INTERACTIVE SHAPE DESIGN

Synthesis Lectures on Computer Graphics and Animation

Interactive Shape Design
Marie-Paule Cani, Takeo Igarashi and Geoff Wyvill
2008

Real-Time Massive Model Rendering
Sung-eui Yoon, Enrico Gobbetti, David Kasik, and Dinesh Manocha
2008

Virtual Crowds: Methods, Simulation and Control
Nuria Pelechano, Jan. Allbeck, and Norman I. Badler
2008

High Dynamic Range Video
Karol Myszkowski, Rafał Mantiuk, and Grzegorz Krawczyk
2008

GPU-Based Techniques For Global Illumination Effects
László Szirmay-Kalos, László Szécsi and Mateu Sbert
2008

High Dynamic Range Imaging Reconstruction
Asla Sa, Paulo Carvalho, and Luiz Velho IMPA, Brazil
2007

High Fidelity Haptic Rendering
Miguel A. Otaduy, Ming C. Lin
2006

A Blossoming Development of Splines
Stephen Mann
2006

Interactive Shape Design

Marie-Paule Cani, Takeo Igarashi, and Geoff Wyvill

www.morganclaypool.com

ISBN: 9781598292107 paperback
ISBN: 9781598292114 ebook

DOI 10.2200/S00122ED1V01Y200806CGR006

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON COMPUTER GRAPHICS AND ANIMATION #6

Lecture #6

Series Editor: Brian A. Barsky, University of California, Berkeley

Series ISSN: 1933-8996 print
Series ISSN: 1933-9003 electronic

INTERACTIVE SHAPE DESIGN

Marie-Paule Cani
Grenoble Institute of Technology
France

Takeo Igarashi
University of Tokyo
Japan

Geoff Wyvill
University of Otago
New Zealand

SYNTHESIS LECTURES ON COMPUTER GRAPHICS AND ANIMATION #6

M&C MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Providing an intuitive modeling system, which would enable us to communicate about any free-form shape we have in mind at least as quickly as with real-world tools, is one of the main challenges of digital shape design. The user should ideally be able to create, deform, and progressively add details to a shape, without being aware of the underlying mathematical representation nor being tied by any constraint on the geometrical or topological nature of the model.

This book presents the field of interactive shape design from this perspective. Since interactively creating a shape builds on the humans ability of modeling by gesture, we note that the recent advances in interactive shape design can be classified as those that rely on sculpting as opposed to sketching metaphors. Our synthetic presentation of these strategies enables us to compare the different families of solutions, discuss open issues, and identify directions for future research.

KEYWORDS

Geometric modeling, Digital shape editing, Geometric design, Surface representations, Sculpting, Sketching

Contents

List of figures

Acknowledgments

A work of this kind depends on the cooperation of many people, students, colleagues, and researchers who have published with us. We would like to specially thank Marc Alexa, Anca Alexe, Alexis Angelidis, Grégoire Aujay, Loïc Barthe, Renaud Blanch, David Bourguignon, Gilles Debunne, Mathieu Desbrun, Guillaume Dewaele, Eric Ferley, Jean-Dominique Gascuel, Laurent Grisoni, John F. Hughes, Shin HyoJong, Yuki Igarashi, Takashi Ijiri, Scott King, Paul Kry, Satoshi Matsuoka, Tomer Moscovich, Andrew Nielen, Makoto Okabe, Shigeru Owada, Adeline Pihuit, Olga Sorkine, Kenshi Takayama, Hidehiko Tanaka, Emmanuel Turquin, Nayuko Watanabe, Jamie Wither, Brian Wyvill, and the anonymous reviewers from Morgan & Claypool who made so many helpful suggestions.

CHAPTER 1
Introduction

1.1 INTRODUCTION

Being able to communicate about a shape is important for both professional purposes and self-expression. However, it is not as straightforward for humans as expressing a new idea of motion or of tune: while we can use our body to mimic motion or hum out a melody, we need the help of an external device to express our idea of a shape.

Sculpting and sketching have been the main techniques used by humans to communicate about shapes. In the first case, the shape is directly made out of dough, clay, wood, or soft rock. Although most people had a chance to play with dough as children, only skilled artists are able to create the shape they want. Even then they are constrained by the limitations and properties of the material they have chosen to work with. Clay, for example, may dry and crack, deform under gravity, or limit the size of the smallest representable feature.

Carrying dough or clay would be unusual in our culture, but we are likely to find paper and pen whenever needed, so most people are more familiar with sketching. However, a sketch will not directly communicate the shape we have in mind. We need to draw several views of it and many if it is complicated or of an unfamiliar type. Such drawings take a great deal of skill, and this restricts the medium to trained and experienced designers. We still lack a simple medium for describing 3D shapes.

Can computers be turned into usable devices to communicate imagined 3D shapes? Can digital design be enhanced to provide the public with user-friendly free-form modeling tools, as well as enabling skilled artists to express any complex shape they can think of? These are the questions we are addressing. Centering our analysis on the different ways to control shapes gives us a particular and possibly unusual viewpoint of the field of digital shape modeling. It also helps us to investigate why, despite more than 30 years of effort toward the development of new surface representations and of algorithms to edit them, most artists first sketch or sculpt in the real world before using a computer.

1.2 DESIGNING SHAPES IN THE REAL WORLD

Let us start with a quick analysis of the way free-form shapes are created in the real world. This will give us some motivation for developing digital modeling techniques and some hints about the users' expectations, when they try to use such systems.

1.2.1 Sculpting

Whatever the material and the approach used, sculpting is characterized by the progressive creation of a physical 3D model, which the user can manipulate and watch from different viewpoints at any stage of the process. In this book, our use of the word *sculpting* will mainly refer to modeling with clay or dough, as depicted in Fig. 1.1 (left). Firstly, this technique is less technical than extracting a shape from hard material, so it is more familiar to, the public. Secondly, it offers more choice of interaction modes. The user does not only use carving tools but

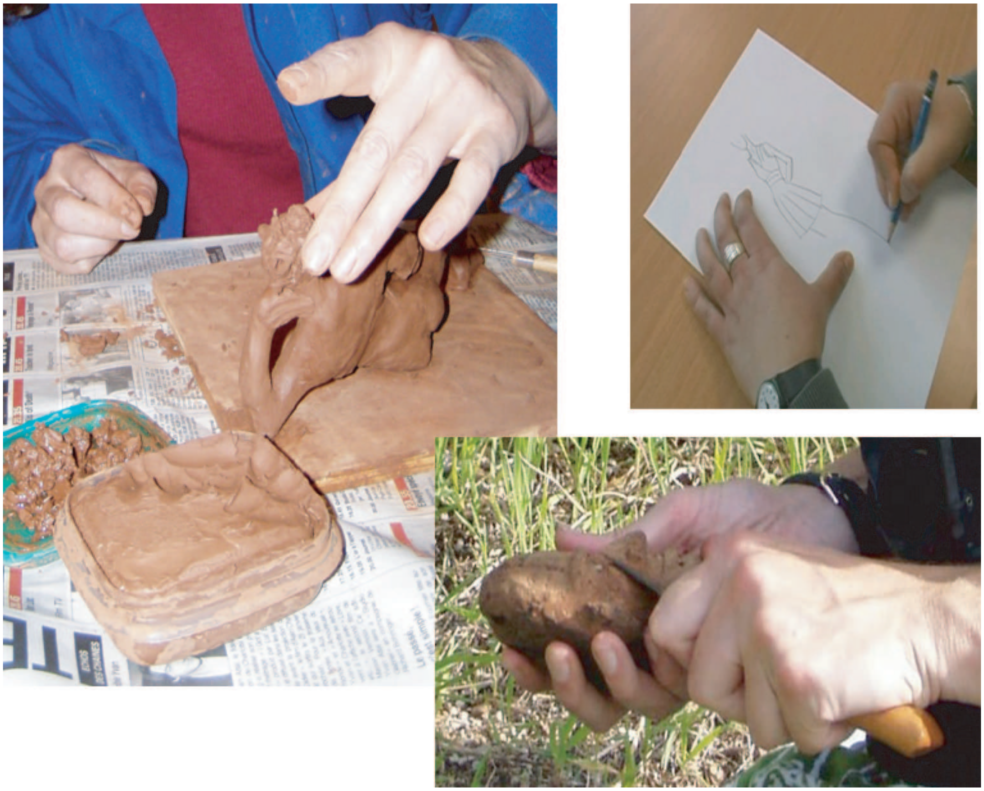

FIGURE 1.1: Shape design in the real world: these examples of clay modeling, wood carving, and sketching show the importance of the artist's hand gestures in the design process.

can also extend the model by adding new chunks of clay and apply local or global deformations at any stage. For instance, a human figure may be sculpted in a symmetrical resting pose to give it the right proportions before being bent or twisted into a livelier posture. Note that applying such deformations is distinct from the action of adding or removing material. Each deformation preserves the model's volume. Lastly, we can, to some extent, change the nature of the material we are using: according to need, we can add some water to make the clay more fluid and the deformations more local or wait till the clay dries a little before adjusting the surface on a larger scale.

In the real world, sculpting techniques are limited by the constraints of the physical medium used. This restricts their practical use in everyday life. Let us look at some of the motivation for using an ideal *virtual clay* as opposed to a real one: Virtual clay would not be affected by unwanted drying or cracking, enabling the user to pause at any time without worrying about the material changing. The material could be made to mutate back and forth between softer and drier states, enabling more or less local deformation. Gravity could be switched off, so we could model thin shapes without having them collapse under their own weight. The artist would be able to work from many directions, turning the work as easily as in the real world and at several different scales, making the size of tools arbitrary with respect to the model's size and thus simplifying the production of fine details as well as of global features. Repetitive operations such as smoothing could be made automatic. Lastly, virtual modeling would allow copy–paste and undo–redo as well as more clay-specific operations such as temporarily removing a part of the model to ease the editing of hard-to-reach areas.

However, let us keep in mind the main features one takes for granted when using clay: direct interaction with the 3D model (ideally through both vision and touch), being able to add and remove material (which includes digging holes through a shape or reconnecting parts), applying constant volume deformations, and tuning the locality of interactions. These will be among the user's expectations, when he or she is offered a sculpting metaphor. This gives the first idea of the difficulty of providing an effective virtual sculpting system. We will come back to the advances made toward this goal in Chapter 2.

1.2.2 Sketching

In real life, the media required for sketching is very common, and this technique is somewhat more familiar to the public than sculpting. One needs only a support surface (generally paper), a pencil, and, ideally, an eraser to enable undo–redo operations. Unlike sculpting, sketching is an indirect way of communicating about a shape. It relies on the sense of human perception: Since our brain is able to interpret the 2D images captured by the eyes into 3D scenes, sketching a shape is most often done by simply drawing a general view as opposed to a special view along an axis or feature. This requires some kind of projection, usually a normal perspective, and a given mode of depiction such as a line drawing of silhouettes and contours, possibly enhanced by some shading.

Sketching is usually, like sculpting, a progressive process, acting from coarse to fine: The user will most often start with the largest features or even with some construction lines, using light strokes to indicate uncertainty; then he or she will progressively refine the shapes by oversketching the main lines and incorporating finer details. Lastly, he or she may enhance the perception of 3D by shading the sketch.

Communicating shapes through sketches works pretty well when the sketch represents a well-known object, easy to recognize from its silhouette (say, a tree). Then the 3D details will be automatically inferred from the a priori knowledge humans have of such objects. The method is also effective for simple, unknown shapes, given the principle of *minimal perception* [1]: basically, humans will always perceive the *simplest* 3D shape that matches a sketch. However, in the general case of a complex, unknown free-form shape, we will need to show it from several viewpoints, raising the problem of maintaining coherency between the different views. Although it can be done, sketching with several views is much more technical and time consuming. This leads us to the advantages digital sketching would have, regarding the expression of imagined 3D shapes: in addition to being able to reproduce a paper–pencil–eraser interface from a given viewpoint, it would enable us to move the camera as soon as the first simple shape is constructed. Then, the user could observe it from different angles and progressively modify its silhouette or add more details without any problem of correspondence between views. As in virtual sculpting, he or she could zoom in to work on fine features and out again for a general view or to make global changes and would expect undo–redo, copy–paste, and automatic smoothing operations. Lastly, he or she would not need to shade the scene by hand (which requires special skills): an appropriate rendering technique would do it automatically.

Providing a digital 2D to 3D sketching system is unfortunately at least as difficult as in virtual sculpting: indeed, the user will take for granted that the system will reconstruct the 3D shape *he* or *she* perceives from the sketch! This requires understanding human perception and extracting the rules needed to infer the third dimension. We will review the different solutions proposed so far in Chapter 3.

1.3 A BRIEF HISTORY OF SHAPE REPRESENTATIONS

Before focusing on the recent advances toward virtual sculpting and sketching, let us look at a brief overview of digital shape modeling. We use a historical perspective to introduce the surface representations developed in the field and discuss their contribution to free-form shape control. Although this section gives the minimal background for understanding the following chapters, a detailed presentation of the mathematical representations used for 3D shapes is beyond the scope of this book. The interested reader should refer to standard texts such as [2, 3, 4].

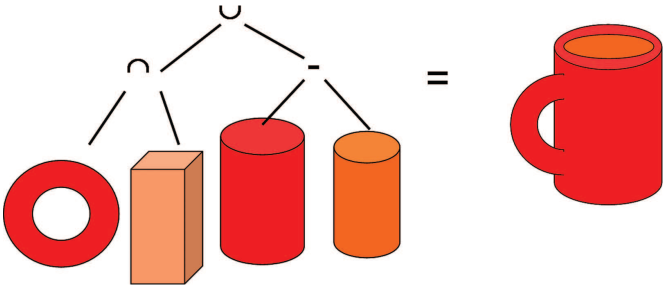

FIGURE 1.2: Constructive solid geometry (CSG) expresses shapes from a construction tree, which shows how they are progressively built out of solid primitives.

1.3.1 Seventies: From Solid Primitives to Free-Form Surfaces

The early work on digital shape design did not focus on easing user interaction toward the creation of complex shapes but rather tackled the problem of developing new shape representations. There are several reasons for that: The first users of digital shape modeling were engineers who wanted to make the production of real pieces, usually made out of metal, more systematic and exact, as well as to provide a straightforward link between shape design and manufacturing. The time spent for creating the initial design of these shapes was not seen to be the problem: the expectations in terms of user interface were low, and the advantage over sculpting physical models was obvious.

One of the first digital representations introduced was to express shapes in terms of solid primitives such as spheres, cylinders, and cubes, defined by their mathematical equations. More interestingly, it enabled engineers to describe the steps needed for the manufacture of a given shape: the geometric primitives were combined into a tree of successive operations such as union, intersection, and difference, expressing the history of manipulations in a *constructive solid geometry* (CSG) formalism [5] (Fig. 1.2). The CSG tree and the primitive's mathematical representations were used for ray tracing the shape, leading to a far from interactive but usable design process: the user controlled the design through an explicit text-based description of the shape; he or she could then visualize it from a given viewpoint, after several minutes to several hours of computation.

Although CSG worked well for designing exact, regular shapes, it was clearly insufficient for the field of automotive industry, where free-form surfaces were required. Car structures made of folded pieces of metal needed some careful design before being built. Once again, a way of mathematically expressing these shapes into a computer—after an early design phase through real sketching and sculpting—was strongly needed: in addition to enabling previsualization, such digital representations were required for simulating deformations under constraint forces, one more step toward the industrial production. In this context, Pierre Bézier, a French engineer working for Renault, took inspiration from metal *splines* early to introduce Bézier curves and surfaces [6, 7]. The use of parametric surfaces spread in the '80s [2], when Bezier surfaces were generalized by the introduction of B-splines, Cardinal splines, Beta-splines, and NURBS (nonuniform rational B-Splines)—the last providing an exact representation of circles and ellipses at the expense of a rational formulation and more user-specified parameters. These free-form parametric representations defined *tensor product surfaces*[1] $S(u, v)$ through the input of a grid of arbitrary *control points*. The last were either interpolated or approximated by a smooth, parametric surface, usually defined using piecewise polynomial or rational functions (fraction of polynomials) of (u, v). Being at least C^1 or C^2 (the minimal level of continuity to get a nice, smooth shading), the spline surfaces could be sampled as precisely as necessary into discrete, smooth-looking triangular meshes used for display.

While developing these representations, researchers discovered the benefits of local shape control: using a series of low degree surface patches, each defined from a few of the control points, had several benefits over embedding all the degrees of freedom into a high degree equation. In addition to being less expensive to compute, it enabled us to deform the surface locally, without having to redesign and recompute everything. Moreover, it was the right way toward an intuitive modeling system, since due to the principle of minimal perception already mentioned the user would expect to get the simplest shape defined from the control points.

Only a few contributions specifically addressed the creation or the editing of parametric surfaces: since directly defining a grid of 3D points is difficult using 2D devices such as a mouse, the control points were either captured from a real model or derived from simple 2D to 3D methods such as surfaces of revolution, lofting, and sweeping techniques. Once a base shape was created, it could be edited by moving the control points one by one, thanks to a graphical interface providing multiple views. In standard spline-based modeling systems, refining the surface by adding more control points in an area of interest, although possible in theory, was not really usable, since it would add a full line or column of new control points, affecting the locality of deformations elsewhere. Moreover, no way of changing the grid-based topology of the control structure was provided. Due to this structure, the control points could not smoothly represent complex geometries (e.g. branching structures such as a human hand, see Fig. 1.3) and topologies (e.g. closed surfaces with several handles). This lead to the development of complex, nonsquare parametric patches: Bézier and NURBS triangles; Gregory, Loops, and Coons surfaces; and more recently T-splines [8, 9, 10]. While enabling us to design more complex shapes, these representations were costlier, since they typically used high degree functions with rational terms. Moreover, ensuring continuity between these complex patches, even when solvable mathematically, often resulted in strange, nonnatural shapes (starting with

[1] Products of spline curves in the u and v parameters, respectively.

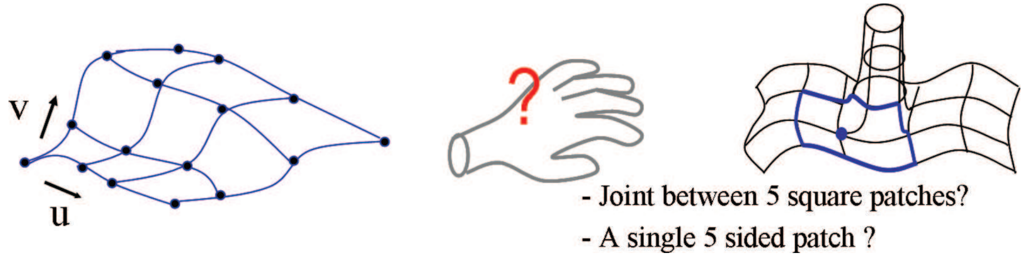

FIGURE 1.3: Tensor product, parametric surfaces built from a grid of control points, are not convenient for representing branching structures, for which either n-sided patches or smooth junctions between an arbitrary number of square patches need to be modeled.

the continuity problem at nonregular vertices between Bézier triangles). These new parametric representations thus mostly remained on the research side rather than becoming standards.

1.3.2 Eighties: Constructive Implicit Surfaces

Implicit surfaces were introduced in the eighties as an alternative to parametric representations, since although smooth and compact the latter were obviously not convenient for representing closed shapes with branches and holes or animating large deformations or topological changes.

Like spline surfaces, the first implicit surfaces introduced in computer graphics—called blobs, metaballs or soft objects [11, 12, 13]—used control points but in a totally different way [Fig. 1.4 (center)]. Each point generated a 3D scalar field decreasing with the distance. These fields were summed into a global scalar field f, and the surface was defined as an isosurface of

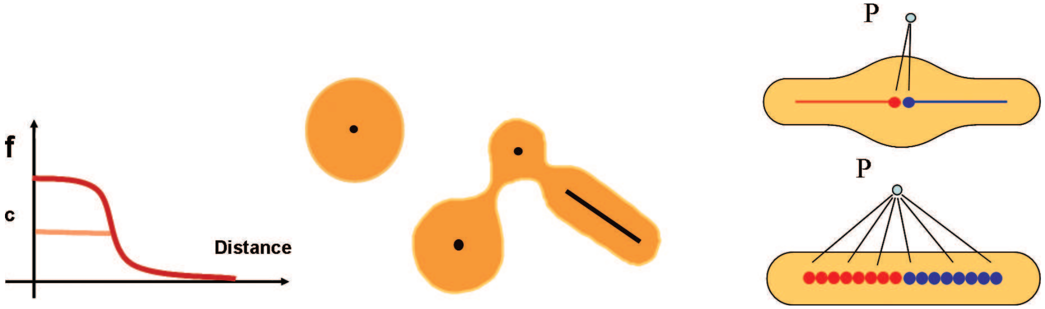

FIGURE 1.4: Constructive implicit surfaces are defined as isosurfaces of a scalar field function. The last is typically a decreasing function of the distance to the skeleton (left and middle). The most standard ones are distance surfaces (top right), for which the contribution of skeletal elements are computed using closest points and then summed, generating bulges at junctions. Convolution surfaces (bottom right) avoid the problem by integrating the field contribution along the skeleton curves or surfaces.

f, using the implicit definition:

$$S = \left\{ P \in \frac{R^3}{f(P)} = c \right\}.$$

This provided several benefits: Implicit surfaces came with a built-in in/out function characterizing not only the surface but also the interior volume of an object. Therefore, they correctly defined closed surfaces with no self-intersections. The in/out function could be used directly for ray tracing the surface or for converting it into a triangular mesh using a continuation method, later called *marching cubes* [13, 14, 15]. The surface would deform in an intuitive way when the point moved. This technique was later extended with the use of more complex implicit primitives such as those generated by skeleton curves or surfaces, the design of blending operators other than the sum and the storage of the constructive operations toward a shape within a construction tree similar to CSG and including deformation nodes [16, 17].

Regarding user control, some local control was provided, thanks to finite-support field contributions. However, the early implicit surfaces could not be edited in an interactive modeling system, since they could not be displayed in real-time: extracting an isosurface—allowing one to interactively change the viewpoint—took a few minutes and ray tracing from a single viewpoint a few hours. So in a similar way to CSG, nice, complex models were designed using a text file to describe the primitives and the blending operations. The first interactive modeling systems with implicit surfaces [18, 19] promoted point-based visualization techniques long before the use of meshless representations spread in computer graphics. However, the shapes created using these interactive techniques never reached the level of complexity of those created by describing the implicit primitives with a text file. The reason is probably that, as the shape was interactively constructed, the number of implicit primitives to be blended increased, so field queries were less and less efficient, causing interactivity to be lost after a while.

As with parametric surfaces, most research work in the implicit modeling domain addressed the improvement of the mathematical representations [20, 21]. Convolution surfaces [22] were introduced to avoid the unwanted bulges at junction between implicit primitives [Fig. 1.4 (right)] and later generalized to analytical convolution, reducing computational costs [23]. The addition of a topological skeleton was investigated to solve the "unwanted blending" problem to prevent parts of the surface close in space, but far from each other along the object's surface, from merging [24, 25]. Although the solution works for some applications, this is still an open problem. As soon as one imposes a structure to say which primitives will blend, one loses the underlying, attractive simplicity of the method. Also the more complicated blending rules sometimes produce surface tangent discontinuities. As in the spline surface case, none of these more expensive and intricate representations spread sufficiently to become a standard.

