Part-Based Representation and Editing of 3D Surface Models

by

Ryan Schmidt

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Part-Based Representation and Editing
of 3D Surface Models

Ryan Schmidt
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2011

The idea that a complex object can be decomposed into simpler parts is fundamental to 3D design, so it is clearly desirable that digital representations of 3D shapes incorporate this part information. While solid modeling techniques based on set-theoretic volumetric composition intrinsically support hierarchical part-based shape descriptions, organic objects such as a human vertebra are more efficiently represented by surface modeling techniques. And although a human observer will easily identify part decompositions in surface models, the homogenous graphs of connected points and edges used in surface representations do not readily support explicit part decompositions.

In this thesis, I will develop a part-based representation for 3D surface models. In abstract mathematics, a surface part can be represented as a deformation of a Riemannian manifold. To create a practical implementation, it is necessary to define representations of the 3D part shape and the region on the target surface where the part is to be placed. To represent the part region I will develop the Discrete Exponential Map (DEM), an algorithm which approximates the intrinsic normal coordinates on manifolds. To support arbitrary part shapes I will develop the COILS surface deformation, a robust geometric differential representation of point-sampled surfaces. Based on this part definition, I will then propose the Surface Tree, which makes possible the representation of complex shapes via a procedural, hierarchical composition of surface parts, analogous to the trees used in solid modeling.

A major theme throughout the thesis is that part-based approaches have the potential to make surface design interfaces significantly more efficient and expressive. To explore this question and demonstrate the utility of my technical contributions, I present three novel modeling tools: an interactive texture design interface, a drag-and-drop mesh composition tool, and a sketch-based Surface Tree modeling environment. In addition to comparative algorithmic evaluations, and a consideration of representational capabilities, I have evaluated this body of work by publicly distributing my modeling tools. I

will close the thesis with a discussion of the extensive feedback provided by users of my drag-and-drop mesh composition tool, called meshmixer. This feedback suggests that part-based approaches have significant benefits for surface modeling.

Dedication

This thesis is dedicated to Ailidh and Grey, who make everything more fun.

Acknowledgements

I would like to recognize the many people who have had an impact on me during my time at the University of Toronto, whether by helping me with my research, keeping me motivated, or just being fun at parties.

First and foremost, my wife Ailidh and my son Grey have been constant supporters. There were lots of days where I forced myself to make some forward progress just so I could get home to them earlier. They kept my life balanced when work was threatening to take over, and they kept me smiling even when the paper rejections were rolling in. I can't imagine a better team. My parents, siblings, and extended families have also been great, and very understanding when I worked instead of coming home for holidays.

My advisor, Karan Singh, has been an excellent collaborator. He was willing to get on board with whatever I was interested in, even when it seemed crazy at first, and for that I will always be grateful. (And he's a lot of fun at parties, too.)

Ok, now on to everyone else. There are a ***lot*** of you, and I know you're just itching to dig in to this thesis, so I don't want to go on for too long. There are specific things I want to thank most of you for, but I would rather do that in person. So, I figured I would just offer to buy each of you a drink the next time we see eachother. Please don't hesitate to remind me! Here is the list, in alphabetical order. It is by no means exhaustive, and I apologize for any accidental omissions.

