface

1.1 Introduction

The goal of 3D texturing is to enhance object appearance by using images to modify surface material attributes, for example, colors, reflection coefficients, or normals.

In existing computer animation and computer-aided design systems, a typical pipeline for adding textures to surface meshes includes several stages, as shown in Figure 1.2, top row. These stages often correspond to separate tools or views.

First, the user selects a region of mesh to work with. The user then assigns texture coordinates to the mesh, using a combination of basic projection operations, more advanced automatic algorithms for mapping meshes to the plane, and manual adjustment of vertex positions. Once the texture coordinates are established, the polygons corresponding to this part of the mesh can be drawn in the image plane and used as a guide for constructing the texture map by adding, distorting, and blending preexisting images, as well as painting directly into the texture. The tools for establishing parameterization, editing the texture, and rendering a complete 3D model are often logically separate, and the user needs to switch between these three views multiple times during the process. The texture placement and

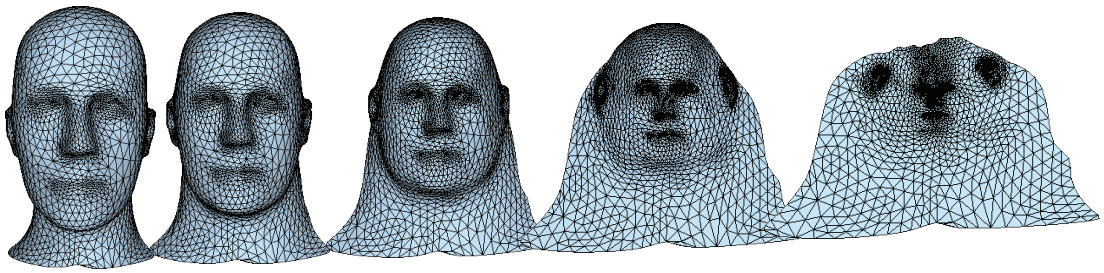


Figure 1.1: A 3D model being flattened into the plane according to its parameterization.

painting process is indirect, as it happens not directly on the three-dimensional geometry but in texture space, where the shapes and sizes of surface triangles are often distorted (Figure 1.1). The user must mentally invert the map from the 3D mesh to texture space, adjusting for this distortion. Photographs and other images need to be warped in texture space to achieve good results in 3D; defining a suitable warping requires considerable experience and careful layout of texture coordinates.

We describe a system which eliminates the separation between these stages and enables direct manipulation of textures on surfaces (Figure 1.2, bottom row). All texture editing can be done directly on the 3D model, similar to 3D painting systems.

In our interaction metaphor, the texture can be thought of as a malleable thin sheet of material that clings to the surface and can be arbitrarily moved and deformed by constraining and moving points and areas of the texture. This directly corresponds to the intuitive idea behind texture mapping: the surface is decorated

with textures. Traditional approaches often require the user to do the opposite: determine how to flatten parts of the surface to the plane of the texture. Our system allows multiple textures to be placed separately and then blended together by specifying the location and size of blend regions directly on the surface.

Our approach to texturing works particularly well with multi-touch and pressure-sensitive hardware. This type of input device allows the user to manipulate multiple points on the texture simultaneously, greatly enhancing his ability to specify complex deformations directly, rather than in multiple stages. Sensitivity to pressure provides a natural framework for adjusting the size of texture areas affected by deformations and blending multiple textures.

We consulted with artists and found that the most cumbersome aspects of the texturing process are establishing a satisfactory parameterization and juggling the three views (3D model, parameterization, and texture editor). We interviewed, at length, an artist from Maxis/Electronic Arts. He characterized the problem of painting into a distorted, flattened region of mesh as surmountable, given training. However, he noted that he had no easy way to map or remap existing textures and photographs onto different models. He also described the iterative process of establishing a satisfactory parameterization: artists make minor adjustments to the parameterization in tandem with texture editing. Finally, he stated that he would prefer as much of the texturing process as possible to take place in the 3D view.

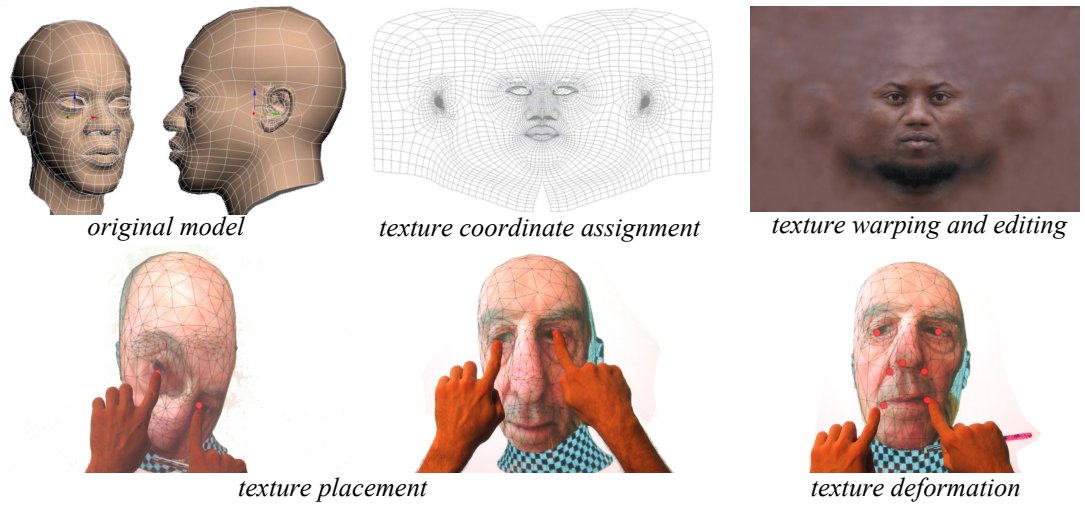


Figure 1.2: (top) The stages of traditional texturing, where the artist must switch contexts between 3D viewing and 2D texture painting and alignment [images courtesy of Jiri Adamec]. (bottom) Our proposed texturing allows the user to manipulate texture parameterization directly on the model.

1.2 Related Work

Our work builds on research in mesh parametrization, painting on 3D meshes, and multi-touch interfaces.

Parameterization.

Mesh parametrization (i.e. computing a map or a collection of maps from a mesh to the plane) is fundamental to a broad range of applications, and we cannot do justice to the spectrum of work in the area. We focus primarily on the work most closely related to ours—specifically, on techniques which emphasize high performance and the ability to add constraints.

Most common techniques for parameterization of disk-like mesh areas rely on minimizing some measure of mesh distortion. A measure of distortion based on elasticity was introduced in Maillot et al. [MYV93]; related stretching techniques are described in Piponi and Borshukov [PB00], Sander et al. [SSGH01], and Yoshizawa et al. [YBS04]. Elasticity-based formulations are most commonly nonlinear, although Yoshizawa et al. [YBS05] describe an approach which yields good results while solving only two linear systems. A different class of methods is based on solving a single linear system; these approaches are particularly suitable for interactive applications. A commonly-used general form with variable weights was described in Floater [Flo97]; specific geometrically-motivated choices of weights were proposed in Desbrun et al. [DMA02] and Lévy et al. [LPRM02]. An alternative approach (directly minimizing angle distortion) was proposed in Sheffer and de Sturler [SdS01]; while the original method requires the relatively expensive so-

lution of a nonlinear problem, a more efficient and robust version is described in Sheffer et al. [SLMB05]. Zayer et al. [ZRS05] introduce a boundary-free parameterization method that achieves good results by solving several linear systems. Parametrization based on tracing geodesics is described in Lee et al. [LTD05].

A number of papers introduce various types of constraints into the parameterization. Lévy [Lév01] uses penalty terms in the energy to approximate constraints. We propose a different energy that matches constraints exactly, which is important for usability, while Lévy weights smoothness of parameterization versus exactness of constraint-matching. We ensure smoothness with a higher-order energy and gradient-matching. Most importantly, our approach to incremental solution updating enables interactive texture manipulation. Desbrun and Alliez [DMA02] suggest using the Lagrange multiplier formulation; Kraevoy et al. [KSG03] point out that not all constraints for a given mesh can be satisfied by a one-to-one mapping and describe an algorithm for adding points to the triangulation so the constraints can be met. A number of papers consider the more difficult problem of consistently parameterizing several surfaces [PSS01, KS04, SAPH04]; while these techniques can also be specialized to constrained parameterization, they are relatively expensive.

An important problem, not addressed in this paper, is partitioning the surface into patches that can be mapped to the plane, or finding the best way to cut the surface such that a single patch is formed. Many authors have developed different automatic approaches to this problem [PFH00, GGH02, KLS03, She03, SWGH03, JWYG04, ZMT05]. Yamauchi et al. [YLHS05] describe a complete automated

texturing pipeline, largely following the traditional stages we have described above.

It is possible to avoid the parameterization problem entirely by storing texture information in a volume data structure (e.g. DeBry et al. [DGPR02]).

Finally, we note that our approach to texturing can be regarded as using image warping on a mesh [BN92], and that the incremental matrix inversion formula we use was applied in James and Pai [JP99] to interactive simulation.

3D painting.

As introduced in Hanrahan and Haeberli [HH90], painting directly on the surface is a natural way to create textures from scratch. In Agrawala et al. [ABL95], a tracker is used to "paint" on a real object; the paint appeared in the texture of the corresponding scanned computer model. Carr and Hart [CH04] show how texture resolution can be increased when painting on a surface. Igarashi and Cosgrove [IC01] discuss an adaptive technique for parametrization while 3D painting. In contrast, our emphasis is on the application and modification of preexisting textures, rather than creating textures from scratch.

Multi-touch interfaces.

There is a large body of relevant work exploring two-handed input. Most of this work precedes the recent advent of practical multi-touch input devices and uses multiple mice/pucks or multimode Wacom tablets. Guiard [Gui87] presents a theoretical framework for asymmetrically assigned roles in two-handed interaction, including the natural partitioning of a graphical task into view manipulation with the non-dominant hand and simultaneous editing or object manipulation with the

dominant hand. 3D camera control and object manipulation with two mice was explored in Balakrishnan and Kurtenbach [BK99] and Zeleznik et al. [ZFS97], and with 6-DOF trackers in Hinckley et al. [HPGK94].

Graphical modeling operations using two hands in a symmetric manner have also been explored. This approach has been applied to simultaneously pan, zoom, or rotate the view context in a 2D map setting in Kurtenbach et al. [KFBB97], to alignment tasks in Balakrishnan and Hinckley [BH00], and for sophisticated 3D mesh modeling operations, as in Twister [LKG+03] which uses a pair of 6-DOF trackers.

While our texturing system can be used with a traditional input device and display, direct-display multi-touch interface devices are a natural match for its capabilities. These devices [DL01, Rek02, Wil04, Han05] provide the advantage of direct graphical interaction, as well as benefits similar to those found in the literature on bi-manual input using multiple-mice: Wu and Balakrishnan [WB03] demonstrate two-finger rotation and scaling operations, while Igarashi et al. [IMH05] describe a system for interactively applying complex free-form deformations of 2D models in the plane, where an arbitrary number of fingers each provide an additional 2-DOF control.

In this work, we use a multi-touch sensing technique recently introduced by Han [Han05], based on frustrated total internal reflection. Our system is implemented in a 36" drafting table form-factor, with a sensing resolution of approximately 20 ppi at 50 Hz. The system uses a compliant surface layer to provide reliable pressure sensitivity, which enables the use of passive styluses. As a result, users are free to

use fingers or styluses as they see fit, for precision as well as comfort.

1.3 User Interface

Next, we describe the interaction operations supported by our system. Our system supports the following modes: texture placement, texture deformation, gluing, and blending.

Texture placement mode.

The first step in the texturing process is to establish an initial mapping to the mesh which we adjust and refine. First, a texture is mapped to the mesh using an automatic distortion-minimizing parameterization with free boundaries, such that the texture patch is in the interior of the mesh. The user can then use a two-point texture placement scheme: two points are placed on the mesh to define the texture's translation, rotation, and scale, as shown in Figure 1.3. Using the multi-touch interface, users can adjust the points simultaneously with two fingers, continuously adjusting texture position on the surface.

This is the only operation in our system that requires multi-touch and can't be performed with a mouse. When using a mouse, the user must separately translate, rotate, and scale the texture.

Texture deformation mode.

Once the initial location of the texture is defined, we fix the texture boundary and deform the texture on the surface in order to bring features of the texture into alignment with features on the mesh. This is achieved by defining a set of *live*

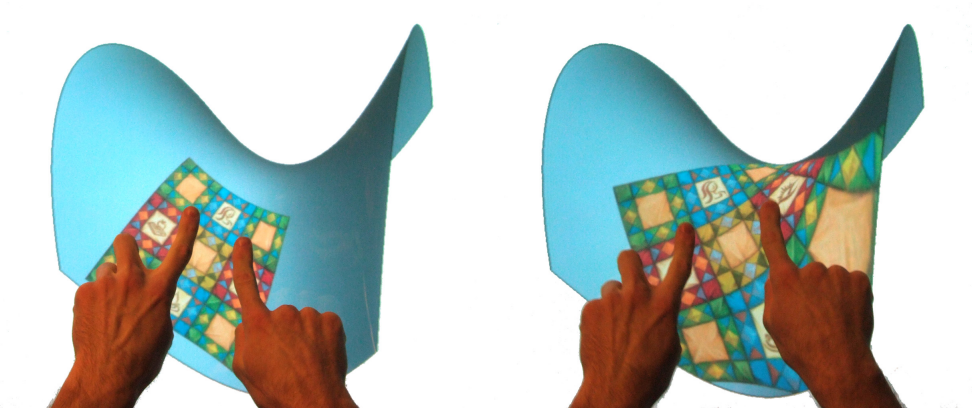


Figure 1.3: Two fingers completely determine the texture's translation, rotation, and scale.

constraints, which are points on the texture moved around by the user. The rest of the texture follows the live constraints; the influence of the constraints gradually decreases with distance. The influence of live constraints is further restricted by constraints defined in the glue tool, described below.

Once the user stops manipulating a live constraint, it converts into a *pushpin*, or dead, constraint. A pushpin constraint can be removed later; texture coordinates then elastically bounce back to their unconstrained positions. We find this behavior to be useful for undoing changes.

Alternatively, the user can choose to convert the deformation fixed by pushpins to *plastic*. As a result of this transformation, the current state of the texture does not change; however, pushpin constraints are no longer necessary to maintain the current mapping and are removed. All subsequent deformations are composed with the current deformation. This is useful for gradual, precise adjustment of

textures without introducing excessive numbers of pushpin constraints, which may prevent smooth texture deformations. Figures 1.4 and 1.5 compare two types of deformations.

Deformable regions: gluing mode and radius widget.

We provide several tools for defining regions on the mesh.

We provide a *glue* brush to restrict deformations to particular areas of the mesh. In this mode, hidden pushpin constraints are applied to the vertices of triangles traversed by the brush. Glued areas can be used to separate regions of the mesh, acting as 'walls' across which changes cannot propagate. By completely encircling an area of the mesh, we limit the effect of edits inside to the interior region and, likewise, protect it from changes made outside. This is convenient for protecting local adjustments.

Alternatively, the user can specify a radius for the deformable area with a simple radius widget. Any constraint point automatically limits its influence approximately to the specified radius. (This is implemented by gluing the ring of triangles lying under a screen space circle of given radius.) Distant such constraint points have non-overlapping regions of influence that divide the mesh into multiple independent circles. These regions can be adjusted simultaneously, which is helpful in situations where multiple edits are taking place at once.

Combining multiple textures, blending.

Our texture adjustment naturally extends to multiple textures. For example, we may take multiple source photographs of an object to capture surface details from

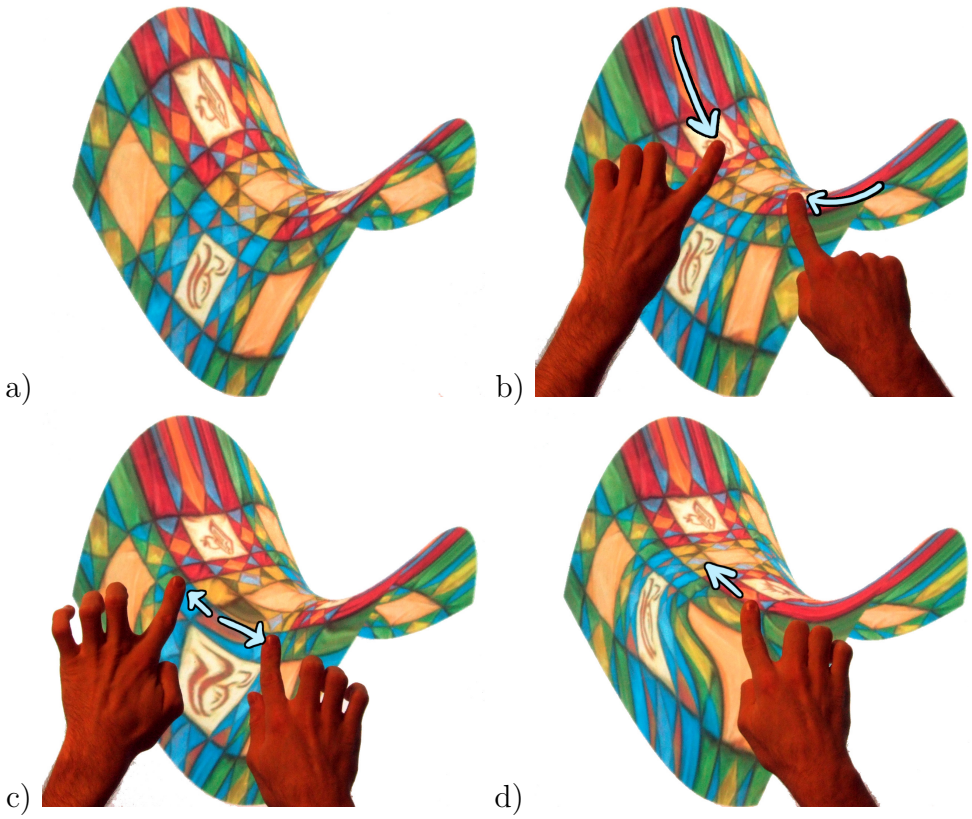


Figure 1.4: Plastic deformations: a) Undeformed parameterization. b) An initial plastic deformation by squeezing. c) A subsequent plastic stretching of the texture. d) A final deformation demonstrates the parameterization's plastic memory.

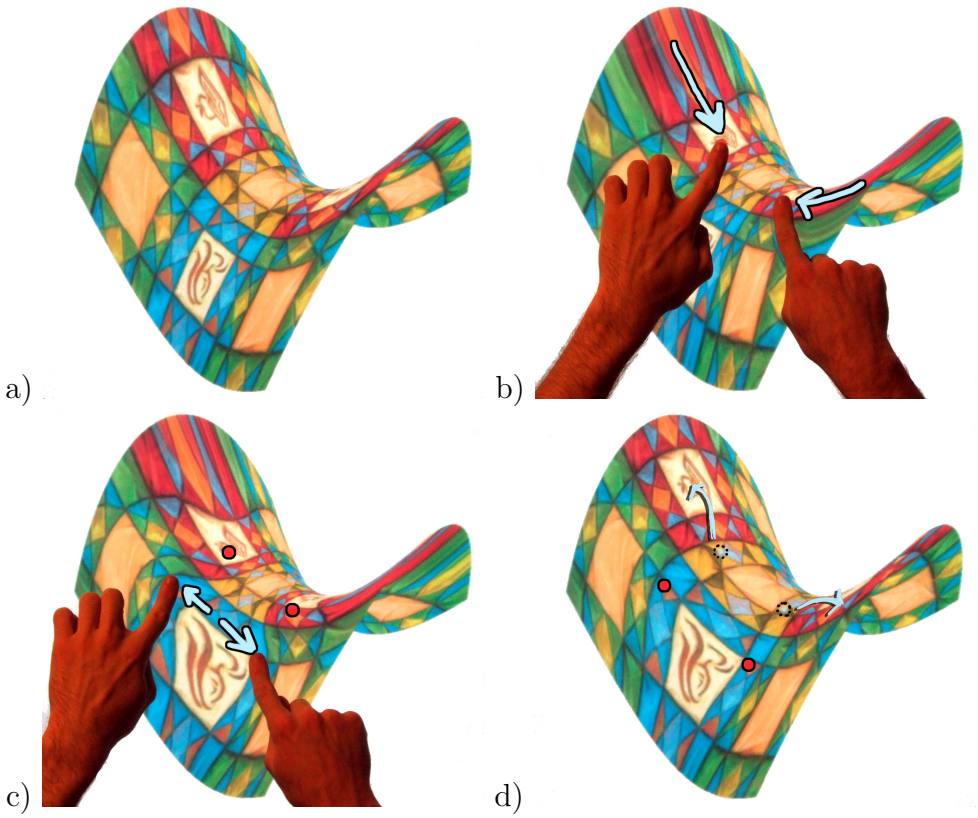


Figure 1.5: Elastic deformations: a) Undeformed parameterization. b) First, two live constraints squeeze the texture. c) Second, two additional live constraints stretch the texture. d) Finally, the pushpin constraints added in b) have been removed and the texture underneath bounces back to its unconstrained configuration.

Figure 1.6: (left) Small and (right) large regions of influence.

different directions. The user can adjust them separately to determine their relative alignment directly on the mesh geometry. This is convenient for accurately merging a number of source textures where the original alignment is not known.

Once relative positioning has been defined, we use an airbrush interface to blend out unnecessary or overlapping portions of the source textures, as illustrated in Figure 1.7. If the user wishes to have continuous, fine brush radius control with his non-dominant hand, the user can place two fingers on the radius widget to "pick it up."

1.4 Algorithms

Next we describe essential algorithms used to implement the user interface described in the previous section.

Figure 1.7: (left) Small and (right) large areas of texture can be blended with the transparency airbrush.

Basic parameterization.

At the heart of our system is an efficient algorithm for mapping a part of the mesh to the plane. It is similar to a widely used class of algorithms that solve a positive-definite linear system of equations.

However, we have observed that commonly used Floater-type algorithms, while performing well with either fixed or natural boundary constraints, do not perform as well in the setting with constrained interior points, especially for large distortion. Large deformations in this case result in mesh fold-overs in the texture domain, causing the same area of texture to be mapped onto multiple surface areas. Intuitively, this can be explained as follows. Qualitatively, the behavior of each coordinate obtained using a parametrization of this type is known to be similar to the behavior of the solution of the Laplace equation, which, in turn, is

similar to the behavior of a soap film. For a point constraint in the interior of the domain, the solution will have a sharp point, thus likely to create a fold for small displacements (Figure 1.8).

While there are relatively complex approaches allowing for the complete elimination of the problem [KSG03], we have found that switching the energy type to the linearized analog of *bending* energy significantly improves the behavior. One can think of this energy as the energy of flattening a membrane which resists bending to the plane (ignoring stretching). Mathematically, the discretization of this energy can be expressed as

$$E = \sum_i \frac{1}{8\, area_i} \left(\sum_{j \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{t}_i - \mathbf{t}_j) \right)^2$$

where $\mathbf{t}_i = [u_i, v_i]$ is the 2D position of vertex i in the texture domain; i varies over all vertices and $N(i)$ is the set of vertices adjacent to i; $area_i$ is computed as described in Meyer et al. [MDSB03]; and the angles α_{ij} and β_{ij} are those shown in Figure 1.9. This energy is based on the discretization for mean curvature presented by Meyer et al..