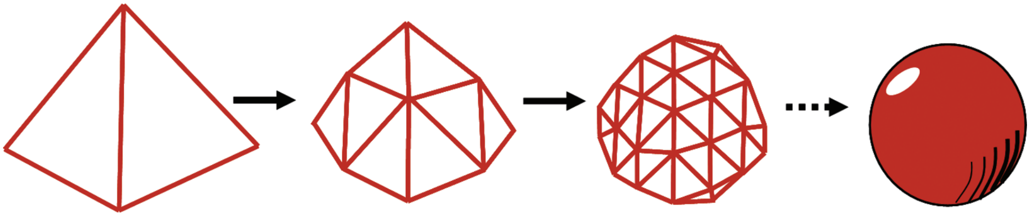

FIGURE 1.5: Subdivision surfaces are procedurally defined through an iterative subdivision scheme to be applied to their control mesh.

1.3.3 Nineties: Subdivision and Multiresolution Surfaces

Multiresolution interactive modeling was first introduced in computer graphics by the early work of Forsey and Bartels [26]: relying on standard tensor product spline surfaces, they replaced the grid of control points by a hierarchical structure, where local grids were progressively introduced to refine the surface with local details. By only editing the adequate control points, these local grids stayed stitched to their parent across the borders, with the desired degree of smoothness. Moreover, expressing the new control points in local frames attached to the parent surface enabled editing the model at a coarser scale when necessary, without altering the details. Since such multiresolution parametric surfaces lacked, until recently [27], the ability of represent arbitrary topologies, the more general framework of multiresolution, subdivision surfaces was quicker to become a standard.

Subdivision surfaces [28] can be seen as a superset of Bézier and B-splines surfaces: they define a shape from an arbitrary control polygon of any topology (which can be a grid of control points and also a cube or a structure with several handles). The control polygon is progressively refined using a given subdivision scheme, leading to finer and finer representations that either interpolate or just approximate the initial set of control points (Fig. 1.5). Most of the early work was devoted to the design of subdivision schemes and to the study of smoothness properties of the limit surface [29, 30, 31]. Although a subdivision surface is not parametric in the general case, it was proved to a be spline surface around *regular vertices* for several of the most standard subdivision schemes [32]. Subdivision was soon extended to more general multiresolution curves and surfaces with the more general idea of incorporating details (through extra point displacements) after each subdivision step [33, 34].

Multiresolution, subdivision surfaces were incorporated into standard interactive modeling systems—and proved very successful. Their advantages with respect to spline surfaces were made evident by their use in several productions: Pixar switched from the use of NURBS in *Toy Story 1* to the use of subdivision surfaces in *Toy Story 2* to get rid of the discontinuity problems between spline patches. The advantages of subdivision surfaces were made even more obvious by the short movie *Gerry's Game* where to get the desired result, the smoothing subdivision was disabled along some of the edges to model skin wrinkles [35]. However, as spline surfaces, subdivision surfaces only provide indirect editing through a control structure, making it difficult to focus on the surface without being aware of the mathematical representations. Moreover, the explicit storage of the control mesh's topology makes it difficult to take into account topological changes such as digging holes into a shape. These have up to now restricted their use in free-form sculpting systems.

1.3.4 Recently: Meshes Versus Meshless Representations

The free-form representations introduced so far were all defined through a limited number of control elements (control meshes for parametric and subdivision surfaces and skeletons for implicit models), making it possible for a user to manually define a shape from scratch and reducing the memory storage requirements to represent a smooth shape. In parallel, meshes attracted interest both as a final representation for models created from the previous high-level models and as the output representation for scanned objects. In the latter case, unorganized 3D data points were converted into a mesh using Delaunay triangulation, most often after a denoising step. The resulting meshes were automatically faired, simplified, or remeshed at different levels of detail, before other operations such as parameterization or editing [36, 37, 38, 39].

Since 2000, the generalization of 3D scanners and the increase of available memory and processing power have made it much easier to capture and process surface data captured from existing objects. Therefore, much of the recent research effort has been put into geometry processing and editing of huge geometric data sets rather than into the improvement of interactive methods for directly modeling digital shapes. The work in geometry processing led to the emergence of several new representations.

As triangulation has become more and more intractable with the huge data sets output by 3D scanners, the direct use of *point-based*, also called *meshless*, representations has been

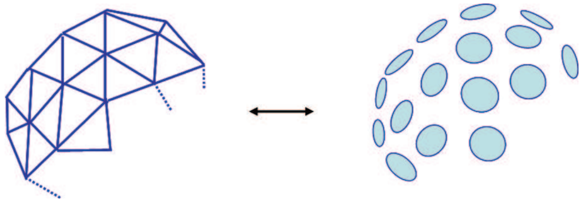

FIGURE 1.6: Point-based, or meshless representations, rendered using oriented splats, have proved to be a good alternative to standard mesh representations for visualizing huge unstructured data sets.

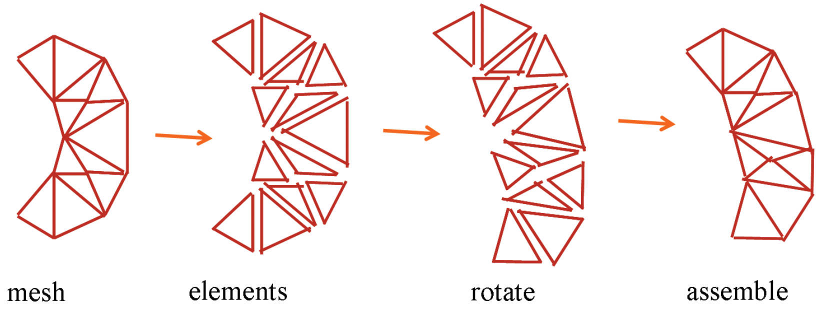

FIGURE 1.7: Differential mesh representations enable us to change global shape attributes while preserving surface details.

explored (Fig. 1.6). Instead of drawing connected triangles on the screen, the models were rendered by drawing a large number of disconnected splats while carefully adjusting their size to avoid gaps between them [40]. *Variational implicit representations* [41] based on RBF (radial basis functions) and moving least squares were used to reconstruct the surface between the existing samples, making it possible to compute local geometric quantities such as normals. These models were later extended to allow progressive levels of detail [42].

Being able to fair or globally edit a large mesh while preserving fine surface details led to the introduction of *differential representations* for meshes [43]. The most popular is vertex laplacian,

$$L(x_i) = x_i - \sum_{j \in N_i} x_i$$

which defines the position of a vertex relative to the average of the neighboring vertices [44]. The advantage of representing a mesh by its differentials is that it provides an efficient storage of the local high frequency details on the surface. Given the local geometry represented by such differentials, the system typically recomputes the mesh by solving a global optimization problem (in the form of a large, sparse linear or nonlinear system of equations), which is feasible thanks to the recent availability of fast matrix solvers (Fig. 1.7). For example, this representation can be used to achieve efficient geometry compression, preserving high frequency details. It also proved very useful for surface fairing, geometry transfer, and hole filling.

Point-based models basically lack an abstract governing structure which would help the user to quickly modify the shape in a meaningful way. They can, however, be combined with higher level laws [45], in the spirit of the early work on interactive modeling with oriented particles [46]. But as yet, they have not been used much for interactive shape design. In contrast, differential representations are intrinsically well suited to shape editing, since they enable us to deform a model—for instance, in order to satisfy user-defined constraints—while preserving local details [47, 48].

1.4 SHAPE REPRESENTATIONS VERSUS SHAPE DESIGN

Although they were all combined at some point with an interactive modeling interface, most of the representations we just discussed were not especially designed to ease creativity or to enhance shape control. As shown by our short overview, they were most often developed to answer specific needs such as representing a given class of existing shapes, easing visualization, allowing computations at different levels of detail, providing a compact way of storing or transmitting geometric data, and optimizing automatic operations like fairing or smoothing. In many cases, user interaction was introduced as a necessary step toward testing the system but had no influence on the mathematical representation developed.

This raises the question: can we identify the important features of a representation that make it suitable for the interactive design of free-form shapes? Firstly, enabling real-time display is an essential feature for interaction. Secondly, the representation must be general enough: it should allow modeling shapes of any topology and geometry, including holes and branching, and, if possible, enable changing the surface's genus as part of the editing process. If the aim is to design solid models, a representation restricted to closed surfaces surrounding a volume can be a good choice, since it will avoid having to explicitly take care of self-intersections and will provide an inside/outside test. To allow long-lasting interactive design sessions, using a representation whose complexity will not increase with each user's interaction gesture is also highly desirable. The complexity of the representation will of course need to increase somewhat to represent all the details progressively added to a shape: to reduce the limits on complexity, a representation enabling visualization and editing at different levels of detail (LODs) may be necessary. Lastly, the shape representation should ideally provide enough degrees of freedom for global and local deformation while incorporating mechanisms to smooth the shape or refine it. In some cases, other features such as deforming at constant volume can be incorporated within the representation rather than be enforced by a deformation technique. We will see in the following chapters that both simple surface-based representations such as meshes and more indirect representations such as implicit surfaces have successfully been used for free-form modeling purposes, although this choice affects the kind of editing techniques that can be developed.

The remainder of this book specifically addresses the problem of interactive shape design, defined as the interactive creation and subsequent refinement of a free-form shape. The methods we present belong to either a *sculpting* or a *sketching* metaphor. Taking inspiration from clay modeling, the first group of methods allows us to interactively mould or carve a 3D shape and to deform it either globally or locally. The methods from the second group, which have become more popular recently in computer graphics, address the creation and progressive refinement of a 3D shape from 2D hand-drawn strokes. As with sketching techniques in the real world, they are great for quickly creating a shape but do not enable us to describe deformations, although oversketching can be used for local editing.

CHAPTER 2

Sculpting Metaphors

A number of interactive shape modeling and editing techniques are based on the metaphor of sculpting soft materials such as clay. After giving some background on deformation techniques in computer graphics, we detail three modeling metaphors closely inspired from clay modeling: volumetric sculpting, physically based virtual clay, and sweepers. These methods enable the interactive creation of a shape from scratch as well as its subsequent refinement and deformation. We show the importance of gesture-based constant volume deformations in this context and discuss the implications of the alternative virtual clay models regarding user interaction.

2.1 BACKGROUND: DEFORMING DIGITAL SHAPES

Deformations are essential to an interactive sculpting metaphor inspired from the manipulation of clay. They have been studied for years, mostly in the context of editing existing models. We stress the difference between *model-based methods*, which directly act on the available parameters of a given model, and *space deformations* and *surface deformations*, which focus on defining a deformation function. With these methods, the deformation is either built everywhere in the embedding space or directly constructed on the surface to be deformed. This presentation is not exhaustive. Other excellent surveys of the same can be found in [49, 44].

2.1.1 Model-Based Methods

A first method for deforming a shape consists in directly editing the available variables or parameters of the original model: interfaces for interactively moving the control points of a spline surface, or for editing the skeletons of an implicit surface, belong to this approach.

Although computer artists used this approach as the only available one in the first generation of modeling systems, editing a complex model this way takes time and effort and can be far from intuitive: model-based deformation methods are based on the assumption that the designer has a perfect understanding of the underlying shape representation and is thus able to optimally use the built-in handles they provide. For instance, he or she will have to predict the locality of the deformations from the mesh resolution, the number of control points, or the size and shape of field functions defining implicit primitives. Moreover, the control he or she

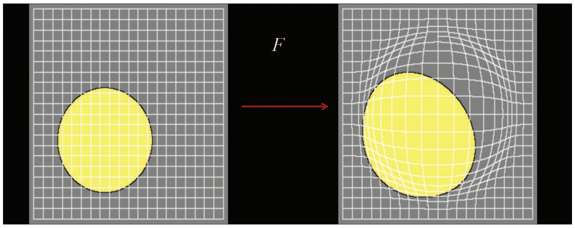

FIGURE 2.1: Space deformations warp space, deforming all the embedded objects. Courtesy of Alexis Angelidis.

gets is often indirect, as control points or skeletons do not lie on the surface to edit. Directly editing a lower level representation, such as a triangular mesh, does not solve the problem either. The editing is tiresome, and smoothness may soon be lost. Besides, a triangular mesh contains no hint of the structural intent of the original design, which makes very difficult to obtain aesthetically pleasing result.

Research in the area addressed these issues by providing specific interfaces for easing the editing of geometric models [50, 51]. Multiresolution representations were developed to provide the user with explicit handles on the locality of deformations [52, 38]. Lastly, some methods made the model's behavior more intuitive by mixing the geometric representation with physical laws [53]. The two first sculpting techniques presented later in this chapter belong to this category of approaches, although their choice for the geometric representation of the shape is quite different.

Overcoming the need for manually controlling a large number of variables also led to the development of alternative space-based and surface-based deformation methods. The latter automatically compute the changes in the model's parameters from intuitive, user-specified constraints on the deformation.

2.1.2 Space Deformations

Space deformations are any warping F from R^3 to R^3 (Fig. 2.1). Unlike model-based deformations, they are defined and tuned independently of the model they are applied to: all the objects embedded in the deformed portion of space, whatever their representation, will be deformed by the warping. To ease the representation of the deformed surface, applying a space deformation is often done in an indirect way, such as applying the transformation to the vertices of a mesh or to the control structure of a spline or a subdivision surface. The surface reconstructed from the deformed control structure is then only an approximation of the space deformation applied to the original shape. In contrast, space deformations can be exactly applied to implicit surfaces, but having an expression for the inverse of F is necessary: if the implicit surface is defined $\{P, f(P) = c\}$ then its deformed image will be $\{P, f(F^{-1}(P)) = c\}$. In practice, this has restricted the class of space deformations applied to implicit models.

Space deformations were first introduced as global deformations defined by a transformation matrix, such as taper, bend, and twist [54]. Subsequent work extended them to user-controlled free-form local deformations. The basic idea in most of these methods is to use a tool to deform space. More precisely, the following steps are applied:

1. the user creates and positions the deformation tool in 3D and presses a "freeze" button;
2. the system attaches the object to be deformed to it, by expressing the object's control points or parameters with respect to the tool;
3. the user edits the tool by manually moving and deforming it; and
4. the object is deformed by recomputing its points or parameters from the new tool position and shape.

If this approach seems indirect, it has the advantage of enabling the user to choose the minimal number of degrees of freedom when creating the tool: this usually makes it much easier to specify the desired deformation and tune its area of influence, among others. Note that the tool's deformation is usually specified using a model-based approach, which is effective if the tool remains quite simple. The different space deformation methods differ by the type of tool they use, the control they give over the locality of deformations, and the way they attach the objects points or parameters to the tool. They can be classified according to the tool's dimension [49].

The first class of free-form space deformations introduced, called FFD (free-form deformations) [55], used 3D lattices of control points as tools: the object's points were attached to the tool by considering their space coordinates as (u, v, w) coordinates in a Bézier volume defined by the tool. This was next extended to more general lattices (EFFD [56]) and, by the use of Bspline, NURBS, or subdivision volumes (SFFD [57]) instead of the Bézier formulation, to ease the definition of smooth, local deformation. Among the practical problems raised by this approach were a poor control of the locality of deformations and the time-consuming and indirect nature of the editing task, done by selecting and dragging lattice control points not belonging to the surface of interest.

Similarly, surface-based tools have been defined for space deformations: the object to be deformed was usually attached to a 2D lattice by projection along the surface's normal. The tools used were often very simplified versions of the object's surface, making it behave similar to multiresolution, subdivision surfaces. Recently, T-FFD [58] used a triangle soup of disconnected, possibly overlapping triangles instead of a lattice, with the advantage that the spherical influence of each vertex could be tuned independent of the others. Applying the tool to the object used a weighted sum of the displacements dictated by each triangle.

Axial deformations based on 1D tools, dedicated to applying local bends, elongations, or twists to parts of an object, proved very useful in practice, since they are much easier to control [59,60]. One of the problems solved was to attach each point of the model to the curve tool, so that there is no discontinuity due to the curve's concavities. The "wires" technique [61] extended curve-based methods by enabling a network of 1D curves to deform a model, thanks to an adequate, normalized blending of deformations: the influence of each wire was defined by a smooth density function similar to those used in constructive implicit modeling, resulting in smooth, local space deformations.

Lastly, using 0D tools was also investigated through the selection and displacement of a set of constraint points, usually located on the model. An optimization method, usually in the form of a system of equations to be solved, was used to find a space deformation fitting these constraints [62,63,64]. More recently, a shape editing method using such point/target constraints (modeled and combined using radial basis function) was implemented in real-time on the GPU (graphics processing unit) [65].

An advantage of space deformations is that this formalism eases the expression of constant volume constraints [66,67,68]: if a space deformation does not compress or dilate space (which can be checked easily from the determinant of its Jacobian matrix), all the objects embedded in space will retain their initial volume. The last sculpting metaphor described in this chapter belongs to space deformations and relies on this property.

2.1.3 Surface-Based Deformations

There is an inherent limitation of space-based methods. Because the deformation is defined in space, it takes no account of the geometry of the model. Suppose, for example, that you want to use deformation to animate the fingers of a hand. The finger tips may be close together in space but need to move independently under deformation. Controlling the locality of deformation then becomes a problem. Points that are close together in space but far apart along the model surface need different treatment.

Surface-based boundary constraint methods address this problem by computing deformation on the surface rather than in space. Like the last group of space deformations we described, they enable the user to define the deformation by defining target positions for a number of surface points, the changes to the representation's parameters being computed automatically. These methods, also called variational modeling [19,69,70,71,72], rely on optimization techniques. They automatically move the control points of a parametric surface, the skeletons of an implicit one, or the vertices of a mesh to fit the user's constraints. The system solves for the surface minimizing a functional or deforms the surface along the gradient of a functional.

Most recent surface-based methods are based on a differential representation of the surface (see Section 1.3.4) and solve a large, sparse linear or nonlinear system of equations to obtain the result [47, 73, 74]. They focus on generating a deformation that will consistently preserve small details in terms of shape and relative orientation. The main difficulty is the treatment of rotational invariance of local frames on the surface. Various approximation methods to represent rotations in a linear system have been proposed, and the recent trend is to solve this nonlinear problem by using an iterative solver in a reduced domain. Some extensions were proposed to generate constant volume surface-based deformations [75, 76].

Surface-based methods heavily rely on the topology of the original mesh structure. They require a good sampling quality for the initial shape, since the solution explicitly uses the initial number of degrees of freedom of the original surface. A more serious problem is that the result of deformation can vary for an identical user control, depending on the sampling of the surface geometry. Therefore, they are better suited for the modification of existing fine meshes than for applying successive deformations to a shape to model it from a simple primitive.

2.1.4 Deforming Versus Sculpting a Shape

Most of the deformation methods we have reviewed are not easily predictable for the user without some insight about either the shape representation they apply to or the mathematical model used for the tool. Moreover, they have complex interfaces requiring several steps of manipulation. None of them allows the user to apply a quick series of real-time modeling gestures, an important mechanism in real life for enabling artists to shape a model quickly. This explains why these methods were mostly illustrated by the editing of existing shapes rather than by the creation of complex models from scratch.

The remainder of this chapter studies in detail three sculpting methods dedicated to interactive shape design. All of them enable gesture-based interaction and address both the quick creation of a rough shape at the early stages of design and its subsequent refinement through deformation. The first two are model-based: they rely on a volumetric representation for clay, the model's surface being an isosurface of a 3D density field. They include a geometric method proven sufficient for adding and carving material and a physically based one for local and global deformations visually close to those of real clay. The last method belongs to space deformations: it sweeps the portion of the model to be deformed, at constant volume if desired, along paths defined by the user's gestures.

2.2 VOLUMETRIC SCULPTING

The first question when developing a virtual clay model is can a purely geometric approach be sufficient, or will a physically based model need to be introduced? If available, a geometric approach will be more efficient and will avoid lots of difficulties raised by physically based modeling, such as time integration, stability, and the like.

This section investigates the use of a purely geometric approach, showing its benefits but also its limitations. We first discuss the geometric representation for clay and describe an implementation which avoids bounding the extent of the sculpted shape in space or in resolution. Then we discuss the different tools which can be introduced for user interaction, as well as the visual and haptic feedback this representation can provide.

2.2.1 Geometric Representation for Clay

Let us think about real clay and its properties, with a view to creating a deformable model: what would be the most appropriate geometric representation for clay, which would provide adequate degrees of freedom—and those only?

Clay is a soft material, able to take any shape, be cut into pieces, and be reconnected, and out of which holes can be dug. The surface of a virtual clay model should thus be free-form, smooth, or easy to smooth out and be a closed surface with no self-intersections, delimiting an interior volume. Therefore, the most standard explicit surface representations such as meshes, spline, and subdivision surfaces are not appropriate. The remaining choices are between an explicit volume representation, such as parametric and subdivision volumes, and implicit representations. We explained in Section 1.3 why implicit surfaces belong to volumetric modeling. Although B-spline and subdivision volumes have been explored for virtual clay [77, 78, 79] and subdivision volumes enable progressive refinement of the model, we concentrate here on the use of *implicit surfaces*, which have the advantage of making topological changes straightforward.

The next question is which kind of implicit surface? As we have seen, implicit surfaces are isosurfaces of 3D scalar fields. The first option is to represent this field in a constructive manner, using a successive combination of implicit primitives [80, 81]. However, if each tool action creates a new primitive, the complexity of the shape continually increases while editing, even if the user erases most of the shape. Field queries then become more and more expensive, which could eventually make interaction impossible after enough operations.

An alternative, used in most previous work in the area [82, 83, 84, 85, 86, 87], is to represent the field function by a number of discrete sample values stored in a grid. Interpolation between these sample values is used to calculate the field. The immediate advantage is that each field query takes a constant time, whatever the length of the sculpting session. Moreover, this representation allows for multiresolution editing if some kind of multiresolution grid is used for storing the field samples (Fig. 2.2).