Anand Agrawala, Marc Alexa, Pierre Alliez, Ramtin Attar, Seok-Hyung Bae, Ravin Balakrishnan, Hanieh Bastani, Jacky Bibliowicz Shuster, Tamy Boubeker, Simon Breslav, Marcus Brubaker, Xiang Cao, Fanny Chevalier, Gerry Chu, Patrick Coleman, Mario Costa Sousa, Keenan Crane, Mike Daum, David Dearman, Jonathan Deber, Martin de Lasa, Tyler de Witt, Julien Diener, Pierre Dragicevic, George Drettakis, Patrick Dubroy, George Fitzmaurice, Eugene Fiume, Dustin Freeman, Michael Garland, Tim Gatzke, Yotam Gingold, Mike Glueck, Rhys Goldstein, Ruslana Goncharenko, Bruce Gooch, Amy Gooch, Cindy Grimm, Tovi Grossman, John Hancock, Sam Hasinoff, Aaron Hertzmann, Takeo Igarashi, Yuki Igarashi, Tobias Isenberg, Pauline Jepp, Joaqium Jorge, Dan Julius, Mike Jurka, Vangelis Kalogerakis, Azam Khan, Alex Kollipolis, Gord Kurtenbach, Kyros Kutulakos, Joe Laszlo, Robert Leclerc, Mike Lee, Julian Lepinski, Christian Lessig, Frank Liu, Noah Lockwood, Shahzad Malik, Michael Massimi, Justin Matejka, James McRae, Igor Mordatch, Nigel Morris, Tomer Moscovitch, Natalie Napier, Michael Neff, Andrew Nealen, Derek Nowrouzezahrai, Peter O'Donovan, Matt O'Toole, Luke Olsen, Mike Polowick, Domi Piturro, Abhishek Ranjan, Gonzalo Ramos, Phil Renato, Nargol Rezvani, Faramarz Samavati, Alla Sheffer, Patricio Simari, Niiti Simonds, Hyunyoung Song, Olga Sorkine, Eron Steger, Jos Stam, Masamichi Sugihara, Moiz Syed, Tony Tang, Kenshi Takayama, Alex Tessier, Yannick Thiel, Khai Truong, Michiel van de Panne, Dan Vogel, Jack Wang, Daniel Wigdor, Brian Wyvill, WeiWei Xu, Koji Yatani, John Yee.

Contents

Chapter 1

Introduction

All but the most basic natural and man-made objects are made up of a set of simpler parts (Figure 1.1). Even an object which appears to be an atomic part to the layperson - say, the volume knob on a speaker - will, to a designer, be further decomposable into bevels, fillets, revolutions and sweeps. There is extensive evidence that even at the lowest levels of our visual perception system, we understand the objects we see in terms of intrinsic parts [Hoffman and Richards 1984; Biederman 1987]. It is understandable, then, that part composition formed the basis for the earliest computer-aided design tools.

Figure 1.2 shows a simple example, where two parts - a cylinder and cube - are functionally combined to create a more complex part. This compositional approach, often referred to as *solid modeling*, provides a well-defined set of rules which, once learned, can be applied to solve a wide range of problems. For example, the composite part in Figure 1.2a can be altered by manipulating the parameters of its underlying components.

When one attempts to model more organic forms, such as those we encounter in the natural world, the limitations of compositional solid modeling quickly become apparent. We tend to see these sorts of shapes not as a set of combined volumes, but rather as a smoothly-varying surface that flows from one part to another. It is natural then that a designer would wish to create such an object by directly pushing or pulling on a virtual 3D

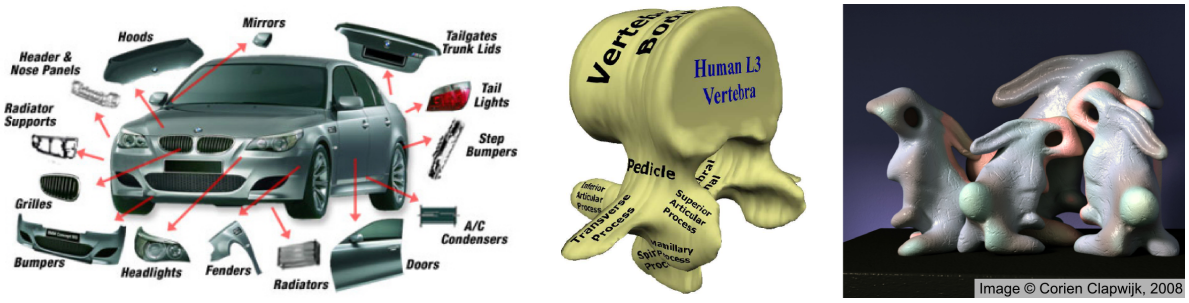


Figure 1.1: *Most objects can be broken down into sets of discrete parts. Some objects, such as the car in this figure, can be broken down into easily understood lists of parts. For many classes of objects, though, the parts are difficult to precisely define. Such parts often have no well-defined boundary; instead they smoothly blend together, resulting in a continuous object surface. How can such parts be represented in a virtual model?*

surface. Hence, *surface modeling* techniques have been developed to support this style of 3D interaction. Surface modeling tools utilize "thin-shell" digital surface representations, or *B-reps*, which can be molded into the desired form using *deformation* operations.

Surface modeling provides great power to the artist, and supports the creation of models with incredible levels of detail. However, virtually all surface modeling approaches are mathematically assembled from networks of *control points* or *handles*, which can be used to push and pull on the 3D surface. These networks are completely homogeneous; individual control points lack the sort of higher-level semantic information found in the primitives of solid modeling. As a result, the notion of underlying parts is lost.