We observe that E is a quadratic function of the texture coordinates. To minimize this energy, we solve the system of linear equations with matrix A obtained by differentiating E twice with respect to all nonfixed points. For boundaries, we use two types of constraints: simple fixed constraints (in deformation mode) as well as natural constraints for free boundaries (see Desbrun et al. [DMA02]) when the texture is initially placed on the surface.

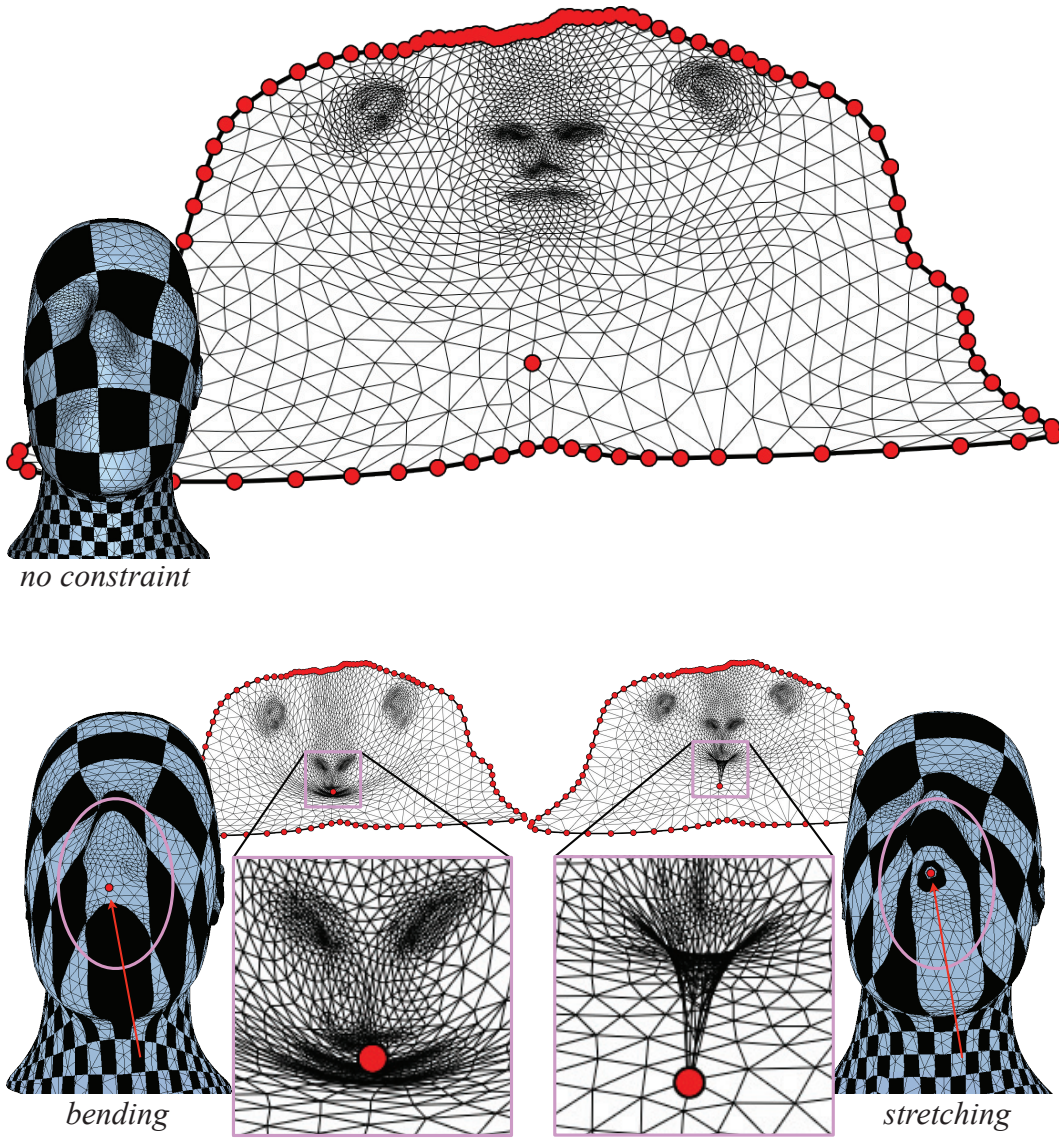


Figure 1.8: Without an interior point constraint, parameterizations obtained by solving the Laplace equation—minimizing stretching energy—and by minimizing bending energy are similar (top). With an interior point constraint, minimizing stretching energy produces fold-overs (lower-right), while minimizing bending energy produces a smooth parameterization with no fold-overs (lower-left).

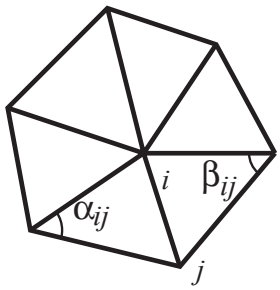


Figure 1.9: Angles referenced in the energy equation.

Constraints.

As described above, a constraint is a point correspondence between an arbitrary point in the texture chosen by the user and a point on the mesh. The point on the mesh need not be a vertex, so we express this constraint as a linear combination of the texture coordinates of the vertices of the triangle in the mesh containing the constrained point:

$$\alpha_1 \mathbf{t}_1 + \alpha_2 \mathbf{t}_2 + \alpha_3 \mathbf{t}_3 = \mathbf{t}^{fixed}$$

where $\alpha_1 + \alpha_2 + \alpha_3 = 1$ are the barycentric coordinates of the constrained point and $\mathbf{t}_i$, $i = 1, 2, 3$, are the texture coordinates of the three vertices of the triangle. In a more concise form we can write these constraints as two equations (one each for u and v) of the form $c^T t = d$, where t is the vector of texture coordinates for all points, c^T is the vector of coefficients with only three nonzero entries, and d is the fixed value.

To ensure the constraints are specified precisely, we use the standard Lagrange multiplier approach, i.e. we add extra variables and equations to our system. In-

stead of minimizing the original quadratic functional E, for each constraint equation $c_i^T t = d_i$ we add a term $\lambda_i(c_i^T t - d_i)$. This leads to a linear system with the matrix

$$A^{ext} = \begin{pmatrix} A & C^T \\ C & 0 \end{pmatrix}$$

where C is a matrix composed out of vectors c_i. If we have M constraints, then the size of C is $M \times N$, where N is the number of vertices. In particular, this means the matrix changes whenever constraints are added and removed, and the linear system has to be solved from scratch. However, we minimize the amount of needed computation by using incremental inverse updates based on the Sherman-Woodbury-Morrison formula, discussed in more detail below.

Incremental matrix updates.

All deformations resulting from live constraints are elastic, i.e. are computed using fixed cotangent coefficients in the energy E. We use the fact that only a relatively small part of the system matrix A^{ext} changes to incrementally compute the inverse whenever the constraints change. Here we briefly summarize the procedure, described in more detail in Appendix A. We rely on the following facts. The crucial observation is that the matrix A^{ext} can be written as

$$\begin{aligned} A^{ext} &= \begin{pmatrix} A & 0 \\ 0 & I_{M\times M} \end{pmatrix} \\ &- \begin{pmatrix} 0 & -C^T \\ I_{M\times M} & \frac{1}{2} I_{M\times M} \end{pmatrix} \begin{pmatrix} -C & \frac{1}{2} I_{M\times M} \\ 0 & I_{M\times M} \end{pmatrix} \\ &= A^0 - UV \end{aligned}$$

where A^0 does not depend on constraints, U and V have dimensions $(N+M)\times 2M$ and $2M \times (N+M)$, respectively, and $I_{M\times M}$ is the identity matrix of size $M \times M$.

The classic Sherman-Morrison-Woodbury formula [Hag89] makes it possible to solve the inverse system with matrix A^{ext} with M live constraints at the cost of solving $2M$ systems with matrix A^0 and different right-hand sides. This yields a substantial increase in performance if M is not very large and the matrix A^0 can be prefactorized. That is, the cost of solving $2M$ systems with a prefactorized matrix is substantially lower than the cost of assembling and solving a single system from scratch. We have observed this to be the case if we use an efficient direct solver and assume that M is no more than 10 (the maximum number of live constraints when using the multi-touch interface with two hands) and typically less.

On a 2.8 GHz P4, the seconds per update of a 2138 vertex mesh is, for 1 through 8 live constraints: .022, .026, .030, .033, .038, .043, .048, .056. The speedup over re-solving the system is between 7 and 9 times.

Plastic deformation.

In plastic deformation mode, once a deformation is completed (in the case of the mouse-based interface, on the button-up event; for the multi-touch interface, when the last finger leaves the table), the entries of matrix A are updated using areas and cotangent weights computed from the texture coordinates. This mimics the plastic deformation of materials such as clay: the undeformed state is tracking the deformation, so when constraints are released the material does not spring back to the original position but stays in the deformed state.

Texture blending implementation.

To implement our transparency airbrush tool, we start with a point C in screen space and pick the corresponding point on the mesh. We construct the matrix converting the picked triangle's texture coordinates to screen (pixel) coordinates. We use the inverse of this mapping to find a rough bounding box of the airbrush in texture coordinates. We then iterate over every texel in this rectangular bounds and, using our texture-to-screen mapping, project the texture coordinate onto the screen and compute its new α value based on its screen space distance from C (the place the user clicked/touched). We use the following α update formula

$$\alpha \leftarrow \alpha - flow\left(1 - 1/(2-s)^2\right)$$

where s is the normalized distance from C to the texel such that $s = 1$ when the distance equals the airbrush radius and $flow$ is the rate of airbrush "spray."

1.5 Results

To demonstrate our system in action, we texture map an existing mesh geometry using casual photographs of a subject (Figure 1.10). We show a simplified workflow to quickly and easily produce a parameterized texture from snapshots taken from a family photo album. By aligning the different photographs directly on the mesh geometry, we avoid awkward operations in 2D space in which the user must map source images to the flattened representation of texture space.

Our workflow is illustrated in Figure 1.11. We load the mesh geometry, along with the first photograph from our set. Our first photograph is a portrait view,

so we adjust the mesh orientation to roughly correspond to that viewing angle. In the two-point affine mapping stage, we align the two eyes to determine boundary conditions (a). From there, we use elastic deformation operations to move different features, such as the mouth, nose, and eyebrows, into alignment with the mesh geometry (b).

In (c), we move to a view of the left side and load in the second photograph. From this view, we can establish a base parameterization using the eye and ear. We move to a partial frontal view (d) to further refine the alignment of features around the lips and eye. We have now brought the separate textures into alignment for their appropriate regions. To eliminate the overlap between the two, we switch to texture blending mode (e), erasing areas of the textures which contain off-angle features or background imagery. Finally, we load the third image, taken from the right side, align it, and blend overlapping regions (f).

We have now obtained (g), a relatively uniform texture mapping on the model from three separate photographs. The relative mappings of the mesh to the separate texture UV spaces of the photographs are shown in Figure 1.12. Some artifacts exist from lighting difference in the original snapshots; these could be adjusted in a secondary image processing iteration. The major task of aligning feature data from the three images to the mesh has been accomplished straightforwardly with our direct-manipulation interface.

Figure 1.10: Geometric model and source photographs.

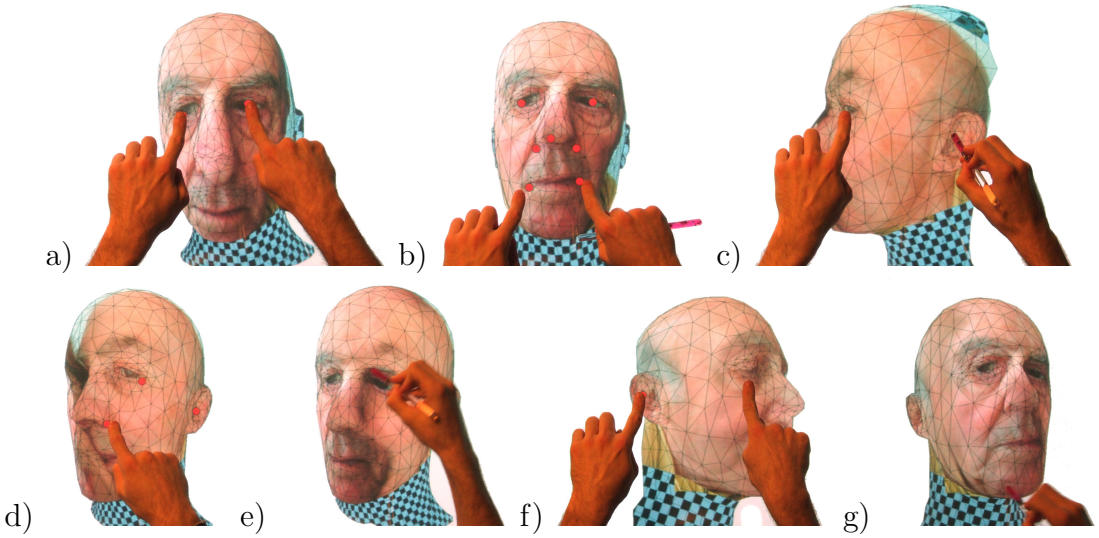


Figure 1.11: A sequence of steps showing the texturing of a model using our system. See the Results section for a description.

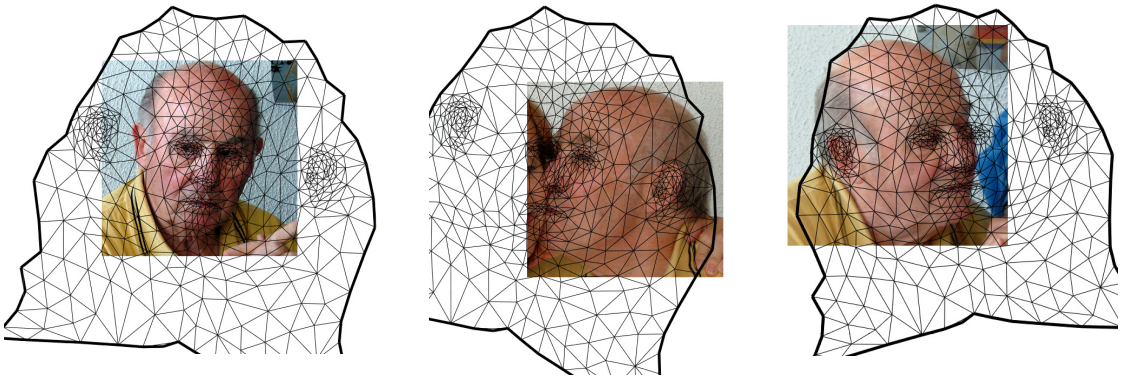


Figure 1.12: Texture maps generated by the system for the model shown in Figure 1.11.

1.6 Conclusions

Our experiments suggest that the interface for texturing described in this paper makes it possible to create textured models faster and requires less skill from the user than conventional techniques.

Artists typically use professional image editing software to edit textures. A limitation of our approach is that the large variety of tools already available in such systems cannot be leveraged directly. Thus, one needs to have a far more extensive collection of image-editing tools in the application itself. Our blending tool is just a first step in this direction.

In the current implementation, we use multitexturing to combine several textures on a model. For many applications this can be regarded as a limitation; it is often desirable to combine all color textures into a single texture for each

part of the mesh. To do this, our interface needs to be combined with global parametrization tools. Texture resolution also may require adjustment, and the approach proposed in Carr and Hart [CH04] is likely to be a useful addition to the system. Increasing the robustness of the parametrization, e.g. by refining the mesh when necessary as in Kraevoy et al. [KSG03], is another important direction for improving the back-end of our system.

2
Shading-Based Surface Editing

2.1 Introduction

Three-dimensional models are often created and manipulated using two-dimensional interfaces. User actions are typically translated into the three-dimensional motion of spline or subdivision surface control points, or into the three-dimensional motion of points on the surface, with the rest of the surface deforming variationally. The relationship between user actions and changes in appearance is indirect: the effect on appearance may be hard to predict, and conversely, it may be difficult to decide which deformation has to be applied to achieve a particular visual effect; e.g., make the outline smoother, remove an unwanted shadow in a view of a model, or reshape a highlight.

In this paper, we describe a sketch-based modeling technique based on changing shaded images of three-dimensional models *directly*, using free-form strokes for two-dimensional image editing. The shape is automatically adjusted to match desired changes of appearance by minimizing a quadratic functional with tangent and positional constraints deduced from user image modifications. This approach complements many other sketch-based modeling techniques. For example, an overall shape can be designed using a system similar to Teddy or FiberMesh [IMT99, NISA07] and further refined using a combination of our system, displacement editing, and silhouette editing [NSACO05].

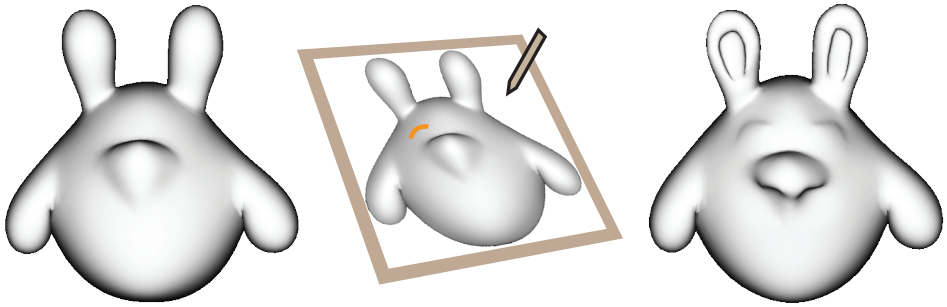


Figure 2.1: In our system, users edit 3D models by drawing 2D shading strokes.

Our choices of algorithms and user interface elements are guided by a general principle: *if a user makes a small change in surface appearance, the resulting shape change should be small*. For most types of modeling interfaces, this continuity of modifications is nearly automatic. For shading-based modeling, because a local modification of shading may require a global shape modification, enforcing this principle is inherently difficult. This can be seen for simple stroke examples in Figure 2.7. Furthermore, the task of inferring shape changes from image changes is closely related to the problem of recovering the shape from a single image. This is a classic problem in computer vision, which in a standard formulation (Lambertian surface, orthographic projection, directional light) is known to be ill-posed.

Compared to recovering a shape from shading or normal information, in our setting we have the advantage of access to the unmodified shape and the ability to control which surface changes are allowed to happen. At the same time, we face additional difficulties, most importantly:

- some types of small shading modifications lead to large and unintuitive model changes (see Section 2.3);

- one needs to preserve existing surface detail during editing;
- it is often necessary to keep the surface exactly unchanged far from the modified area and the transition between the modified area and the rest of the surface smooth;
- surface updates should happen at interactive rates.

Our technique to solve these problems has two complementary parts. First, we design user-interface tools (primarily stroke-based) which retain the intuitive feel of two-dimensional drawing and painting brushes. These tools ensure that image modifications that may lead to unexpected and discontinuous surface changes are not possible (Section 2.3). Second, the restricted class of image modifications permitted by the stroke-based interface allows us to update the surface by solving a quadratic optimization problem, without assuming small deformations or image changes. In contrast, typical shape-from-shading techniques solve a much more general problem, but require solving more complex nonlinear equations.

2.2 Related Work

We build on a broad range of work in the areas of variational surface modeling, sketch-based modeling, and shape-from-shading reconstruction (SfS). Our work is most closely related to [vO96], [BCCD04], [KGB05], and [WTBS07, NWT07].

[vO96] is, as far as we know, the first paper to introduce the idea of shading-based modeling. This paper describes a system for editing height fields sampled

on a regular grid. The user modifies the gradients of the height field directly. [RGB+03] presents a technique for mesh modification and repair based on a variational shape-from-shading algorithm. [BCCD04] uses equal height region propagation to reconstruct height fields from a set of initial equal height contours specified by the user and hand-drawn shading information. [WTBS07] introduces a paradigm for modeling shapes by transferring normal information from a a reference shape (shape palette) to the modeled height field. [NWT07] describes a method for recovering a surface from dense or sparse normal information using radial basis functions in a variational context, without restricting the resulting surface to be a height field, also found to be essential in our context.

In contrast to previous work, we do not use SfS or gradient recovery methods directly. Rather, we focus on modification tools for existing surfaces that yield predictable and controllable surface changes. Most of our operations are based on simple strokes, painted by the user, considerably narrowing the scope of the problem and allowing it to be posed as a quadratic optimization problem. A related idea of painting strokes (silhouettes and suggestive contours [DFRS03]) appeared in [NSACO05].

The techniques we use for surface updates extend Laplacian and gradient-based surface editing [SCOL+04, YZX+04] (see a recent comprehensive survey [BS08]), which can be thought of as data-driven variational modeling [CG91, MS92, WW92, BK04]. Following other sketch-based modeling work (e.g., [IMT99, CCP+04, LF04, KDS06, KH06, NISA07]), we use free-form sketching as a user interface for surface shape changes. Silhouette-based techniques [NSACO05, ZNA07] are complemen-

tary to ours: our system allows for creating silhouettes, which can then be edited using these techniques.

Using modifiable strokes also resembles curve-based modeling tools [SF98, SWZ04, NISA07].

SfS is a well-studied problem in computer vision (the state of the art is surveyed in detail in [Pra04]). In its simplest setting, the scene is assumed to have a single Lambertian, directional light source and an orthographic camera. A variety of techniques for SfS were proposed, including variational (such as in [RGB+03] for modeling) and various types of front propagation methods (e.g. in [BCCD04]). Variational techniques typically use a smoothness penalty term, which can be thought of as a special case of differential coordinate surface deformation with the reference surface being flat.

Some recent methods use user-specified normals or gradients as additional information to solve the SfS problem [ZMQS05], or they infer the surface from sparse normal information [ZDPSS01].

2.3 Shading Changes to Shape Changes

An ideal shading-based modeling system would allow the user to make arbitrary changes to the rendered image of an object, and the resulting modified surface would appear visually indistinguishable from the user-modified image, *while guaranteeing the stability of surface changes and satisfying boundary constraints.* (We use stability in the sense that small changes produce small effects.)

Several fundamental aspects of the relationship between the shaded image and

the corresponding shape make such an ideal system impossible. Rather than attempting to match an arbitrary change of the surface image exactly, our system restricts the types of changes that can be applied, and attempts to approximate the target image modification as closely as possible, without causing unstable changes. We briefly consider several aspects of converting a change in shading into a shape change to motivate our design choices.

Shading changes along curves. Our interface is stroke-based; a typical case for which we need to solve the shape recovery problem is darkening or brightening an image along a curve of uniform thickness, while keeping it unchanged elsewhere. Consider a vertical stroke across a flat square patch, with the light and view directions coinciding (Figure 2.2). In this case, there are three areas of constant shading. We assume the solution to be continuous and to have well-defined normals in each area. If the left boundary is fixed, the solution can be recovered uniquely for any darkening stroke, up to a choice of one of two slopes for each area (the *slope ambiguity*, which we discuss below). We observe that the surface modification amounts to rotating the normals in the area under the stroke about the stroke direction. An additional complication arises when two sides of the patch are fixed: the target image can no longer be matched exactly. However, the solution obtained by minimizing the L_2 norm of the error in shading is also obtained by rotating the normals under the stroke about the stroke direction.