Next, the kind of discrete scalar field to be used is to be discussed. Signed distance fields, whose "outside part" stores an approximation of the Euclidian distance to the sculpted shape, are used for sculpting purposes [85, 86]. They ease the construction of a shape from real data or

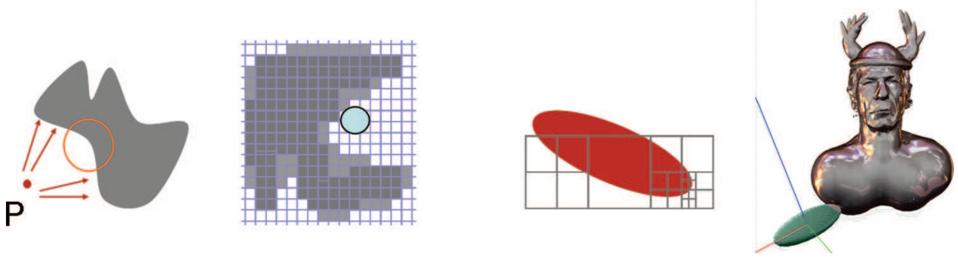

FIGURE 2.2: Distance fields (left) require global edits and are discontinuous in concave areas, while density fields (middle) are local. They can be stored in multiresolution grids that are refined according to the tool's shape, leading to the easy sculpting and storage of large to fine structures (right). From [89].

its conversion from another representation. However, a distance field requires global editing, as the value of closest distance may need to be updated in an unbounded region while editing the shape. Moreover, this field is discontinuous in concave regions, so smoothed approximations of it need to be computed to preserve a good level of continuity for the isosurface. In this book, we rather advocate the use of a scalar field representing the density of clay. Firstly, this is a more intuitive choice which makes, as we will see in Section 2.2.3, the creation of intuitive editing tools quite straightforward. Secondly this representation reduces memory requirements: the density field being zero where there is no clay, these regions do not need to be explicitly stored (and practical implementation will be described in Section 2.2.2). Where clay is present, the density value smoothly rises (ideally with a constant slope everywhere) from zero to a given maximal constant value inside the clay. In this book, we take the usual convention that the maximal value is 1, the clay surface being the isosurface at density 1/2. Such a density field still locally provides a local distance to the clay in the outside regions where the density is not zero. The closest point on the clay will lie approximately in the direction of the field's gradient.

Moreover, it makes field updates more efficient, since they are now local: using a tool to locally edit the surface will not require recomputing anything outside of the relatively small area of influence for the tool.

2.2.2 Providing a Multiresolution, Unbounded Clay Model

The early work [82] used a fixed coarse grid for storing the density field. The drawback of this simple representation is that a very large grid is needed to provide an extent of space, big enough for the sculpture while offering sufficient precision. Such a large fixed grid wastes memory, as the user will fill only part of it with clay. It also restricts the user's actions, since extending the shape outside of the grid is not possible. Also, the user will soon be limited in resolution: applying large, smooth tools will take time because of all the field samples to be updated, while tiny details will soon be missing in the locations the user would like to refine. To overcome the space extent problem, the commercial application using this method started with a rectangular block of material which could only be carved; but we are discussing here a sculpting system where the user can both add and remove material.

A practical solution to these problems is proposed by Ferley et al. [88]: the density field is stored in a virtual unbounded grid, implemented using hash tables. Cells are created and deleted when and where needed, only nonempty cells being stored in the hashing structure. This framework can easily be extended to multiresolution [89], enabling to sculpt fine details with virtually no bounds on the space resolution: the unbounded virtual grid is now a multigrid, whose cells can be split on demand into a fixed number of subcells. In practice, the local resolution in a given area is dictated by the size of the editing tools used. A different hash table stores field samples at each different resolution, and each update after an editing action is progressive, from coarse to fine, in order to always provide an interactive display. The finest resolutions can be updated, if necessary, at idle moments when the user is not editing.

Although volumetric rendering could be used, the commonest way of visualizing the sculpted isosurface is to convert it into triangles using a continuation algorithm such as marching cubes (see Section 1.3). Once this conversion is done, the mesh representation enables the user to rotate the model and view it from any direction at interactive rates. This visualization method can be efficiently used in the unbounded virtual grid representation we just discussed. In practice, Ferley recommends using three hash tables instead of one: in addition to the main grid storing the nonempty cells with their field values, two structures are added to optimize visualization—a hash table storing the set of cells that cross the isosurface and one storing the current surface triangles (Fig. 2.3). After each surface edit, the three structure are updated in the influence region covered by the tool. Each new cube crossing the surface is converted, on the fly, into surface triangles, and each one removed from the second structure results in the removal of the associated triangles. Visualization is then done using the surface triangle structure. This leads to a real-time incremental tessellation. In the multiresolution case (Fig. 2.2 right), tessellation

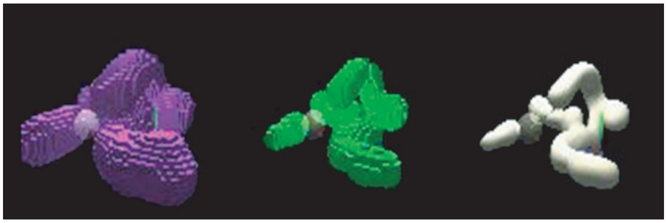

FIGURE 2.3: Data structures for the virtual unbounded grid, stored in three hash tables: nonempty cells, cells crossing the surface, and surface triangles. These three structures are locally updated after each user edit, so rendering is real-time. From [88].

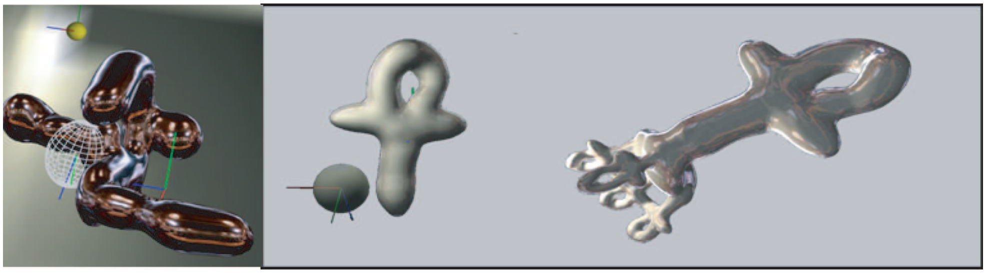

FIGURE 2.4: Sculpting tools are either analytic implicit primitives (e.g. spheres and ellipsoids) or discrete implicit shapes sculpted within the application. From [88].

is performed at each level of detail. The field updates are computed from coarse to fine, and the visualization uses the finest up-to-date surface in each area. With this method, the surface display is sometimes coarse, and cracks may appear between regions tessellated at different resolutions, but it ensures a real-time visual feedback to the user, whatever the number or speed of his or her editing actions, which is the essential point for effective, interactive sculpting.

2.2.3 Adding, Carving, and Smoothing Clay

Let us now discuss the different tools which can be provided for progressively creating and editing a shape in a volumetric sculpting system: The user may want to add some matter, carve it, or locally smooth out the shape. All these operations will be performed by locally editing the clay density values sampled over the 3D space. Locally increasing density will add material, decreasing it will remove some, while applying local smoothing filters to the density field will smooth the isosurface.

The simplest solution to achieve these editing actions is inspired by the constructive implicit surfaces methodology presented in Section 1.3. We define a sculpting tool (Fig. 2.4) as a couple formed by a local implicit primitive and a given action:

> The implicit primitive is any smooth continuous field function of finite support attached to a local frame called the *tool frame*. It can be either analytically defined, from a set of skeletal elements or a construction tree, or a discrete implicit surface itself, previously sculpted with the system.
>
> The action describes the way the tool's contribution is to be combined with the clay density values covered by the tool's support. Typical actions are *sum* to add some more clay and *difference* to remove some.

Tools used to add and remove material are usually represented as opaque rigid objects, visualized at their isosurface 0.5. Intuitively, the volume they cover is added to or removed from the clay. If their field contribution smoothly drops to zero, the shapes modeled progressively by their application will have the same degree of smoothness when correctly sampled. Other tools, such as those used for smoothing or painting the clay, just apply a specific action over their regions of influence, which can be visualized directly. Smoothing can be implemented using a low pass filter which recomputes each field value covered by the tool as a mean of itself and of its closest neighbors, using a matrix of coefficients approximating a Gaussian curve. Painting just changes the color attributes of the underlying field samples affected by a given tool.

Note that an adding tool can be extended to implement copy–paste and cut–paste actions: a part of the model is copied or cut, attached to a user-controlled local frame, and then used to add the same field contribution elsewhere. In addition to being moved the copied part can be scaled and rotated before pasting.

2.2.4 User Interaction: Visual and Haptic Feedback

One of the advantages of the volumetric sculpting approach we have just described is that the user directly interacts with the model's surface, without any need to be aware of its underlying mathematical representation. Therefore, he or she can just concentrate on shape design.

To enable a proper interaction, the user interface should however be designed with care. Firstly, the user should always get a good perception of the 3D shape he or she is sculpting. If a standard 2D display is used, then the perception of the third dimension can be improved using environment textures [88]. The use of 3D glasses is also a good solution (Fig. 2.11). Secondly, if sculpting tools are to be positioned with a 2D mouse while using a trackball to change viewpoint, it will be tiresome to move in the third dimension to reach the next location on the surface for editing. Using a device with 3–6 degrees of freedom is thus much preferable. Six degrees of freedom enables us to change the tool's position and orientation within the same gesture. Lastly, the user needs to know exactly whether the tool is slightly in front of, intersecting, or inside the model before applying any action. Therefore, Ferley et al. [88, 89] has used a semitransparent surface for the model, so that the tool's color changes when it is behind the surface. This however proved insufficient for sculpting complex models. Adding actions are usually made just in contact with the existing clay. Very often they were made too much in front by mistake, so the shape in Fig. 2.5 was achieved in only 3 hours, with lots of undo–redo operations.

A much easier and more effective positioning was achieved using force feedback [90]. Blanch and Ferley studied the use of a phantom device for haptic interaction and discussed several solutions to generate a useful feedback force in real-time (at 1000 Hz). Basically, two useful forces can be computed in real-time at the center of the tool: a viscous friction force, computed by multiplying the local clay density value by the tool's speed, and applying the force to reduce this speed; and a contact force, oriented along the field gradient and active when the

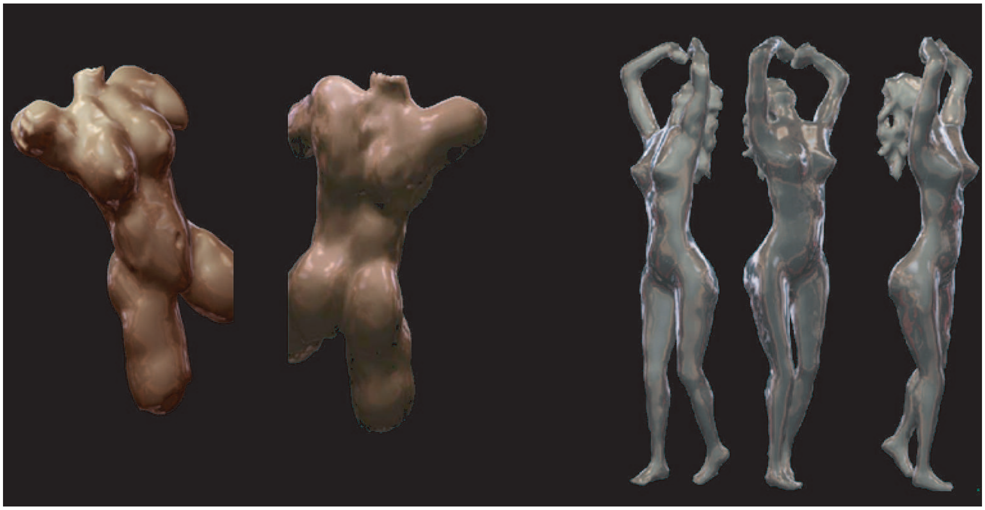

FIGURE 2.5: Using haptic feedback accelerates volumetric sculpting, since it eases the positioning of the tools. The figure on the left (from [88]) was sculpted in 3 hours using a simple mouse, while the one on the right (from [90]) was sculpted in 1 hour using haptic feedback.

tool touches the surface. By analogy with nonphotorealistic expressive rendering, Blanch has proposed a "non-hapto-realistic" or "expressive" way to combine these forces while giving as much useful feedback to the user as possible. The idea is to use both forces but to modulate their action according to the "applied" or "nonapplied" status of the tool. When the user moves a tool before applying it, he or she needs to feel the surface, to be able to position the tool with respect to it: then, a larger coefficient is used for the contact force. During tool application, the user should be able to move in space but feel the density of clay he or she is traversing. So a larger coefficient is used for viscous friction.

This methodology proved really efficient: the user could feel the virtual clay with the tool and position it much faster with respect to the model. The dancer of Fig. 2.5 could be sculpted in only 1 hour with some copy–paste editing for the legs.

2.2.5 Limitation: Volumetric Clay Does Not Deform!

The approach we described up to now is a good solution for adding or carving claylike material. This mode of interaction, which enables no deformation, is somewhat similar to painting and erasing matter in a 3D space. However, when offered a clay modeling metaphor, a user would expect much more as stressed in Section 1.2.1: clay should be able to deform, as real clay does.

In real life, clay deformations are used for two purposes: local deformations made by pushing clay with rigid tools or with the fingers help to shape the model's details (e.g., eye cavities

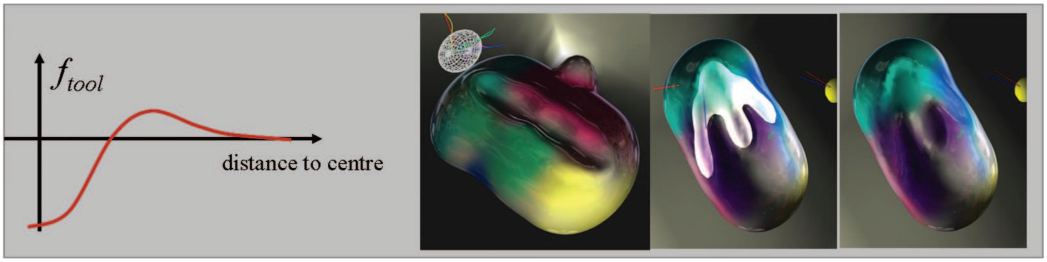

FIGURE 2.6: A geometric solution for mimicking the local deformation produced by a rigid tool: matter is suppressed inside the tool and added in an outside ring, forming a bulge. From [88].

and eyebrows are usually achieved with such gestures). Secondly, models such as characters are first created in a standard posture, which eases finding the right proportions, before large scale deformations such as bends and twists are used to make them more expressive. Note that both of these local and global deformations are done at constant volume.

A geometric solution for mimicking the action of a rigid tool locally pushing clay in front of it was proposed in [88]. The tool used had a smooth field contribution going from negatives values inside the displayed surface for the tool to positive ones in an outside ring around it and then smoothly back to zero. Applying such a tool with an additive action would thus remove material inside the tool while adding some around it, mimicking the bumps a print leaves on real clay (Fig. 2.6). However, no way of ensuring that such an action would preserve volume was proposed: the profile of the local deformation field function was interactively tuned by the user, and its exact effect on the clay it was applied to, such as reducing or increasing volume, would depend on the local slope of the density of clay function.

Applying large scale deformations in the volumetric sculpting approach we described is still more of a challenge. These deformations are different in nature from local ones: indeed, they should be able to propagate outside any predefined boundary of the tool's influence (Fig. 2.7).

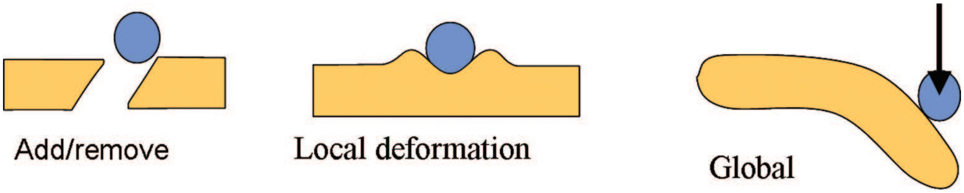

FIGURE 2.7: Global deformations such as bending and twisting are of a different nature from local ones, since they can propagate outside of any predefined boundary of the tool's influence.

Obviously, the purely geometric model we have discussed for the clay is not sufficient anymore: some kind of physically based modeling is required.

2.3 PHYSICALLY BASED VIRTUAL CLAY

This section describes a real-time simplified physically based model for clay which was set up for interactive sculpting purposes. Like the previous one, this model belongs to a model-based approach to deformations. Geometrically speaking, it relies on the same representation for clay, namely a density field stored in a grid. After describing a method for simulating real-time constant volume deformations in this framework, we discuss the benefits of this model in terms of visual results and its consequences for user interaction.

2.3.1 Physical Models for Clay

The properties of real clay lie between those of plastic solids and viscous liquids [91]. Clay can undergo local and global deformations, including changes of topology such as splitting and merging. Among the most important properties of clay for sculpting purposes are plasticity and mass conservation. The shape of a plastic object depends on its initial shape and on the history of deformations applied to it. This is unlike an elastic object whose shape at any time is a rest shape modified by applied external forces. At the speed and scale at which a user applies interaction, clay deforms to its new shape with almost no dynamical effect such as vibrations and waves. Such effects are almost ideally damped. Mass conservation during deformations implies the geometric property of constant volume, since clay is incompressible within the range of forces a human hand applies. Lastly, clay does have some rigidity: otherwise, it would progressively spread under its own weight when lying on a table. In the absence of gravity, this rigidity can be modeled by "surface tension" opposing any spreading into empty areas next to the clay.

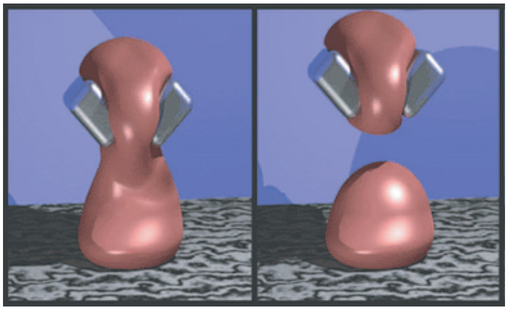

FIGURE 2.8: Viscous fluids have been modeled using particles surrounded by an implicit surface (picture from [92]). © 1997 IEEE.

Standard physically based deformable models in computer graphics include Lagrangian (Fig. 2.8) and Eulerian approaches. Let us discuss their benefits and limitations as regards modeling clay in real-time.

Lagrangian models use markers such as mass points and nodes to follow matter as it moves in space. Networks of mass nodes of a fixed topology have been used to simulate deformations, combined with smooth deformable models such as NURBS [53]. To get models that can change topology, particle systems where interactions are based on the distance between a pair of mass points are more appropriate. The first ones used Lennard-Jones forces [92,93] inspired from molecular dynamics, later extended to smoothed particles (SPH) whose kernels model the mass distribution around them [94]. All these particle systems preserve mass. They model compressible material (the internal forces being due to a local change of density and thus of pressure) but tend to come back to their initial volume after deformation. The last model of that kind proposed an efficient implementation, real-time up to 2000 particles, and introduced a surface tension effect thanks to a double kernel mechanism and modeled plasticity thanks to breakable springs [95]. However this material is still very far from the clay we need for sculpting. When the interactions are frozen into springs, it oscillates like jello after deformations, and without them, it spreads flat when lying on a table. Lastly, the use of a Lagrangian approach in an interactive sculpting process would require adding more and more smaller particles as details are sculpted, but one needs to be able to sculpt at constant cost whatever the length of the session.

Eulerian approaches which consider a fixed grid over space and express the flow of matter through the grid cell are closer to the geometric representation we studied in the previous section. These models were used for simulating realistic viscous fluids [96, 97], but however not in real-time due to the free surface to be tracked. Even if they were done in real-time, viscous liquids models would spread too much to be usable for clay, for which a specific physically based model needs to be designed. Closer to the work presented below, local claylike deformations have been modeled within a Eulerian representation using cellular automata [91].

A final way of using physically based modeling, when accuracy is not the main goal, is to follow the *layered models* methodology first introduced by [98]. The idea is to combine simple submodels to get a complex behavior more efficiently. The best adapted submodels are designed for each of the physical properties we would like to get; these models are then connected or combined to get a coherent general behavior, leading to the efficient animation of complex models, at the expense of physical accuracy. The requirement of real-time response is more important than physical accuracy for an interactive virtual clay model. The next section presents an approach based on this *layered model methodology*.

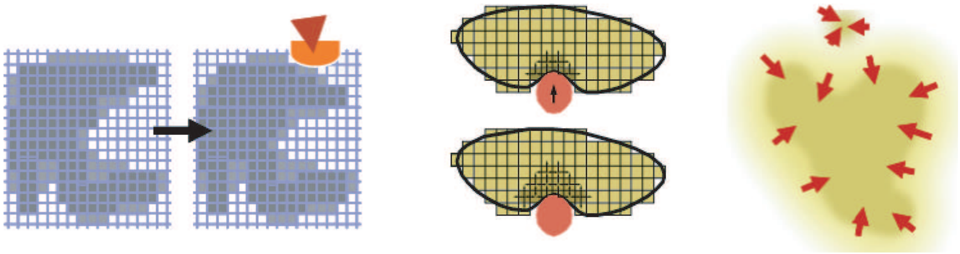

FIGURE 2.9: A layered model for virtual clay. Large-scale deformations produced by the first layer modeling plasticity (left). Local deformations generated by the second layer insure constant volume (middle). The third layer generates an internal rigidity which prevents clay from spreading over space (right). From [99].

2.3.2 A Layered Model for Virtual Clay

We describe here a model loosely inspired by physics, which visually mimics clay and is real-time [99, 100]. It overcomes the difficulties of standard physically based modeling, since it is static instead of dynamic and has thus no problem with time integration.

As discussed earlier, the three main properties we would like to get for clay are:

1. large-scale, plastic, fully damped deformations;
2. volume conservation; and
3. some inherent surface tension, preventing clay from spreading too much over space.

Let us discuss independent submodels for each of these properties. During simulation, these submodels will be emulated in turn, over the same virtual clay representation (the 3D grid storing the density field), each resulting in some transport of clay material from one grid cell to another. The underlying common geometric representation will ensure the coherency of the whole animation. Figure 2.9 gives a schematic representation of the three layers.

2.3.2.1 Large-Scale Deformations

This first layer should allow the user to bend or twist parts of the sculpted model using several rigid tools. The deformations are plastic: the clay will not come back to its initial state after the deformation is applied. Moreover, there is no need of a dynamic model here: getting a static "equilibrium shape" after each user's action is sufficient.

The aim is thus to compute the displacement δ to apply to the clay material lying in each grid cell as some combination of the displacements dictated by the different tools simultaneously acting on the clay. Let us get loosely inspired by physics: Since clay is a viscous fluid, the displacement of a user-controlled tool basically moves the clay around it in the same direction. Then, this action decreases with the distance. However, it is not the Euclidian distance which is important here but rather the shortest path *inside the clay* between the tool and the current cell. When several tools are applied at once, these pseudodistances inside clay can be combined to define something akin to Voronoi regions around the tools. More precisely, Dewaele and Cani [99,100] computes the weighting coefficient k_i to be applied to the motion of each tool T_i as follows:

$$k_i = \frac{1 - \frac{d_i - min_j(d_j)}{min_j(d_{ij})}}{2} \quad \textbf{and} \quad \delta = \frac{\sum_i k_i \delta_i}{\sum_i k_i}, \tag{2.1}$$

where j refers to all the other tools involved, d_i is the pseudodistance from the current cell to T_i, and d_{ij} is the pseudodistance between two tools. The pseudodistance is computed using a propagation mechanisms from each tool position. This approach results in Voronoi-like regions of influence for each tool but with a continuous blending on the generated displacements between them.