A reasonable question is whether the idea of a *part* even makes sense in a surface modeling context. Perhaps 3D sculptors truly think of their models as holistic objects, formed in a single atomic operation. But consider the example in Figure 1.2d. If asked for a description of this shape, I believe most readers will agree that "a sphere with a bump on it" is a reasonable answer. This suggests that we do have some intuitive sense of parts - there is a sphere and a bump, and they have somehow been combined.

Consider further the description "a sphere with a bump on it". How can we convert this description into a 3D shape representation? We know what a sphere is, but the definition of a *bump* is less certain, as is the mechanism for placing it *on* the sphere. To discuss such operations in a surface modeling context, we must resort to mathematical minutia such as NURBS control points or mesh vertices. Contrast this with the similar statement "a sphere with a hole through it". Not only can we easily imagine such an object, we can unambiguously interpret it as a CSG Tree without needing to decide if we are using B-reps or implicit surfaces.

Although an artist may use language like "a sphere with a circular bump on it" to describe a 3D surface model he or she has created, the *part semantics* of the statement will not be represented in the underlying mathematical model. We have no way to talk about *where* the bump is on the sphere, or even which part of the surface is the bump. This limits our ability to predict what will happen if we replace the sphere with a cube, or *move* the bump somewhere else.

I believe that our inability to concisely describe a surface model in a representation-

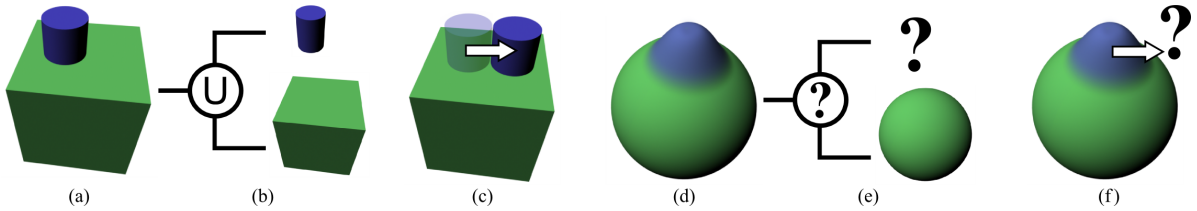


Figure 1.2: *We have a good understanding of part semantics for compositional, volume-based modeling operations. A shape like that in (a) can be described by a tree of solid modeling operations (b). Changes like "move the cylinder to the right" (c) can be expressed as modifications to this tree. But what about for surface modeling operations? Figure (d) is clearly a sphere with a bump on it, but what* is *the bump? Can we decompose this shape into a tree of operations (e)? What would it mean to "move" the bump (f)?*

independent language is a significant problem. It seems that the fundamental issue is that we lack a concrete sense of what a *part* is for surface models. As we will see, once we define a surface model PART, many of the ambiguities noted above are easily resolved. In addition, once PARTsare well-defined, it will become relatively straightforward to construct a hierarchical, procedural representation for surface models which is analogous to the CSG Tree of solid modeling.

From a practical standpoint, PART-based surface modeling will allow us to take a step towards the "holy grail" of representation-agnostic 3D modeling interfaces. To this end, I will present three novel PART-based 3D design tools. By operating at the PARTlevel, these tools easily abstract away the underlying surface mathematics, allowing the artist to focus on the artistic aspects of shape modeling tasks. The feedback from artists who have experimented with these tools has been highly positive.

1.1 Thesis Overview

In the following chapters I will explore issues relating to PART-based 3D surface modeling. In some sense, the underlying goal of my work is to convince the reader that a representation which explicitly represents a PART decomposition of a 3D surface has the capability to more effectively capture the *part semantics* of the model than traditional surface representations. These semantics include not only the PARTs of the surface, but also their relationships and the operations we might wish to apply to them.

This is of course quite a broad problem, so in the context of this thesis my goal will be more modest. My primary aim is only to determine how to *represent* a PART decomposition of a 3D surface at the data structure level. Although at some conceptual level it does seem obvious that this should be possible, existing approaches which could be interpreted as attempting to address the same problem [Forsey and Bartels 1988; Barghiel et al. 1994], involved many restrictive assumptions. Hence, at the practical level it was initially unclear that a generic PARTbased representation of 3D meshes would be possible. So, I could perhaps pose my *research question* as:

> Can the data structure underlying a modern 3D surface model, namely a single mesh of homogeneous polygons, be decomposed into a data structure based on a set of discrete PARTs and procedural composition operators, such that the product of these PARTs and operators reproduces a similar mesh?