The cases of darkening and brightening strokes are asymmetrical. For certain combinations of light direction and degree of brightening, there is no continuous solution. The reason for this is that brightening to values close to maximal essentially

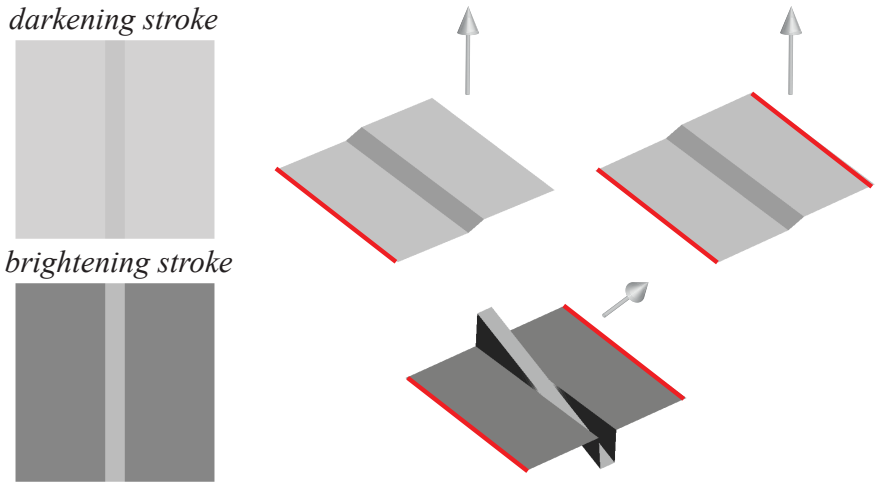


Figure 2.2: A vertical stroke across a flat square patch. Red lines are fixed edges and the arrow represents the coinciding light and view directions. Above: a darkening stroke; a continuous solution for one side fixed; an approximate solution for two sides fixed. Below: a brightening stroke. For this light direction and stroke intensity, all solutions are discontinuous (with either one or two sides fixed).

prescribes the normal everywhere on the stroke; it has to point towards the light source. Furthermore, this discontinuous solution is not stable; for sufficiently long strokes, a small shading change (brightening) can produce a large shape change. In our model square patch example, continuity requires that normals are rotated about the stroke direction.

In our system, we use this observation to maintain stability: normals can rotate only in the planes perpendicular to the stroke (Section 2.5.2).

Figure 2.3: Unstable change of intensity: a stroke, shown in yellow, applies very small (less than 1%) darkening to a highlight, an imperceptible change. To effect this change, a part of the exactly recovered surface has to flip.

Instability near highlights. In some situations, a small change in shading may require a large change in the surface shape (Figure 2.3), contradicting the interface stability principle we have adopted. A common situation of this type is related to *highlight removal*. If the viewer and light positions coincide, the highlight points are also the extremal points of the distance from the surface to the image plane, requiring a large change to the surface to remove. Clearly, for a closed surface there is always a point closest to the light; we conclude that generally, *a highlight cannot be erased* by a smooth surface deformation, so decreasing highlight intensity even by a small amount requires a large change in the surface shape. To prevent this, we terminate strokes which attempt to erase highlights before the stroke reaches the highlight (Section 2.5.3).

Slope ambiguity. It is well known that images can be ambiguous: concave and convex objects can have exactly the same image (Figure 2.4a). A closely related

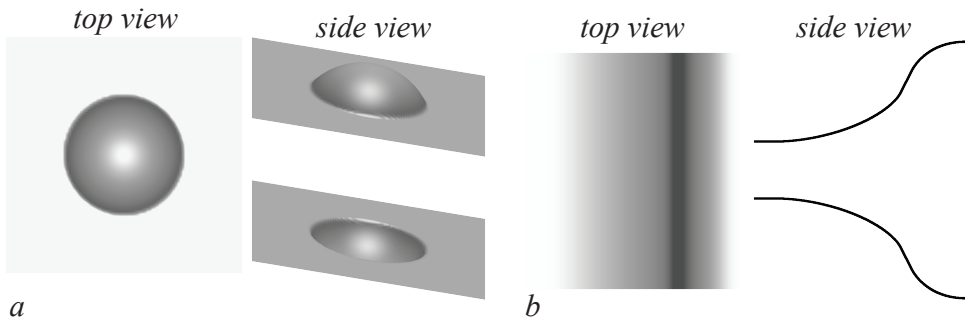


Figure 2.4: (a) Convex-concave ambiguity; (b) slope ambiguity

type of ambiguity, particularly relevant to stroke-based editing, is slope direction ambiguity (Figure 2.4b). In our system, strokes modify an existing surface, so we can resolve the ambiguity by choosing the slope which changes the surface the least.

2.4 Overview of the System

The input to our system is an arbitrary manifold mesh, possibly with boundary. The type of interaction we describe is most suitable for relatively smooth meshes; otherwise, shading is not likely to provide easily understandable information concerning surface shape. The user arbitrarily positions the mesh, chooses its material properties, and positions the light source. Only one light source can be used.

Most of the interaction is done using several types of brushes: a shading modification brush, highlight motion brush, and silhouette brush. These brushes are demonstrated in Figure 2.5. The user can choose a brush's width, opacity, smooth-

ness, and other attributes. An applied stroke can be modified after application (i.e., its attributes can be changed).

The shading modification brush is used to change shading, primarily away from highlights. A darkening shading stroke which crosses a point highlight is terminated, but a darkening stroke can cross a highlight line or highlight area. The user has explicit control over the surface tilt ambiguity: by default, the direction is chosen to minimize the change in slope under the stroke, but can be flipped by pressing a button after the stroke is drawn. Silhouette strokes add silhouette lines to the surface. The width of the stroke in this case controls the size of the created fold. The highlight motion tool is used to reposition highlights (including pushing them to merge with other highlights). A combination of highlight repositioning and shading modification can be used to change a highlight's shape.

Last but not least, it is possible to fix a region of interest (ROI) on the surface to ensure that no changes are made to the surface outside this area (Figure 2.6).

Additional views are provided so that the user can observe the effects of modifications from different points of view.

2.5 Problem Formulation

In this section, we provide a mathematical definition of our stroke parameters (Section 2.5.1) and demonstrate how the problem of converting shading changes specified by strokes into shape changes can be formulated as a quadratic surface optimization problem with linear constraints (Section 2.5.2). The problem formulation is independent of the choice of discretization, and can be applied, for example,

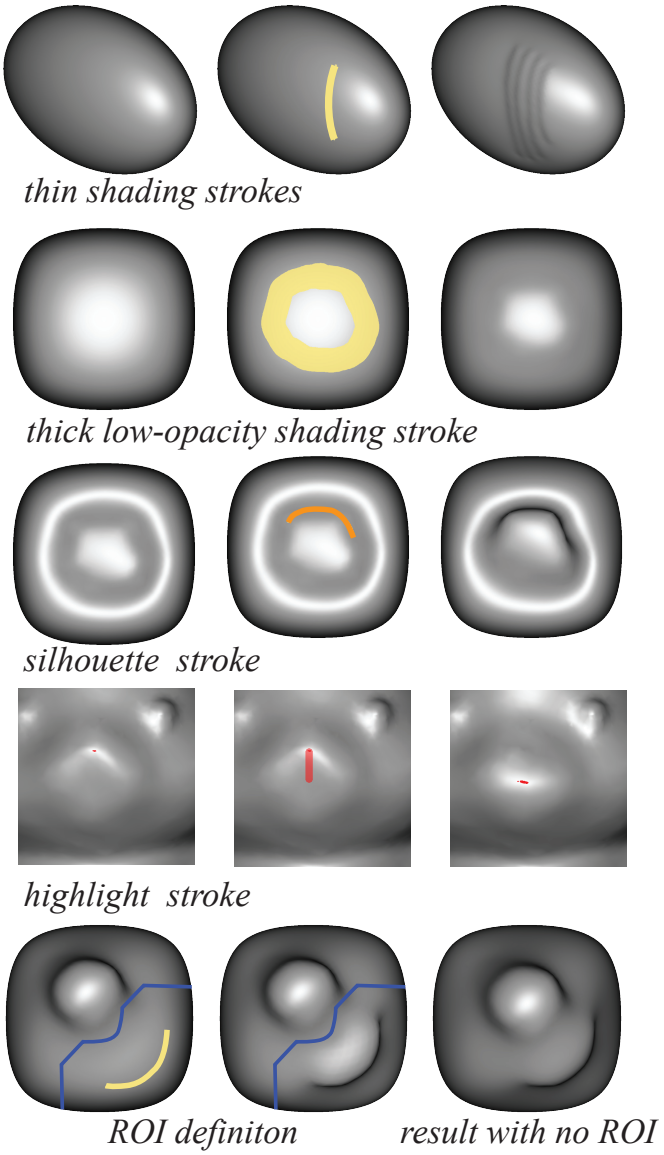


Figure 2.5: Examples of different tools applied to simple surfaces.

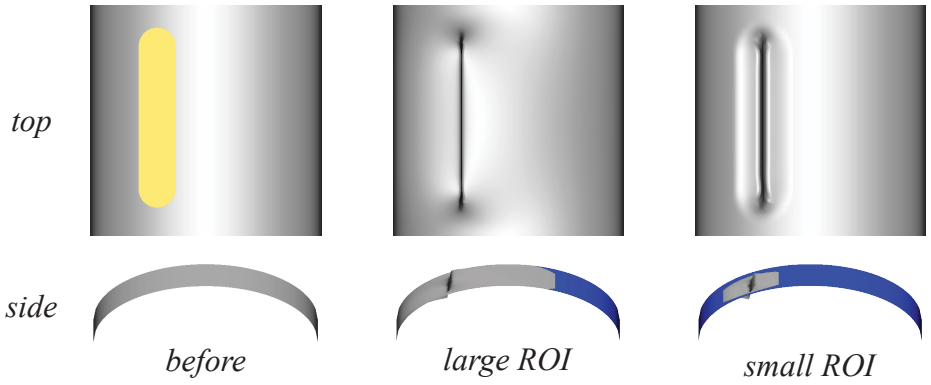


Figure 2.6: A darkening stroke applied to a simple surface with varying regions of interest. The blue region is frozen and unaffected by the stroke. The stroke and original surface (left). The result of the stroke with a sufficiently large region of interest (center). A small region of interest leads to a bright ringing artifact (right).

to spline or subdivision surfaces. We consider its discretization in Section 2.6.1.

The goal of our technique is to modify a given surface M, with or without boundary. We assume that M is a smooth (C^1) surface, such that the normals are defined everywhere, there are no self-intersections, and it is approximated by a mesh.

A single light source is located at a point $\mathbf{p}_l$, or at infinity in direction $\mathbf{l}$. For simplicity of discussion, we use a directional light source. Similarly, the camera is located at a point $\mathbf{p}_v$, or at infinity, and the view direction (the normal to the image plane) is $\mathbf{v}$. The projection to the image plane is denoted P; e.g., for an orthographic projection to a plane passing through zero, $P = I - \mathbf{v}\mathbf{v}^T$.

We assume the material has uniform properties specified by a reflectance function $\rho(\mathbf{n})$, where $\mathbf{n}$ are surface normals. We use simple reflection functions with Lambertian and glossy reflection terms of the form $(1-\beta)\langle \mathbf{n}, \mathbf{l}\rangle + \beta\langle \mathbf{n}, \mathbf{h}\rangle^p$, where β is the degree of glossiness, p is the Phong exponent, and $\mathbf{h} = (\mathbf{v}+\mathbf{l})/\|\mathbf{v}+\mathbf{l}\|$ is the halfway vector. We emphasize that we use this specific type of shading as a user interface widget. We do *not* attempt to make it possible to do shading-based modeling in realistic lighting conditions. Multiple light sources, complex reflection models, and variable surface properties make it much more difficult for the user to visualize desired changes in appearance.

In the context of this work, the *image* of the surface M is a function $I(\mathbf{q}) = \rho(\mathbf{n}(\mathbf{p}))$, where $\mathbf{p}$ is the closest point of M projecting to $\mathbf{q}$. (I is defined in the area of the image plane corresponding to $P(M)$, the projection of M.)

The user modifies the image function, $I(\mathbf{q})$, to obtain a new function, $\tilde{I}$. The

modification is confined to the smooth areas of I (i.e., we fix the silhouettes). Our goal is to construct a new surface, $\tilde{M}$, whose image matches $\tilde{I}$ as closely as possible, subject to a number of restrictions. Most modifications are based on *strokes*, defined by curves C in the image plane, corresponding to curves $P^{-1}(C)$ on the surface.

2.5.1 Stroke types

Strokes determine how the target intensity field is specified. While the attributes of strokes are similar to those found in two-dimensional drawing programs, a few aspects of these strokes need to be adapted to our application.

Strokes have an intuitive informal geometric interpretation: they correspond to variable rotations of the tangent planes about the centerline of the stroke. The attributes of the stroke determine how the planes are rotated and how the rotation propagates from the stroke. The surface change required for *simple* changes in appearance, such as uniform darkening, may be quite complex, although qualitatively we ensure that the behavior is intuitive; in particular, the rotation changes continuously along the stroke. The primary attributes of a stroke are the base curve C, width w, and value I_v.

Strokes have *softness* f, and *opacity* α, and can be applied in one of two modes, *multiply* or *replace*, which determine the interpretation of I_v. Stroke parameters, with the exception of softness and width, are used to determine target intensities for the surface. Suppose the original intensity of the surface at a point under the stroke is I_0. Then the target intensity I_{trg} is determined according to the following

formulas.

- replace mode:

$$I_{trg} = \alpha I_v + (1 - \alpha) I_0$$

  In this mode, I_v is interpreted as intensity and ranges from $0 \ldots 1$, with blending between the old and new intensity determined by opacity. In geometric terms, for Lambertian surfaces at 100% opacity, this corresponds to twisting the surface about the stroke in a way that forces the surface normals to have a given angle with the light direction.

- multiply mode:

$$I_{trg} = \alpha min(1, I_v I_0) + (1 - \alpha) I_0$$

  In this mode, the stroke darkens or brightens the underlying surface by a percentage determined by I_v, which ranges from $0 \ldots 1.5$. As a surface is painted over with a darkening stroke multiple times, it gets increasingly darker, asymptotically approaching zero intensity. Geometrically, this also twists the surface at each point, by an amount proportional to the angle of the normal with the light direction (again, for Lambertian surfaces).

Softness determines the sharpness of the transition between the stroke and the rest of the surface. The mechanism used for this is described in Section 2.5.3. A softness of zero corresponds to a sharp transition (normal discontinuity), and the maximal softness is one. The effects of changing different stroke attributes are shown in Figure 2.7.

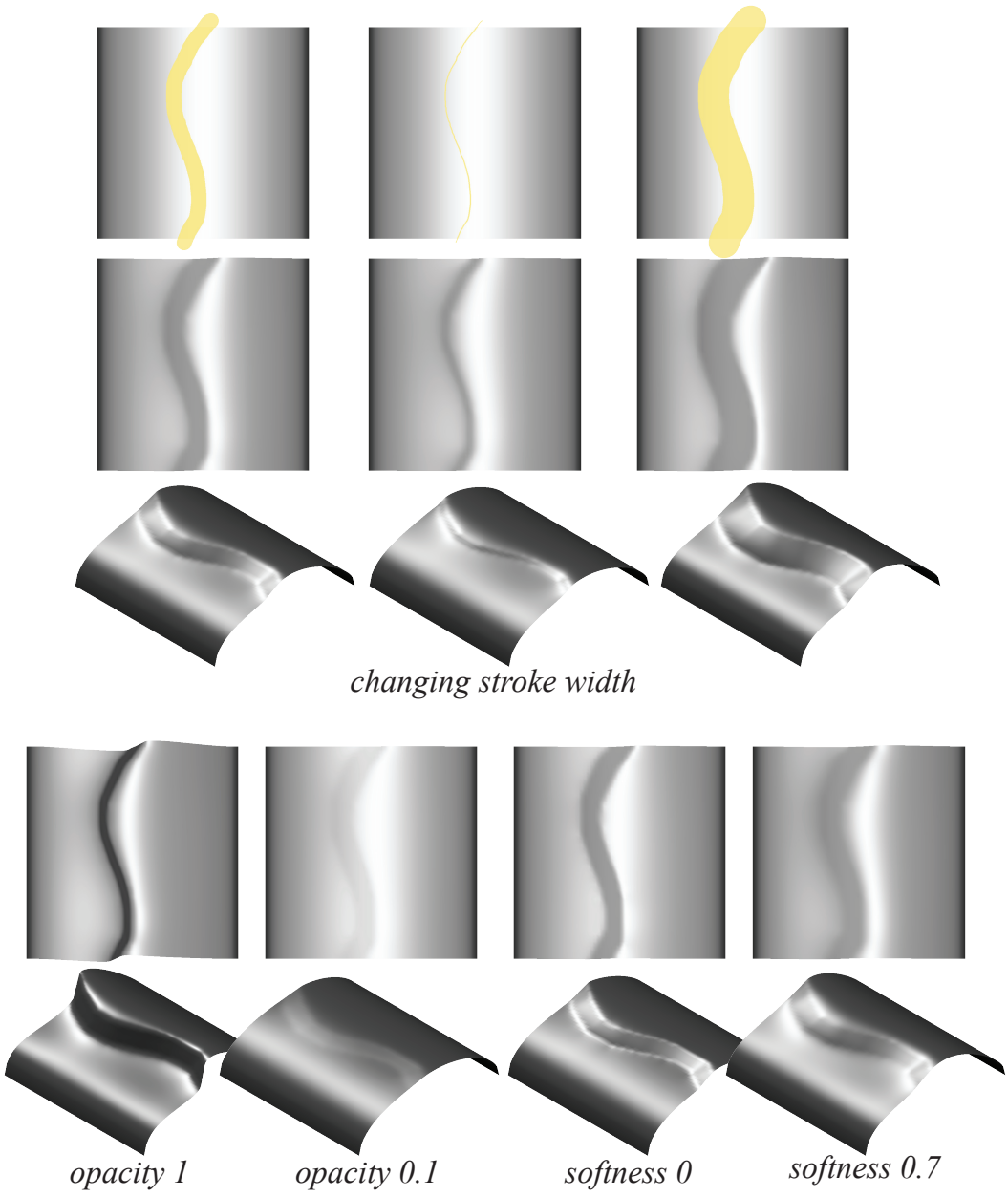


Figure 2.7: Changing stroke attributes: width, smoothness, and opacity.

2.5.2 Constrained Surface Optimization

In this section, we describe our surface optimization functional and basic stroke constraints.

In our system, the user modifies the surface one stroke at a time, generally changing shading in a small area of the surface. As a consequence, the goal of finding a modified surface which agrees with the updated image can be separated into two parts: recovering a (relatively small) part that matches the modified surface under the stroke, and keeping the rest of the surface as close as possible to the original.

The central idea of our approach is to treat the stroke as a special type of constraint and use a weighted detail-preserving functional to minimize the changes in the rest of the surface.

We motivate our choice for the detail-preserving functional first and then explain how stroke constraints are defined.

Preserving appearance outside strokes. There is a fundamental conflict between shading-based modifications and preserving the *surface itself* exactly unchanged outside the stroke. This can be clearly seen in Figure 2.7: if an area on a plane is darkened, the parts of the plane on two sides of the stroke can remain flat, but they need to be displaced. Similar to observations in [SCOT03] and [ACSD$^+$03], we note that high-frequency error matters more for appearance preservation; i.e., low-frequency error is preferred.

A natural choice for functionals of this type are those based on surface gradients

[YZX+04] and Laplacians [SCOL+04], used in a variety of contexts. The vector Laplacian is the normal scaled by the mean curvature: $\Delta_M \mathbf{x} = H\mathbf{n}$, where Δ_M is the Laplace-Beltrami operator on the original surface M, and H is the mean curvature. If the surface changes remain close to isometric, the Laplacian operator does not change [WBH+07], and the Laplacian difference

$$\int_M (\Delta_M \mathbf{x} - \Delta_M \mathbf{x}_0)^2 dA = \int_M (H\mathbf{n} - H_0\mathbf{n}_0)^2 dA$$

is a change in the normal orientation scaled by mean curvature. While we do not restrict our deformations to be isometric, if the triangle distortion stays small, one can view the Laplacian difference energy as a weighted normal change penalty. This closely matches what is needed for appearance preservation. A first-order Poisson approach uses the functional

$$\int_M (\nabla x - \nabla x_0)^2 dA.$$

In this case, the relationship to normal preservation can be seen from the Euler-Lagrange equation:

$$\Delta_M \mathbf{x} = \Delta_M \mathbf{x}_0.$$

The difference between the two functionals is primarily in supported boundary conditions. While the gradient-based functional allows only for positional or normal constraints on the boundary of the region of interest, the Laplacian-based functional makes it possible to join the modified patch with the surface smoothly, which is more suitable for our problem. We use a weighted Laplacian-based functional

$$\int_M g(\mathbf{x}_0)(\Delta_M \mathbf{x} - \Delta_M \mathbf{x}_0)^2 dA. \tag{2.1}$$

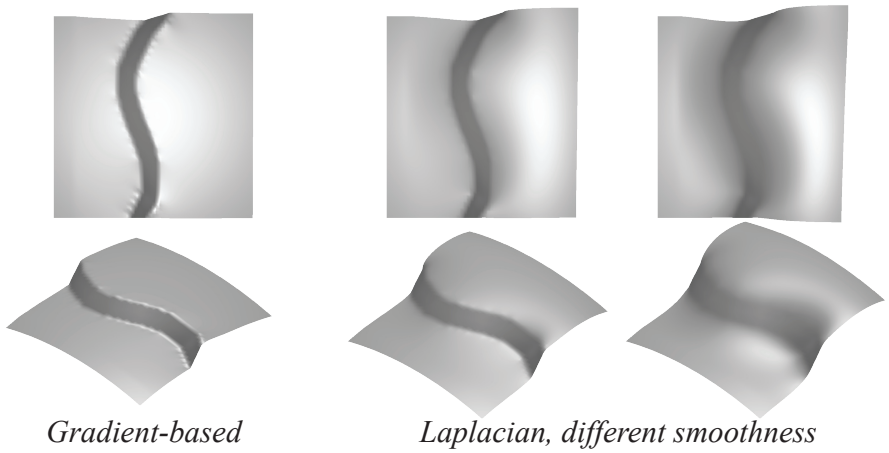


Figure 2.8: Laplacian vs. Poisson penalty outside of the stroke.

(The weighting function g is used to implement stroke smoothness, discussed in Section 2.5.3.)

We note that detail-preserving differential coordinate techniques are often complemented by various types of rotations applied to the differential coordinates (Laplacians and gradients) (e.g., [LSLCO05, BS08]). In the context of large deformations, it is desirable to preserve the orientation of details with respect to some coarse reference surface, as opposed to preserving details' world-space orientations. In our context, however, these rotations are clearly undesirable; for the surface to retain unchanged appearance, we *do* want the normals to retain their spatial direction with respect to the viewing direction and the light source.

Strokes as constraints. We treat strokes as normal and positional constraints on our weighted, detail-preserving functional.