Up to this point, we assumed that the tools' motion would be translations when they are in contact with the clay. However, the user may also rotate tools, expecting to produce rotations or twists in the clay. The motion of a solid rotating object cannot be described by a simple displacement vector. Instead, a point A rigidly linked to the tool moves according to displacement field:

$$\delta_A = \delta_O + OA \times \omega, \tag{2.2}$$

where δ_O is the translation of point O (the center of the tool), and ω is the twist applied to the tool. To take into account the rotation of the tool, we simply replace the previous δ_i in Eq. (2.1) by the δ_A which would be obtained at the correct cell using Eq. (2.2) for tool i. This way, more general deformations can be generated. For instance, a bar of clay can be twisted by simply turning a tool at one end of the bar.

2.3.2.2 Mass Conservation Layer

Nothing in the deformation generated by the first layer ensures that volume, or even mass, will be preserved. The displacement field generated can easily send too much clay to one cell. Thanks to the underlying density grid representation, detecting such excesses of clay is easy: they are the cells whose density value exceeds 1.

Let us take some loose inspiration again from fluid mechanics. In a fluid, when there is an excess of pressure locally, a flow will be generated from the area of high pressure to areas with lower ones until the pressure is uniform. Unlike fluids, clay should remain solid (density 1) rather than spread and occupy the whole space to equalize pressure. We thus use a slightly different iterative mechanism: The clay in excess in a dense cell is poured into its six neighboring cells, regardless of their own density. If those cells are not full, then the process terminates. If there is an excess of matter, they distribute it among their closest neighbors, and so on. Matter will move from cells to cells and finally reach the object's border, where it will find some room to remain. This results of iterating this very simple process over the dense regions of the grid results in intuitive folds and local bulges when the user deforms or presses the clay.

2.3.2.3 Surface Tension

After several deformations using the two layers above, the matter tends to become less and less compact. Clay pushed by the tools can be dispersed around the object, and the transition from inside (density 1) to outside (void cells) gets slower and slower. One of the problems with cells with low densities is that the user does not see them, so strange effects can arise if a tool pushes these small quantities of clay in front of it: clay popping from nowhere when density, due to action of the tool, rises to the threshold; inaccurate changes of the surface location; and so on. Moreover, since matter in cells of low density is no longer visible, the object's volume will seem to decrease, even if matter does not really disappear.

The surface tension layer used in [99] aims at limiting these problems. It keeps the gradient of density near the surface of the clay to an acceptable value. Basically, matter in cells with very low densities is moved to nearby cells with higher densities along the gradient direction. This way, the object remains compact.

2.3.3 Interacting with Virtual Clay

The user can use three kinds of tool. In addition to tools for adding or erasing clay, which work just the same as in the previous volumetric sculpting system, he or she is provided with rigid tools to interact with the clay at constant volume. As with volumetric sculpting, rigid tools are modeled using implicit surfaces. When a rigid tool touches the clay, it affects its deformations in two ways: the tool's displacement and rotation are communicated to the surrounding clay through the first layer, while the clay inside cells covered by the tool is poured to the neighboring cells in an outward direction with respect to the tool. This can produce both large-scale and local deformations.

Figure 2.10 shows the ability of the model to take into account both translations and rotational motion of the tools in contact with the clay, as well as the use of an arbitrary number of tools. The side-by-side comparison with photographs shows that this model adequately captures the main features of real clay. However, there is a good reason why this virtual clay model was only used, up to now, to perform very simple, proof-of-concept deformations: the closer a model is to real clay, the more difficult the interaction is. Let us discuss the interface needed for interaction.

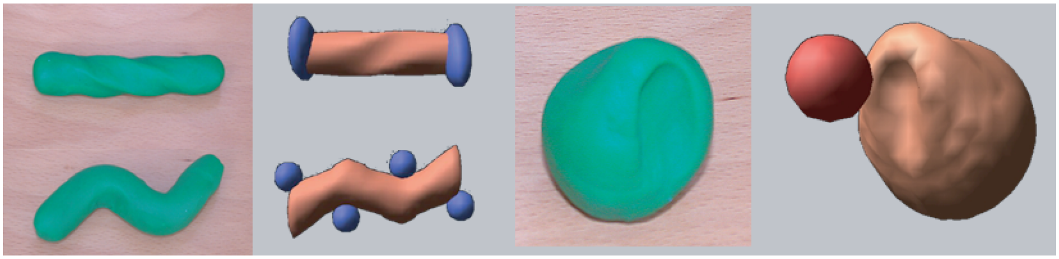

FIGURE 2.10: Comparison between real clay and Dewaele's virtual clay. Local and global deformations at constant volume are adequately modeled. They can be controlled by an arbitrary number of tools. From [99].

Although possible in theory, sculpting a complex shape using a single rigid tool controlled with the mouse such as in Fig. 2.10 (right) would be difficult and time-consuming. Indeed, most people would use their ten fingers and maybe the palm of their hand for modeling real clay. Simultaneously interacting with an arbitrary number of tools is possible with the model we just described, but it is far from sufficient. An easy way of interactively controlling these multiple tools needs to be defined.

The most intuitive solution is to provide some real-time control of virtual hands interacting with the clay, for instance, by duplicating the motion of the user's hands, using either a data glove or a system based on cameras designed to do so (Fig. 2.11). However, one doubts the user's ability to model a shape without touching it. Without the sense of contact with virtual clay, it is very likely that the real hands' gestures would overshoot their target. In real life, designers use the compliance of their hands and the interaction forces with real clay to finely tune their gestures. We discuss this in more detail in Chapter 4.

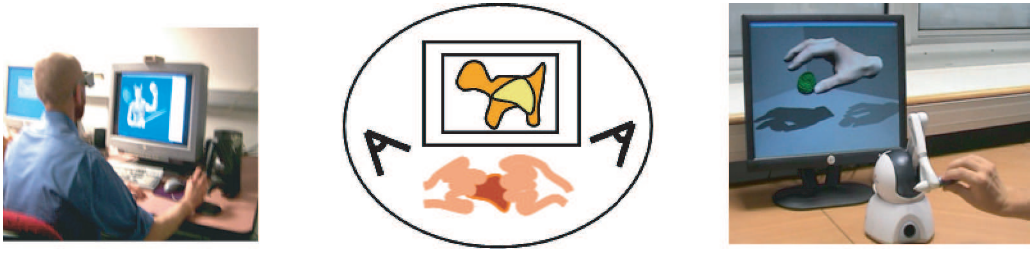

FIGURE 2.11: Three possible interfaces for interacting with virtual clay: using a single tool, moved in 3D using a force feedback device (left); capturing the deformations the user applies on a real object serving as an avatar (middle); and controlling the gestures of a virtual hand interacting with the clay (right). Left from [90], Right courtesy of Paul Kry.

This leads to the question: how far should a digital model for clay try to mimic the behavior of real material? The last section in this chapter studies a totally different way of modeling the constant volume deformations which characterize real clay: instead of using a model-based deformations, it investigates the use of space deformations but keeps the clay modeling metaphor in that the deformation is controlled interactively by the user's gestures.

2.4 SCULPTING WITH SWEEPERS

This section presents a sculpting metaphor based on space deformations. It is applicable to virtually any model representation. In practice, the sculpted shape needs to be accurately, yet efficiently, represented: an adaptive mesh automatically refining where needed is a good solution. Typically, an edge splits whenever a deformation would otherwise take its midpoint too far away from the average of its new end points. The sweepers' method focuses on generating intuitive space deformations, controlled by the user's gestures. It can be extended to constant volume swirling sweepers, still using the same kind of interaction. Note that other similar methods such as warp sculpting [101], and constant volume vector field–based deformation [102] were developed in parallel. We focus on the sweepers' version [103, 104], since the simple gesture-based interface it provides makes it a very relevant solution toward virtual clay.

2.4.1 Sweepers: Space Deformations Controlled by Gesture

We already reviewed space deformations in Section 2.1. As we have seen, most of them use a tool to define the deformations: the user freezes the tool within the object, interactively deforms it, and then applies the resulting deformation to the model, which takes typically a few seconds to compute. Even if it is only a fraction of a second we have a step-by-step process where we make a deformation abstractly and then see it applied in each step. In contrast, a *sweeper* is a space deformation defined by gesture: more precisely, the user sweeps a geometric tool that deforms space along a motion path. The region of space covered by the tool, where the model to deform is partly embedded, follows its motion, while the deformation smoothly vanishes outside the tool's region of influence. This way, the user can ignore the fact he or she defines a space deformation and rather concentrate on real-time interaction with the sculpted surface.

The challenges are first to find a way of defining a smooth deformation that sweeps space with the tool while vanishing at a distance and second to ensure that this deformation is foldover free: preventing space foldover will prevent the sculpted shape from self-intersecting, whatever the user does. This is mandatory, since changing a closed surface into a self-intersecting shape would be very bad in terms of the clay modeling metaphor. Moreover, if the model did self-intersect, no subsequent space deformation would be able to set it back to a nonintersecting state.

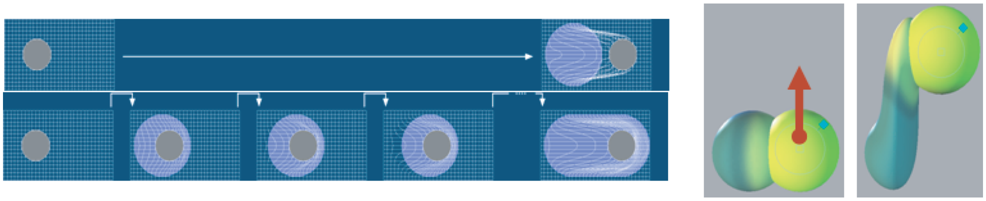

FIGURE 2.12: Sweepers use the user's translation gestures to wrap space in a foldover. Small steps used to make the deformation foldover free (left). The user positions a tool in intersection with a shape, the tool's region of influence being depicted in yellow (right): when applied and moved, the tool sweeps a part of the shape with it while keeping it smooth. From [104].

Angelidis et al. [103] defines a tool very closely to the way it was done by Ferley and Dewaele and Cani [88, 99]: a tool is a 3D shape defined by a user-controlled moving frame and a field function $\varphi(P) \in [0, 1]$, modeling its influence in space, which smoothly decreases and vanishes at a distance. It is visualized as an isosurface of the influence function.

Intuitively, sweeping means weighting over space the transformation M defined by the tool's motion with the local influence value φ. The transformation is given by a 4×4 matrix, a combination of scale, translation, and rotation, defined by a gesture such as a mouse move. But what is a natural way of weighting a transformation, to apply just a portion of it? The answer is to take a power of the matrix. As shown by Alexa [105], this can be implemented using matrix logarithm and exponentiation and leads to the formulation:

$$F(P) = (\varphi(P) \odot M) \cdot P \tag{2.3}$$

where F is the deformation function for this gesture, and the operator $\odot$ is defined by $\alpha \odot \mathrm{M} = \exp(\alpha \log(\mathrm{M}))$ (see [105]).

Self-intersections of the deformed shape could occur if the above deformation was used directly. In practice, the deformation is instead applied in a series of small steps, computed so that space foldover is prevented (see [103] for details).

2.4.2 Constant Volume Swirling Sweepers

Sweepers, as just presented, compress and dilate space and thus perform a function equivalent to adding and removing material from the sculpted shape. As we have already emphasized, constant volume deformations are much more intuitive. Let us discuss an extension of sweepers to achieve constant volume.

In order to set up constant volume deformations, Angelidis et al. [104] introduces a particular case of sweeper—a *swirl*, defined as a rotation whose magnitude decreases away from its center, and smoothly vanishes at a distance. Formally, a swirl is defined by a center point

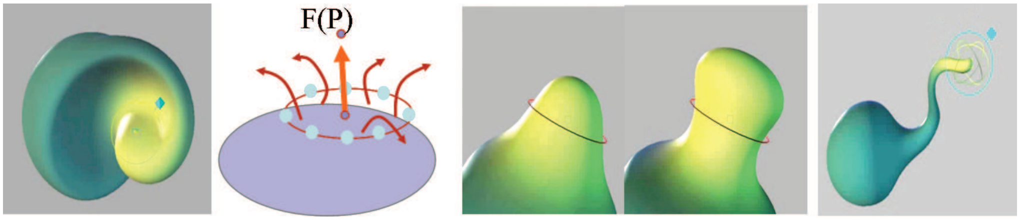

FIGURE 2.13: Constant volume swirls (left) are placed along rings to move matter along a translation vector defined by the user's gesture (center). In practice, the use of swirls is transparent to the user. He or she simply sweeps constant volume clay by gesture (right). From [104].

C together with a rotation of angle θ around an axis V [Fig. 2.13 (left)]. A radial function φ defines how the amount of rotation decreases away from C, within a sphere of radius λ. Informally, a swirl twists space locally around axis V without compression or dilation (see proof and implementation details in [104]). It thus preserves the volume of any shape embedded in the deformed space.

Although rotations can be useful for bending and twisting 3D shapes, being able to define a constant volume sweeping gesture is mandatory for being able to sculpt a shape through deformations. To convert swirls into translations, Angelidis uses a *ring of swirls*: n swirls are positioned around a circle, their axes being along the local tangent direction to the circle. Each of them swirls matter through the inside of the circle. Summing their contribution has the effect of locally puling space through the circle, at constant volume.

A way for the artist to control the amount of deformation is still to be defined. Obviously, he or she would like to control the amount of translation locally applied to the shape and its space extent, but one does not need to be aware of the underlying mathematical model such as the ring of swirls. Therefore, Angelidis computes the parameters of each swirl from the translation T the user would like to apply and which is defined, for basic sweepers, by sweeping a spherical tool over space. At each deformation step, n swirls (usually eight) are placed along a circle perpendicular to the current translation vector T. The swirl's common radius of influence is twice the circle's radius r. The angle θ_i of each swirl is set up so that a point at the center of the user-controlled tool is translated to its position plus T, using:

$$\theta_i = \frac{2\|T\|}{nr}. \tag{2.4}$$

Figure 2.13 (right) illustrates the deformation applied to a shape.

2.4.3 Vector Field–Based Deformation

The key to volume preservation is that the swirl defines a divergence-free vector field, and their combination is still divergence-free. A volume-preserving deformation can be constructed from any such field, and, more recently, von Funck et al. [102] have done this in a more general way. They construct a divergence-free vector field from two scalar field functions, p and q, using $\boldsymbol{v}(x, y, z) = \nabla p(x, y, z) \times \nabla q(x, y, z)$. By choosing different ways to construct the field they produce deformations from a swept tool in the manner of swirling sweepers and also a variety of other deformations including twists and bends. Volume is preserved and large-scale deformation can be done in a way that preserves local detail.

2.4.4 Results and Discussion

As the results show, the use of a geometric constraint such as constant volume deformations can be sufficient to get very natural deformations, with no need of any physically based simulation. With swirling sweepers, the model behaves very much like a physical piece of soft clay. Swirling

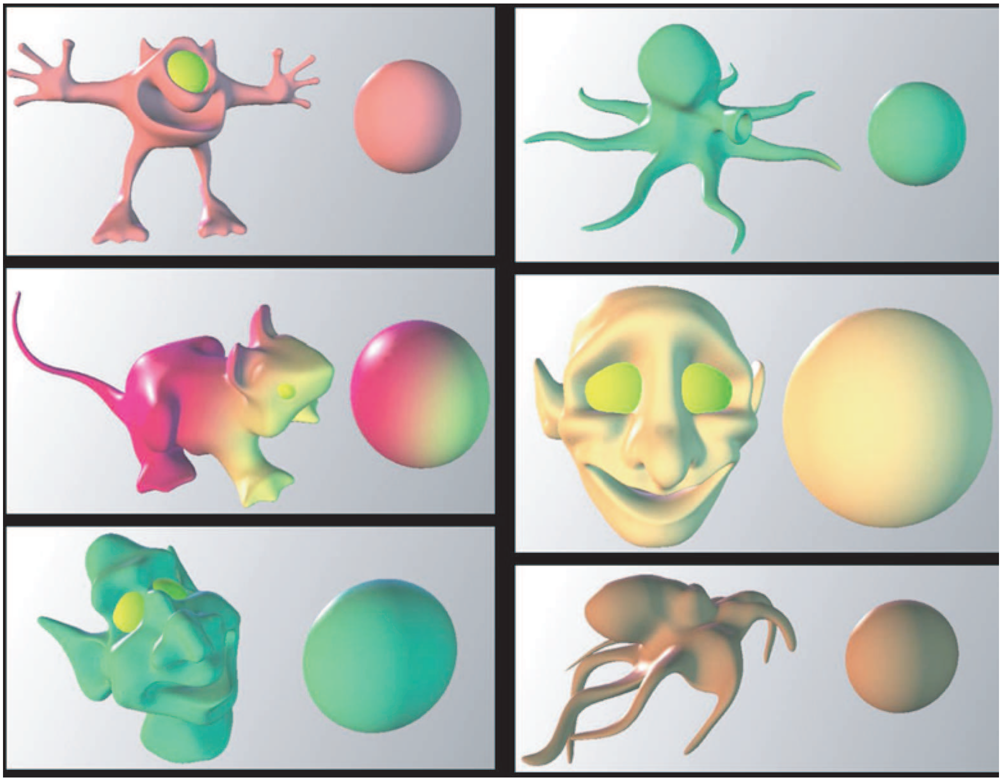

FIGURE 2.14: Examples of shapes modeled from a ball of clay in a few minutes, using swirling sweepers. From [104].

sweepers were illustrated by the creation from scratch of a series of complex shapes, represented using adaptive meshes (Fig. 2.14). They attest to the practical usability of the method.

Let us however discuss the limitations of this approach, thinking of the different expectations we listed in Section 1.2.1 for sculpting metaphors:

A first problem is the space-based nature of the definition of a tool's area of influence: When the shape to be sculpted is complex, all the parts of it that come close to the tool will be deformed at once. The user thus has to sculpt from coarse to fine levels, with smaller and smaller tools to avoid deforming the neighboring regions. And he or she cannot easily apply large-scale deformations to a detailed model, although this can be done to some extent (see [102]). As with Dewaele's virtual clay model [99, 100], the deformation should rather propagate according to the shortest path *inside clay* between the location on which the deformation is applied and the current point. However, this would then be a model-based deformation rather than a space deformation, since the shape of the model would affect the result.

A second expectation to be discussed is the ability of the method to model large-scale deformations such as bending or twisting parts of the shape. Although Angelidis chose to only use translation gestures, a single swirl could be used to apply a local bend or a twist. These rotation-based tools were specifically introduced in other similar formalisms, such as vector field based deformations [102]. However, as shown by this work, to specify these deformations requires an interface in the spirit of standard space deformation tools. This supports the idea that such deformations cannot easily be controlled interactively by gestures or, at least, by 2D gestures specified with the mouse.

Lastly, the use of foldover-free space deformation makes it impossible to change an object's topology, such as digging holes in soft material and welding two pieces together. Although less general, the method is still very useful in an interactive sculpting context, where the user starts with a simple primitive of same volume and topology as the model he or she wants to create and applies translation gestures to shape it.

2.5 CONCLUSION: WHERE SHALL WE STOP TOWARD VIRTUAL CLAY?

This chapter has described three approaches based on a sculpting metaphor. Although the first and the third ones could be successfully used to create complex shapes from scratch, none of them provides a full solution, answering all that one would expect for sculpting metaphors: The first system basically enables us to deposit and to erase matter in 3D, providing an effective, but quite limited, version of sculpting. The second one successfully incorporates claylike deformations in the first method but is limited in practice by the lack of proper user interface for applying deformations. The third system just borrows from clay modeling the ideas of modeling by gesture and of constant volume deformations. In practice, the gestures are translation gestures controlled by the mouse in which the user sweeps matter along paths, which deforms the model at constant topology and without any rotation-based deformation.

If the first and third models proved immediately usable, it could be because they could rely on 2D rather than 3D sculpting gestures from the user: most of the models were created with a simple 2D mouse and translation gestures while progressively changing the viewpoint. The third model is able to pull or push the clay instead of just depositing or erasing it. An advantage of 2D gestures is that they are less tiring and thus much more precise than a 3D gesture without an adequate sense of touch. In contrast, the second model raised the questions: Where shall we stop toward virtual clay? If provided with an ideal, hyperrealistic virtual clay model, would we be able to develop the proper interface to use it on a computer? The future directions of research this question raises will be discussed further in Chapter 4.

An alternative to making 3D sculpting usable is to switch from sculpting to *sketching metaphors*: the latter would intuitively be the most appropriate way of taking a full benefit of the user's 2D input gestures. As we will see in the next chapter, these systems also raise a number of challenges.

CHAPTER 3

Sketching Systems

Sketching systems allow the user to construct a 3D shape via sketching. The user draws the outline of the desired shape as a 2D sketch, and the system automatically generates a corresponding 3D model. This chapter first describes the background of the approach, discussing how people depict 3D shapes in 2D form and how people perceive a 3D shape from a 2D representation. Then we describe three sketch-based shape modeling approaches that use different surface representations, plain mesh, implicit surfaces, and optimization surfaces. Each representation has its own strengths and limitations, and we describe how sketching interfaces are adapted to these representations.

3.1 BACKGROUND

Sketching methods are built on top of knowledge in art and science. Here we describe two important ingredients.The first is from art: how people depict a 3D shape using a 2D medium. The second is from science: how people perceive the third dimension from a 2D medium. We first describe what has been discussed before the emergence of computers and then describe computational methods to replicate these processes.

3.1.1 2D Depiction of 3D Shapes

Humans have a long history of depicting 3D shapes and scenes in 2D form. Cave painting is considered to be the beginning of such activity. Line drawing is a specific form of such depiction, where the artist depicts 3D shapes via a collection of lines. These lines are roughly divided into three categories: silhouettes, features, and shading (Fig. 3.1). A silhouette line shows the boundary of the shape. One side of the line is the inside of the object, and the other side is the outside. A feature line shows that there is a discontinuity on the surface such as a sharp edge. Shading lines are used to indicate shading. Willarts discusses how silhouette lines and feature lines are used to represent shapes [106]. In the case of objects with planar faces, lines denote edges and line junctions indicate corners and occlusions. In the case of smooth objects, lines denote contours and line junctions indicate occlusions. Various labeling schemes have been proposed to analyze such drawings.

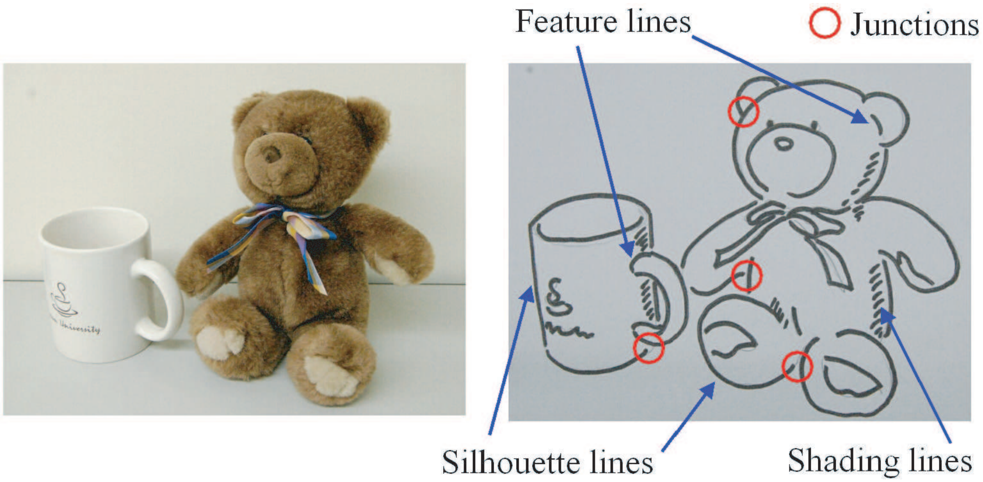

FIGURE 3.1: Line drawing of real-world objects.