The above is one of the major problems I will consider. However, my goal is not simply representational *capability*, but also *utility*. Hence, I would expand the above question to include the following;

> Would such a procedural PART-based surface decomposition support novel 3D shape design techniques that are not readily available without said decomposition?

The main result of this thesis is to answer the above two questions in the affirmative, at least for relatively broad classes of surface PARTs. I will describe a suite of algorithms,

data structures, and user interfaces that, in aggregate, support my claim that PART decomposition of surfaces is not only possible, but useful. To simplify the presentation of this material, I will consider the following three more basic questions:

- How can a 3D surface PART be defined and represented?
- How can surface PARTs be combined into a procedural representation?
- How can PART-based techniques be applied in 3D design tools?

As we will see in Chapter 2, there are many existing works which have to some degree explored PART-based surface modeling. For example, any spatial deformation with local support applied to a surface model implicitly defines a PART. However, these sorts of PARTs are generally transient, and will be lost when the current modeling operation is completed. Surface analysis techniques like symmetry detection and mesh segmentation are fundamentally attempts to infer the PART breakdown of a surface. To date these techniques do not infer any sort of layering relationships, leading to seemingly arbitrary segmentation boundaries on smooth surfaces. We will also consider examples where meaningful PART boundaries do not have any of the intrinsic geometric features such algorithms depend on.

To motivate a more robust definition of a surface PART, in Chapter 3 I will first consider what PARTs might be used for, in the context of current shape modeling practices. The main finding of this exploration is that certain shape modeling operations which are intuitively simple are in practice virtually impossible to carry out efficiently with current tools. For example, the simple operation of resizing the sphere that lies underneath the bump in Figure 1.2d, while keeping the size of the bump fixed, is monumentally difficult for a NURBS or SubD representation.

A small set of such examples will lead us to a set of properties that are desirable for any definition of a surface PART. An elegant way to satisfy these requirements is to utilize the *manifold* formalism, which provides a global parameterization for smooth 3D surfaces. We will then decompose a surface PART into the region of the surface that it modifies - a *domain* - and the new spatial configuration for that region - a *shape*. The PART domain can be encoded in the manifold parameterization of the underlying surface on which the PART rests, and will then be well-defined even after significant deformation of the underlying surface. Similarly, the shape of a PART can be encoded as a deformation of the portion of the underlying manifold specified by the PART domain. Hence, we will see that a well-defined surface PART involves no more than a region in a parameterization, and a deformation operation applied to that region.

Unfortunately, imposing a full manifold structure on an existing 3D surface is very difficult. The few techniques which have been developed are based on computationally intensive iterative fitting of meshes, and do not specifically handle manifold deformations. Hence, I will make the simplifying assumption that the PART domain can be bounded by a geodesic disc centered at some point $\mathbf{p}$. In this case the underlying surface can be intrinsically parameterized using the *normal coordinates* at $\mathbf{p}$, defining a local 2D coordinate system "on the surface" in which the PART domain can be stored. In Chapter 4 I will introduce the *Discrete Exponential Map* (DEM), an algorithm which approximates

normal coordinates on point-sampled surfaces. The DEM is robust, reasonably accurate, and most importantly, very efficient. I will consider several simple modifications that enhance the stability of the original DEM, and perform empirical evaluations of numerical accuracy and convergence. To demonstrate the utility of this technique, I present a simple but powerful PART-based texture design interface[1].

The second component of my PART definition involves deforming a portion of the manifold to create the intended PART shape. In Chapter 5 we will consider various methods to implement such operations on discrete surfaces. We will focus on *differential representations* which represent the desired shape as a field of local geometric information. Combined with suitable boundary conditions embedded in the underlying manifold parameterization, this approach allows the shape to be reconstructed at any location on a surface while avoiding dependency on global coordinate frames. Existing work focuses on variational approaches to the reconstruction problem, however we will see that there are several practical drawbacks to such methods. To address these issues I will introduce the COILS deformer, a geometric analog of variational techniques. Combined with the DEM parameterization, this algorithm provides an implementable surface PART definition. To explore PART-based interaction in a shape modeling context, I will present a novel *drag-and-drop* interface for editing and compositing 3D surfaces.