The simplest case is *hairline strokes* of zero width. We treat this type of stroke as the combination of normal constraints along the stroke curve $P^{-1}(C)$ and positional constraints on the projection of $P^{-1}(C)$ to the image plane. The target intensities for the stroke are computed as explained in Section 2.5.1. In principle, in the case of a hairline stroke, the normal directions can be defined arbitrarily (integrability constraints need to be satisfied only if the normals are specified on an area). We find that the most predictable behavior is obtained if the normal is rotated in the plane perpendicular to the tangent to the stroke. If the initial normal is $\mathbf{n}_0$, and the stroke unit tangent is $\mathbf{t}$, the choice of normals is restricted to

$$\mathbf{n}(\alpha) = \cos(\alpha)\mathbf{n}_0 + \sin(\alpha)\mathbf{t} \times \mathbf{n}_0. \tag{2.2}$$

For each point, we find the minimal angle of rotation α such that $\rho(\mathbf{n}(\alpha)) = I_{trg}$, to obtain the target normal $\mathbf{n}_{trg}$. Figure 2.13 portrays these terms along with their discrete counterparts (discussed in Section 2.6.1). (We note that this works even if the reflectance function has multiple maxima, although we believe that simpler lighting choices are best for modeling.) If the surface is smooth, this angle changes continuously along a stroke, excluding the case of strokes passing through a highlight, as discussed in Section 2.5.3.

We discuss the discretization of stroke constraints in Section 2.6.1.

Choice of variables. Most work on shading-based surface recovery operates on *height fields*; that is, the surface is allowed to move only in the view direction.

Since we regard the problem as one of maintaining the three-dimensional surface shape away from the stroke, there is no particular reason to restrict motion of the surface to this single dimension. We find that distributing the error to all three dimensions, rather than restricting it to the view direction, is preferable. As shown in Figure 2.14, restricting deformations to height fields leads to extreme distortion of the mesh even in the simple situation of a single darkening stroke.

2.5.3 Realization of Stroke Attributes

In this section, we present the implementation of stroke smoothness and thick strokes, silhouette strokes, the interaction of strokes with highlights, and highlight motion strokes.

Stroke smoothness and thick strokes. Using hairline strokes to define constraints for optimizing a detail-preserving functional would not provide much control over the width of the stroke and the sharpness of the transition between modified and unmodified parts of the surface. One of the crucial elements of our construction is adding a weighting function to the Laplacian functional, making it possible to control stroke width and softness. The base curve of a thick stroke places the same constraints as a hairline stroke; they differ only in this weighting function.

The idea of the weighting function is to "weaken" the link between the stroke area and the rest of the surface, allowing the surface to bend more flexibly at the stroke boundary or even form a sharp feature for hard strokes. At the same

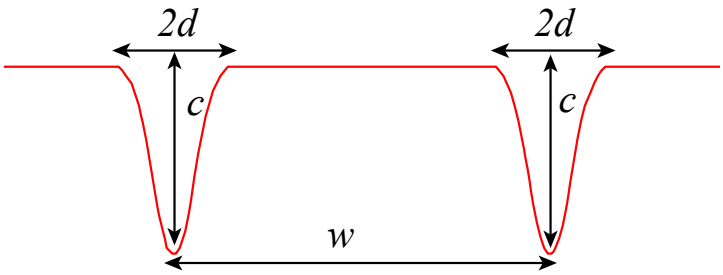


Figure 2.9: Weighting function profile across the stroke.

time, part of the surface remains tightly linked by the detail-preserving energy and rotates with the normals along the stroke base curve.

Substituting for g in Equation 2.1, we have our complete surface-preserving functional,

$$\int_M h(d(P(\mathbf{x}_0), C))(\Delta_M \mathbf{x} - \Delta_M \mathbf{x}_0)^2 dA, \tag{2.3}$$

where $d(P(\mathbf{x}_0), C)$ is the image plane distance from the projection of a point on the surface to the stroke, and $h(r)$ is a weighting function defined by the width w and softness f of the stroke as follows.

$$h(r) = \begin{cases} 1, \text{ for } -w/2 + d < r < w/2 - d \text{ and } |r| > |w/2 + d| \\ 1 - cB((r + w/2)/d), \text{ for } -w/2 - d \leq r \leq -w/2 + d \\ 1 - cB((r - w/2)/d), \text{ for } w/2 - d \leq r \leq w/2 + d \end{cases}$$

where $B(t)$ is a quadratic spline function satisfying $B(0) = 1$, $B(-1) = B(1) = B'(-1) = B'(1) = 0$. As can be seen in Figure 2.9, the weighting function is constant, excluding two areas at the stroke boundaries, of width d, and depth c. Generally, d has a very subtle effect on appearance; for meshes, it is selected to be sufficiently wide so that there is a closed chain of mesh edges entirely within d

of the stroke boundary; this ensures that when the functional is discretized, the non-unit weighting is applied to an area of the mesh surrounding the stroke. Once d is chosen, c is computed from the condition $(1-c)/d = f$ (very small values of d require c close to 1 for hard strokes).

The motivation for this choice is that the integral under the bump remains constant. Analytic computations for one-dimensional stroke cross-sections indicate, and experiments confirm, that the shape of the stroke is insensitive to the choice of d, as long as $(1-c)/d$ remains constant, within a broad range of d.

We found that this relatively simple approach works remarkably well. While it does *not* ensure uniform darkening if the stroke is applied to a surface area with lots of details, it effectively ensures average darkening while preserving the geometry of small-scale detail, as shown in Figure 2.10. An attempt to darken all points uniformly results in detail "smudging." While this is desirable in some cases, we believe this may be best controlled by a separate attribute.

Silhouette strokes. Silhouette strokes are implemented similarly to shading strokes, but they have no opacity α or value I_v.

Specifically, silhouette strokes are also based on constraining normal orientations along the base curve of the stroke, in this case finding $\mathbf{n}(\alpha)$ perpendicular to the view direction (see Equation 2.2). Silhouette strokes do have smoothness, implemented exactly in the same way as for shading strokes. Combined with shape-preservation optimization, this typically leads to some faces becoming back-facing. (Silhouette strokes would not be possible with a height field representation.)

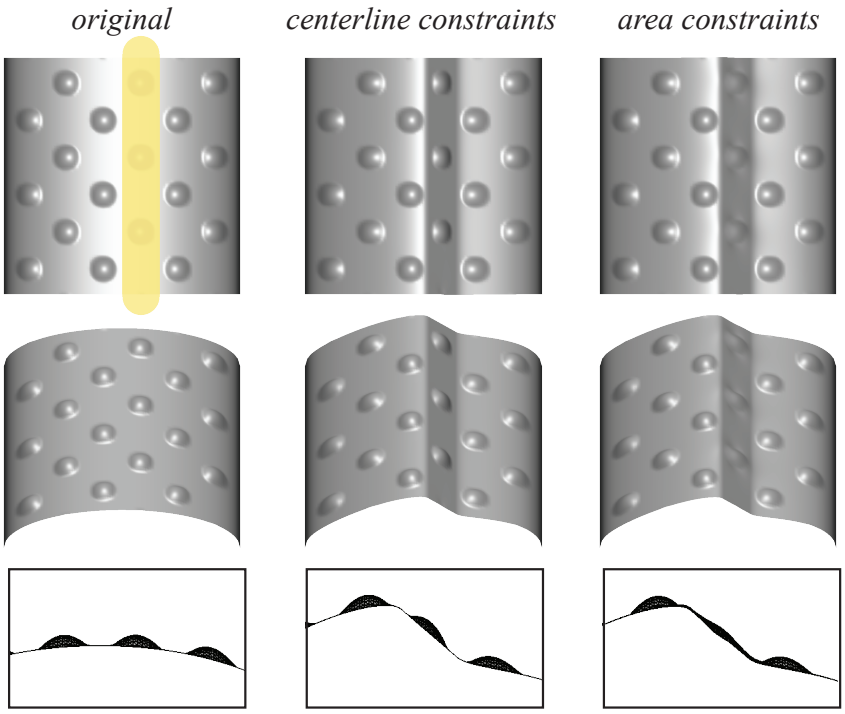


Figure 2.10: Detail preservation: a thick stroke with base curve constraints only, and the same stroke with constraints on normals imposed over the whole area in the least-squares sense.

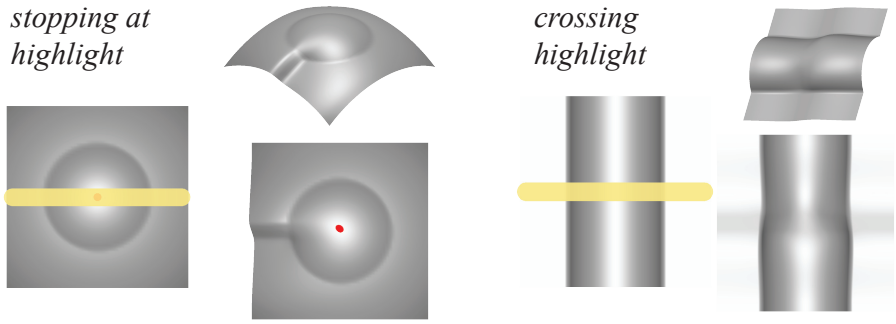


Figure 2.11: Types of stroke behavior at highlights.

Interaction with highlights. If a stroke passes a highlight, i.e., a local maximum of $\rho(\mathbf{n})$, then the normal rotation may experience a jump at this point: the preferred direction of rotation may change (but there are highlights for which it does not). This leads to nonintuitive behavior which we prefer to avoid. Thus, when a discontinuity of this type is detected, the stroke is terminated at the highlight (Figure 2.11).

Highlight motion strokes. A highlight motion stroke is designed to move a highlight. In highlight motion mode, the highlights are detected by thresholding the intensity at vertices, and the user can draw a stroke starting at a highlight.

Overall, the stroke operates the same way shading strokes work; however, the target normals along the stroke are defined differently. Let $\mathbf{x}_{old}$ and $\mathbf{x}_{new}$ be the old and new positions of the highlight on the surface. The constraints for a highlight motion stroke are defined as follows, and illustrated in Figure 2.12.

- The target position $\mathbf{x}_{new}^{trg}$ of $\mathbf{x}_{new}$ has the same position in the image plane

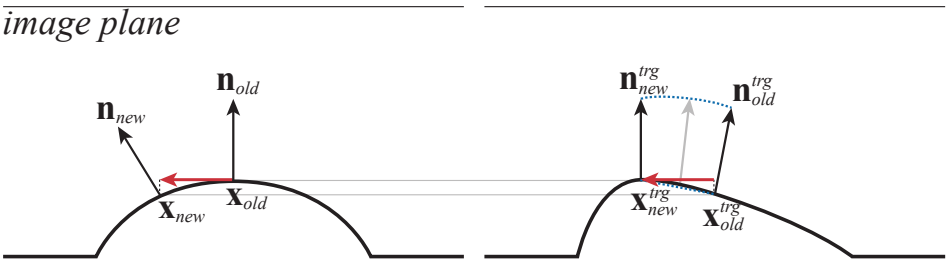


Figure 2.12: The effect of a highlight motion stroke, viewed from the side. The highlight motion stroke is depicted as a red arrow parallel to the image plane. The initial configuration is shown at left, and the result is shown at right.

as $\mathbf{x}_{new}$ and displacement from the image plane equal to that of $\mathbf{x}_{old}$. The normal at $\mathbf{x}_{new}$ is constrained to be the negative view direction.

- The target position $\mathbf{x}^{trg}_{old}$ of $\mathbf{x}_{old}$ has the same position in the image plane as $\mathbf{x}_{old}$ and displacement from the image plane equal to that of $\mathbf{x}_{new}$. The target normal at $\mathbf{x}_{old}$ is constrained to be perpendicular to $\mathbf{x}^{trg}_{new} - \mathbf{x}^{trg}_{old}$.

- The rotations of normals between these two points interpolate the rotations at endpoints.

2.6 Discrete problem

In this section, we describe the discretization we use for our surface optimization functional and for stroke constraints.

The discretization of the weighted Laplacian functional (Equation 2.3) is standard, using the well-known cotangent formula for Laplacians. (At the boundary of

the mesh, the edges get only a single cotangent in their weight [Sor08].) We refer to the work on Laplacian editing for details.

2.6.1 Discretization of Stroke Constraints

The constraints we impose for strokes affect vertices of edges intersecting the stroke. The stroke curve (if necessary, smoothed by subsampling and spline interpolation) is projected to the mesh and intersected with mesh edges. The target normals along the stroke are defined using the tangent to the stroke, the original normal, and the target intensity. Furthermore, the image plane position of the stroke, C, is constrained to be fixed.

We discretize these constraints as follows. An *edge* normal, $\mathbf{n}_e$, is computed for every edge, e, intersected by the projected stroke. A target normal, $\mathbf{n}_e^{trg}$, is computed for the edge, according to the stroke type and its attributes. Figure 2.13 portrays these terms along with their continuous counterparts.

The simplest linear constraint would be to require the new position of the tangent, $\mathbf{p}_2 - \mathbf{p}_1$, where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the endpoints of the edge, to be orthogonal to $\mathbf{n}_e^{trg}$; i.e., $\langle \mathbf{p}_2 - \mathbf{p}_1, \mathbf{n}_e^{trg} \rangle = 0$. However, this constraint is also satisfied if the edge has zero length, and we observe that the triangles often do degenerate. Instead, we obtain a target tangent vector e^{trg} by applying the minimal rotation mapping $\mathbf{n}_e$ to $\mathbf{n}_e^{trg}$ to the edge, resulting in the constraint $\mathbf{p}_2 - \mathbf{p}_1 = e^{trg}$. The advantages of this approach are shown in Figure 2.14. Both types of constraints are linear in vertex positions. In addition to constraining the edge direction, we also constrain the projected position of the point $P(\mathbf{p}) = P(a\mathbf{p}_1 + (1-a)\mathbf{p}_2)$ where the stroke

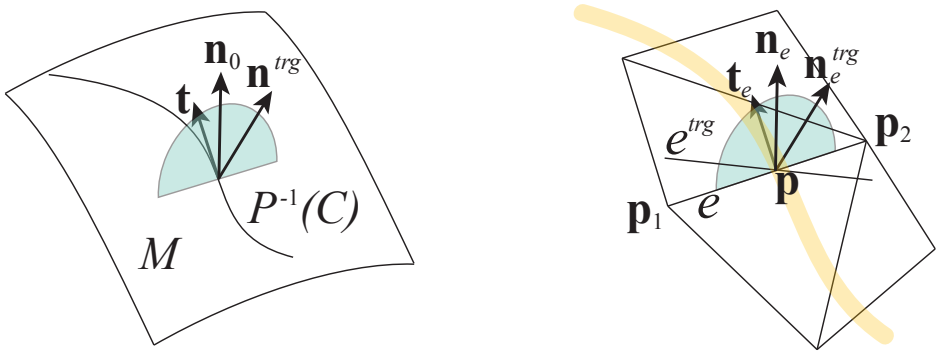


Figure 2.13: Notation for stroke constraints. A continuous (left) and discrete (right) surface patch.

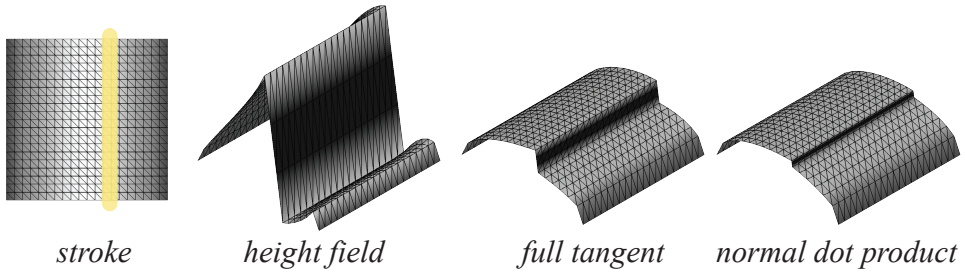


Figure 2.14: Dot product with target normal vs. full tangent constraints.

intersected the edge. This insures that the position of the projection of the stroke to the image plane does not change.

Adaptive refinement. One of the problems with image-space strokes is that it may not be possible to reproduce the stroke on the surface exactly—especially strokes with low softness or thin strokes—if the mesh is coarse. We use adaptive $\sqrt{3}$-subdivision to refine the mesh in the area of the stroke (Figure 2.15). The

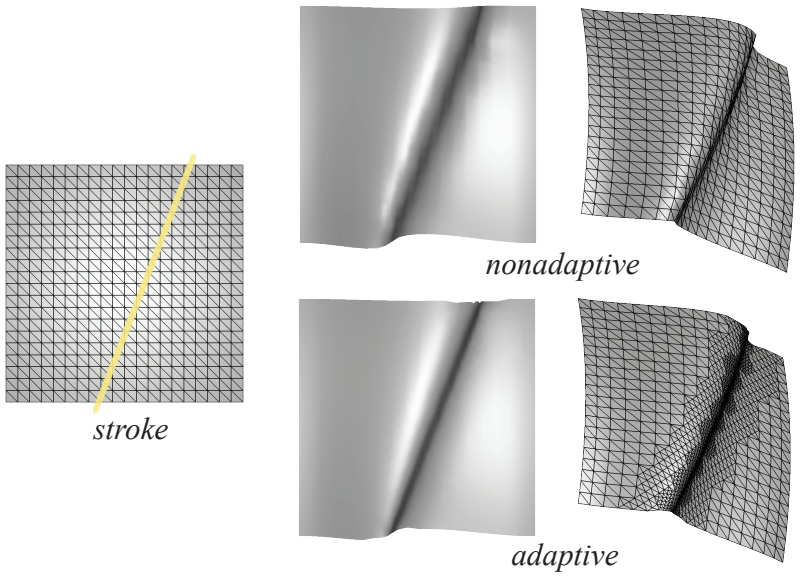


Figure 2.15: Adaptive refinement. The unrefined mesh has 441 vertices, which increases to 1302 vertices in the refined mesh.

criterion for refinement is to reduce the image-space length of the edges overlapped by the stroke boundary to a user-specified threshold (in pixels). Note that if the view of the model is zoomed, and a stroke is reapplied, additional refinement will occur, as the criterion is defined in image space.

2.6.2 System Assembly

Summarizing the above, we describe the construction of the linear system of equations that we solve. We minimize the following energy:

$$E = \sum_{i=0}^{N} g(\mathbf{x}_0^i) A_0^i \|\Delta\mathbf{x}^i - \Delta\mathbf{x}_0^i\|^2 + w_{lsq} E_{\text{constraint}}$$

where g is defined (by substitution) in Equation 2.3, $\mathbf{x}^i$ is the position of vertex i, $\mathbf{x}_0^i$ is the undeformed position of vertex i, A_0^i is the area corresponding to vertex i in the undeformed mesh, and N is the number of vertices in the ROI. $E_{\text{constraint}} = E_{\text{tangent}} + E_{\text{midpoint}}$, where

$$E_{\text{tangent}} = \sum_{(i,j)=e \in C} \|(\mathbf{x}^i - \mathbf{x}^j) - e^{trg}\|^2$$

and

$$E_{\text{midpoint}} = \sum_{(i,j)=e \in C} \|P(a_e \mathbf{x}^i + (1 - a_e)\mathbf{x}^j) - P(\mathbf{p}_e)\|^2$$

and C is the set of constrained edges. e^{trg} and $\mathbf{p}$ are defined in Section 2.6.1. We use $w_{lsq} = 1e7$ for all examples.

Since we fix silhouettes, all vertices belonging to back-facing faces are considered to be outside the ROI. (An exception is made for small back-facing components—those whose area is less than 10% of the front-facing ROI area—to prevent small cavities such as nostrils from becoming fixed.) In addition, an automatic ROI determination is available, which sets the ROI to the part of the surface whose image plane projection lies within a user-specified distance of the stroke. This reduces the system size for large meshes.

If the positions are transformed so that z is the view direction, then the system is decoupled.

2.7 Results

We demonstrate how our system can be used to modify a variety of models. The two main scenarios are modifying existing detailed models or refining a simple existing model. Examples can be seen in Figures 2.16-2.21. Shading strokes are shown in yellow and silhouette strokes are shown in orange.

In Figure 2.16, an eye is added to a simple horse model using many thin, smooth shading strokes. The initially coarse mesh is adaptively refined to allow for the addition of fine detail. In this example, all strokes are drawn from a single point of view. Muscles and sharp features are added to a simple male model in Figure 2.17. Shading strokes are used at varying scales to create medium as well as fine geometry changes. Figure 2.18 depicts eyes, nostrils, and ear cavities added to a model created in the FiberMesh system ([NISA07]). The eyes are added with several thin, sharp shading strokes, creating ridges. Such features are difficult to create using displacement editing tools. The nostrils and ear cavities are each created with a single smooth silhouette stroke. In Figure 2.19, wear and tear is added to a couch. The creases are added using with silhouette strokes. The rest of the features are added with shading strokes. In Figure 2.20, a thick shading stroke is used to give the mannequin head puffy cheeks. A silhouette stroke creates a more pronounced lip. In Figure 2.21, a sharp silhouette stroke is used to deepen a brow line on an elephant, and smooth shading strokes are used to create a bulging leg muscle.

All interaction sessions were recorded on a MacBook Pro with a 2 GHz Intel

Core Duo processor. A single core is used for computations. A graphics tablet is used as the input device.

The time for a stroke to be applied greatly depends on the stroke size, ROI setting, mesh size, and degree of adaptive refinement, ranging from instantaneous to seconds. The total optimization time is dominated by building and solving the system of equations. In our implementation, we use the sparse direct LU solver in the PETSc package. Any direct LU solver will suffice, and faster ones are available.

Performance is summarized in the following table. Each row represents a stroke applied to a model. Models indicated appear in Figures 2.16–2.21. The ROI column indicates the number of vertices in the ROI of the stroke; the system size is three times this number, squared. The constraints column indicates the number of constraints equations induced by the stroke. The "total" column summarizes the total computation for the stroke, in seconds. The "building" and "solving" columns indicate the percentage of the total computation spent building and solving the system.

model	ROI	constraints	total (s)	building	solving
figure 2.18	931	162	.25	37%	39%
horse	3859	237	1.38	44%	51%
elephant	9950	267	3.71	44%	52%

2.8 Conclusions and Future Work

We have described a general framework for controllable shading-based surface editing, providing a direct and intuitive interface for a broad range of surface modifi-

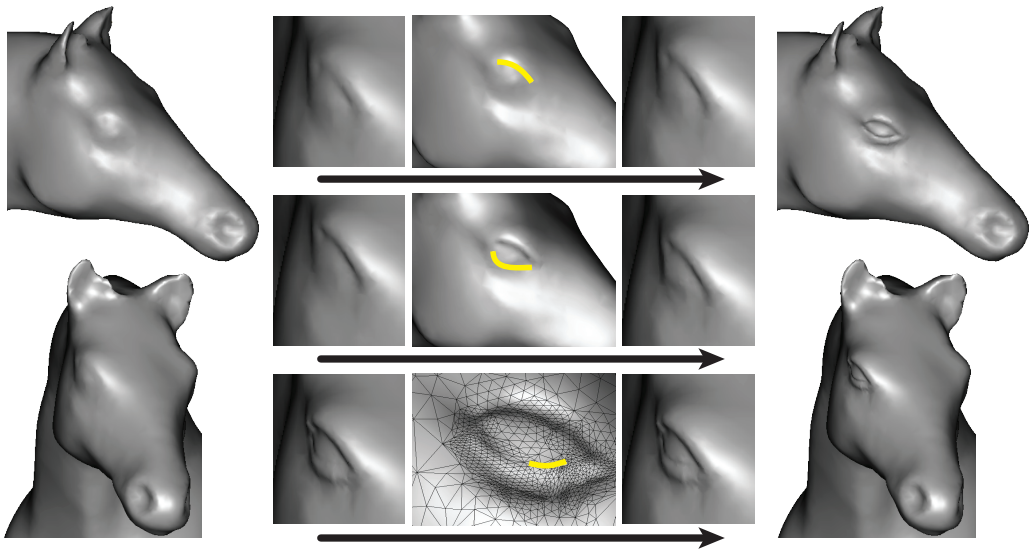


Figure 2.16: Adding an eye to a horse model with shading strokes; the mesh is adaptively refined. The mesh begins with 19851 vertices and is refined to 21116 vertices.