Automatic construction of line drawings from given 3D shapes has been the subject of experiment since 1990 as a part of the so-called nonphotorealistic rendering techniques. Instead of shading all visible surfaces, these techniques depict the given 3D scene as a 2D line drawing, combining silhouette, feature, and shading lines. They are roughly divided into two approaches (Fig. 3.2). One is image space processing, and the other is object space processing. Image space methods generate silhouettes in image space [107, 108]. A popular method is to render a depth image and a normal image and then trace discontinuities in these images. Object space approaches explicitly detect silhouette edges that are adjacent to both back-facing and front-facing faces and render them on the screen [109, 110]. Shape modeling by sketching can

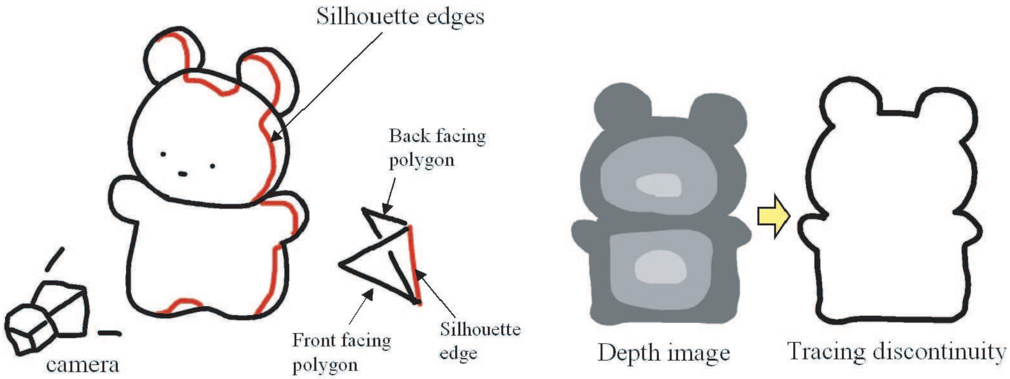

FIGURE 3.2: Silhouette rendering in object space method (left) and image space method (right).

be seen as the reverse of such 2D depiction. Given a 2D line drawing consisting of silhouette, feature, and shading lines, the system automatically generates a 3D shape whose 2D depiction matches the input.

3.1.2 Human Perception of the Third Dimension

How does the human brain convert a single 2D picture into a mental 3D shape? This is an important problem in cognitive psychology, and many descriptions have been proposed. One way to describe the process is the notion of maximal simplicity, that is, humans will imagine the 3D shape as the simplest way to connect the depicted lines. The question is what is a simple shape for a human? Most people seem to consider that it is the shape which will minimize the variations of curvature, which leads to setting up a curvature in the third dimension which fits the one we observe of the 2D silhouette strokes. Some cognitive studies seem rather to show that humans tend to imagine developable surfaces [111].

Hoffman suggests that, when a person see a 2D image, the human visual system infers the most stable 3D structure whose appearance does not change significantly when the viewing angles changes slightly [1]. This leads to more detailed rules useful for reconstructing 3D geometry from give 2D line drawings. One example is that human vision assumes one silhouette is hidden behind another silhouette when seeing T-shaped silhouette line. Another example is that human vision assumes a convex surface when seeing a convex silhouette and assumes a saddle-shaped surface when seeing a concave silhouette (Fig. 3.3).

When they recognize a well-known object from a sketch, humans will unconsciously use a priori knowledge they have about the object to infer the third dimension. For example, if a person knows that what he is looking at is a flat cloth, he can predict its complete geometry with high accuracy from a single view. Similarly, a person can infer the depth of body parts very accurately when seeing a 2D stick figure, using the knowledge of human body constraints. Although we can leverage this ability in designing shape modeling systems, we will not discuss

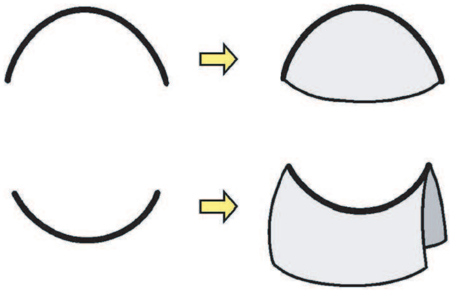

FIGURE 3.3: 3D surface reconstructions from 2D contours.

this issue of modeling with a priori knowledge in this book because our main focus is on general free-form shape design. Interested readers are requested to refer to the literature on this, such as garment design [112], posing of human models [113], and plant modeling [114].

Computational methods to mimic such 2D to 3D conversion are relatively new. The computer vision community has focused on the problem of constructing 3D shapes from natural images captured by camera. However, if the source image is a hand drawn sketch, the process is entirely different. The information in a natural image is created by the interaction of light with the 3D shape. A sketch is deliberately designed to communicate the shape to another human using knowledge about how it will be interpreted. Reconstruction of objects consisting of planar surfaces is relatively well studied [115], but reconstruction of free-form surfaces only dates back to the seminal work by Williams in the early 90s. He generated a 3D surface by inflating a closed region, using a level set method [116]. The systems described in the following sections extend this idea by combining various gestural operations.

There are two distinguishable approaches for 3D model construction by means of 2D line drawings. One is to complete a single complex sketch from a fixed viewpoint and then ask the computer to reconstruct a 3D geometry [117,118]. The other is to progressively construct a complex geometry by interweaving sketches with viewpoint changes [119, 120, 121]. All three systems we describe next follow this progressive approach. The first approach more closely follows the convention of traditional artwork and can appear more intuitive for first time users. However, there is too much ambiguity in a complex sketch, and it is very difficult to build a practical system based on this approach. The second approach addresses the problem by decomposing complex sketches into many small steps to clearly capture the user's intention, opening a way to build practical modeling systems using sketching.

3.2 SKETCHING MESH MODELS

This section describes a sketching interface for constructing polygonal mesh models. The prototype implementation is called Teddy [119]. The user interactively draws the silhouette of the target model, and the system automatically generates a plausible 3D mesh. Polygonal mesh representation is chosen instead of volume, since sketching directly specifies and operates on surfaces, and it is easy to apply global deformation to a mesh. The Teddy system is recognized as one of the first sketch-based modeling systems that extensively use gestural operations. Instead of trying to infer a 3D shape from a single static 2D line drawing, the user interactively constructs a shape, combining sketching and view control.

3.2.1 User Interface

The Teddy system emphasizes a simple pen-and-ink drawing metaphor, and most operations are done by drawing simple free-form strokes on the canvas. The user draws interactively, and

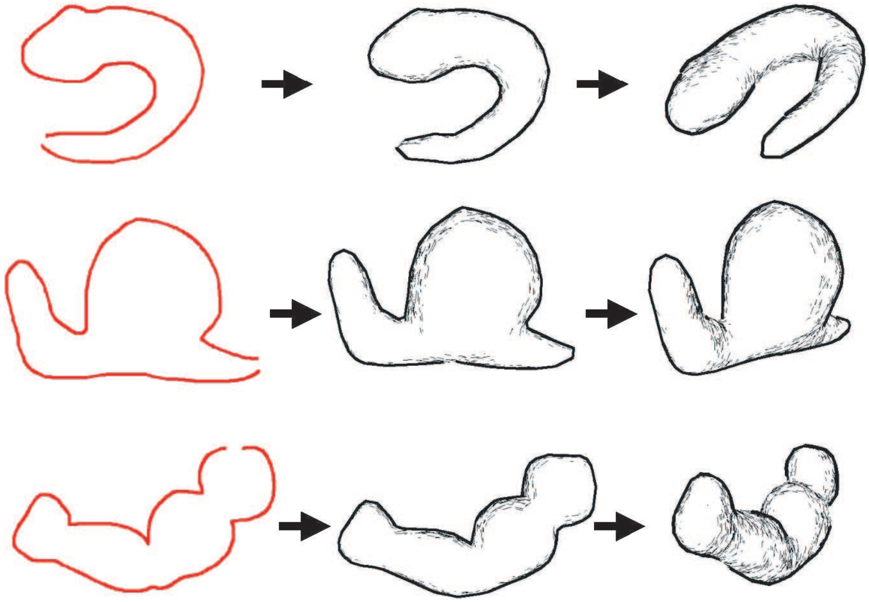

FIGURE 3.4: Creation. From [119]. © 1999 ACM, Inc. Reprinted by permission.

the system applies an appropriate operation to the model, depending on the location and shape of the stroke. The user can also change viewing angle and zoom level at any time with a simple dragging operation.

3.2.1.1 Creating a New Object

Starting with a blank canvas, the user creates a new object by drawing its silhouette as a closed free-form stroke. The system automatically constructs a 3D shape based on the 2D silhouette. Figure 3.4 shows examples of input strokes and the corresponding 3D models. The algorithm to calculate the 3D shape is described in detail in the next section. Briefly, the system inflates the closed region with the amount depending on the width of the region: that is, wide areas become fat, and narrow areas become thin.

3.2.1.2 Extrusion

Extrusion is a two-stroke operation: a closed stroke on the surface and a stroke depicting the silhouette of the extruded surface. When the user draws a closed stroke on the object surface, the system highlights the corresponding surface line in red, indicating the initiation of "extrusion mode." The user then rotates the model to bring the red surface line sideways and draws a silhouette line to extrude the surface. This is basically a sweep operation that constructs the 3D shape by moving the closed surface line along the skeleton of the silhouette. The direction of extrusion is always perpendicular to the object surface, not parallel to the screen. Users can

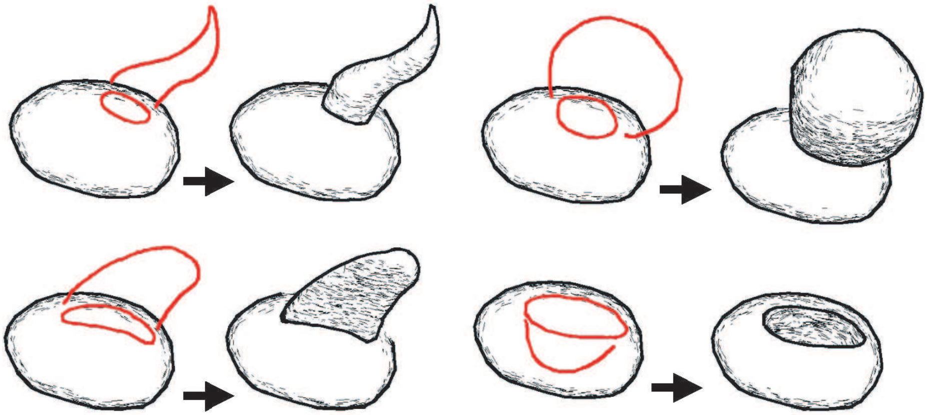

FIGURE 3.5: Extrusion. From [119]. © 1999 ACM, Inc. Reprinted by permission.

create a wide variety of shapes using this operation, as shown in Fig. 3.5. They can also make a cavity on the surface by drawing an inward silhouette.

3.2.1.3 Cutting

A cutting operation starts when the user draws a stroke that runs across the object, starting and terminating outside its silhouette. The stroke divides the object into two pieces at the plane defined by the camera position and the stroke. What is on the screen to the left of the stroke is then removed entirely (as when a carpenter saws off a piece of wood). The user can also bite the object using the same operation (Fig. 3.6).

3.2.1.4 Smoothing

One often smoothes the surface of clay models to eliminate bumps and creases. Teddy lets the user smooth the surface by drawing a scribble during "extrusion mode." The system first

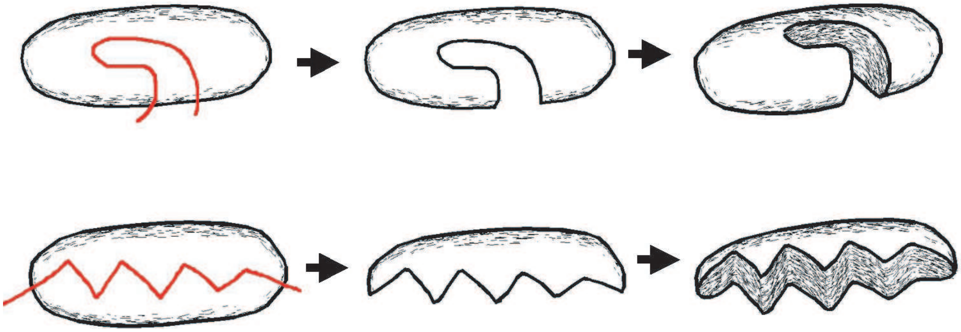

FIGURE 3.6: Cutting. From [119]. © 1999 ACM, Inc. Reprinted by permission.

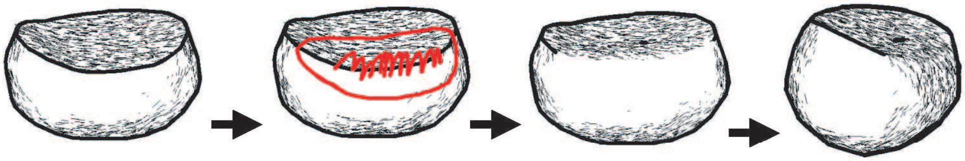

FIGURE 3.7: Smoothing. From [119]. © 1999 ACM, Inc. Reprinted by permission.

removes all the polygons surrounded by the closed red surface line and then creates an entirely new surface that covers the region smoothly (Fig. 3.7). This operation is useful for removing unwanted bumps and cavities or smoothing the creases caused by earlier extrusion operations.

3.2.1.5 Transformation

This operation distorts the model while preserving the polygonal mesh topology. It starts when the user presses the "bend" button and uses two free-form strokes called the reference stroke and the target stroke to modify the model. The system moves the vertices of the polygonal model so that the spatial relation between the original position and the reference stroke is identical to the relation between the resulting position and the target stroke. This movement is parallel to the screen, and the vertices do not move perpendicular to the screen. This algorithm is described in [122]. Transformation can be used to bend, elongate, and distort the shape (Fig. 3.8). A couple of extensions to this transformation operation have been proposed recently [123, 124].

3.2.2 Algorithms

This section describes how the system constructs a 3D polygonal mesh from the user's free-form strokes. A model is represented as a polygonal mesh. Each editing operation modifies the mesh to conform to the shape specified by the user's input strokes (Fig. 3.9). The resulting model

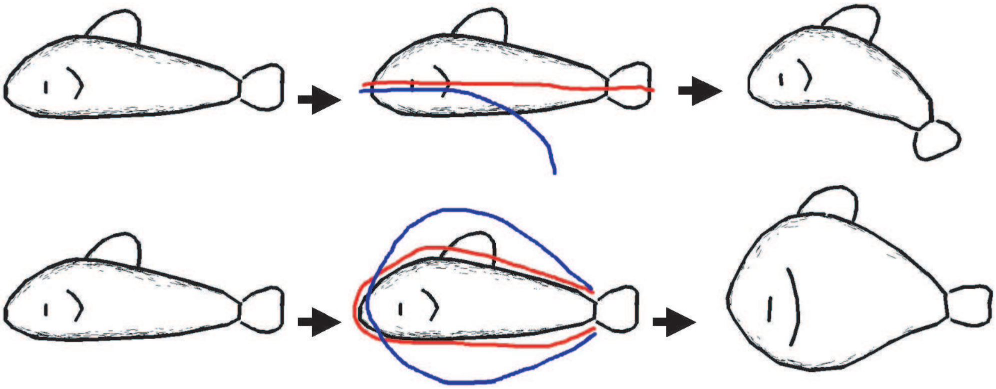

FIGURE 3.8: Transformation. (Red lines are reference stroke and blue lines are target strokes.) From [119]. © 1999 ACM, Inc. Reprinted by permission.

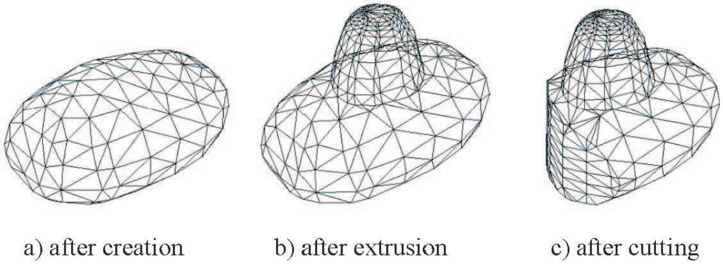

FIGURE 3.9: Internal representation: (a) after creation, (b) after extrusion, and (c) after cutting. From [119]. © 1999 ACM, Inc. Reprinted by permission.

is always topologically equivalent to a sphere. We focus on the main creation operation here. See [119] for the detail of other modeling operations.

The system creates a new closed polygonal mesh model from the initial stroke. The overall procedure is this: we first create a closed planar polygon by connecting the start point and end point of the stroke and determine the skeleton or axes of the polygon using the chordal axis introduced in [125]. We then elevate the vertices of the skeleton by an amount proportional to their distance from the polygon. Finally, we construct a polygonal mesh wrapping the skeleton and the polygon in such a way that sections form ovals. When constructing the initial closed

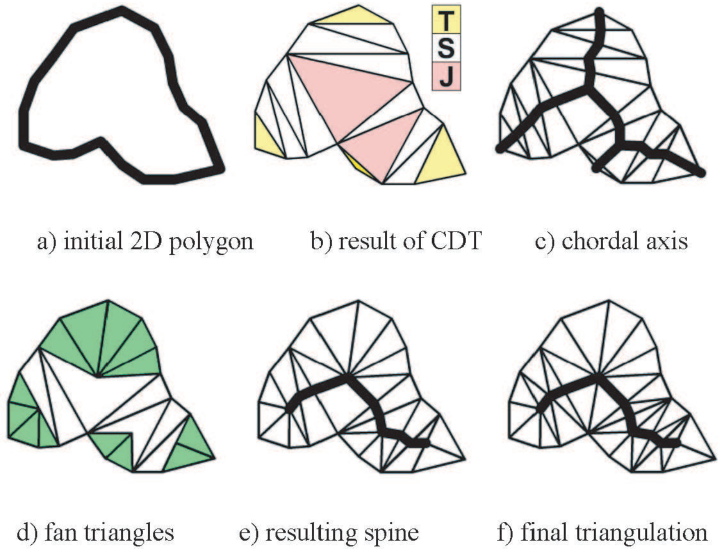

FIGURE 3.10: Finding the skeleton: (a) initial 2D polygon, (b) result of CDT, (c) chordal axis, (d) fan triangles, (e) resulting spline, and (f) final triangulation. From [119]. © 1999 ACM, Inc. Reprinted by permission.

planar polygon, the system makes all edges a predefined unit length [Fig. 3.10(a)]. The edges of this initial polygon are called external edges, while edges added in the following triangulation are called internal edges. The system then performs constrained Delaunay triangulation of the polygon [Fig. 3.10(b)]. We then divide the triangles into three categories: triangles with two external edges (terminal triangle), triangles with one external edge (sleeve triangle), and triangles without external edges (junction triangle). The chordal axis is obtained by connecting the midpoints of the internal edges [Fig. 3.10(c)], but our inflation algorithm first requires the pruning of insignificant branches and the retriangulation of the mesh.

The pruning process starts from each terminal triangle and remove triangles inward as long as the removed vertices are within a semicircle whose diameter is the interior edge. When the eating stops, the system generates fan triangles by connecting the midpoint of the last interior edge and removed vertices [Fig. 3.10(d)]. The pruned skeleton is obtained by connecting the midpoints of remaining sleeve and junction triangles' internal edges [Fig. 3.10(e)]. The next step is to subdivide the sleeve triangles and junction triangles to make them ready for elevation. These triangles are divided at the skeleton, and the resulting polygons are triangulated, so that we now have a complete 2D triangular mesh between the skeleton and the perimeter of the initial polygon [Fig. 3.10(f)].

Next, each vertex of the skeleton is elevated proportionally to the average distance between the vertex and the external vertices that are directly connected to the vertex [Figs. 3.11(a) and 3.11(b)]. Each internal edge of each fan triangle, excluding skeleton edges, is converted to a quarter oval [Fig. 3.11(c)], and the system constructs an appropriate polygonal mesh by sewing together the neighboring elevated edges, as shown in Fig. 3.11(d). The elevated mesh is copied to the other side to make the mesh closed and symmetric. Finally, the system applies mesh refinement algorithms to remove short edges and small triangles.

3.2.3 Results and Discussions

Figure 3.12 shows a couple of examples of 3D models created by novice users using Teddy. It shows that a wide variety of 3D shapes can be created by combining basic editing operations.

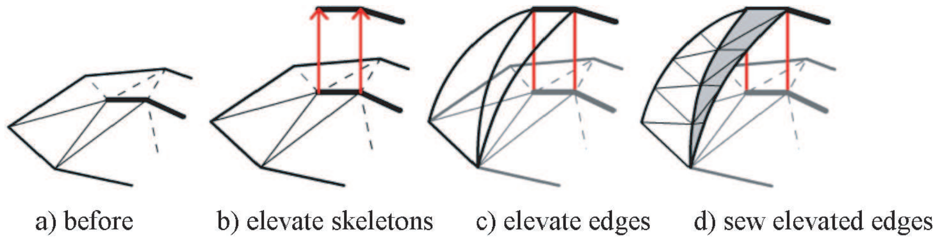

FIGURE 3.11: Polygonal mesh construction: (a) before, (b) elevate skeletons, (c) elevate edges, and (d) sew elevated edges. From [119]. © 1999 ACM, Inc. Reprinted by permission.

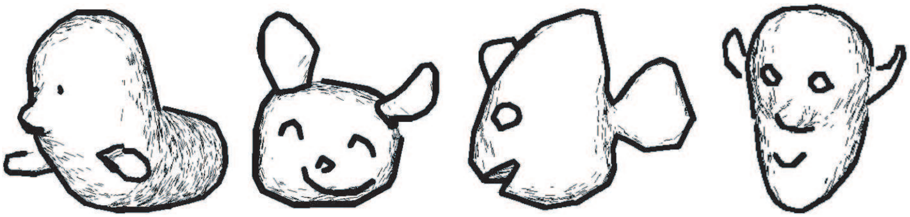

FIGURE 3.12: 3D models designed by novice users. From [119]. © 1999 ACM, Inc. Reprinted by permission.

It is important that it uses pen-and-ink style rendering to give an impression that the model is incomplete or imprecise. It hides the irregularity of the underlying mesh and allows the user to concentrate on the overall shape without worrying about the details. A problem occurs when one wants to use the resulting 3D model outside of Teddy, for example, putting them in a 3D scene designed using standard modeling software. Then, the roughness of the polygonal mesh model becomes apparent, and better surface representation becomes necessary.