Given our well-defined surface PART, we can now attempt to construct a procedural surface representation with properties similar to the CSG Tree. In Chapter 6 I propose the *Surface Tree*, which can represent a complex 3D surface as a hierarchy of procedurally-combined PARTs. Unlike the CSG Tree, in which all PARTs exist in $\mathbb{R}^3$, each PART in a Surface Tree is defined in an abstract and independent two-dimensional space embedded in some underlying manifold. This complicates conceptually simple operations like inserting a node into the tree, or moving a PART to a different location on the manifold. I will describe how to carry out such operations, and also present data structures which allow a Surface Tree to be efficiently implemented. These techniques will then be put to use in a Surface Tree creation and editing tool. This interface will also be used to demonstrate some of the benefits of PART-based procedural modeling which have never before been possible on surface models, such as the linking of parameters of layered PARTs.

In Chapter 7, I will evaluate my work along two axes. The interactive tools described in Chapters 4, 5, and 6 provide an opportunity to consider the *utility* of PART-based surface modeling, particularly with respect to current 3D modeling practices. I have had some success distributing my tools to artists, and will discuss feedback gleaned from this experience. I will then consider the question of *representational capability*, focusing on the limits of my theoretical continuous-manifold PART definition, and its practical DEM+COILS implementation. We will see that the DEM is the biggest stumbling block, and I will propose some potential alternatives for future study. Other possible applications of PART-based modeling and the Surface Tree will also be considered, including animation and layered NURBS/SubD modeling.

[1] Many surface texture elements can be thought of as PARTs "in the plane"

Chapter 2

Background

A wide range of previous works have addressed PART-based shape representation to some extent, although often not explicitly. Similarly, most surface representations and modeling techniques have some procedural aspects, and hence may be adaptable for use in compositional surface hierarchies. And many geometric analysis techniques aim to detect salient PARTs that coincide with those a human observer would identify.

In this chapter I will review works that I consider to be relevant to the general problem of procedural PART-based modeling and interaction. This survey will by no means be exhaustive, but does form the basis for my understanding of the issues, and I hope will give the reader a sufficient background to follow the rest of this thesis. Background material relevant to specific chapters, such the normal coordinates and mesh parameterizations used in Chapter 4, will be provided when necessary.

Although my interest is in surface representations, I will first consider volumetric solid modeling, as it perhaps best embodies the properties of PART-based procedural modeling that I wish to develop for surfaces. After discussing these properties and some general procedural modeling concepts, I will consider the various approaches to surface modeling which have been explored, focusing on techniques that have some PART-based or procedural component. I will then briefly explore the state-of-the-art in shape analysis, which will inform the discussion of surface PARTs in the next chapter, and consider some PART-based interaction techniques which have been proposed.

2.1 Solid Modeling

Much of the early work in shape modeling was devoted to *solid modeling* techniques [Requicha and Voelcker 1983; Mantyla 1988; Hoffmann 1989]. Using a solid modeling system, the designer can construct complex models by combining simpler volumes. Solid modeling generally utilizes a tree of procedural operations known as a *CSG Tree*. Simple *primitives* lie at the leaves of the tree, and interior nodes define compositions of their inputs (which can be either primitives, or the outputs of other nodes). The *root node* of the tree defines the current model. The volume of a primitive is defined as sets of points in $\mathbb{R}^3$, and composition operations are simply set operations on $\mathbb{R}^3$, with the Boolean operators (*union*, *intersection*, and *difference* or *subtraction*) being the most common. Within

this framework any solid PART can be composed with any other, using any operator. This statement holds even if we don't know what the PARTs and operators are.

Early off-line, script-based CSG systems have evolved into powerful interactive tools, which are utilized extensively in modern engineering design. Some tools attempt to shield the designer from having to explicitly interact with the CSG Tree, by inferring the intended tree modifications from the designer's actions. The use of *regularized* Boolean operations [Foley et al. 1993] avoids anomalous cases such as surfaces with zero volume, ensuring that a *valid* solid model is produced at each node in the tree.

The description above of solid modeling as set operations on $\mathbb{R}^3$ is an abstraction which can be realized in many ways [Hoffmann and Rossignac 1996]. Functional techniques [Rvachev 1963; Ricci 1973] are perhaps the most mathematically rigorous, although more flexible boundary representations, or *B-Reps* [Requicha and Voelcker 1983], are dominant in commercial systems.