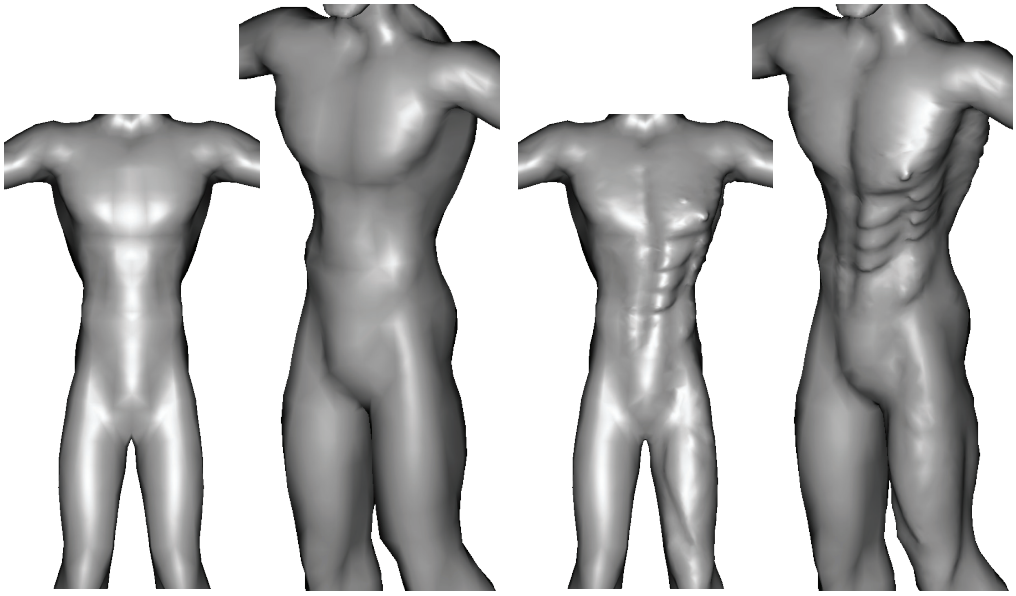


Figure 2.17: Refining a simple male model. The mesh begins with 5914 vertices and ends with 9151 vertices.

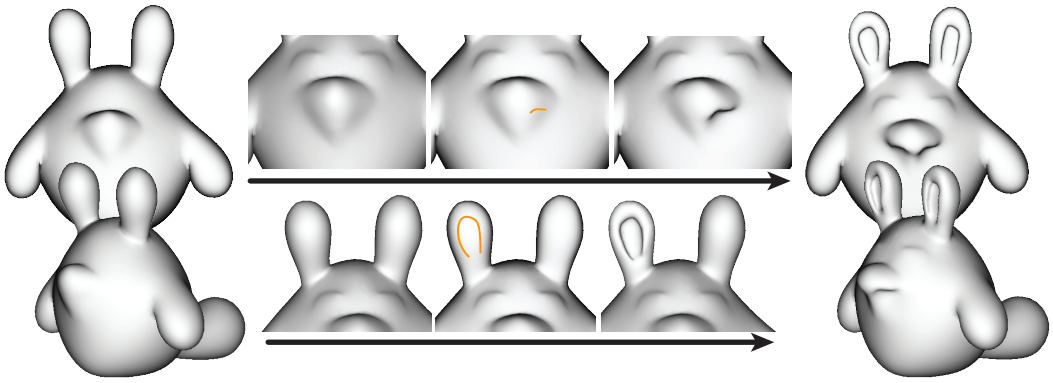


Figure 2.18: Refining a model created in the FiberMesh system ([NISA07]). The eyes are added using shading strokes; the nostrils and ears are added with silhouette strokes. The mesh begins with 3498 vertices and is refined to 5330 vertices.

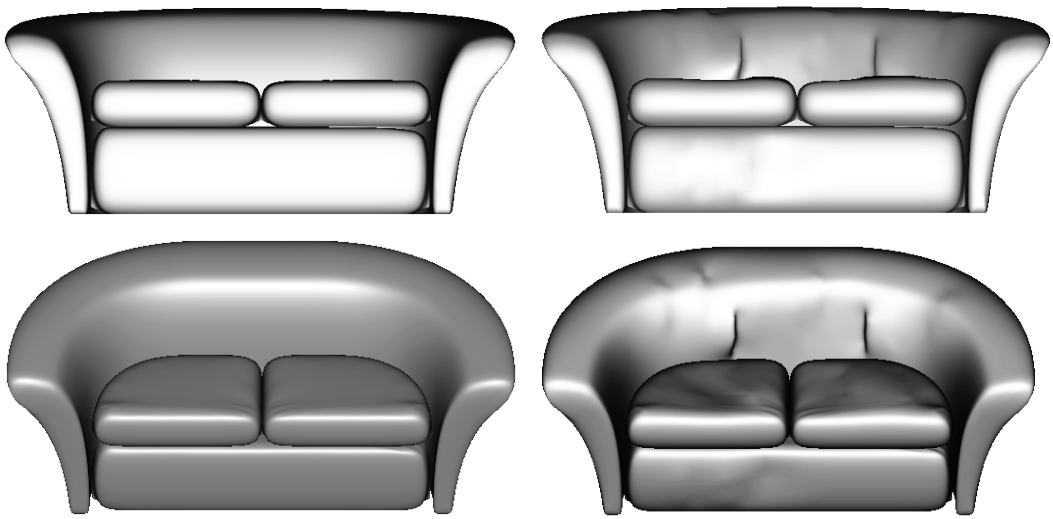


Figure 2.19: Adding features to a couch. The couch contains 31013 vertices.

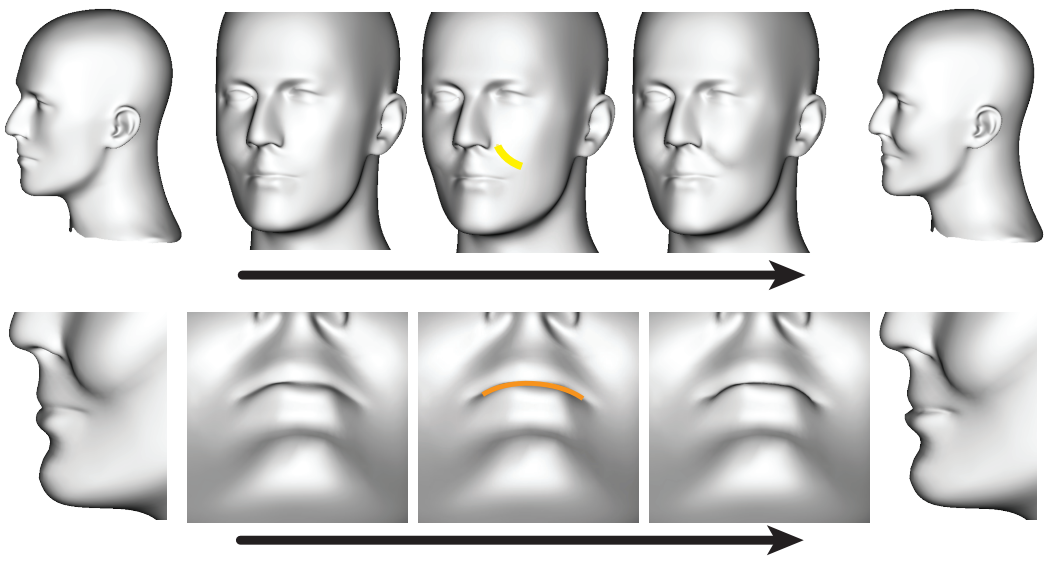


Figure 2.20: A shading stroke and a silhouette stroke applied to the mannequin head. The mesh begins with 10883 vertices and is refined to 12515 vertices.

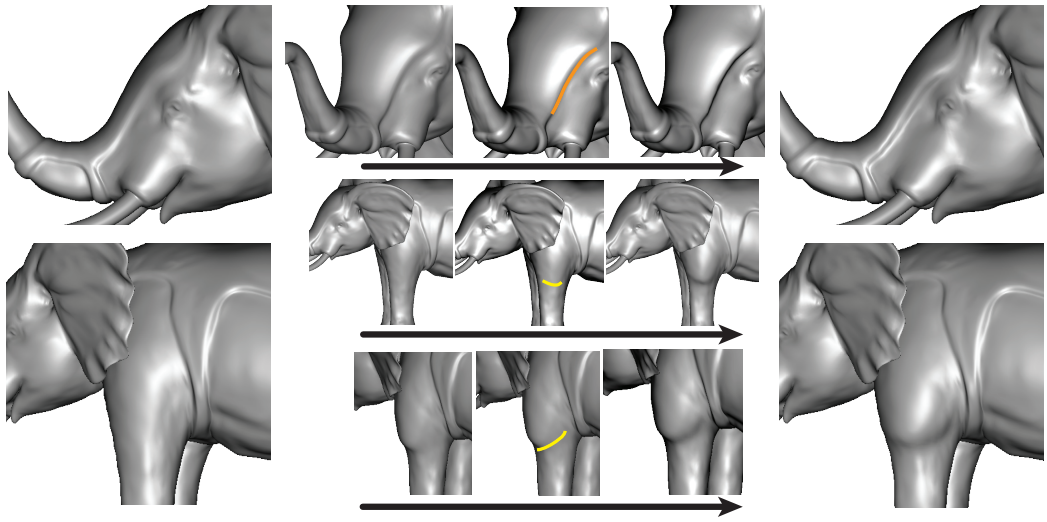


Figure 2.21: Deepening a crease and adding a muscle on an elephant model. The mesh contains 45682 vertices.

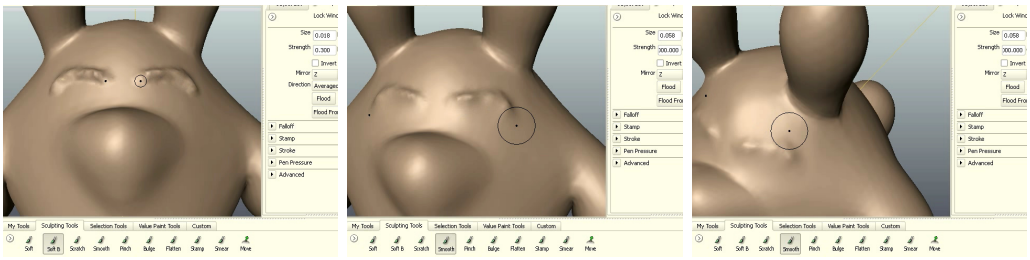


Figure 2.22: In a 3D sculpting system such as Mudbox [Aut08], adding eyes as in Figure 2.18 requires three operations instead of one: creating a thin ridge, smoothing below the ridge, and smoothing above the ridge.

cations. Our primary tool, shading strokes, have both an intuitive meaning when viewed as two-dimensional strokes in the image plane, and a predictable geometric behavior. A set of attributes makes them sufficiently flexible to achieve fine control over surface appearance.

We plan to extend this work in a number of ways. The control of highlights our tool provides is still quite limited. We found detail-preserving strokes most useful. However, in some cases it is desirable to eliminate details in a controllable way, so adding a "blur" attribute to the stroke would be useful. Our system naturally complements a number of other sketch-based approaches, so we will explore integration with other systems.

3
Structured Annotations for 2D-to-3D Modeling

3.1 Introduction

Traditional 3D modeling tools (e.g. [Aut09]) require users to learn an interface wholly different from drawing or sculpting in the real world. 2D drawing remains much easier than 3D modeling, for professionals and amateurs alike. Professionals continue to create 2D drawings before 3D modeling and desire to use them to facilitate the modeling process ([TS08, TBSR04, EHBE97]). Sketch-based modeling systems, such as Teddy [IMT99] and its descendants, approach the 3D modeling problem by asking users to sketch from many views, leveraging users' 2D drawing skills. In these systems, choosing 3D viewpoints remains an essential part of the workflow: most shapes can only be created by sketching from a large number of different views. The workflow of these systems can be summarized as "sketch-rotate-sketch." Because of the view changes, users cannot match their input strokes to a guide image. Moreover, finding a good view for a stroke is often difficult and time-consuming: In [SSB08], a 3D manipulation experiment involving users with a range of 3D modeling experience found that novice users were unable to complete their task and became frustrated. These novice users "positioned the chair parts as if they were 2D objects." The change of views is a *major bottleneck* in these

systems.

Sketching is also used in the context of traditional modeling systems: a workflow often employed by professional 3D modelers is placing axis-aligned sketches or photographs in the 3D scene for reference. This workflow could potentially allow amateurs who cannot draw well in 2D to create 3D models from sketches produced by others. Yet, paradoxically, this approach requires a *higher* level of skill despite relying on easier-to-produce 2D sketches as a modeling aid. This is because of the difficulty of using conventional tools, which require constant changes to the camera position, whereas a single view is needed to match an existing image.

The goal of our work is to design a user interface that simplifies modeling from 2D drawings and is accessible to casual users. Ideally, an algorithm could automatically convert a 2D drawing into a 3D model, allowing a conventional sketch (or several sketches) to serve as the sole input to the system. This would eliminate the need for view point selection and specialized 3D UI tools. However, many (if not most) drawings are ambiguous and contain inconsistencies, and cannot be interpreted as precise depictions of any 3D model (Section 3.3). This limits the applicability of techniques such as Shape-from-Shading ([Pra04]) and reconstruction from line drawings ([VC07]). Humans apparently resolve many of the ambiguities and inconsistencies of drawings with semantic knowledge. Our work provides an interface for users to convert their interpretation of a drawing into a 3D shape. Instead of asking the user to provide many sketches or sketch strokes from multiple points-of-view, we ask the user to provide all information in 2D from a *single view*, where she can match her input to the underlying sketch. In our tool, user

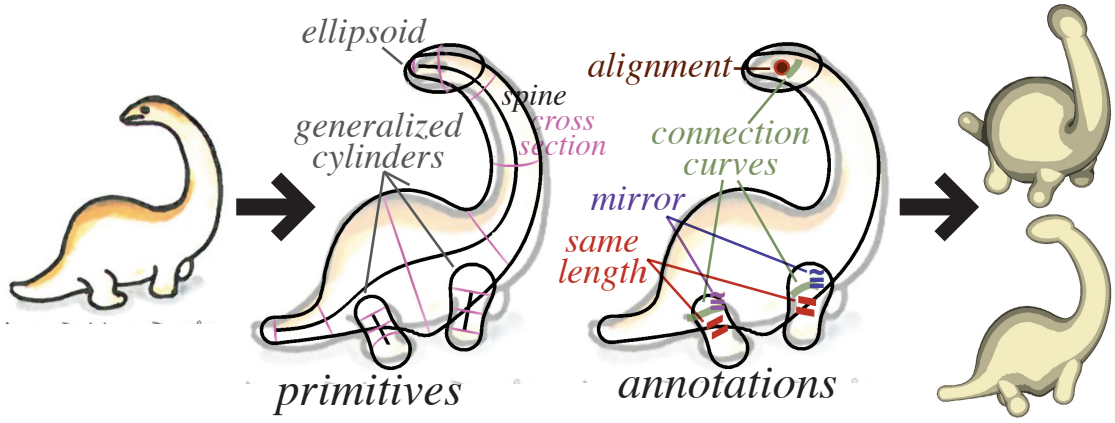


Figure 3.1: Our modeling process: the user places primitives and annotations on an image, resulting in a 3D model.

input takes the form of (1) *primitives* (generalized cylinders and ellipsoids) with dynamic *handles*, designed to provide complete flexibility in shape, placement, and orientation, while requiring a single view only, and (2) *annotations* marking similar angles, equal-length lines, connections between primitives, and symmetries, to provide additional semantic information. Our system generates 3D models entirely from this user input and does not use the 2D image. We do not expect that users have a consistent 3D mental model of the shape and are specifying primitives precisely; we aim to create plausible, reasonable quality 3D models even if a user's input is inconsistent.

Contributions. We have designed a system of user interface elements implementing an intuitive and powerful paradigm for interactive modeling from existing 2D drawings, based on the idea of "describing" an existing drawing by placing

primitives and annotations.

We present the results of a small user study showing that our interface is usable by artists and non-artists after minimal training and demonstrating that the results are consistently better compared to tools using the "sketch-rotate-sketch" workflow. We demonstrate that our system makes it possible to create consistent 3D models qualitatively matching inconsistent illustrations.

Our resulting 3D models are collections of primitives containing structural information useful in applications such as animation, deformation, and further processing in traditional modeling tools. We do *not* argue that one should perform all 3D modeling operations in a 2D view. Our goal is to demonstrate that it is possible to accelerate the creation of initial, un-detailed 3D models from 2D sketches, which can be further refined and improved using other types of modeling tools.

3.2 Related Work

Interactive, single-view modeling. Our approach is most similar in spirit to [ZDPSS01] and [WTBS07], in which users annotate a single photograph or drawing with silhouette lines and normal and positional constraints; the systems solve for height fields that match these constraints. In our system, the primitives and annotations added by a user are structured and semantic, and we are able to generate 3D models from globally inconsistent drawings. Our system rectifies the shape primitives placed by a user in order to satisfy the user's annotations (symmetries and congruencies).

Interactive, multiple-view modeling. In [DTM96] and [SSS+08] and [vdHDT+07], users mark edges or polygons in multiple photographs (or frames of video). The systems extract 3D positions for the annotations, and, in fact, textured 3D models, by aligning the multiple photographs. (In [DTM96], users align them to edges of a 3D model created in a traditional way). In our system, users have only a single, potentially inconsistent drawing; these computer vision-based techniques assume accurate, consistent input and hence cannot be applied to our problem.

Automatic, single-view sketch recognition. Sketch recognition techniques convert a 2D line drawing into a 3D solid model. These approaches also typically assume a simple projection into the image plane. Furthermore, a variety of restrictions are placed on the line drawings, such as the maximum number of lines meeting at single point, and the implied 3D models are assumed to be, for example, polyhedral surfaces. For a recent survey of line-drawing interpretation algorithms, see [VC07]. One notable recent work in this direction is [CKX+08], which allows for imprecise, sketched input by matching input to a domain-specific database of architectural geometry. More relevant to free-form modeling, the recent works of [KH06] and [CS07] generate 3D models from a single view's free-form visible silhouette contours. These works are primarily concerned with generating surfaces ([KH06]) or skeletons ([CS07]) correctly embedded in $\mathbb{R}^3$ with visible contours matching the user's input. They do not represent modeling systems per se, but rather a necessary component for any system taking silhouette contours as input. Our approach can be viewed as a form of user-assisted 2D-to-3D interpretation.

Because a human uses our tool to annotate the 2D image, we are able to receive user input that eliminates ambiguity and rectifies inconsistencies in the image.

Interactive 3D modeling. There are a variety of sketch-based modeling tools based on the concept of sketching curves from various angles (sketch-rotate-sketch). The earliest of these is [IMT99], and this direction has been explored in a variety of later works. A good overview can be found in the recent survey of [OSCSJ08]. These works assume users are capable of sketching a model from multiple points-of-view, and that users can find good views for sketching and manipulating the model. As such, users cannot trace a guide image. We do not assume such skill, and believe that that rotation and sketching from novel views is the most difficult aspect of these systems.

The work of [CSSJ05] deserves further mention. The goal of this work is to minimizing the number of strokes a user must draw to create a surface. They introduce "rotational blending surfaces," which are similar to our generalized cylinders. Notably, these surfaces can have a single arbitrary cross section, although the user must sketch the cross section from a rotated view. In addition, these surfaces' "spines" are planar xy curves unless over-sketched from a rotated view. The authors observe that the majority of modeling time taken for complex (multi-part) models was not in sketching, but in assembly (translation and rotation).

Two early sketch-based modeling systems relevant to our work are [ZHH96] and [EHBE97]. [ZHH96] introduced SKETCH, a gestural interface for 3D modeling. In SKETCH, users are capable of performing most modeling operations from a single view. For this reason, SKETCH could almost be re-purposed as a tool for

annotating existing 2D drawings. However, inconsistencies commonly present in 2D drawings preclude this application of SKETCH. In addition, SKETCH only supports a subset of CAD primitives and cannot be used for free-form modeling. [EHBE97] introduced constraints for beautifying users' imprecise, sketched input. We, too, use constraints, although our constraints are also designed to reconcile globally inconsistent user input.

3.3 Motivation

3D shapes in 2D sketches. 2D sketches are remarkably efficient at conveying the information sufficient to perceive a 3D object from a single point of view. Many 2D drawing approaches are based on composing (or decomposing, if drawing reality) a model out of primitive shapes (Figure 3.2, bottom row). The 3D shape of each primitive is relatively simple; primitives are primarily depicted with outlines and additional "scaffolding" ([SIJ+07]), typically lines indicating the shape of several *cross-sections*. In [Vil97], these primitives are sphere-, cylinder-, and cube-like shapes. Similarly, our interface is based on the idea of users placing primitives over a 2D drawing and modifying primitives' cross-sections to control their shape and match their appearance to the drawing.

Global inconsistency and ambiguity. Many 2D drawings are globally inconsistent and contain various ambiguities

It is well known that drawing and paintings, from quick sketches to classical works of art, contain elements drawn from different perspectives [AZM00]; in many

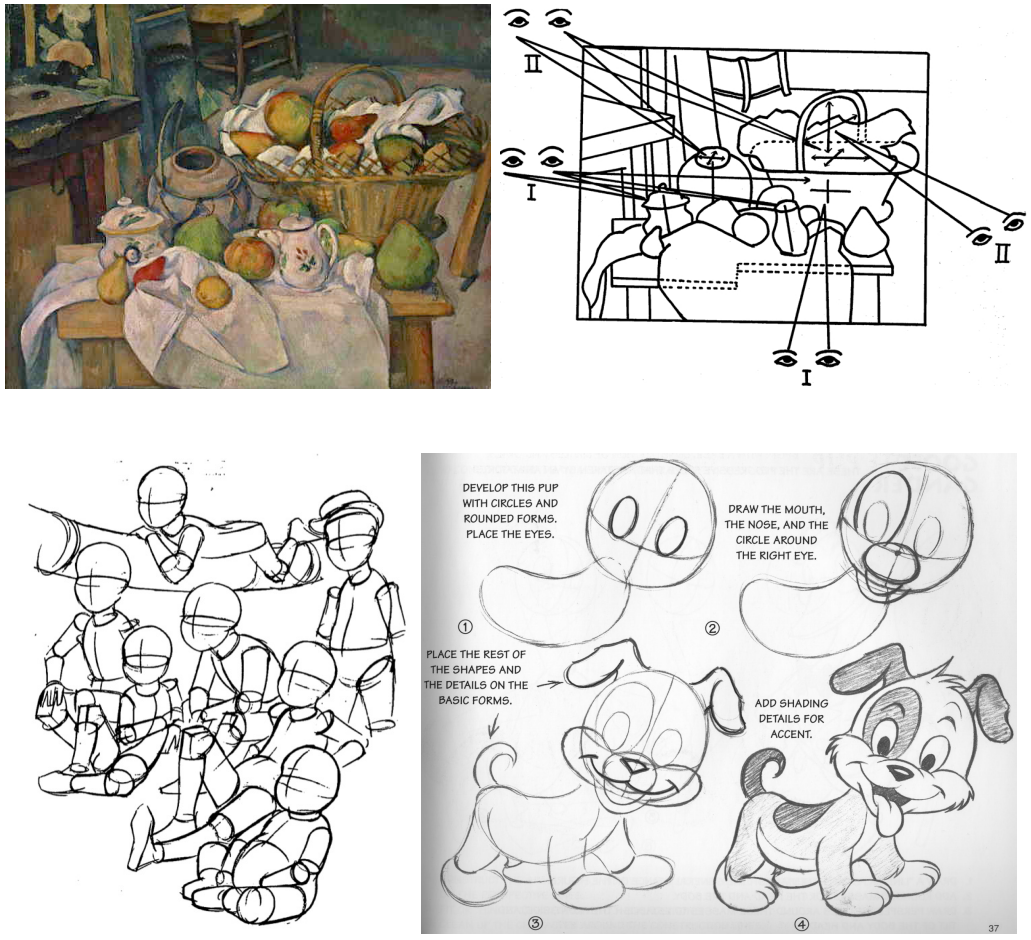


Figure 3.2: Top row: Cezanne's *Still Life with a Fruit Basket* (diagram from [Lor43]), reproduced from [AZM00]. Bottom row: Drawing using primitive shapes from [Vil97] (left) and [Bla94] (right).

cases, one cannot unambiguously infer the 3D view used for the 2D image.