The creation and extrusion algorithms described here essentially generate a surface around a skeleton, so implicit surface representation can be a natural choice. Teddy chose not to use implicit surface and directly constructed a mesh combining sequential mesh processing operations. This is mainly due to the performance reasons at the time when the system was developed (1998), but this heuristic method fails for unexpected inputs as shown in Fig. 3.13. More seriously, the polygonal mesh is not associated with any high-level semantics, and it is difficult to edit the geometry afterward in a consistent way. Different representations for sketch-based modeling to address these problems will be introduced next.

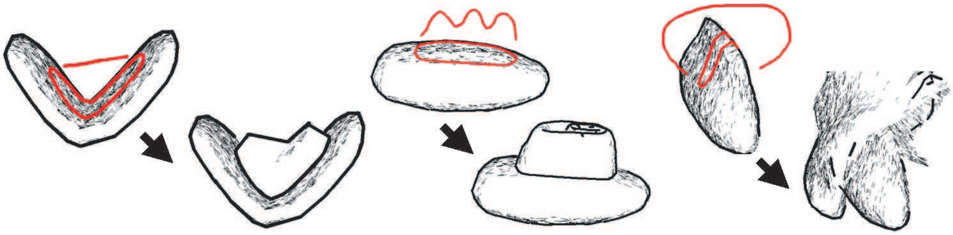

FIGURE 3.13: Unintuitive extrusion results. From [119]. © 1999 ACM, Inc. Reprinted by permission.

3.3 SKETCHING IMPLICIT MODELS

A common approach to construct a smooth surface or any topological genus is the use of implicit representation (Section 1.3.2). This section describes a sketching interface that uses such an implicit representation. Unlike the mesh-based model introduced in the previous section, the system described here computes a globally smooth 3D field function in space from the user input and then extracts an isosurface of this field. It gives a more natural definition to the surface geometry and makes it easier to add new parts to the shape while preserving smoothness.

There are two distinct approaches for defining an implicit surface. One is a variational approach that computes a surface interpolating the sketched contour via optimization [41,126]. This approach can generate globally smooth surfaces, but the optimization step is rather costly and it is difficult to obtain local control. Similarly, Schmidt [127] uses the fitting of a 2D variational curve to the sketch and conversion into a field image before inflation to 3D; the successively drawn primitives are then composed using implicit CSG operations. The other approach is to infer a skeleton from the input sketch (as in the previous section) and then construct a volumetric implicit surface thanks to the skeleton-based implicit surface formalism (Section 1.3.2).

This gives a more straightforward definition to the surface and is well suited for subsequent editing (changing thickness, skeleton-driven deformation, and so on). Alexe has shown that the use of convolution surfaces of varying radius enables us to fit the contour without any optimization step [120]. This is the approach we describe here.

The system described in this section is different from the one in the previous section in several ways. Firstly, the user input is not limited to a single stroke. The user can paint a rather complicated shape using multiple strokes before sending it to the surface construction process. Secondly, the surface is everywhere smooth, even at the junctions between the different parts. Thirdly, the user can adjust the thickness of each part. Finally, although it is not explicitly addressed in the prototype system introduced here, it is also easy to deform the model by manipulating the skeleton.

3.3.1 User Interface

The user constructs a model by progressively adding or removing parts to or from a shape. Each part is generated from a user-drawn sketch by means of inflation. The user can create either a rotund part by drawing a closed stroke or a tubular shape by drawing an open stroke. Moreover, these shapes can be made flat with no extra effort, thanks to the embedded thickness control mechanism. Figure 3.14 shows an example of this iterative sketching process.

Part construction starts with 2D sketching. The system uses a graphical tablet and a digital pen to obtain pressure and inclination information. The user sketches with as many strokes as he or she wants. As long as the strokes have not been used for reconstruction, one is free to erase and modify them.

Once the contour has been completed the user strongly presses the digital pen against the tablet, requesting 3D surface construction. The system then constructs a 3D shape component such that its projection on the screen fits the user-drawn contour [Figs. 3.15(a) and 3.15(b)].

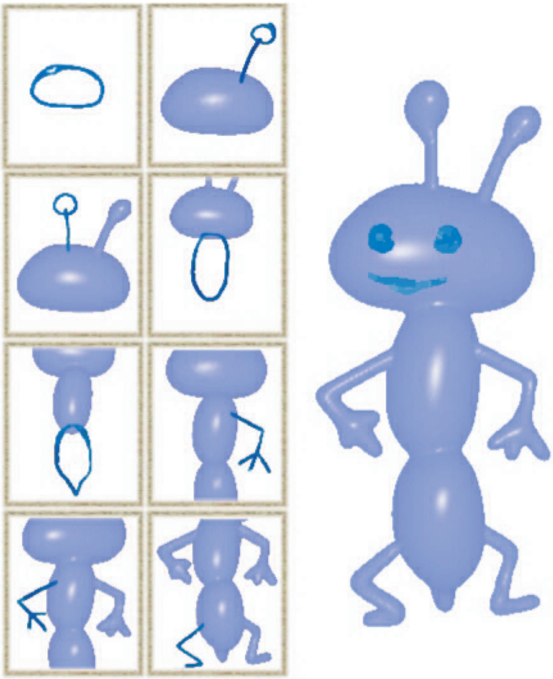

FIGURE 3.14: Progressive implicit modeling by sketching: simple implicit primitives drawn from different viewpoints are blended using a zoom-dependent sharpness (from [120]). From [120].

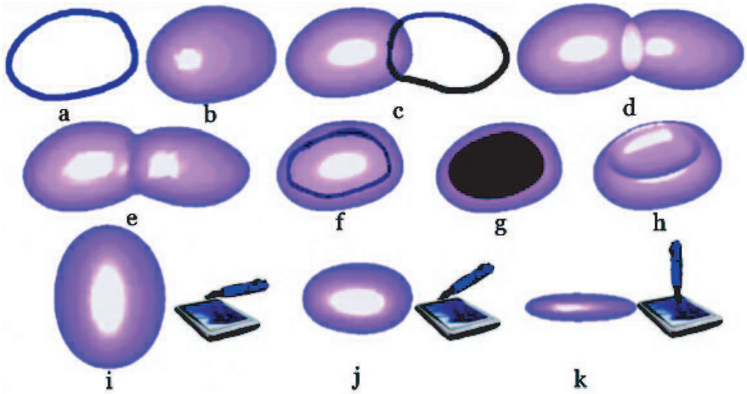

FIGURE 3.15: (a), (b) Creating a part. (c), (d), (e) Adding a part to an object. (f),(g),(h) Carving. (i),(j),(k) Thickness control (side view). From [120].

The user controls the thickness of the shape through the pen's angle when he or she presses the tablet to call reconstruction: if the pen is held orthogonal to the tablet while pressing, the shape is a flat one [Fig. 3.15(k)], otherwise it is a thicker one [Fig. 3.15(i)].

The interface for extending or carving an existing shape is the same as for creating the first component. When the user sketches over an existing surface, the point where the sketch first covers the object gives the depth of the new shape component to be constructed. When the new set of strokes is completed, the user presses the stylus on the tablet to add the shape to the existing object, or the eraser at the other end of the pen to carve it into the object. See Figs. 3.15(c)–3.15(h). Small details can be modeled by zooming in to get closer to the object. While large object parts smoothly blend with each other, the smaller components (e.g., eyes and nose of a character) are automatically composed using a detail-preserving, sharper blending.

3.3.2 Extraction of a Skeleton From 2D Sketch

The first step of the algorithm is to extract a geometric skeleton from the sketch. It gives a basis for a volumetric, implicit reconstruction of the shape.

Skeleton extraction has been studied for years in computer vision for various purposes such as shape matching and classification. The method described here is tailored to the subsequent construction of organic shapes as convolution surfaces. A popular representation for a skeleton is the medial axis (defined as the locus of centers of maximal balls included in the 2D shape), but such linear geometry can only support tubular shapes or rounded cross-sections when used as skeletons for generating implicit surfaces. In order to be able to reconstruct both flat and rounded shapes according to the user-controlled thickness, the system generates a skeleton which includes both polygonal and linear components.

When the stylus pressure indicates that the drawing is finished, the strokes' image is converted to a bitmap. Its size is decreased using a pixel averaging technique in order to smooth out the input and to reduce the amount of computation for the skeleton. The computation then proceeds as follows:

1. The object is separated from the background [Fig. 3.16(b)].
2. The weighted distance transform (WDT) is computed [Fig. 3.16(c)]. The system then detects the set of centres of maximal discs (CMDs) which will be used in skeleton extraction.
3. The object is thinned iteratively keeping all the CMDs. This produces a connected pixel graph [Fig. 3.16(d)], which is pruned to eliminate the insignificant branches. This graph provides the line segments of the skeleton.
4. The second pass is performed on the initial WDT image and thins the object without preserving the CMD, the result being an eroded shape of the object [Fig. 3.16(e)].

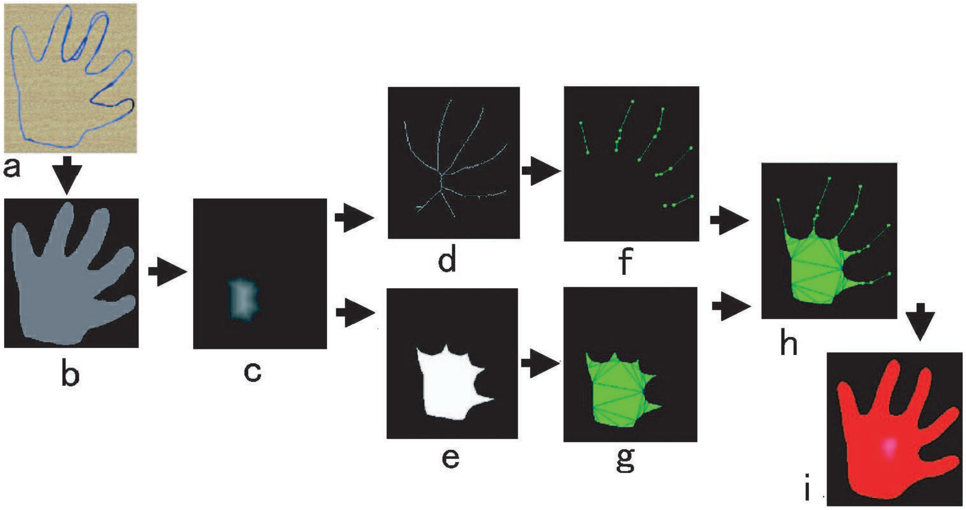

FIGURE 3.16: Algorithm overview: (a) user strokes, (b) identification of the object boundaries, (c) WDT transform, (d) iteratively thinning with preservation of the CMDs, (e) iteratively thinning without preservation of the CMDs, (f) sampling the segments, (g) sampling and triangulating the polygons, (h) final skeleton graph, and (i) final convolution surface. From [120].

This image is used to compute the skeleton's polygons, as described in the rest of the steps.

5. The image from Fig. 3.16(e) is subtracted from Fig. 3.16(d), and the result is a collection of free-form lines of one pixel width. Each line is resampled to reduce the number of segments [Fig. 3.16(f)].
6. To compute polygons, the object contour is recovered from the eroded image [Fig. 3.16(e)] and sampled [Fig. 3.16(g)]. If the image has several disconnected components, then each part produces a polygon. The polygons are split into triangles using constrained Delaunay triangulation [Fig. 3.16(g)].
7. The graph connections are computed from the two images in Figs. 3.16(f) and 3.16(g). This produces the final geometric skeleton, made of triangles and line segments [Fig. 3.16(h)].

3.3.3 Construction of a Convolution Surface

Standard distance-based implicit surfaces [11] would produce bulges at joints between skeleton components. To get a nice, smooth implicit shape from the complex skeleton generated from the sketch, a convolution surface is required. Moreover, a formulation enabling a varying radius is needed. Alexe uses Angelidis's closed-form convolution surfaces with varying radius [25]. In their method, a convolution surface is defined as an isosurface of a field function defined in space as the convolution product of a geometric skeleton function g(p) and a kernel function h(p), which smoothly varies according to weights attached to the skeleton vertices:

$$f(p) = \int_{r \in S} g(r)h(p-r)dr = c\,. \tag{3.1}$$

To fit the sketched contour, the weights have to be chosen according to the distance from the vertex to the contour (which can be directly read from the WDT image) and the level of zoom. In practice, these weights are set from a table of precomputed correspondence values, with polynomial interpolation to get intermediate values. This results in a smooth approximation of the user-drawn sketch, which avoids any unwanted oscillation of the surface [Fig. 3.16(i)].

The control of the object's thickness orthogonal to the viewing direction is achieved by providing every implicit shape with a depth-scaling factor: The field function is composed with a scale in the depth direction, the scaling factor being computed from the angle between the pen and the tablet, when the user clicks to construct the shape component. The reconstructed shape is positioned either at a default depth or at a depth computed from the existing object covered by the sketch, allowing progressive addition of details to a shape in an intuitive way.

3.3.4 Progressive Blending and Rendering

Each time the user draws a new shape component, the level of detail of the geometric skeleton remains constant in the image space, so it increases with respect to the 3D shape when the user zooms in. This zoom-dependent level of detail is also used to determine the sharpness of blending with the existing object and the size of the polygonization cell for the shape to be reconstructed.

The way the new shape blends with the previously existing object is described by a blending function. Alexe uses the functions defined by Barthe et al. [128], setting the blending parameter according to the current zoom level. This ensures that small details are added using a sharper blending than large ones. Otherwise they tend to fade out when they are composed with a much larger shape.

The polygonization of the new object part is computed and displayed immediately. If the component is to be subtracted from the existing shape it is displayed semitransparently. Meanwhile, a process in the background computes the final surface polygonization, taking the blending between surface components into account, and the user continues his modeling task. When the full polygonization is available, the final mesh is displayed, replacing the disconnected meshes. If the object has already been updated, the polygonizing process is notified, and it restarts the computation. This maintains interactive rates during the modeling process, so that the user feels free to pursue his modeling activity.

3.3.5 Results and Discussion

The system we just presented reduces user interface to sketching and navigation actions, with no need to use menus or scroll bars: all the parameters are set up automatically from the sketch and the current zoom factor; the user just has to adjust the viewpoint, sketch a shape with no restriction of topology or geometry, and click to either add or remove this new component—being careful of pen orientation to get the desired thickness. This provides an intuitive interface for quick shape prototyping: the results in Fig. 3.17 show a wide variety of free-form shapes, modeled in less than 5 minutes each.

Looking into the underlying shape representation, the method relies on the computation of a nonmanifold geometric skeleton (a graph of branching polylines and polygons) coated by a convolution surface of varying radius. This formulation yields several advantages: No optimisation step is required for fitting the 3D shape to the 2D contours, which ensures interactive performance and avoids any accidental oscillation of the reconstructed surface. Each shape component can have an arbitrary topological genus and can vary, according to the user's desire, between rounded and flat shapes. Moreover, both smooth blending and smooth removal operations can be performed. Lastly, the implicit CSG operators used include sharpness control parameters, so small features are not simply blurred into a larger shape.

One of the modeling choices is still to be discussed: this system represents the final shape analytically as a hierarchy of blended implicit primitives. This has the benefits of providing an explicit skeleton which can be used for subsequent shape edition, deformation, or animation. However, as noted in Chapter 2, this may lead to shapes which become costlier and costlier to compute and display if the sketching session lasts a long time. Thanks to the rendering process which first displays disconnected primitives, the user will still get some real-time feedback, but he or she will have to wait longer and longer before getting a smooth view of the shape.

An alternative would be to combine this way of constructing implicit primitives with the data structure described in the volumetric sculpting system presented in Chapter 2. The field function would be stored in a discrete way within a multiresolution, adaptive data structure. It would still be edited by the blending of new analytic shape components after each reconstruction step, keeping the same method for skeleton extraction and component construction while adapting the volumetric grid to the level of zoom. Such a system would be an interesting direction for future work. Note that a volumetric representation for sketch-based modeling, based on a constant resolution storage of volumetric data, was already investigated in [129] and gave good results when no sharp feature was required.

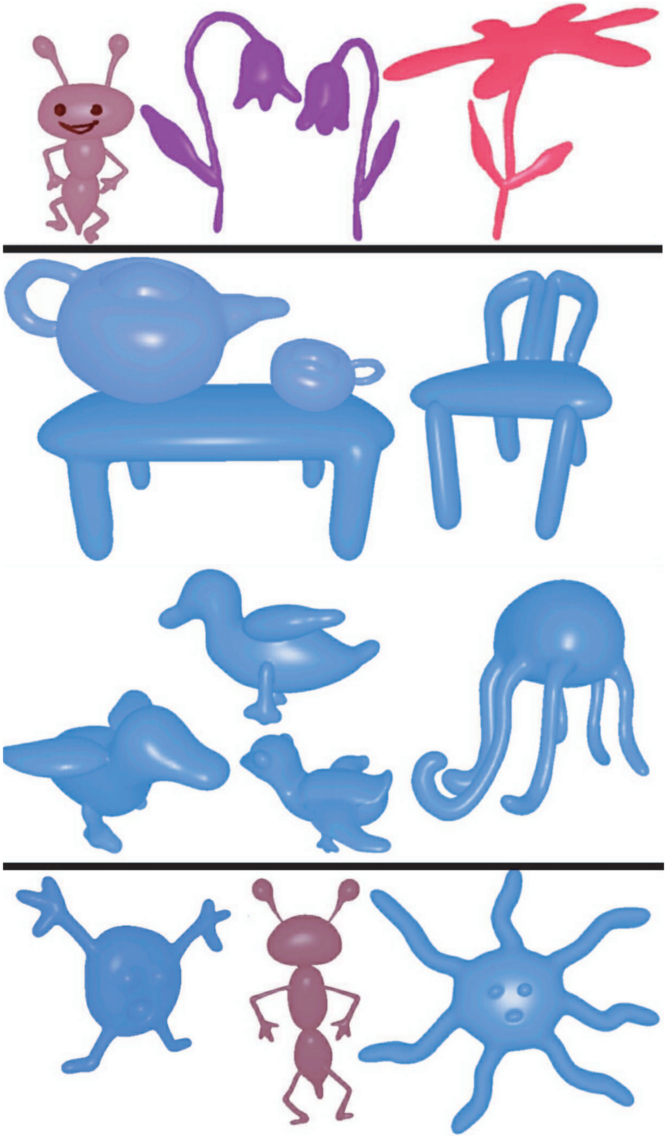

FIGURE 3.17: Objects modeled with the system. The user took 2–5 minutes for modeling each object and used three to nine strokes, each of them modeling a different object part. From [120].

3.4 SKETCHING OPTIMIZATION SURFACES

This section describes a sketch-based modeling system that uses surface optimization for the geometry construction. The implicit representation described in the previous section is defined in space, and the surface is constructed by means of isosurface extraction. In contrast, the method described in this section uses surface-based representation. It starts with a manifold mesh and moves vertices so as to minimize local variation of curvature without changing mesh topology. This process is essentially the variational modeling method described in Section 2.1.1 but is tailored for interactive modeling by using the original sketching as a means to define locality of optimization.

Surface-based representation has several advantages over implicit representation. Firstly, surface-based representation makes it possible to control the surface by means of constraints *on the surface*. This is more direct than controlling implicit surface by means of constraints along the axis. Secondly, it is easier to implement locally sharp features. Typical implicit representation is globally smooth, and it is not easy to represent sharp features. CSG operation makes it possible to represent sharp features by subtraction, but it is not easy to represent locally sharp features smoothly disappearing to smooth regions (Fig. 3.22). Thirdly, surface-based representation makes it possible that two different surface areas coexist in the same spatial location. For example, when the user defines a snake-shaped model where the head and the tail are touching, the head area and the tail area require different geometry definition, although they are at the same spatial location. It is straightforward to treat them differently in surface-based representation, but it requires special treatment when using simple implicit representation.

This section introduces a prototype implementation called FiberMesh [121]. Its interface closely follows that of the original Teddy (Section 3.2). However, unlike Teddy, the user's original stroke stays on the model surface and serves as a handle for further geometry control. The user can push and pull these curves interactively, and the surface geometry changes accordingly. In addition, the user can freely add and remove control curves on the surface. These extensions enable the design of far more elaborate shapes than those possible with sketching alone (Fig. 3.24).

3.4.1 User Interface

The system provides several modeling tools, and the user selects a tool by pressing a GUI (graphical user interface) button. A sketching tool allows the user to construct a model by drawing free-form strokes on the screen. Other tools are provided to edit control curves to obtain fine control of surface geometry.

3.4.1.1 Sketching Tool

The sketching tool is basically identical to that of Teddy. The user can create, cut, and extrude a model by drawing free-form strokes. Unlike the original Teddy, the user's original strokes stay on the surface as control curves that define the surface geometry. In the case of creation, a single loop control curve appears around the model created [Fig. 3.18(top)]. The curve is shown in blue, indicating that it is a smooth constraint (requiring normal continuity across the curve).

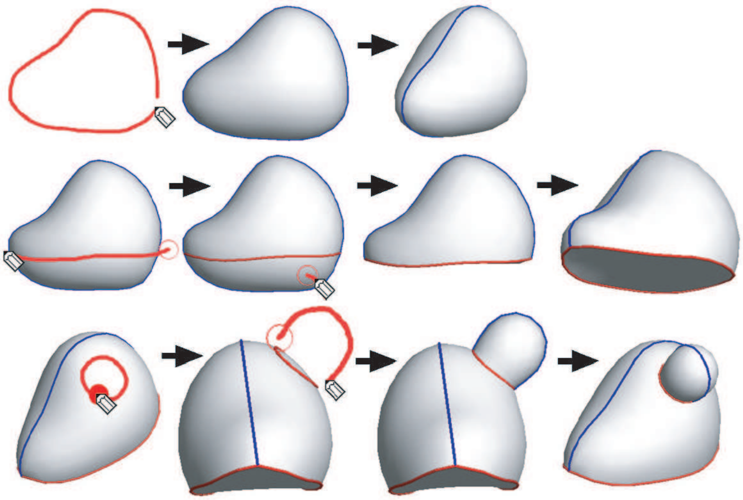

FIGURE 3.18: Sketching operations of FiberMesh, modifying geometry. From [121]. © 2007 ACM, Inc. Reprinted by permission.

In the case of a cut, a single loop control curve appears at the boundary of the cross-section [Fig. 3.18(middle)]. The curve is shown in red, indicating that it is a sharp constraint and does not require normal continuity. In the case of extrusion, a red closed curve appears at the base of extrusion and a blue open curve appears at the side of the extrusion [Fig. 3.18(bottom)].

The user can also add a new control curve at an arbitrary location by drawing a free-form stroke. When the stroke starts and ends on the surface, it becomes an open control curve on the surface [Fig. 3.19(top)]. The user can also put a closed loop control curve on the surface by drawing a closed loop on the surface [Fig. 3.19(middle)]. When the stroke starts and ends outside the surface crossing the surface in the middle, it generates a closed loop control curve on the surface beneath the stroke [Fig. 3.19(bottom)].