2.1.1 Implicit Modeling

The key limitation of most solid modeling systems is that they do not easily support the creation of the wide classes of shapes that have smoothly-varying surfaces. These types of surface, often referred to as "freeform" or "organic", are difficult to classify but generally have the property that they do not have obvious breakdowns into a set of geometric primitives and composition operations.

In response, implicit volumetric modeling techniques [Blinn 1982; Wyvill et al. 1986][1] have been developed to augment the standard CSG framework. Briefly, implicit volumes are defined by the iso-contours of a 3D scalar field $F(\mathbf{x})$. For example, the scalar field of a sphere could be defined by $F(x, y, z) = x^2 + y^2 + z^2$. Then a spherical *implicit surface* of radius r is defined by $F(x, y, z) = r^2$, and an *implicit volume* by $F(x, y, z) \leq r^2$. These functions can be trivially combined; for example, the union of two implicit functions can be defined by $\min(F_1, F_2)$. Similar functional compositions can be used to create smooth blends between volumes. The resulting "extended" CSG Tree, sometimes referred to as the *BlobTree*, integrates support for hierarchical blending and functional warping [Wyvill et al. 1999; Pasko et al. 1995]. Recent systems based on this framework can be used to interactively create complex, hierarchical freeform models [Schmidt 2006].

Still, this approach is based on composition of primitive shapes, which limits the designer's control over the model surface. Recent work has begun to explore ways to lift this restriction, with curve-based implicit sweeps [Schmidt 2006] and space warps that mimic surface push-and-pull [Sugihara et al. 2010]. However, the current state-of-the-art provides nowhere near the flexibility of surface representations in this respect.

One approach may be to automatically convert surface representations to volumetric formats for procedural composition, as in the Hybrid Tree [Allègre et al. 2004]. Such conversions are computationally expensive and tend to cause a loss in geometric fidelity. More importantly, converting to volumes results in volumetric-style compositions, rather than the deformation+stitching that is more in line with surface-based modeling.

[1] These representations are often grouped with other functional surface representations under the term *implicit surfaces*. Here we refer specifically to the volumetric representations.

2.2 Procedural Modeling

In the field of computer graphics, the term *procedural model* generally refers to a 3D model which is produced by a well-defined series of functions. Such functions can either take a set of parameters and output a 3D model, or take an input model, modify it, and output a new model. As in CSG, we refer to either type of function as a *node*, and the former specifically as a *primitive node*. Primitives are often understood to be the Parts of the model, although the intermediate output of any node could also reasonably be considered a Part.

In some sense, all 3D modeling is procedural, however in most cases node $n+1$ is highly dependent on node n. This dependency limits any advantages in usability or expressive power that could possibly be derived from the sequence of nodes. Utility only arises when the procedural operations can be abstracted in some useful way. For example, in CSG models defined by a binary tree of Boolean compositions, we can make high-level changes to the model by manipulating the abstract tree representation. Representations which allow a complex model to be represented in a useful abstract form have many advantages. I will refer to this property as the level of *abstraction* supported by a representation.

One of the key properties which enables abstraction is the *independence* of nodes in the procedural definition. By this I mean that the nodes are self-contained; they do not depend on any external geometric information. For example, compositions in a CSG tree take "black boxes" of points in $\mathbb{R}^3$ as input, so there are no constraints on how Parts are used. This is perhaps the most difficult property to reproduce in Part-based procedural surface modeling. Note that independence can be decomposed into technical and aesthetic components. We will see many representations where the form of a Part is tightly coupled with the state of the surface at the time of Part creation. Although technically these Parts can be transferred to another surface, the form will be lost, which provides little utility for the artist.

Another desirable property of a procedural representation is the ability to organize nodes into some meaningful *hierarchy*. Hierarchy is beneficial because it allows complexity to be encapsulated, which is important for both usability and computational efficiency. Managing complex models is much more difficult with representations that lack hierarchy. For example, the Fibermesh [Nealen et al. 2007] system generates a surface from a sparse network of 3D curves, and hence could be considered a procedural representation. To increase surface detail, more and more curves must be added to the network. Each new curve adds complexity, and ultimately the overhead of managing the curve network may limit the expressive power of the representation.