Due to inconsistencies and inaccuracies, intended or unintended, one cannot in general take a drawing to be a precise projection of a 3D model (cf. Figure 3.2, top row). For this reason, we cannot hope to reconstruct a good-quality 3D model without additional information. Nevertheless, even poor quality drawings are understandable by humans; they are able to convey local shape information and are usually sufficient to recognize the object. Our system provides several types of *annotations* for users to input essential semantic information. We combine the primitives' local shape information with the annotations' semantic information to create a plausible, globally consistent 3D model.

3.4 User Interface

Users begin a modeling session by choosing an arbitrary image file containing their drawing (or other source material). Our interface presents users with a window displaying the image and a palette of primitive and annotation tools (Figure 3.3). *Primitives* (generalized cylinders and ellipsoids) are used to create simple object parts; they are manipulated with several types of handles for single-view shape editing. *Annotations* describe geometric relationships between these shapes, such as equality of lengths and angles. Users proceed to place primitives and annotations over the image, which is visible as an underlay; only primitives' silhouettes and cross-sections are visible in the interaction view.

Primitives (e.g., a character's body) are initially created with their spines flat in the image plane. The main tools allowing implicit out-of-image-plane shape

deformation and 3D positioning are cross-section tilt handles and connection curves attaching two primitives together. By tilting cross-sections, the user "lifts" a shape out of the plane, while preserving its outline. This process is similar to the way artists suggest the 3D shape and orientation of an object by sketching cross-sections, even if these are erased in the final image. Primitives are positioned with respect to each other (e.g., legs with respect to the body) using connection curves. Connection curve annotations connect two primitives where they overlap, and determine the depth ordering. Tilting cross sections and attaching primitives together make it possible to define a 3D shape matching a sketch using single-view interactions.

Additional relationships between primitives can be specified by other annotations: alignment, same-length and angle, and mirroring. Alignment and mirror annotations place primitives with respect to the *symmetry sheet* of another primitive. Symmetry sheets are controlled by the user and generalize symmetry planes; unlike a mathematically defined symmetry plane or axis, symmetry sheets do not necessarily capture a precise geometric property of a shape; rather, it provides a means to communicate to the system the semantics of the object. For example, consider a cylinder deformed to a snake-like shape. Before the deformation, the cylinder had well-defined symmetry planes which became non-planar sheets; the sheets retain the semantic of symmetry planes.

A 3D preview window is available at all times, allowing the user to view and rotate the 3D model. This window is only used for verifying the model, not for interaction. For inconsistent sketches, the 3D model cannot match the 2D sketch

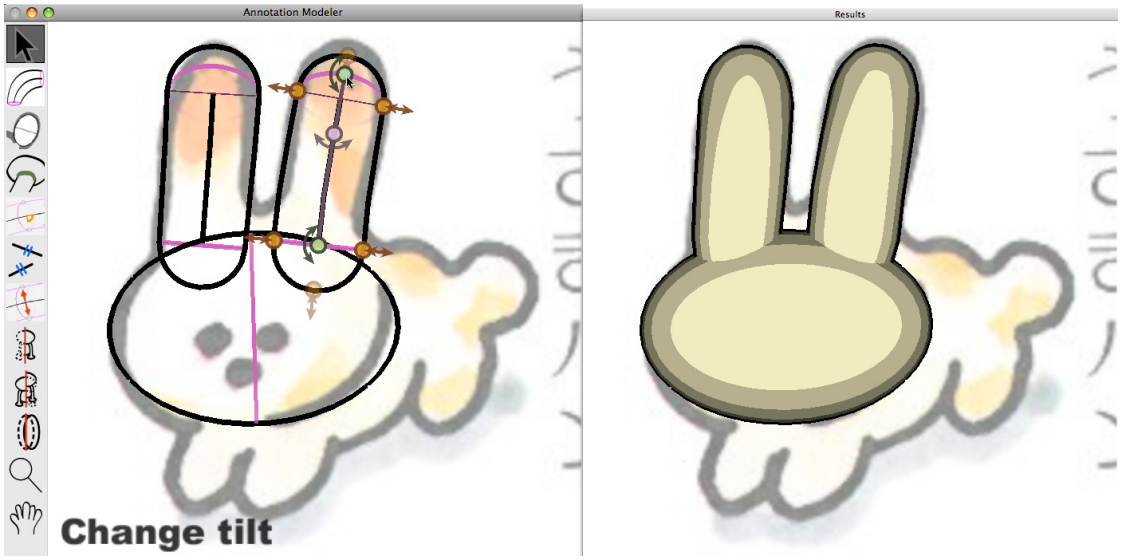


Figure 3.3: A screenshot of our interface.

and user input exactly, and the ability to evaluate the 3D model directly is essential.

3.4.1 Primitives

In the interaction view, primitives are depicted as 2D outlines with handles for manipulating their degrees of freedom. The primitives resemble Autoshapes [Mic03], with an important difference that the number of handles is not predefined and can be altered by the user. Following our observations in Section 3.3, we provide two kinds of primitives, generalized cylinders and ellipsoids (Figure 3.4). While an ellipsoid can be treated as a special case of a generalized cylinder, we have found it important to consider it a separate primitive type, as the set of suitable handles is different. The guiding design principle for all controls is direct manipulation of the

appearance of 2D projections of curves associated with a primitive, for example, a cross-section or a silhouette. In standard 3D modeling systems, the user controls parameters such as 3D rotations, translations, and spatial dimensions. In our system, those parameters are affected indirectly when users match a primitives' curves to curves in a 2D sketch. This indirect manipulation of 3D parameters by direct manipulation in 2D enables the single-view manipulation necessary for easy construction of 3D models from sketches.

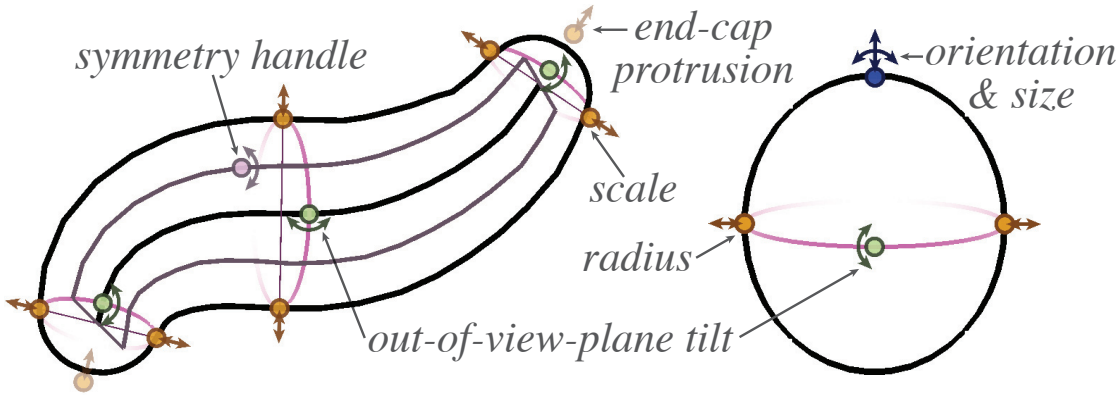


Figure 3.4: Primitives: A generalized cylinder (left) and an ellipsoid (right).

Generalized Cylinders. The first primitive is the generalized cylinder (Figure 3.4, left). It is defined by a 2D spine curve and attributes associated with cylinder cross-sections: cross-section shape (a closed curve), out-of-image-plane cross-section tilt, cross-section scale, and the symmetry director used as a reference for orienting cross-sections and to define the symmetry sheet as explained below. Most of these attributes are directly controlled by handles, excluding the cross-section shape, for which a separate 2D sketching mode is used. These at-

tributes are smoothly interpolated along the spine. At either end, the user can choose the end-cap protrusion as well. The 3D shape is defined as the union of scaled cross-sections (with tangent-continuous end-caps); the exact definition is presented in Section 3.5.1. To create a generalized cylinder, the user draws a free-form spine in 2D (Figure 3.5). A newly created generalized cylinder has a circular cross-section, a symmetry director parallel to the image plane, no out-of-image-plane tilt, a scale 5% of the screen width, and hemispherical end-caps. The newly created primitive has scale handles and tilt handles at both ends of the spine. The user can add and remove handles freely along the spine, provided that there is at least one handle of each type.

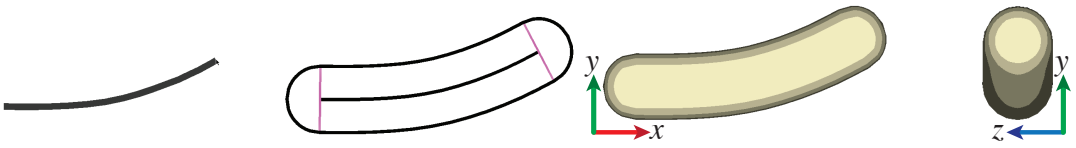


Figure 3.5: Creating a generalized cylinder from a stroke.

Out-of-image-plane tilt handle. This is one of the most important shape manipulation tools we provide. Out-of-image-plane tilt handles make it possible to bend a primitive out of the image plane while preserving its silhouette, which typically matches a silhouette in the 2D guide image. With no tilt, the cross-section plane is perpendicular to the image plane and so appears as a straight line. By dragging the handle, the user tilts the cross-section, rotating it about the axis perpendicular to the spine and parallel to the image plane; the cross-section's 2D projection changes to a closed curve (Figure 3.6). The 3D preview window is often

important for cross-section tilt manipulation. While specifying a tilt in the 2D view allows the user to maintain consistency with the sketch, small changes in the cross-section tilt can sometimes result in large changes in the 3D shape; the ability to evaluate the result from an additional viewpoint allows for more reliable control of orientation.

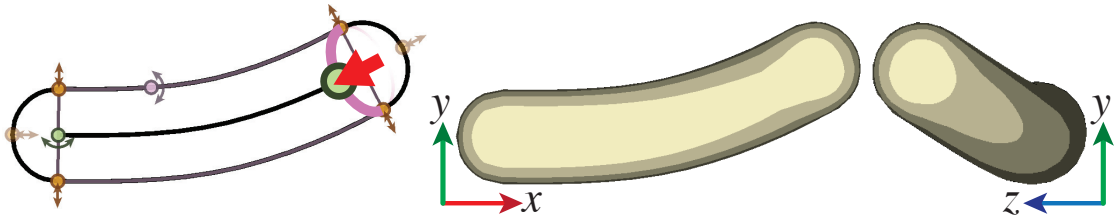


Figure 3.6: Tilting a circular cross-section out of the image plane. (Editing the generalized cylinder from Figure 3.5.)

Cross-section scale handle. A scale handle is used to change a cross-section's size. A scale handle is added anytime a user clicks and drags on the primitive's silhouette. As the user drags, the opposite point on the silhouette remains fixed, while the silhouette point under the mouse follows (Figure 3.7, top); the corresponding point on the spine is adjusted to remain in the center of the cross-section. Optionally, the user can scale the cross-section symmetrically, keeping the spine fixed (Figure 3.7, bottom).

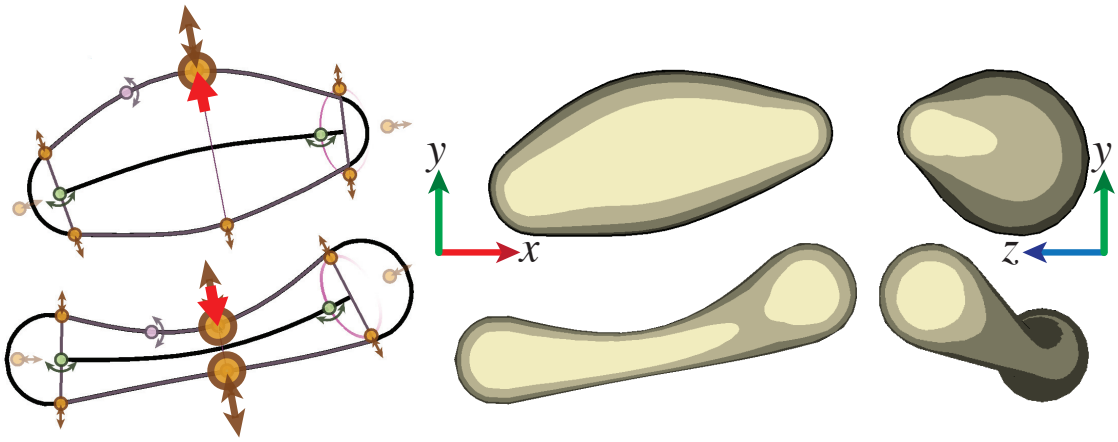


Figure 3.7: Top row: Changing the scale of a cross-section; the handle opposite remains fixed. Bottom row: Changing the scale of a cross-section; the handle opposite moves synchronously. (Both rows edit the generalized cylinder from Figure 3.6.)

Symmetry sheet handle. The symmetry sheet is defined by the 3D spine and the interpolated directors. The sheet is the ruled surface obtained as a union of the director lines at all points along the spine. We provide a handle to the user to select the director in any cross-section. The intersection of the sheet with the surface of the generalized cylinder is displayed in purple (Figure 3.8).

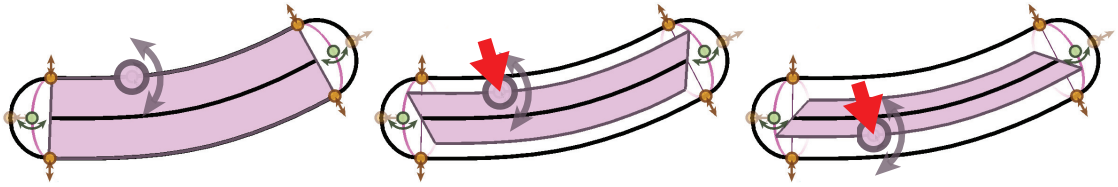


Figure 3.8: Adjusting the symmetry sheet of a generalized cylinder.

Cross-section shape adjustment mode. Arbitrary cross-section shapes can be drawn at any point along the spine, via a separate 2D mode accessible in a contextual menu (Figure 3.9). In this mode, a canvas is displayed and the user may draw the arbitrary cross-section curve or choose from a palette of common cross-sections. Cross-section curves are oriented so that the x direction of the 2D cross-section curve is aligned with the symmetry director and normalized to fit inside a unit circle. They are then scaled by the corresponding scale along the spine.

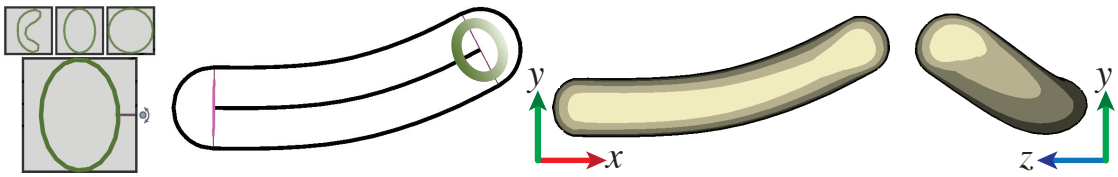


Figure 3.9: Editing a generalized cylinder's cross-section curve. The edited cross-section is drawn in green. (Editing the generalized cylinder from Figure 3.6.)

Spine manipulation. Finally, the user can click and drag any point on the spine to initiate a curve deformation with a peeling interface, as in [IMH05]. Al-

ternatively, the user can also over-sketch the spine, replacing all or a portion of it, depending on whether the over-sketching curve begins or ends near the spine. These operations are shown in Figure 3.10.

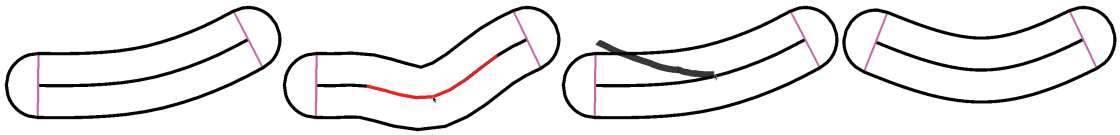


Figure 3.10: From left to right: A generalized cylinder; deforming its spine by peeling; over-sketching its spine; the result of over-sketching.

End-cap handles. End-cap handles determine the protrusion of the end-caps at either end of the generalized cylinder. These handles lie on the shape outline at the points obtained by continuing the spine to the silhouette tangentially. (Figure 3.11).

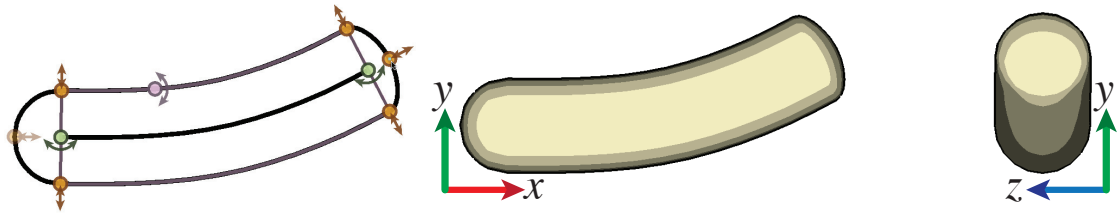


Figure 3.11: Adjusting the end-cap protrusion. (Editing the generalized cylinder from Figure 3.5.)

Ellipsoids. The second primitive is the ellipsoid (Figure 3.4, right). With this primitive, the user draws a free-form curve, to which a 2D ellipse is fit (in a least-

squares sense). Handles are provided to change its out-of-image-plane tilt and the length and orientation of the axes of its 2D projection (Figure 3.12). The out-of-image-plane tilt handle appears and operates identically to that of a generalized cylinder. Ellipsoids' two smaller axes are constrained to have the same length, so all cross-sections perpendicular to its longest axis are circular. An ellipsoid's symmetry plane passes through its center and is perpendicular to its long axis; as a result, it is tied to and controlled by the the same handle as the cross-section tilt.

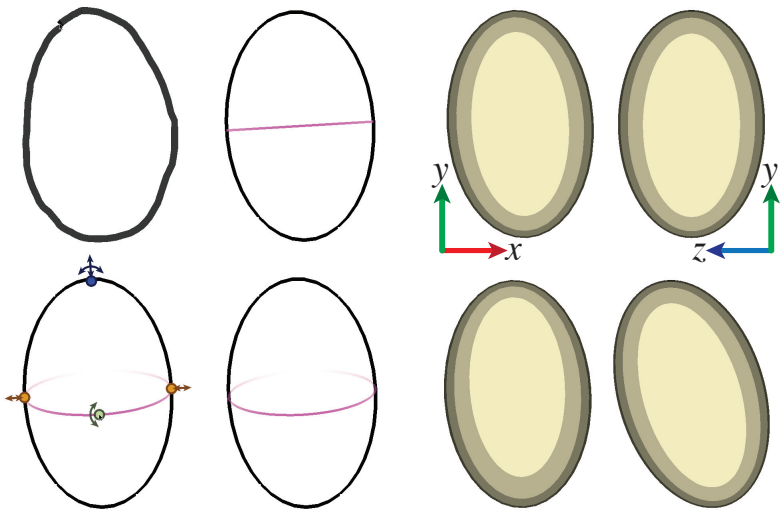


Figure 3.12: Ellipsoids. Top row: Creating an ellipse from a stroke. Bottom row: Tilting a circular cross-section out of the image plane.

3.4.2 Annotations

Annotations are the key to establishing a consistent 3D model that reconciles primitives placed on an inconsistent 2D image with geometric properties the user knows to be true. For example, the arms of a character should be the same length, even if there is no 3D model with arms of the same length that would project to the provided 2D sketch precisely. In addition, a character's arms should be attached to its body symmetrically opposite each other.

Connection curve annotations attach two primitives together and establish the (relative) depth between them. Users connect two overlapping primitives to each other—which places them in depth—by clicking and dragging in their 2D overlapping region. The primitives are translated in the z direction so that their 3D surfaces intersect, and the intersection curve passes under the mouse; the intersection curve is drawn in green and remains in the interior of both (2D) primitives (Figure 3.13). Alternatively, users may draw the free-form intersection curve he or she wishes the 3D surfaces would make with each other. This stroke must begin on the overlap of exactly two primitives, which determines which two primitives are to be connected. By default, the connection curve is taken to be the intersection curve between the front faces of the 3D surfaces; this can be changed to the back faces via a menu.

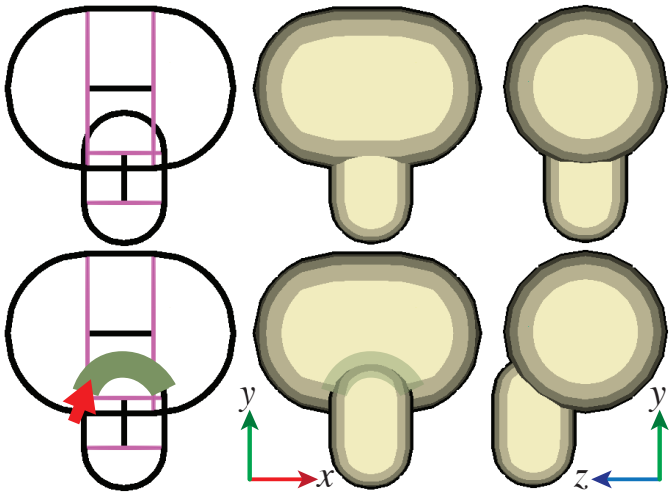


Figure 3.13: Attaching two primitives with a connection curve annotation.

Mirror annotations create a copy of a primitive (and its attached primitives) reflected across another primitive's symmetry sheet (Figure 3.14). Mirror annotations can be used to create characters' occluded, symmetric arms or legs.