3.4.1.2 Deformation Tool

The deformation tool allows the user to grab a curve at any point and pull the curve in the desired direction. The curve deforms accordingly, preserving local details as much as possible [130], and the surface geometry is updated accordingly (Fig. 3.20). Editing operations are always applied to the control curves, not directly to the surface. Surface geometry is automatically defined by the optimization process, and the user is not allowed to control it directly. If the user wants more control, he or she must explicitly add new control curves on the surface. This two-step process can be seen as a bit awkward. However, if we allow the user to directly deform a surface independent of control curves, the definition of the surface becomes invisible, and it becomes

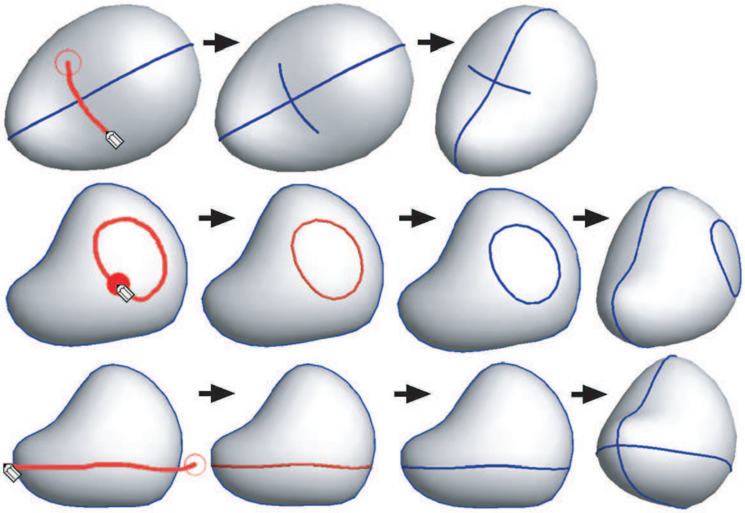

FIGURE 3.19: Sketching operations of FiberMesh, adding control curves. From [121]. © 2007 ACM, Inc. Reprinted by permission.

difficult to maintain global smoothness. Explicit addition of control curves makes the structure of the surface clearer, and each curve serves as a convenient handle for further editing.

The system uses a peeling interface for the determination of deformed area (region of interest, ROI) [130]. The size of the curve segment to be deformed is proportional to the amount of dragging. The more the user pulls, the larger the segment of the curve deformed. This frees the user from manually specifying the ROI before starting deformation and allows the user to dynamically adjust ROI during deformation. This peeling effect propagates to the other curves connected to the deformed curve and allows the user to deform larger areas of the surface.

3.4.1.3 Rubbing Tool

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

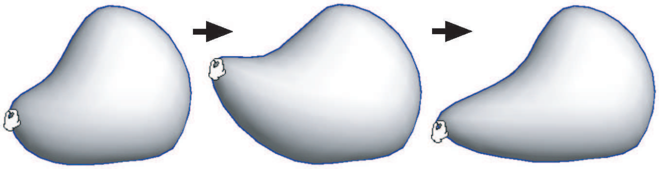

FIGURE 3.20: Pulling a curve. The deformed curve segment is determined by how much of it the user peels off. From [121]. © 2007 ACM, Inc. Reprinted by permission.

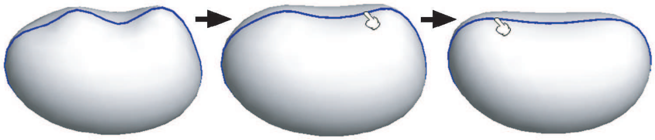

FIGURE 3.21: Rubbing (smoothing) a curve. From [121]. © 2007 ACM, Inc. Reprinted by permission.

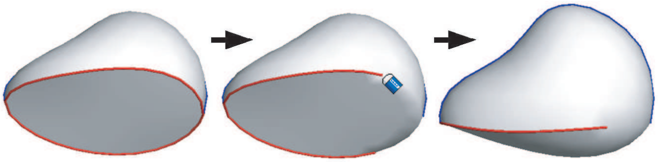

FIGURE 3.22: Erasing a control curve: (left) before erasing, (middle) immediately after erasing, and (right) after optimization. From [121]. © 2007 ACM, Inc. Reprinted by permission.

smoother the curve becomes (Fig. 3.21). This tool is very important because the curves resulting from sketching always contain some noise, and deformation can also introduce roughness to a curve. It might be possible automatically to apply denoising after each user interaction, but the amount of smoothing to apply is not clear. The rubbing tool provides an intuitive and convenient interface for specifying the target area for smoothing as well as the amount of smoothing to apply.

3.4.1.4 Erasing Tool and Type Change Tool

The erasing tool works as a standard eraser. The user drags the cursor along a control curve to erase it (Fig. 3.22). This is equivalent to removing constraints that define the surface. The system applies functional optimization on the fly as the user erases a stroke to observe the consequence of the erasing operation. The change tool is for changing the type of a control curve. Like the erasing tool, the user drags the cursor along a curve to change the property (Fig. 3.23). If the curve is a sharp curve, it converts it to smooth curve, and vice versa. As

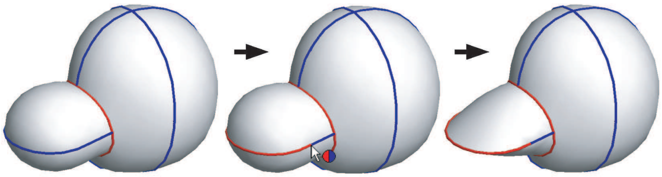

FIGURE 3.23: Changing curve type: (left) before the change, (middle) immediately after the change, and (right) after optimization. From [121]. © 2007 ACM, Inc. Reprinted by permission.

with the erasing tool, the system continuously updates the surface geometry according to the property changed and presents the result to the user.

3.4.2 Algorithm

We describe here how to compute the surface geometry for a given set of control curves. The curve geometry is given by a sketching operation and subsequent deformation operations. The topology of the surface is defined when the surface mesh is first constructed by sketching operations. The surface construction algorithm described here computes the position of mesh vertices, so that the resulting surface is locally smooth everywhere except on the red control curves. We do not describe the curve deformation algorithm here. Interested readers may refer to the original paper [121].

The system generates a fair surface that interpolates the control curves specified by the user by means of functional optimization. There are a couple of possible objective functions to choose. Welch and Witkin minimized squared mean curvatures [131], and Moreton and Sequin minimized variation of curvature [132]. Bosch compared optimization results minimizing various degrees of linear differentials and showed that higher order differentials produce better results but become unstable [133]. Based on these previous results and some experiments, minimizing the variation of curvature is chosen in FiberMesh. Since it is a nonlinear optimization problem and is too expensive to obtain an exact solution, the FiberMesh system approximates the computation in a specific way to solve it efficiently, using a fast sparse linear solver.

The method used in FiberMesh is based on a hole-filling fair surface construction method presented by Schneider and Kobbelt [134]. Their basic idea is to factorize the fourth-order problem into two second-order problems and solve them sequentially to achieve interactive speed and obtain stability. They first compute target mean curvatures that smoothly interpolate the curvatures specified at the boundary and then move the vertices to satisfy the target curvature one by one. Since prescribed curvature values are not given at the boundary, the FiberMesh system computes curvatures together with the vertex positions. Since it is a nonlinear problem, an iterative approach is adopted to quickly obtain an approximate solution. Starting from an approximate initial mesh, the current curvatures are computed first, and then the target curvature values are obtained by smoothing (diffusing) the current curvatures. The system solves the following least squares minimization problem to obtain smoothly varying scalar curvature values:

$$\arg\min_{c} \left\{ \sum_i \|L(c_i)\|^2 + \sum_i \|c_i - c_i'\|^2 \right\}, \tag{3.2}$$

where $L(\cdot)$ denotes the discrete graph Laplacian, which approximates scalar mean curvature values. The first term requires that the neighboring curvature values be close (smooth), and the second term requires that the resulting curvatures at all vertices be near the current curvatures.

To obtain a geometry that satisfies these target curvatures they use the uniformly discretized Laplacian as an estimator of the integrated mean curvature normal [135]. The integrated target Laplacian $\delta_i = A_i \cdot c_i \cdot n_i$ per vertex is given as the product of an area estimate A_i for vertex i, the target curvature c_i, and the normal n_i from the current face normals. Then new positions could then be computed by solving the following global least squares system:

$$\arg\min_{v} \left\{ \sum_{i} \|L(v_i) - \delta_i\|^2 + \sum_{i \in C} \|v_i - v_i'\|^2 \right\}, \tag{3.3}$$

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

However, the assumption that the uniformly discretized Laplacian estimates the mean curvature does not hold when the edges around a vertex are not of equal length. Rather than use a geometry-dependent discretization, which would require us to recompute the system matrix in each iteration, the system tries to achieve equal edge lengths by prescribing target edge vectors. For this, the system first computes desired scalar edge lengths, similar to the computation of desired target curvatures, by solving:

$$\arg\min_{e} \left\{ \sum_{i} \|L(e_i)\|^2 + \sum_{i} \|e_i - e_i'\|^2 \right\} \tag{3.4}$$

for a smooth set $\{e_i\}$ of target average edge lengths from the current set of the average lengths e_i' of edges incident on vertex i. Again, the iteration starts by using only the edge lengths along the given boundary curve. Note that the matrix for this linear system is identical to the system for computing target curvatures, so that one can reuse the factored matrix. From these target average edge lengths, the system derives target edge vectors for a subset B of the edges in the mesh:

$$\eta_{ij} = (e_i + e_j)/2 \cdot (v_i - v_j)/\|v_i - v_j\|. \tag{3.5}$$

Using this set of target edge vectors, they modify the linear system in Eq. 3.3 to derive the updated vertex positions as follows:

$$\begin{aligned} \arg\min_{v} \Bigg\{ & \sum_{i} \|L(v_i) - \delta_i\|^2 + \sum_{i \in C} \|v_i - v_i'\|^2 + \\ & \sum_{(i,j) \in B} \|v_i - v_j - \eta_{ij}\|^2 \Bigg\}. \end{aligned} \tag{3.6}$$

It has been found that it is sufficient only to constrain edges incident to the constrained curves, as setting the uniformly discretized Laplacian equal to vectors in the normal direction automatically improves inner fairness at all free vertices [136]. The two-step process, consisting of solving for target curvatures and edge lengths and then updating the positions, is repeated until convergence. In practice, it is observed that the computation converges rather quickly, in approximately five to ten iterations. The system needs to repeatedly solve a few sparse linear systems, but the expensive matrix factorizations are required only once at the beginning (because left-hand side matrices remain unchanged during iteration). The system only needs to run backsubstitutions during the iterations, which is very fast.

3.4.3 Results and Discussion

Figure 3.24 shows a couple of 3D models designed using FiberMesh. These detailed models cannot be created using the original Teddy system. It is also difficult to represent these surface details containing both sharp and smooth features with implicit representation. The combination of sketching and subsequent curve editing interface efficiently support the skill transfer from traditional 2D sketching to 3D modeling. the artist, Uruma, tried the system and commented, “One great thing about this system is that one can start doodling without having a specific goal in mind, as if doodling on paper. One can just create something by drawing a stroke, and then gradually deform it guided by serendipity, which is very important for creative

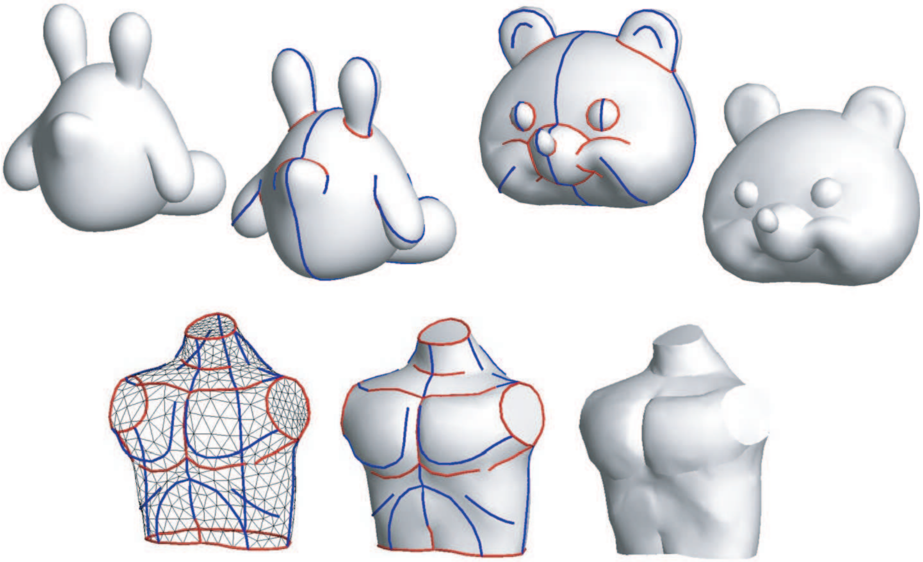

FIGURE 3.24: 3D models designed using FiberMesh.

work. Traditional modeling systems (parametric patches and subdivision surfaces) require a specific goal and careful planning before starting to work on the model, which can hinder the creative process."

The surface-based method described in this section and the skeleton-based method described in the previous section are complementary to each other. The surface-based method is good for representing surface features such as a human face but not very good with protruding structures such as arms and legs. Interesting future direction would be to combine these two methods as seen in the early work by Welch and Witkin [WW92], where they used both surface constraints and axial constraints. Using the curves as animation control is another promising future direction.

3.5 CONCLUSION: WHAT IS THE BEST REPRESENTATION FOR SKETCH-BASED MODELING?

This chapter has described three sketch-based modeling systems that use different geometrical representations: plain mesh, implicit surfaces, and optimization surface. Plain mesh is simple and fast but lacks surface smoothness and is not suitable for detailed control. Hierarchical implicit surface can generate perfectly smooth surfaces and makes subsequent deformation possible, driven by a skeleton. However, it is difficult to define features directly on the model surface. Optimization surface allows the user to directly specify the features on the surface and can continuously present a smooth surface to the user for subsequent deformations. Part of the reason for this progression is the availability of computational resources. Interactive optimization became feasible only recently thanks to ever-increasing CPU speed and fast matrix libraries.

What else to do? One interesting research direction is to consider physical properties in the modeling process. The essence of sketching interface is that the system automatically infers missing information using some assumption or some rule. Domain specific physical constraint is a natural choice, and a few experiments have been reported. We have already seen some consideration of physical properties in Section 2.3. Other existing experiments include garment draping for sketching [112], virtual garments, and simulation of stuffed materials for plush toy design [137].

CHAPTER 4

Future directions: Modeling by Gesture

The defining feature of any interactive modeling process in the real world is that the shape progressively emerges from a large series of repetitive user gestures. Even for skilled users, a first gesture is often not exactly the right one due to the physical interaction with the support paper or surface. Oversketching a stroke several times, or repeating a sculpting gesture to remove or add more material, are extremely common ways to progressively improve and refine a shape. In the real world, gestures are enhanced by the feeling of the interaction with the support paper or sculpted surface; this feeling can of course be much more difficult to communicate in the virtual context. In addition, while each real gesture has an immediate effect on the shape, it does not necessarily increase its complexity: many gestures are rather aimed at simplifying the feature lines or smoothing the shape. An important observation is that even when changing the viewpoint is possible, as in sculpting, a typical user will not do it as frequently as applying the gestures that create the shape. This is to be expected since reaching visual harmony in shape design requires us to model parts from a consistent viewing angle.

Let us discuss the ways the digital modeling systems we have reviewed exploit users' gestures. This will enable us to identify the main challenge and indicate some open directions for future research.

4.1 MODELING THROUGH ADDING AND ERASING GESTURES

Let us first discuss the interactive modeling systems which do not use any kind of deformations but rather enable the user to design a 3D shape by progressively adding matter to and removing matter from it.

If we try to identify the differences between sculpting and sketching systems of this kind, we come to the following observations:

Sculpting systems enable the user to drag volumetric tools along paths that correspond to the geometric skeleton of a shape to be added to or removed from the work. In contrast, sketching systems enable the user to sketch contours and will somehow compute a geometric skeleton from them to infer 3D. In theory, sculpting gestures can indeed be more complex, since they do not need to be planar. However, without a proper interaction with a physical surface, moving in 3D is tiresome and can be less precise: in practice, 2D gestures similar to sketching are mostly used. So, is it more effective to sketch a contour of a shape or to sketch the skeleton? Although some proper user studies would be required here, *designing shapes from contours* looks more intuitive, since it is closer to the way humans perceive real shapes.

A second difference between progressively sculpting and progressively sketching by blending primitives is that 3D sculpting systems require the user to define explicitly the positions of the 3D parts he or she adds or removes. As we have seen in Chapter 2, this either requires using a multiple DOF (depth of field) device with some force feedback or changing viewpoint to check for contacts with the already existing parts of the model. In contrast, most sketching systems automatically guess a relative position for successive primitives, with the hypothesis that when the user oversketches some existing part, the new primitive will be located at the same depth. This hypothesis has proved very effective in practice. It greatly reduces the need for changing viewpoint, which as mentioned above, helps to reach a harmonious design.

These two remarks are probably the reasons why, at least for common shapes, simple sketching systems based on 2D contours can be much faster to use than volumetric sculpting, with about the same quality of results.

4.2 ENABLING DEFORMATION GESTURES

Enabling deformations is indeed one of the main advantage of sculpting systems. In a nonvirtual modeling context, one of the most important factors that affects the artist's technique is the amount of available material. This notion is not only familiar to professional artists but also to children who play in kindergarten with dough and to adults through everyday life experience. A sculpting technique that preserves volume will take advantage of this familiarity and will hopefully be genuinely intuitive to use. As has been demonstrated by Aardman Studios in the Wallace and Gromit series, successive constant volume deformations can even be used to animate a shape, while sketching would more or less require the artist to start again from scratch and great skill to ensure consistency.

However, even if all the elements are there to provide real-time deformable models behaving as constant volume real clay, an effective method for the user to define deformation gestures is still needed. Although local deformations such as pushing and pulling material based on translation gestures can be easily controlled even with a mouse, large-scale deformations such as bending and twisting a shape are much harder to specify. They may require using, in the virtual world, an avatar object such as a sponge, so that the user has something in his hands during manipulation [138]. However, an avatar object will never be as good as the clay model, with the right shape: In real life, the mastery we have of physical interactions helps us to manipulate clay gently enough not to apply unwanted local prints but still strongly enough to deform it at a large scale. This range from local to global would probably be much harder to create in the virtual world, although extra interfaces such as interaction capture systems [139] could be used. In summary, there is still a long way to go for offering a really usable virtual clay modeling system.

An alternative open research direction would be to find out how sketches could depict deformations and try to incorporate these in sketch-based designs. To our knowledge, this is still an open issue: only local deformations based on oversketching have yet been incorporated into sketching interfaces.

4.3 TOWARD FAST AND EASY CREATION OF DIGITAL CONTENT

To conclude, let us step backward and look at the whole field of the creation of digital content: with the improvement of computer speed and memory capacities, the demand for complex digital scenes has grown higher and higher. There is an increasing gap between the scenes a computer can handle in real-time and those the artists can interactively create. This is a reason for the recent developments in capturing and processing real-shapes as well as for all the image-based techniques proposed in the field. However, relying on existing objects is more restrictive, and physically building them is much costlier than creating them as virtual objects.

We thus believe that there is some strong motivation for developing interfaces which will accelerate the creation of digital content. In addition to the problem of individual shape modeling studied in this book, specifying the relative placements of objects in a scene, or quickly indicating the amount of similarity between and variety of a large number of elements [140], are important input needed for digital content. In this context, sketching has been exploited only a little, although it has huge potential. This opens interesting challenges: the ability to interpret complex sketches drawn from a single viewpoint and the utilization of the combination of sketching and procedural methods for creating complex scenes.

Bibliography

[1] Donald D. Hoffman. *Visual Intelligence: How We Create What We See*. W.W. Norton & Company, Inc., New York, 1998.

[2] R.H. Bartels, J.C. Beatty and B.H. Barsky. *An Introduction to Spline for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, San Francisco, 1987.

[3] C.M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, CA 1989.

[4] J.D. Foley, A. van Dam, Steven K. Feiner and John F. Hughes. *Fundamentals of Interactive Computer Graphics*, second edition. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA,1990.

[5] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973. doi:10.1093/comjnl/16.2.157

[6] P. Bézier. *Numerical Control: Mathematics and Applications*, R. Forrest, Translator. John Wiley, Somerset, New Jersey, 1972.

[7] P. Bézier. General distortion of an ensemble of biparametric patches. *Computer Aided Design*, 10(2):116–120, 1978. doi:10.1016/0010-4485(78)90088-X

[8] L. Piegl. Coons-type patches. *Computers and Graphics*, 12(2):221–228, 1988. doi:10.1016/0097-8493(88)90033-7

[9] T. DeRose and C. Loop. The S-patch: A new multisided patch scheme. *ACM Transactions on Graphics*, 8(3):204–234, 1989. doi:10.1145/77055.77059

[10] Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov and Ahmad Nasri. 2003. T-splines and T-NURCCs. *ACM Transactions on Graphics*, 22, 3, 477–484. doi:10.1145/882262.882295

[11] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982. doi:10.1145/357306.357310

[12] T. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa and K. Omura. Object modeling by distribution function and a method of image generation (in Japanese). *Journal of Papers at Electronics Communication Conference'85*, J68-D(4):718–725, 1985.

[13] B. Wyvill, C. McPheeters and G. Wyvill. Data structure for soft objects. *The Visual Computer*, 4(2):227–234, August 1986. doi:10.1007/BF01900346

[14] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21, 4, ACM, 163–169. doi:10.1145/37402.37422

[15] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988. doi:10.1016/0167-8396(88)90013-1

[16] A. Pasko, V. Adshiev, A. Sourin and V. Savchenko. Function representation in geometric modeling: Concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995. doi:10.1007/BF02464333

[17] Brian Wyvill, Eric Galin and Andrew Guy. Extending the csg tree. Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, June 1999. doi:10.1111/1467-8659.00365

[18] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109–116, March 1990. doi:10.1145/91394.91427

[19] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. *Proceedings of ACM SIGGRAPH 1994*, Orlando, Florida, 269–277.

[20] B. Wyvill and G. Wyvill. Field functions for implicit surfaces. *The Visual Computer*, 5:75–82, December 1989. doi:10.1007/BF01901483

[21] Z. Kačić-Alesić and B. Wyvill. Controlled blending of procedural implicit surfaces. *Graphics Interface'91*, 236–245, Calgary, Alberta, Canada, 1991.

[22] J. Bloomenthal and K. Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, July 1991. *Proceedings of SIGGRAPH'91*, Las Vegas, Nevada, July 1991. doi:10.1145/127719.122757

[23] Andrei Sherstyuk. Kernel functions in convolution surfaces: A comparative analysis. *The Visual Computer*, 15(4):171–182, 1999. doi:10.1007/s003710050170

[24] Andrew Guy and Brian Wyvill. Controlled blending for implicit surfaces. *Implicit Surfaces'95*, 107–112, Grenoble, France, April 1995.