Again, virtually any sequence of editing operations can be considered a "hierarchy" (albeit a linear one). An important property of representations such as CSG is that the hierarchy can have branches. Branching is made possible by *composition* nodes that take multiple input primitives. Composability allows us to form complex Parts from simpler ones, which can themselves be used as Parts. For example, in CSG an entire branch of the model can be used as an atomic Part in another CSG Tree.

Although not completely orthogonal or complete, these properties - *abstraction*, *independence*, *hierarchy*, and *composability* - are useful axes on which to evaluate approaches to procedural modeling A somewhat more complex property, which is critical to this

thesis, is suitability for use in interactive 3D modeling tools. An infinite number of arbitrarily complex surfaces can be described using L-systems [Prusinkiewicz and Lindenmayer 1991] and other specialized procedural languages for generating geometry, but the designer must think in terms of recursive functions, scripting commands, and abstract parameters. These non-interactive techniques have been a focus of extensive work [Ebert 2002], but are outside the scope of this thesis.

2.2.1 Dependency Graphs and Construction Histories

The CSG tree is a somewhat abstract concept, but once understood, designers can easily interact with it. Each node is dependent only on its inputs, so changes to a node can automatically propagate up the tree. This type of structure is generally referred to as a *dependency graph*.

Solid modeling is only one use of dependency graphs in computer graphics. Dependency graph architectures have been applied to most major problems in computer graphics, some key examples are the ConMan visual programming environment [Haeberli 1988], the apE dataflow modeling and visualization framework [Schröder 1995], and the shader networks used in rendering systems [Cook 1984]. Commercial freeform modeling and animation packages such as Maya [Autodesk Inc. 2010e] are built on top of generalized dependency frameworks in the form of *directed acyclic graphs*, or *DAGs*. For example, in Maya, a designer can sweep one NURBS curve along another, creating an *extrusion*. Each NURBS curve, the output extrusion, and even parameters of the extrusion operation, can be represented as nodes in the DAG. Changes made to any of these nodes propagate forwards to the output extrusion. Even basic mesh-editing operations are represented in the Maya DAG.

The generalized DAGs in modern interactive systems are often automatically generated, although the user can directly manipulate them (and is usually required to do so if non-trivial behavior is desired). Such DAGs represent the *construction history* of the current model. This notion of construction history is a powerful one, with potentially far-reaching consequences for modeling capability and efficiency. However, only a few commercial systems manage to exploit this possibility to any significant effect. The main challenge lies in designing suitable visualizations which can expose construction history to the user, and interfaces which can be used to effectively manipulate the history.

Automated construction history tracking does have some limitations, in particular with respect to the manipulation of control meshes, which are common to both NURBS and subdivision modeling. Many useful editing operations are implemented such that they depend on specific control network topology (faces, iso-contours, edge-rings, etc) which currently cannot easily be generalized, resulting in undesirable constraints on the upstream DAG. Hence, although DAGs tend to support *composition* and *hierarchy*, they have limited capabilities for *abstraction* and *independence* of nodes.

2.2.2 Parametric Modeling

Dependency graphs (DAGs) are a powerful tool for procedural surface modeling. However, because abstraction is often limited, complex models in tools like Maya [Autodesk

Inc. 2010e] tend to be constructed out of many smaller, independent DAGs, rather than a single large one. A related limitation is that the one-way flow of data in a DAG limits the types of constraints which can be specified between objects. Hence, the designer may need to update many dependencies by hand if a Part of the model changes.

As a result, DAG frameworks can be cumbersome if the model must satisfy a large set of design constraints. Any design change which must propagate to many model components results in large amount of manual labour. *Parametric modeling* [Shapiro and Vossler 1995; Shah and Mantyla 1995] allows the designer to directly specify arbitrary dependencies between parameters of the model. A trivial example would be a parameterized chair, where changing the length of a leg re-scales the entire model. The key difference here is that the constraints are specified as equivalence relationships between parameters, rather than directed links, so changing the scale on some other Part of the model will update the radius of the leg.

More advanced systems support complex, possibly conflicting geometric constraints, and even engineering-analysis constraints. For example, the constraint solver can propagate changes from one Part of the chair to the others, while also ensuring that the chair both supports a maximum weight and does not become too heavy.