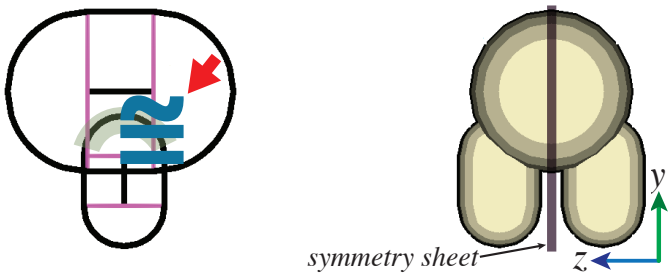


Figure 3.14: Mirroring one primitive about another. (Annotating the primitives from Figure 3.13.)

Alignment annotations align one or two primitives with respect to a connected primitive's symmetry sheet. If only one primitive is chosen to be aligned, it is translated so that its attachment origin (defined in Section 3.5) lies on the symmetry plane (Figure 3.15, top row). If two primitives are chosen to be aligned, they are translated so that their attachment origins are a reflection of each other with respect to the symmetry sheet (Figure 3.15, bottom row).

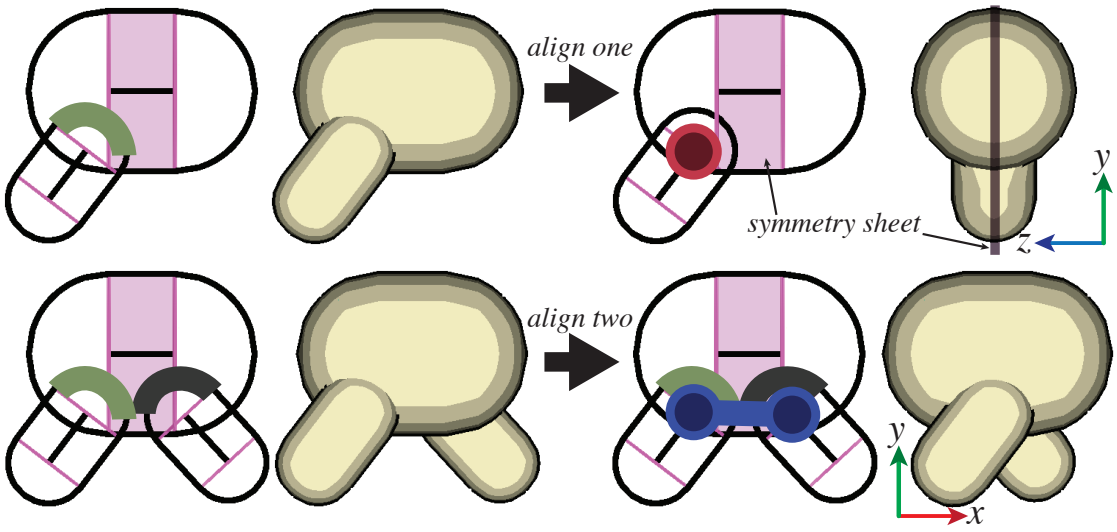


Figure 3.15: Top row: Aligning one primitive on another's symmetry plane. Bottom row: Aligning two primitives with respect to another's symmetry plane. (The dark connection curve connects back faces.)

Several annotations mark equal geometric measurements.

Same-length annotations mark two primitives as having the same long axis length, for an ellipsoid, or 3D spine length, for a generalized cylinder. Users

mark two primitives, and same length markings, familiar from geometry textbooks, appear along the primitives' lengths (Figure 3.16). Note that the primitives in the interaction view do not change; in general, primitives are placed to match a guide image.

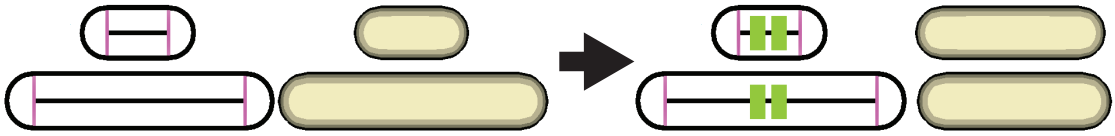


Figure 3.16: A same-length annotation.

Same-tilt annotations mark two or more cross-sections as having planes whose tilt angles with respect to the image plane are the same. As users mark circular cross-sections, angle markings, familiar from geometry textbooks, appear near the center of the titled cross-sections (Figure 3.17).

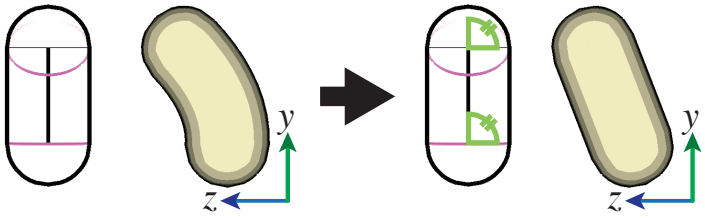


Figure 3.17: A same-tilt annotation.

Same-scale annotations mark two or more cross-sections as having the same scale. The arrows spanning the diameter of the chosen cross-sections are marked with familiar same length markings (Figure 3.18).

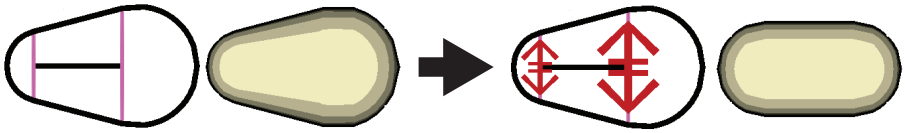


Figure 3.18: A same-scale annotation.

3.5 Implementation

In this section, we describe how we build a 3D model from user input. Because we assume that images are globally inconsistent, some primitives' parameters may contradict annotations: for example, the user may indicate that two legs have equal length, yet the primitives defining the legs are not. Because annotations provide semantic information the user knows to be true, annotations take precedence over primitive parameters set using handles.

Annotations are applied in the following order: (1) same-scale (2) same-tilt (3) connection curves (4) same-length (5) alignment (6) mirror. Same-scale and same-tilt annotations modify the shape and orientation of primitives; connection curves position primitives; same-length and alignment annotations adjust the shape and position of primitives; and mirror annotations create additional, aligned instances of primitives.

3.5.1 Primitives

Generalized cylinders. A generalized cylinder is defined by a 3D spine curve $\gamma(t)$ ($t \in [0, 1]$), a cross-section scale function $s(t)$, symmetry directors $d(t)$, and

closed, 2D cross-section curves $\alpha_t(u) = (\alpha_t^1(u), \alpha_t^2(u))$ $(u \in [0,1])$. The simplest definition of a generalized cylinder's surface is

$$c(u,t) = \gamma(t) + s(t)\left(\alpha_t^1(u)d(t) + \alpha_t^2(u)d^{\perp}(t)\right),$$

where $d^{\perp}(t)$ is the vector perpendicular to both the spine's tangent and the symmetry director (Figure 3.19). Unfortunately, if the curvature of $\gamma(t)$ is high compared to the cross-section scale, this definition will cause the surface to pinch and self-intersect. To eliminate this problem, we use integral curves along a *smoothed* distance field [PKZ04] to offset the cross-sections, instead of straight lines perpendicular to $\gamma(t)$. To obtain the 3d position for a cross-section point $\alpha_t(u)$, we move along the (curved) integral line of the smoothed distance function starting at point $\gamma(t)$ with an initial direction along $\alpha_t(u)^1 d(t) + \alpha_t(u)d^{\perp}(t)$, by distance $|\alpha_t(u)|$ (this computation is described in more detail in [PKZ04]). Conceptually, this process corresponds to bending and compressing the planes of cross-sections to avoid self-intersections.

The user's out-of-view-plane tilted cross-sections imply sparse tangent constraints on $\gamma(t)$. Given these constraints and the user-specified 2D spine curve, we find smoothly varying z coordinates for γ, minimizing $\int(\Delta\gamma(t))^2 dt$ with respect to the z coordinate of γ. Similarly, we find a smoothly varying scale function $s(t)$ given the user's sparse scale constraints, as well as smoothly varying symmetry directors $d(t)$ and cross-section curves $\alpha_t(u)$.

Finally, we attach end-caps in a tangent continuous manner. Let e_0 be the end-cap protrusion at $\gamma(0)$. We extend the surface by sweeping the cross-section at the end in the direction tangent to (without loss of generality) $\gamma(0)$; we scale

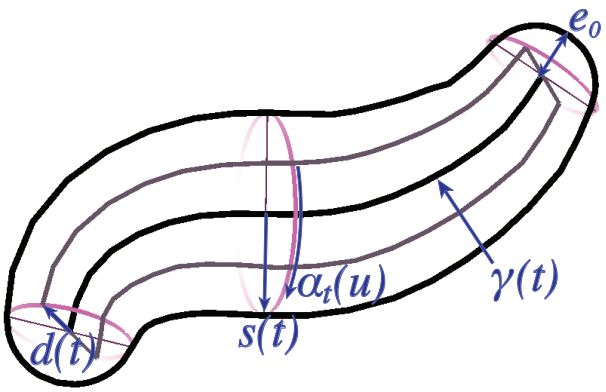


Figure 3.19: Notation introduced in Section 3.5.1.

these cross-sections by the y value of a cubic Bézier curve whose tangent at its start is parallel to $(1, s'(0))$ and whose y value falls to 0 at $x = e_0$. The four control points are $(0, s(0)), (\frac{e_0}{2}, s(0) + \frac{s'(0)e_0}{2}), (e_0, \frac{s(0)+s'(0)e_0}{2}), (e_0, 0)$.

Spine deformation is implemented using Laplacian curve editing [SCOL+04]. The spine is either deformed directly or, more commonly, during cross-section scale manipulation, when it is constrained to pass through the center of the cross-section. (During cross-section scale manipulation, the one of the cross section's silhouette points remains fixed while the other follows the mouse.)

Ellipsoids are implied by the 2D ellipse and the out-of-view-plane tilt of the circular cross-section. (The ellipsoid's two shorter axes have the same length as the ellipse's short axis and span the tilted cross-section's plane; the long axis is perpendicular to the tilted cross-section with length such that the ellipsoid's silhouette projects to the 2D ellipse.)

3.5.2 Annotations

Connection curves are user-drawn 2D curves $\beta(t)$ ($t \in [0,1]$) whose projection onto the surfaces of two shapes we wish to be identical; in other words, we wish for the shapes to intersect each other along the projection of β. Let P_f and Q_f be functions which project a point in the image plane to a point on the front of the first and second shapes, respectively. Allowing for translation of the surfaces in z (our depth coordinate), we minimize

$$\int_0^1 \left[P_f\left(\beta(t)\right)_z - Q_f\left(\beta(t)\right)_z + c\right]^2 dt$$

with respect to c, a relative offset between the two shapes. We assume that Q has already been fixed in z and translate P by c in the z direction. When dragging the intersection curve, $c = Q_f(a)_z - P_f(a)_z$, where a is the image-plane point under the mouse. A depth-first search is used to ensure that we only translate each shape in depth once. We do not support multiply-connected graphs of primitives.

Mirror annotations. To implement mirror annotations, we duplicate and then reflect the 3D shape of a primitive n (and those of its attached primitives) across the symmetry plane of another primitive m. In case of a symmetry sheet, we find the closest point on the sheet to n's attachment origin and use the tangent plane there as the symmetry plane. A primitive's *attachment origin* is either its center or, in the case of a generalized cylinder, one of the 3D endpoints of its spine, whichever is closest (in 2D) to the connection curve attaching the primitive to m.

Alignment annotations. To implement alignment annotations, described in Section 3.4.1, we find the smallest satisfying translations in a least squares sense. In case the alignment is with respect to a symmetry sheet, we simply average the tangent planes for each to-be-aligned primitive's attachment origin, just as with mirror annotations.

Same scale, tilt, and length annotations. The congruency annotations are all implemented similarly. A same-scale annotation marks a set of cross-sections as having the same scale. To satisfy this constraint, every cross-section in the set is simply assigned the average scale of the entire set. Same tilt angle annotations are implemented similarly, averaging angles. Same length annotations are also implemented similarly; changing the length of a 3D spine curve or ellipsoid axis simply scales the 2D spine curve or corresponding 2D ellipse axis. However, when scaling a primitive, attached primitives are also translated so as to remain connected.

3.6 Results

We have used our interface to generate models from a variety of found 2D images. These are shown in Figures 3.20 and 3.21. A typical modeling session lasted less than ten minutes (longer for the more complex models shown in Figure 3.21). Results created by user testers are presented in Section 3.7. Source material included drawings from a children's book, user-created 2D drawings, concept artwork, and cartoons.

Models creating with our system contain structural information useful in a

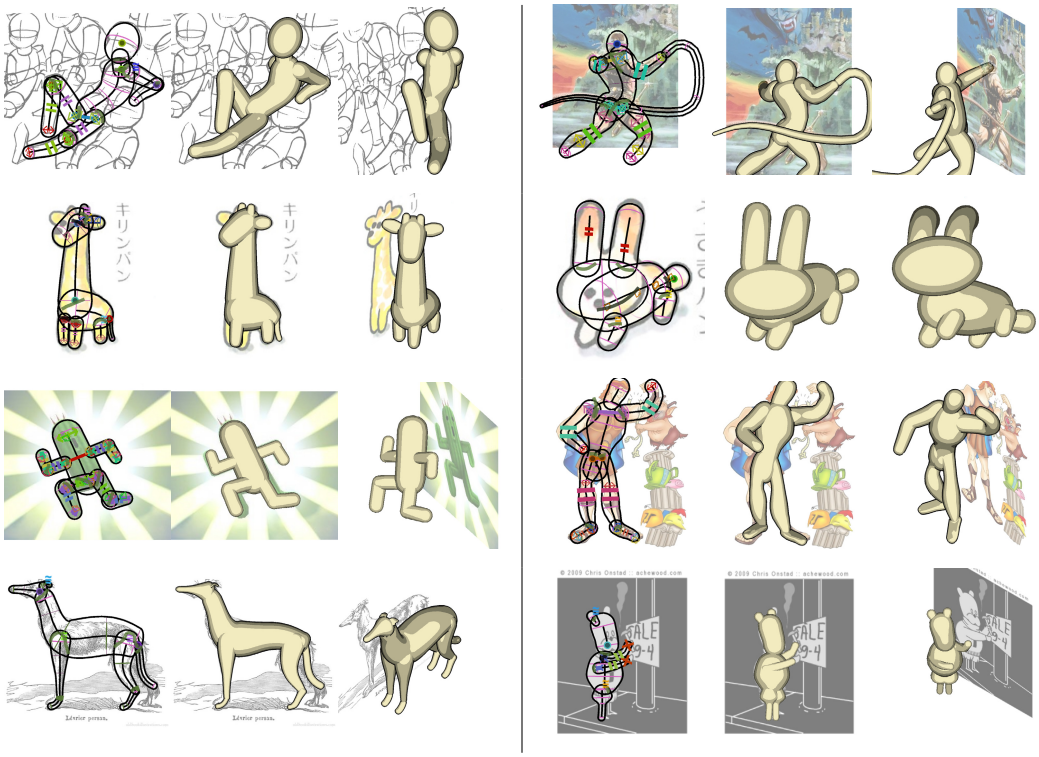


Figure 3.20: Models created using our interface. Each took less than 10 minutes to create. Primitives and annotations are shown on the left, and the resulting model from the same angle and a different angle is shown in the middle and on the right. On the left, source images are [Vil97], © Satoshi Kako, and © Square Enix. On the right, source images are © Konami, © Satoshi Kako, © The Walt Disney Company, and © Chris Onstad.

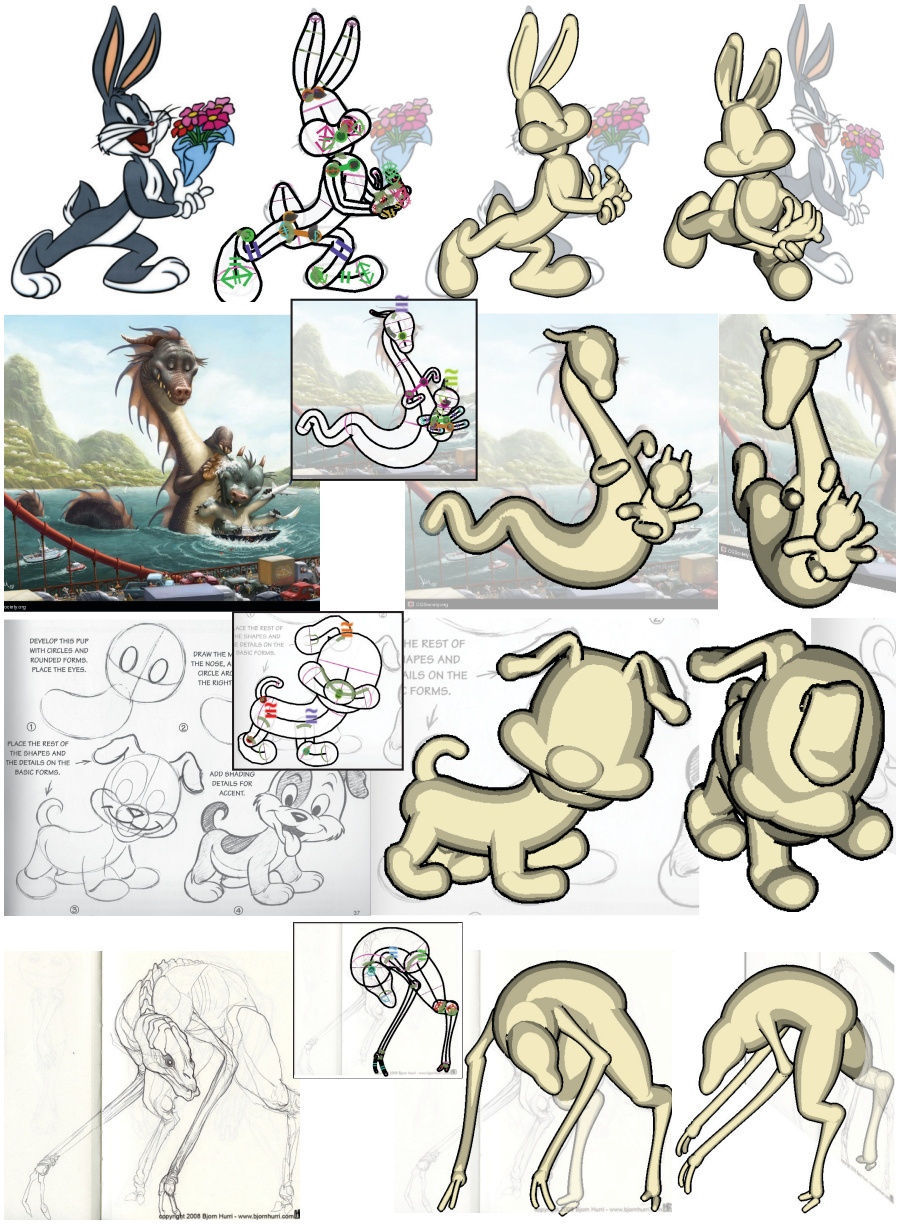


Figure 3.21: More complex models created using our interface. In each row: the guide image; the primitives and annotations; the resulting model from the same and a different angle. The guide images in each row are © Warner Brothers, © Kei Acedera, [Bla94], and © Björn Hurri.

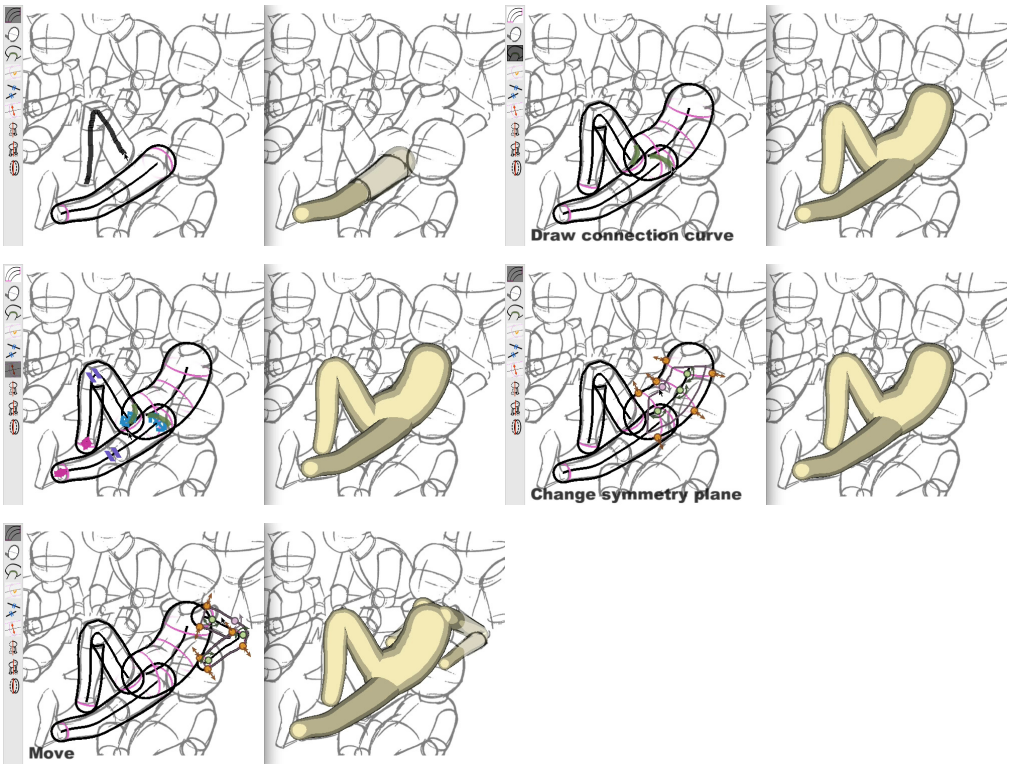


Figure 3.22: Several frames from a modeling session: drawing the second leg; attaching a leg to the body with a connection curve annotation; marking same length and scale annotations; adjusting the symmetry sheet; adjusting a mirrored arm. The guide image is from [Vil97].

variety of applications. This information includes spines, the location of joints, and a mesh whose parts are segmented into distinct connected components. Furthermore, annotations encode symmetries in the model and lengths which should remain equal when, e.g. deforming the model. One use for symmetry information is automatic propagation of mesh refinement performed in a surface editing tool such as a 3D sculpting tool. The spines and joints can be used for animation.

3.7 Evaluation

We have found it easy to create a variety of models consisting of rotund and tubular areas, and, through the use of non-circular cross sections, flat shapes such as ears. When tilting cross section, we find it necessary to verify the results in the 3D view. This is consistent with [KvDK92], in which humans are shown to make (consistent) errors when estimating the magnitude of surface slopes. (It is perhaps worth noting that tilting circular cross sections is reminiscent of the gauge figures adjusted by subjects in the experiment described in [KvDK92].)

We performed a small, informal user study, consisting of six people, two of whom had 2D artistic experience. None of the subjects had significant 3D modeling experience, but four were familiar with 3D manipulation concepts. After a 15 minute training session, users were able to create their own models, several of which are shown in Figure 3.23. Users unfamiliar with 3D manipulation concepts were significantly slower, however, but reported feeling more comfortable over time.