[25] Alexis Angelidis and Marie-Paule Cani. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. *Solid Modelling and Applications*, Saarbrucken, Germany, June 2002. ACM Press.

[26] David R. Forsey and Richard H. Bartels. Hierarchical B-spline refinement. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22, 4, ACM, 205–212. doi:10.1145/378456.378512

[27] Alex Yvart, Stefanie Hahmann and Georges-Pierre Bonneau. Hierarchical triangular splines. *ACM Transactions on Graphics*, 24(4):1374–1391, 2005. doi:10.1145/1095878.1095885

[28] Weiyin Ma. Subdivision surfaces for CAD—An overview. *Computer-Aided Design*, 37(7):693–709, 2005. doi:10.1016/j.cad.2004.08.008

[29] Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. *Proceedings of ACM SIGGRAPH 1998*, Orlando, Florida, 395–404.

[30] Adi Levin. Interpolating nets of curves by smooth subdivision surfaces. *Proceedings of ACM SIGGRAPH 1999*, Los Angeles, California, 57–64.

[31] Henning Biermann, Adi Levin and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. *Proceedings of ACM SIGGRAPH 2000*, New Orleans, Louisiana, 113–120.

[32] Denis Zorin and Daniel Kristjansson. Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer*, 18(5–6):299–315, 2002. doi:10.1007/s003710100149

[33] Eric J. Stollnitz, Tony D. DeRose and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, San Francisco, 1996.

[34] Michael Lounsbery, Tony D. DeRose and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997. doi:10.1145/237748.237750

[35] Tony DeRose, Michael Kass and Tien Truong. Subdivision surfaces in character animation. *Proceedings of ACM SIGGRAPH 1998*, Orlando, Florida, 85–94.

[36] Hugues Hoppe. Progressive meshes. *Proceedings of ACM SIGGRAPH 1996*, New Orleans, Louisiana, 99–108.

[37] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of ACM SIGGRAPH 1997*, Los Angeles, California, 209–216.

[38] L. Kobbelt, S. Campagna, J. Vorsatz and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Proceedings of ACM SIGGRAPH 1998*, Orlando, Florida, 105–114.

[39] Jonathan Cohen, Marc Olano and Dinesh Manocha. Appearance-preserving simplification. *Proceedings of ACM SIGGRAPH 1998*, Orlando, Florida, 115–122.

[40] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers and Graphics*, 28(6):801–814, December 2004. doi:10.1016/j.cag.2004.08.009

[41] Greg Turk and James F. O’brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, October 2002. doi:10.1145/571647.571650

[42] Shachar Fleishman, Daniel Cohen-Or, Marc Alexa and Cláudio T. Silva. Progressive point set surfaces. In John C. Hart, Editor, *ACM Transactions on Graphics*, Vol. 22(4), 997–1011. ACM Press, 2003. doi:10.1145/944020.944023

[43] Marc Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2–3):105–114, 2003.

[44] Olga Sorkine. Laplacian mesh processing. In Yiorgos Chrysanthou and Marcus Magnor, Editors, *STAR Proceedings of Eurographics 2005*, Dublin, Ireland, 53–70, September 2005. Eurographics Association.

[45] Matthias Zwicker, Mark Pauly, Oliver Knoll and Markus H. Gross. 2002. Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21, 3, 322–329. doi:10.1145/566654.566584

[46] R. Szeliski and D. Tonnesen. Surface modeling with oriented particles systems. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26, 2, ACM, 185–194. doi:10.1145/142920.134037

[47] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian R ssl and Hans-Peter Seidel. Laplacian surface editing. *Eurographics Symposium on Geometry Processing*, Nice, France, 2004, 179–188.

[48] Yaron Lipman, Olga Sorkine, Marc Alexa, Daniel Cohen-Or, David Levin, Christian Rössl and Hans-Peter Seidel. Laplacian framework for interactive mesh editing. *International Journal of Shape Modeling*, 11(1):43–62, 2005. doi:10.1142/S0218654305000724

[49] Dominique Bechmann and James Gain. Point, curve, surface, volume-based spatial deformation. *Internal Report*, Strasbourg University, in press.

[50] T.I. Vassilev. Interactive sculpting with deformable non-uniform b-splines. *Computer Graphics Forum*, 16(4):191–199, 1997.

[51] Hiromasa Suzuki, Yusuke Sakurai, Takashi Kanai and Fumihiko Kimura. Interactive mesh dragging with an adaptive remeshing technique. *The Visual Computer*, 16(3–4):159–176, 2000. doi:10.1007/s003710050205

[52] Denis Zorin, Peter Schröder and Wim Sweldens. Interactive multiresolution mesh editing. *Proceedings of ACM SIGGRAPH 1997*, Los Angeles, California, 259–268.

[53] D. Terzopoulos and H. Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, April 1994. doi:10.1145/176579.176580

[54] A.-H. Barr. Global and local deformations of solid primitives. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18, 3, ACM, 21–30. doi:10.1145/964965.808573

[55] T.-W. Sederberg and S.-R. Parry. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, 20, 4, ACM, 151–160. doi:10.1145/15886.15903

[56] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24, 4, ACM, 187–193. doi:10.1145/97880.97900

[57] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. *Proceedings of ACM SIGGRAPH 1996*, New Orleans, Louisiana, 181–188.

[58] Kazuya G. Kobayashi and Katsutoshi Ootsubo. t-FFD: Free-form deformation by using triangular mesh. *ACM Symposium on Solid Modeling and Applications,* Seattle, Washington, 2003, 226–234. doi:10.1145/781606.781641

[59] Yu-Kuang Chang and Alyn P. Rockwood. A generalized de Casteljau approach to 3D free-form deformation. *Proceedings of ACM SIGGRAPH 1994,* Orlando, Florida, 257–260.

[60] Francis Lazarus, Sabine Coquillart and Pierre Jancène. Axial deformations: An intuitive deformation technique. *Computer-Aided Design*, 26(8):607–613, 1994. doi:10.1016/0010-4485(94)90103-1

[61] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. *Proceedings of ACM SIGGRAPH 1998*, Orlando, Florida, 405–414.

[62] Paul Borrel and Dominique Bechmann. Deformation of n-dimensional objects. *ACM Symposium on Solid Modeling and Applications,* Austin, Texas, 1991, 351–369.

[63] P. Borrel. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics*, 13(2):137–155, April 1994. doi:10.1145/176579.176581

[64] Laurent Moccozet and Nadia Magnenat-Thalmann. Dirichlet free-form deformations and their application to hand simulation. *Proceedings of Computer Animation,* 93–102, Geneva, 1997. IEEE Computer Society.

[65] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005. doi:10.1111/j.1467-8659.2005.00886.x

[66] Ari Rappoport, Alla Sheffer and Michel Bercovier. Volume-preserving free-form solids. *IEEE Transactions on Visual Computer Graphics*, 2(1):19–27, 1996. doi:10.1109/2945.489383

[67] Fabrice Aubert and Dominique Bechmann. Volume-preserving space deformation. *Computers & Graphics*, 21(5):625–639, 1997. doi:10.1016/S0097-8493(97)00040-X

[68] Gentaro Hirota, Renee Maheshwari and Ming C. Lin. Fast volume-preserving free form deformation using multi-level optimization. *ACM Symposium on Solid Modeling and Applications,* Ann Arbor, Michigan, 1999, 234–245.

[69] W. Welch and A.P. Witkin. Variational surface modeling. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26, 2, ACM, 157–166. doi:10.1145/142920.134033

[70] W.M. Hsu, J.F. Hughes and H. Kaufman. Direct manipulation of free-form deformations. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26, 2, ACM, 177–184. doi:10.1145/142920.134036

[71] Gabriel Taubin. A signal processing approach to fair surface design. *Proceedings of ACM SIGGRAPH 199*5, Los Angeles, California, 351–358.

[72] Denis Zorin. Curvature-based energy for simulation and variational modeling. *Shape Modeling International 2005*, 198–206, Cambridge, Massachusetts, 2005. IEEE Computer Society.

[73] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo and Heung-Yeung Shum. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics*, 23, 3, 644–651. doi:10.1145/1015706.1015774

[74] Rhaleb Zayer, Christian Rossl, Zachi Karni and Hans-Peter Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24(3):601–609, 2005. doi:10.1111/j.1467-8659.2005.00885.x

[75] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo and Heung-Yeung Shum. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Transactions on Graphics,* 24, 3, 496–503. doi:10.1145/1073204.1073219

[76] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo and Heung-Yeung Shum. 2006. Subspace gradient domain mesh deformation. *ACM Transactions on Graphics,* 25, 3, 1126–1134. doi:10.1145/1141911.1142003

[77] Alon Raviv and Gershon Elber. Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design*, 32(8–9):513–526, August 2000. doi:10.1016/S0010-4485(00)00039-7

[78] Kevin T. McDonnell, Hong Qin and Robert A. Wlodarczyk. Virtual clay: A real-time sculpting system with haptic toolkits. *ACM Symposium on Interactive 3D Graphics*, Research Triangle Park, North Carolina, 2001, 179–190. doi:10.1145/364338.364395

[79] Kevin T. McDonnell and Hong Qin. Dynamic sculpting and animation of free-form subdivision solids. *The Visual Computer*, 18(2):81–96, 2002. doi:10.1007/s003710100138

[80] A. Pasko, V. Savchenko and A. Sourin. Synthetic carving using implicit surface primitives. *Computer Aided Design*, 33(5):379–388, 2001. doi:10.1016/S0010-4485(00)00129-9

[81] Alexei Sourin. Functionally based virtual computer art. *ACM Symposium on Interactive 3D Graphics,* Research Triangle Park, North Carolina, 2001, 77–84. doi:10.1145/364338.364369

[82] T.A. Galyean and J.F. Hughes. Sculpting: An interactive volumetric modeling technique. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25, 4, ACM, 267–274. doi:10.1145/127719.122747

[83] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. *ACM Symposium on Interactive 3D Graphics*, Monterey, California, 1995, 151–156. doi:10.1145/199404.199430

[84] R.S. Avila and L.M. Sobierajski. A haptic interaction method for volume visualization. *Proceedings of Visualization'96*, 197–204, San Francisco, 1996. IEEE Computer Society.

[85] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of ACM SIGGRAPH 2000*, New Orleans, Louisiana, 249–254.

[86] Ronald N. Perry and Sarah F. Frisken. Kizamu: A system for sculpting digital characters. *Proceedings of ACM SIGGRAPH 2001*, Los Angeles, California, 47–56.

[87] J. Andreas Bærentzen and Niels Jørgen Christensen. Volume sculpting using the level-set method. *Shape Modeling International 2002*, 175–182, Banff, Canada, 2002. IEEE Computer Society.

[88] Eric Ferley, Marie-Paule Cani and Jean-Dominique Gascuel. Practical volumetric sculpting. *The Visual Computer*, 16(8):469–480, December 2000. A preliminary version of this paper appeared in *Implicit Surfaces'99*, Bordeaux, France, September 1999. doi:10.1007/PL00007216

[89] Eric Ferley, Marie-Paule Cani and Jean-Dominique Gascuel. Resolution adaptive volume sculpting. *Graphical Models (GMOD)*, 63:459–478, March 2002. Special issue on Volume Modelling. doi:10.1006/gmod.2001.0558

[90] Renaud Blanch, Eric Ferley, Marie-Paule Cani and Jean-Dominique Gascuel. Non-realistic haptic feedback for virtual sculpture. Technical Report RR-5090, INRIA, U.R. Rhone-Alpes, January 2004. Projet EVASION, Theme 3.

[91] Sébastian Druon, André Crosnier and Loïc Brigandat. Efficient cellular automata for 2D/3D free-form modeling. *WSCG, 2003,* Plzen-Bory, Czech Republic.

[92] M.-P. Cani and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):39–50, March 1997. Published under the name Marie-Paule Cani-Gascuel. doi:10.1109/2945.582343

[93] David Tonnesen. Modeling liquids and solids using thermal particles. *Graphics Interface'91*, 255–262, Calgary, Alberta, Canada, 1991.

[94] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles : A new paradigm for animating highly deformable bodies. *EG Computer Animation and Simulation'96*, Springer-ComputerScience, 61–76, 1996. Proceedings of the Eurographics Workshop in Poitiers, France, August 31–September 18, 1996.

[95] Simon Clavet, Philippe Beaudoin and Pierre Pouliny. Particle-based viscoelastic fluid simulation. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 219–228, Los Angeles, California, 2005. Eurographics Association.

[96] Mark Carlson, Peter John Mucha, R. Brooks Van Horn, III, and Greg Turk. Melting and flowing. *ACM SIGGRAPH Symposium on Computer Animation*, San Antonio, Texas, 2002, 167–174. doi:10.1145/545261.545289

[97] Tolga Goktekin, Adam W. Bargteil and James F. O'Brien. 2004. A method for animating viscoelastic fluids. *ACM Transactions on Graphics*, 23, 3, 463–468. doi:10.1145/1015706.1015746

[98] J.E. Chadwick, David R. Haumann and Richard E. Parent. Layered construction for deformable animated characters. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23, 3, ACM, 243–252. doi:10.1145/74334.74358

[99] Guillaume Dewaele and Marie-Paule Cani. Interactive global and local deformations for virtual clay. *Graphical Models*, 66:352–369, September 2004. A preliminary version of this paper appeared in the proceedings of Pacific Graphics 2003. doi:10.1016/j.gmod.2004.06.008

[100] Guillaume Dewaele and Marie-Paule Cani. Virtual clay for direct hand manipulation. *Eurographics'04* (short papers), Grenoble, 2004.

[101] James E. Gain and Patrick Marais. Warp sculpting. *IEEE Transactions on Visual Computer Graphics*, 11(2):217–227, 2005. doi:10.1109/TVCG.2005.36

[102] Wolfram von Funck, Holger Theisel and Hans-Peter Seidel. 2006. Vector field based shape deformations. *ACM Transactions on Graphics*, 25, 3, 1118–1125. doi:10.1145/1141911.1142002

[103] Alexis Angelidis, Geoff Wyvill and Marie-Paule Cani. Sweepers: Swept deformation defined by gesture. *Graphical Models (GMOD)*, 68(1):2–14, January 2006. Special issue on SMI 2004. doi:10.1016/j.gmod.2005.08.002

[104] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill and Scott King. Swirling-sweepers: Constant volume modeling. *Graphical Models (GMOD)*, 68(4):324–332, July 2006. Special issue on PG'04. doi:10.1016/j.gmod.2005.11.001

[105] Marc Alexa. 2002. Linear combination of transformations. *ACM Transactions on Graphics*, 21, 3, 380–387. doi:10.1145/566654.566592

[106] John Willats. *Art and Representation.* Princeton University Press, 1997.

[107] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24, 4, ACM, 197–206. doi:10.1145/97880.97901

[108] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. *ACM Symposium on Interactive 3D Graphics*, Atlanta, Georgia, 1999, 135–140. doi:10.1145/300523.300539

[109] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. *Proceedings of ACM SIGGRAPH 1997*, Los Angeles, California, 415–420.

[110] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. *Proceedings of ACM SIGGRAPH 2000*, New Orleans, Louisiana, 517–526.

[111] Davi Geiger Hiroshi Ishikawa. Illusory volumes in human stereo perception. *Vision Research*, 46(1–2):171–178, 2006.

[112] Emmanuel Turquin, Jamie Wither, Laurence Boissieux, Marie-Paule Cani and John Hughes. A sketch-based interface for clothing virtual characters. *IEEE Computer Graphics & Applications*, 27:72–81, January/February 2007. doi:10.1109/MCG.2007.1

[113] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popovi and David Salesin. A sketching interface for articulated figure animation. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation,* San Diego, California, 2003, 320–328.

[114] Takashi Ijiri, Shigeru Owada, Makoto Okabe and Takeo Igarashi. 2005. Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints. *ACM Transactions on Graphics,* 24, 3, 720–726. doi:10.1145/1073204.1073253

[115] Kokichi Sugihara, *Machine Interpretation of Line Drawings.* The MIT Press, 1986.

[116] Lance Williams. Shading in two dimensions. *Graphics Interface'91*, 143–151, Calgary, Alberta, Canada, 1991.

[117] H. Lipson and M. Shpitalni. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design* 28(8):651–663, 1996. doi:10.1016/0010-4485(95)00081-X

[118] Olga A. Karpenko and John F. Hughes. 2006. Smoothsketch: 3D free-form shapes from complex sketches. *ACM Transactions on Graphics*, 25, 3, 589–598. doi:10.1145/1141911.1141928

[119] Takeo Igarashi, Satoshi Matsuoka and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. *Proceedings of ACM SIGGRAPH 1999*, Los Angeles, California, 409–416.

[120] Anca-Ileana Alexe, Loïc Barthe, Véronique Gaildrat and Marie-Paule Cani. A sketch-based modelling system using convolution surfaces. Rapport de recherche IRIT-2005-17-R, IRIT, Université Paul Sabatier, Toulouse, July 2005.

[121] Andrew Nealen, Takeo Igarashi, Olga Sorkine and Marc Alexa. 2007. Fibermesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics*, 26, 3, 41:1–41:9. doi:10.1145/1276377.1276429

[122] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26, 2, ACM, 35–42. doi:10.1145/142920.134003

[123] Andrew Nealen, Olga Sorkine, Marc Alexa and Daniel Cohen-Or. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics*, 24, 3, 1142–1147. doi:10.1145/1073204.1073324

[124] Youngihn Kho and Michael Garland. Sketching mesh deformations. *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, 2005, Washington, District of Columbia, ACM, 147–154. doi:10.1145/1053427.1053452

[125] Lakshman Prasad. Morphological analysis of shapes. *CNLS Newsletter*, 139:1–18, July 1997. LALP-97-010-139, Center for Nonlinear Studies, T-DOT, Theoretical Division, Los Alamos National Laboratory.

[126] O. Karpenko, J.F. Hughes and R. Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3):585–594, 2002. doi:10.1111/1467-8659.t01-1-00709

[127] R. Schmidt, B. Wyvill, M. Sousa and J. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. *Proceedings of the 2nd Eurographics Workshop on Sketch Based Interfaces and Modeling, 2005*, Dublin, Ireland, 53–62.

[128] Loïc Barthe, Véronique Gaildrat and René Caubet. Combining implicit surfaces with soft blending in a CSG tree. *Proceedings of CSG 98*, CSG Conference Series, Ammerdown, UK, April 17–30, 1998, 17–31, Information Geometers Ltd., April 1998.

[129] Shigeru Owada, Frank Nielsen, Kazuo Nakazawa and Takeo Igarashi. A sketching interface for modeling the internal structures of 3D shapes. *Proceedings of 3rd International Symposium on Smart Graphics*, Lecture Notes in Computer Science, July 2–4, 2003, Springer, Heidelberg, Germany, 49–57.

[130] Takeo Igarashi, Tomer Moscovich and John F. Hughes. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24, 3, 1134–1141. doi:10.1145/1073204.1073323

[131] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. *Proceedings of ACM SIGGRAPH 1994*, Orlando, Florida, 247–256.

[132] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26, 2, ACM, 167–176. doi:10.1145/142920.134035

[133] Mario Botsch. *High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes*. PhD Thesis, RWTH Aachen, Shaker Verlag, Aachen, 2005.

[134] Robert Schneider and Leif Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design*, 18(4):359–379, 2001. doi:10.1016/S0167-8396(01)00036-X

[135] Max Wardetzky, Miklos Bergou, David Harmon, Denis Zorin and Eitan Grinspun. Discrete quadratic curvature energies. *Computer-Aided Geometric Design*, 24(8–9):499–518, November 2007. doi:10.1016/j.cagd.2007.07.006

[136] Andrew Nealen, Takeo Igarashi, Olga Sorkine and Marc Alexa. Laplacian mesh optimization. *Proceedings of the 4th International Conference on Computer graphics and Interactive Techniques in Australasia and Southeast Asia*, 2006, Kuala Lumpur, Malaysia, ACM, 381–389. doi:10.1145/1174429.1174494

[137] Takeo Igarashi and Yuki Mori. 2007. Plushie: An interactive pattern design for plush toys. *ACM Transactions on Graphics*, 26, 3, 45:1–45:8.

[138] Jia Sheng, Ravin Balakrishnan and Karan Singh. An interface for virtual 3D sculpting via physical proxy. *GRAPHITE '06: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, 2006, Kuala Lumpur, Malaysia, ACM, 213–220. doi:10.1145/1174429.1174467

[139] Paul Kry and Dinesh Pai. Interaction capture and synthesis. 2006. *ACM Transactions on Graphics,* 25, 3, 872–880. doi:10.1145/1141911.1141969

[140] William Baxter and Ken ichi Anjyo. Latent doodle space. *Computer Graphics Forum*, 25(3):477–485, 2006. doi:10.1111/j.1467-8659.2006.00967.x

Author Biography

Marie-Paule Cani is a professor of Computer Science at the Grenoble Institute of Technology, France, and the head of the INRIA research group EVASION, part of Laboratoire Jean Kuntzmann, a joint lab of CNRS, INRIA and Grenoble University. She received her PhD from the University of Paris in 1990. In 1999, she was awarded membership in the Institut Universitaire de France and inn 2007, she received the national Irène Joliot Curie award to acknowledge her action towards women in Computer Science. She has served on the editorial boards of Graphical Models and the SIGGRAPH advisory board and is currently on the board of IEEE Transactions on Visualization and Computer Graphics.

Her main research interest is the creation of digital content for animated virtual worlds, which leads to developing techniques for both shape design and animation. Her contributions include the use of implicit surfaces and of constant volume, space deformations in interactive sculpting systems, 3D clothing design from sketches, physically-based models such as super-helices for hair animation and the design of layered models for efficiently animating natural scenes. In addition to research papers, she contributed to several courses and co-received the "Best course notes for a new course" award for "Strands and hair" at SIGGRAPH 2007.

Takeo Igarashi is an associate professor in the Department of Computer Science, The University of Tokyo. He was a post doctoral research associate at Brown University Computer Graphics Group, June 2000–Feb 2002. He received his PhD from the department of Information Engineering, The University of Tokyo in March, 2000. He also worked at Xerox PARC, Microsoft Research, and CMU as a student intern. His research interest is in user interface in general and current focus is on interaction techniques for 3D graphics. He received 2006 Significant New Researcher Award from ACM SIGGRAPH and Katayanagi Prize in Computer Science from CMU and Tokyo University of Technology in 2006.

Geoff Wyvill is a professor in the Department of Computer Science, University of Otago, New Zealand, and a director of Animation Research Limited who make commercial animation and animation software for sports. He has a BA in physics from Oxford University and MSc and PhD in computer science from Bradford (UK). Over many years he has encouraged scientists into artistic pursuits and artists to learn science. He has published over one hundred technical articles and papers and given numerous invited talks and courses. He has served on the editorial boards of Computer Graphics Forum and The Visual Computer and is currently on the board of Computer Animation and Virtual Worlds.

His present research interests are in virtual reality, modeling; and the analysis of musical sound by computer.