We also performed a comparison study, consisting of five people, between our system and FiberMesh [NISA07]. In our experiment, subjects were given 15 min-

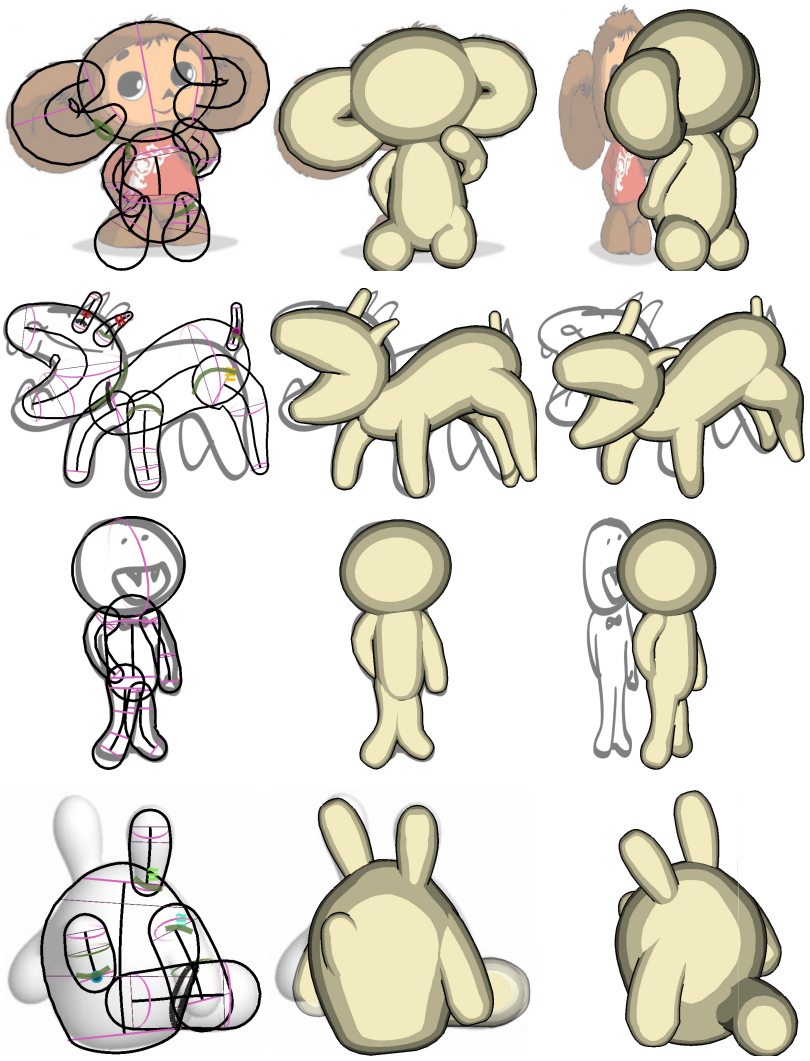


Figure 3.23: Models created by first-time users. The primitives and annotations are shown on the left, and the resulting model from the same angle and a different angle is shown in the middle and on the right. From top to bottom: a cartoon character (20 minutes); a monster (10 minutes); a vampire (10 minutes); a cartoon character from [NISA07]. The monster and vampire images were drawn by the users.

utes to create a 3D model from the same 2D illustration in each system. All subjects received the same image. Half were randomly assigned to use our system first. Before using each system, subjects were given 15 minutes of training, which consisted of a brief video and hands-on, guided experimentation. Most subjects reported some casual 2D artistic experience, and none had 3D modeling experience. The given illustration and example results are shown in Figure 3.24. When asked to state an overall preference, four subjects preferred our system and one stated no overall preference. One subject remarked that being able to "draw right on the image" made the task "so much easier." We asked an independent person to evaluate subjects' results. He chose, for each subject, which model was higher quality and which model better matched the guide image. In all cases, the evaluator preferred the model created in our system.

Our informal and comparison studies found that, overall, users reported satisfaction at creating 3D models from their sketches or illustrations, and comfort with our primitives which resembled shapes in 2D drawing programs. Users reported understanding and liking tilting cross sections, but noted that they verified their manipulations in the 3D results window. Connection curves were similarly received, but several users desired more direct depth control. All annotations were used during testing, but not uniformly. Some users preferred to adjust primitives manually until they achieved the desired result. Most users appreciated the symmetry-related annotations (mirroring and alignment) for their efficiency (relative to duplicating effort and manual tweaking).

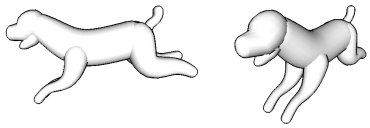
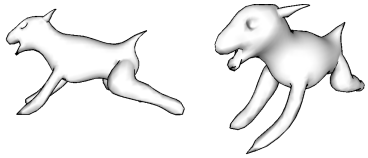
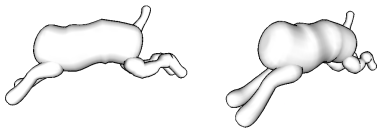
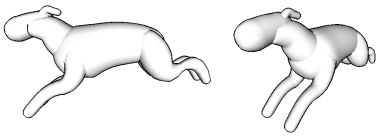
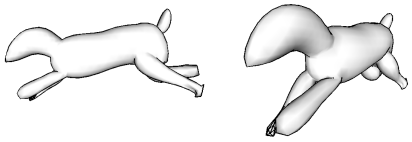
	Our System	[NISA07]
Subject 1		
Subject 2		
Subject 3		

Figure 3.24: Our comparison study. The given 2D illustration (top) along with the 3D model created using our system (left column) and FiberMesh [NISA07] (right column). Each row corresponds to a single subject.

3.8 Conclusion, Limitations, and Future Work

We have presented an interface for 3D modeling based on the idea of annotating an existing 2D sketch. Although we have shown that many 2D drawings have no consistent 3D representation (Section 3.3), with a small degree of training, even novices are able to create 3D models from 2D drawings. Our interface eliminates the need for constant rotation inherent to many previous sketch-based modeling tools, which precludes users from matching their input to a guide image. Our primitives and annotations are structured and persistent, and provide semantic information about the output models useful in a variety of applications. An additional benefit of our approach is that the entire modeling process is visible in a single, static 2D image, which makes it easy to explain and learn how to create a given model.

While we are able to create a variety of models using our existing primitives and annotations, our approach is not without its limitations. First, it is not possible to do away with a 3D view altogether. The 3D view is necessary for verification. Second, our interface has operations which feel like modeling rather than sketching, and it takes some training to learn the 2D-to-3D mapping. Third, our interface suffers from visual clutter. We plan to explore solutions used for 2D drawing programs (e.g. [RRC+06]). Fourth, we provide no way to color or texture 3D models, even though drawings may have been colored.

We see our current interface as an initial step in the direction of structured, two-dimensional 3D modeling. In the future, we plan to create primitives and

annotations for precise CAD modeling, for other commonly used free-hand drawing primitives, for relatively flat shapes, and for additional geometric relationships. We also foresee annotations for specific applications, such as annotating primitives' motions for animation and rigidity for deformation. We would like to automate (or simplify) tracing curves in an image, as in [TBSR04]. We would also like to support adding geometry to an existing 3D model, by sketching over it on a 2D overlay plane, and then placing primitives and annotating the overlaid sketch. Finally, support for oblique projections would allow for alternative, possibly less surprising 3D interpretations of user input.

Conclusion

We have presented several systems for creating and editing 3D models, each designed to leverage users' 2D skills in a different way. In Chapter 1, we presented a system that allows users to easily texture 3D models with casual photographs or even naturally drawn 2D images. This system hides the parameterization as an implementation detail, allowing the user to concentrate on creating a 2D texture in a convenient manner. Applying the texture to the 3D surface is achieved by sliding the texture along the surface and deforming it with constraints. In this way, we have removed the need for users to invert an unintuitive 3D-to-2D mapping, the parameterization, and replaced it with direct manipulation of the texture on the surface—a simple linear, WYSIWYG projection.

In Chapter 2, we presented a system for adding small-to-medium scale surface features by shading over an image of a 3D model as if with a pencil. Although there are a variety of existing techniques for surface editing, none allow users to "draw what they want to see," the ideal workflow for users who prefer to draw 2D shaded images. Our system provides a fast, controllable solution to a special case of the shape-from-shading problem.

In Chapter 3, we presented a novel interface for creating a 3D model from a 2D drawing by leveraging users' interpretive skills. With our system, users are free to create 2D drawings in their preferred tool, which may be pencil and paper, or use found images. In our system, users import a drawing and "describe" the 3D model it depicts by placing primitives which specify local shape information and

annotations which specify global semantic information. Together, the primitives and annotations imply a 3D model.

Taken together, these tools provide users with a primarily 2D workflow for creating detailed, textured models from scratch. First, the user creates a drawing of a model. Next, the user loads the drawing into the system presented in Chapter 3. The user describes the 3D shape and congruencies in the model, and obtains an undetailed 3D model as output. Then, the user loads the resulting undetailed 3D model into the system presented in Chapter 2 and shades on top of the model to obtain a detailed one. Finally, the user chooses several photographs or draws 2D, color images of the model from several points of view and, using the system presented in Chapter 1, places and deforms the images to match features of the 3D model.

The tools presented in this thesis are not intended to be used to the exclusion of other, useful tools. Rather, they are intended to provide alternative workflows which are advantageous in certain scenarios. A user skilled at sculpting in the physical world may prefer a 3D sculpting tool to the system presented in Chapter 2. (Strictly speaking, these two approaches are complementary; the class of edits easy to do with one is difficult with the other, and vice versa.) Similarly, a user skilled at painting sculptures in the physical world may prefer a 3D painting tool to the system presented in Chapter 1. Yet, given pre-existing textures (such as photographs or textures from another 3D model), even a sculpture painter will appreciate being able to "adjust" (place and deform) an existing texture rather than painting a new one from scratch.

The tools presented in this thesis are also not intended to be the last word in their respective areas. For the system presented in Chapter 1, in addition to technical improvements, we foresee a system for automatically aligning and unprojecting photographs onto 3D models of humans. For the system presented in Chapter 3, the process could be greatly sped by providing the user with higher-level templates composed of bundles of primitives and annotations, such as templates for quadrupeds and humans. Rather, the tools presented in this thesis are intended to be useful, to complement existing tools, and to inspire future work.

A
Sherman-Woodbury-Morrison
formula

We follow Hager [Hag89] to update the inverse of an $n \times n$ matrix A with a low-rank modification of the form UV, where U is $n \times m$, V is $m \times n$, and m is much smaller than n. The Sherman-Morrison-Woodbury formula

$$B^{-1} = [A - UV]^{-1} = A^{-1} + A^{-1}U(I - VA^{-1}U)^{-1}VA^{-1}$$

leads to the following algorithm for solving a system of equations $Bx = b$:

1. Solve $Ay = b$ for y.
2. Compute the $n \times m$ matrix $W = A^{-1}U$ by solving m linear systems $Aw_i = u_i$, where w_i and u_i are columns of W and U.
3. Form a small $m \times m$ matrix $C = I - VW$ and solve $Cz = Vy$ for z.
4. Set $x = y + Wz$.

This algorithm yields a considerable speedup if A can be factorized to a form which makes it possible to solve the system $Ay = b$ for different right-hand sides as needed in step 2. This is typical for direct solvers; e.g. we use the PARDISO solver [SG04, SG06, KK98], which prefactors the matrix A as LDL^T, where L is a lower triangular and D is a diagonal matrix. Solving a system with A represented in this form is an order of magnitude faster than the cost of factorization.

Bibliography

[ABL95] M. Agrawala, A. C. Beers, and M. Levoy. 3D painting on scanned surfaces. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 145–150, 1995.

[ACSD+03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics (TOG)*, 22(3):485–493, 2003.

[Aut08] Autodesk. Mudbox, 2008. http://www.mudbox3d.com.

[Aut09] Autodesk. Maya, 2009. http://www.autodesk.com/maya.

[AZM00] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 125–136, London, UK, 2000. Springer-Verlag.

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

[BH00] R. Balakrishnan and K. Hinckley. Symmetric bimanual interaction. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40, 2000.

[BK99] R. Balakrishnan and G. Kurtenbach. Exploring bimanual camera control and object manipulation in 3D graphics interfaces. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 56–63, 1999.

[BK04] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (TOG)*, 23(3):630–634, 2004.

[Bla94] Preston Blair. *Cartoon Animation*. Walter Foster, Laguna Hills, California, 1994.

[BN92] T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 35–42, 1992.

[BS08] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

[CCP+04] V. Cheutet, C. E. Catalano, J. P. Pernot, B. Falcidieno, F. Giannini, and C. Leon. 3D sketching with fully free form deformation features (δ-f4) for aesthetic design. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM)*, pages 9–18, sep 2004.

BIBLIOGRAPHY

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 25(4):257–266, 1991.

[CH04] N. A. Carr and J. C. Hart. Painting detail. *ACM Transactions on Graphics*, 23(3):845–852, 2004.

[CKX+08] Xuejin Chen, Sing Bing Kang, Ying-Qing Xu, Julie Dorsey, and Heung-Yeung Shum. Sketching reality: Realistic interpretation of architectural designs. *ACM Transactions on Graphics (TOG)*, 27(2):1–15, 2008.

[CS07] Frederic Cordier and Hyewon Seo. Free-form sketching of self-occluding objects. *IEEE Computer Graphics and Applications*, 27(1):50–59, 2007.

[CSSJ05] Joseph Jacob Cherlin, Faramarz Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Sketch-based modeling with few strokes. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 137–145, New York, NY, USA, 2005. ACM.

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

[DGPR02] D. G. DeBry, J. Gibbs, D. D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics*, 21(3):763–768, 2002.

[DL01] P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, 2001.

[DMA02] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21(3):209–218, 2002.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1996. ACM.

[EHBE97] Lynn Eggli, Ching-Yao Hsu, Beat D. Bruderlin, and Gershon Elber. Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29(2):101–112, February 1997.

[Flo97] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[GGH02] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

BIBLIOGRAPHY

[Gui87] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinetic chain as a model. *The Journal of Motor Behavior*, 19(4):486–517, 1987.

[Hag89] W. W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.

[Han05] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, 2005.

[HH90] P. Hanrahan and P. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 215–223, 1990.

[HPGK94] K. Hinckley, R. Pausch, J. C. Goble, and N. F. Kassell. Passive real-world interface props for neurosurgical visualization. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 452–458, 1994.

[IC01] T. Igarashi and D. Cosgrove. Adaptive unwrapping for interactive texture painting. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 209–216, 2001.

[IMH05] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24(3):1134–1141, 2005.

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

[JP99] D. L. James and D. K. Pai. ArtDefo: accurate real time deformable objects. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 65–72, 1999.

[JWYG04] M. Jin, Y. Wang, S.-T. Yau, and X. Gu. Optimal global conformal surface parameterization. In *15th IEEE Visualization 2004 (VIS'04)*, pages 267–274, 2004.

[KDS06] Levent Burak Kara, Chris M. D'Eramo, and Kenji Shimada. Pen-based styling design of 3D geometry using concept sketches and template models. In *Proceedings of the ACM Symposium on Solid and Physical Modeling (SPM)*, pages 149–160, 2006.

[KFBB97] G. Kurtenbach, G. Fitzmaurice, T. Baudel, and B. Buxton. The design of a GUI paradigm based on tablets, two-hands, and transparency. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 35–42, 1997.

BIBLIOGRAPHY

[KGB05] B. Kerautret, X. Granier, and A. Braquelaire. Intuitive shape modeling by shading design. In *Smart Graphics: 5th International Symposium*, 2005.

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

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[KLS03] A. Khodakovsky, N. Litke, and P. Schröder. Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics*, 22(3):350–357, 2003.

[KS04] V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics*, 23(3):861–869, 2004.

[KSG03] V. Kraevoy, A. Sheffer, and C. Gotsman. Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics*, 22(3):326–333, 2003.

[KvDK92] Jan J. Koenderink, Andrea J. van Doorn, and Astrid M. L. Kappers. Surface perception in pictures. *Perception & Psychophysics*, 52(5):487–496, 1992.

[Lév01] B. Lévy. Constrained texture mapping for polygonal meshes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 417–424, 2001.

[LF04] J. Lawrence and T. Funkhouser. A painting interface for interactive surface deformations. *Graphical Models*, 66(6):418–438, 2004.

[LKG+03] 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 Transactions on Graphics*, 22(3):663–668, 2003.

[Lor43] E. Loran. *Cezanne's Composition*. University of California Press, 1943.

[LPRM02] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

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

[LTD05] H. Lee, Y. Tong, and M. Desbrun. Geodesics-based one-to-one parameterization of 3D triangle meshes. *IEEE Multimedia*, 12(1):27–33, 2005.

[MDSB03] M. Meyer, M Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Vi-*

sualization and Mathematics III, pages 35–57. Springer-Verlag, Heidelberg, 2003.

[Mic03] Microsoft. Office, 2003. http://office.microsoft.com.

[MS92] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 167–176, 1992.

[MYV93] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 27–34, 1993.

[NISA07] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (TOG)*, 26(3):41, 2007.

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

[NWT07] Heung-Sun Ng, Tai-Pang Wu, and Chi-Keung Tang. Surface-from-gradients with incomplete data for single view modeling. In *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[OSCSJ08] L. Olsen, F. F. Samavati, M. Costa Sousa, and J. Jorge. A taxonomy of modeling techniques using sketch-based interfaces. In *Eurographics State of the Art Reports*, April 2008.

[PB00] D. Piponi and G. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 471–478, 2000.

[PFH00] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 465–470, 2000.

[PKZ04] Jianbo Peng, Daniel Kristjansson, and Denis Zorin. Interactive modeling of topologically complex geometric detail. *ACM Transactions on Graphics (TOG)*, 23(3):635–643, August 2004.

[Pra04] E. Prados. *Application of the theory of the viscosity solutions to the Shape From Shading problem.* PhD thesis, University of Nice-Sophia Antipolis, 2004.

[PSS01] E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 179–184, 2001.

[Rek02] J. Rekimoto. SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. In *CHI '02: Proceedings of the SIGCHI*

conference on Human factors in computing systems, pages 113–120, 2002.

[RGB+03] H. Rushmeier, J. Gomes, L. Balmelli, F. Bernardini, and G. Taubin. Image-based object editing. *Proceedings of the Fourth International Conference on 3D Digital Imaging and Modeling (3DIM)*, pages 20–28, 2003.

[RRC+06] Gonzalo Ramos, George Robertson, Mary Czerwinski, Desney Tan, Patrick Baudisch, Ken Hinckley, and Maneesh Agrawala. Tumble! Splat! helping users access and manipulate occluded content in 2D drawings. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 428–435, 2006.

[SAPH04] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. *ACM Transactions on Graphics*, 23(3):870–877, 2004.

[SCOL+04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP)*, pages 175–184, 2004.

[SCOT03] Olga Sorkine, Daniel Cohen-Or, and Sivan Toledo. High-pass quantization for mesh encoding. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP)*, pages 42–51, 2003.

[SdS01] A. Sheffer and A. S. E. de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17(3):326–337, 2001.

[SF98] Karan Singh and Eugene Fiume. Wires: a geometric deformation technique. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 405–414, 1998.

[SG04] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.

[SG06] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.

[She03] A. Sheffer. Skinning 3D meshes. *Graphical Models*, 65(5):274–285, 2003.

[SIJ+07] Ryan Schmidt, Tobias Isenberg, Pauline Jepp, Karan Singh, and Brian Wyvill. Sketching, scaffolding, and inking: A visual history for interactive 3D modeling. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 23–32, 2007.

BIBLIOGRAPHY

[SLMB05] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 24(2):311–330, 2005.

[Sof08] Smith Micro Software. Poser, 2008. http://my.smithmicro.com/mac/poser/.

[Sor08] Olga Sorkine. Personal communication. 2008.

[SSB08] Ryan Schmidt, Karan Singh, and Ravin Balakrishnan. Sketching and composing widgets for 3D manipulation. *Computer Graphics Forum*, 27(2):301–310, 2008.

[SSGH01] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 409–416, 2001.

[SSS+08] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3D architectural modeling from unordered photo collections. *ACM Trans. Graph.*, 27(5):1–10, 2008.

[SWGH03] P. V. Sander, Z. J. Wood, S. J. Gortler, and H. Hoppe. Multi-chart geometry images. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 146–155, 2003.

[SWZ04] Scott Schaefer, Joe D. Warren, and Denis Zorin. Lofting curve networks using subdivision surfaces. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP)*, pages 103–114, 2004.

[TBSR04] Steve Tsang, Ravin Balakrishnan, Karan Singh, and Abhishek Ranjan. A suggestive interface for image guided 3D sketching. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 591–598, 2004.

[TS08] Thorsten Thormählen and Hans-Peter Seidel. 3D-modeling by ortho-image generation from image sequences. *ACM Transactions on Graphics (TOG)*, 27(3):1–5, 2008.

[VC07] Peter Varley and Pedro Company. Sketch input of 3D models: Current directions. In *VISAPP 2007: 2nd International Conference on Computer Vision Theory and Applications*, pages 85–91, Barcelona, Spain, March 2007.

[vdHDT+07] Anton van den Hengel, Anthony Dick, Thorsten Thormählen, Ben Ward, and Philip H. S. Torr. VideoTrace: rapid interactive scene modelling from video. *ACM Transactions on Graphics (TOG)*, 26(3):86, 2007.

[Vil97] Glen Vilppu. *Vilppu Drawing Manual*. Vilppu Studio, Acton, California, 1997.

BIBLIOGRAPHY

[vO96] C. W. A. M. van Overveld. Painting gradients: free-form surface design using shading patterns. In *Proceedings of the conference on Graphics Interface (GI)*, pages 151–158, 1996.

[WB03] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 193–202, 2003.

[WBH+07] Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. Discrete quadratic curvature energies. *Computer Aided Geometric Design*, 24(8-9):499–518, 2007.

[Wil04] A. D. Wilson. TouchLight: an imaging touch screen and display for gesture-based interaction. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76, 2004.

[WTBS07] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. ShapePalettes: interactive normal transfer via sketching. *ACM Transactions on Graphics (TOG)*, 26(3):44, 2007.

[WW92] William Welch and Andrew Witkin. Variational surface modeling. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 157–166, 1992.

[YBS04] S. Yoshizawa, A. Belyaev, and H. P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *SMI '04: Proceedings of the Shape Modeling International 2004 (SMI'04)*, pages 200–208, 2004.

[YBS05] S. Yoshizawa, A. Belyaev, and H. P. Seidel. A moving mesh approach to stretch-minimizing mesh parameterization. *International Journal of Shape Modeling*, 11(1):25–42, 2005.

[YLHS05] H. Yamauchi, H. P. A. Lensch, J. Haber, and H. P. Seidel. Textures revisited. *Visual Computer*, 21(4):217–241, 2005.

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

[ZDPSS01] Li Zhang, Guillaume Dugas-Phocion, Jean-Sebastien Samson, and Steven M. Seitz. Single view modeling of free-form scenes. *CVPR*, 01:990, 2001.

[ZFS97] R. Zeleznik, A. Forsberg, and P. Strauss. Two pointer input for 3D interaction. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 115–120, 1997.

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: an interface for sketching 3d scenes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 163–170, 1996.

BIBLIOGRAPHY

[ZMQS05] Gang Zeng, Yasuyuki Matsushita, Long Quan, and Heung-Yeung Shum. Interactive shape from shading. *CVPR*, 1:343–350, 2005.

[ZMT05] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24(1):1–27, 2005.

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

[ZRS05] R. Zayer, C. Rossl, and H.-P. Seidel. Setting the boundary free: A composite approach to surface parameterization. In *Symposium on Geometry Processing*, pages 91–100, 2005.