Parametric modeling is a very powerful concept, which is applied to great effect in commercial solid-modeling systems [Dassault Systemes / Solidworks Corp. 2010; Autodesk Inc. 2010d], but is relatively un-explored in surface modeling domains. It seems likely that this disparity is in part due to the lack of widely available procedural surface representations and modeling interfaces. Techniques for defining and solving systems of parametric constraints over arbitrary surfaces cannot be developed until Parts of surfaces can be represented parametrically.

2.2.3 Interactive Procedural Modeling

Procedural models in general have a large amount of "hidden complexity" - abstract parameters and structures that are difficult to visualize in a way that is immediately obvious to the user. Many tools simply expose this complexity in abstract views, such as traditional 2D tree widgets, dialog boxes, and large numbers of abstract parameters. This type of interface generally has a very steep learning curve, necessitating extensive and often expensive training. Hence, many systems based on DAGs or parametric modeling support some level of direct interaction with the 3D model.

Visual manipulation of a procedural node is most straightforward when the "parameters" are themselves geometric elements, such as the profile curve that generates an extrusion [Havemann 2005]. Editing the curve directly controls the procedural output. In the domain of grammar-based procedural building generation, several authors have attempted to develop interactive tools to manipulate the underlying shape grammars. The *dynamic components* module of Google SketchUp [Google Inc. 2007] allows spatial relationships and simple geometric constraints to be assigned to solid Parts using a spreadsheet-style authoring tool, and then manipulated using visual widgets. Lipp et al. [Lipp et al. 2008] describes *semantic selection* tools that interrogate the model hierarchy for Parts whose *semantic tags* match a given query. For example, all the windows who have the same "column" value can be selected, even if they are in different branches of

the model tree. The selected windows can then be edited using 3D handles, or replaced by a different window type in a shape library. Similar interfaces have been developed for editing 2D road networks [Bruneton and Neyret 2008; Chen et al. 2008].

Note, however, that most of these tools involve a set of interface elements that are highly specialized to the underlying shape grammar in use. Although smooth surfaces are sometimes generated [Havemann 2005], current interactive procedural modeling tools essentially present solid modeling interfaces to the designer.

One of the main issues in interactive procedural tools is the *persistence problem*, which roughly is the challenge of maintaining a consistent procedural model as the artist makes arbitrary changes to different components. For example, layered geometric components (windows on a wall, etc) are often positioned relative to specific faces in the model. If those faces are modified, the system may be unable to sensibly position the dependent elements. As we will see in Chapter 6, handling changes to the model hierarchy is one of the more difficult problems in Part-based procedural surface modeling.

2.3 Surface Modeling

Although implicit solid modeling can generate organic shapes, it does so at great cost, and provides at best indirect control over the surface shape. Surface-based modeling techniques have thus been developed to satisfy the demands of designers who wish to directly manipulate 3D surfaces. Although a wide range of surface representations have been developed, only the few I will now discuss have seen any measurable adoption:

Spline Patches are based on three smooth coordinate mappings from a *parameter domain* in $\mathbb{R}^2$. Perhaps the most common examples are bicubic Bezier and B-Spline patches, in which the coordinate maps are defined via a grid of *control points* in 3D [Farin 2002; Bartels et al. 1995].

Polygonal Meshes are formed by stitching together a set of 3D polygonal *faces*, creating a graph of *edges* and *vertices*. Polygonal meshes are sometimes cast as piecewise-linear spline patches.

Point Set Surfaces are defined by a set of points in 3D, to which a series of locally-supported approximation functions are fit and functionally composed [Alexa et al. 2001]. A standard example is Moving Least-Squares (MLS) surfaces [Lancaster and Salkauskas 1981; Levin 1998], based on fitting planes to ϵ-ball neighbourhoods. More complex projection operators [Guennebaud and Gross 2007; Alexa and Adamson 2009] allow for complex surfaces to be defined by a relatively small number of points.

Subdivision Surfaces are defined by progressive refinement rules applied to an initial polygonal mesh. For example, *Catmull-Clark* subdivision inserts new vertices at the center of each edge and face [Catmull and Clark 1978], and then displaces the vertices based on simple linear weights. If the initial mesh is a grid of regular quads, this subdivision process converges to the bi-cubic B-spline surface defined by the grid.

Manifold Surfaces are also based on an initial polygonal mesh, but define a set of smooth *transition functions* between parameter spaces defined on each face, edge, and vertex [Grimm and Hughes 1995; Ying and Zorin 2004]. The resulting surfaces are computationally more expensive than Subdivision surfaces, but mathematically smoother and have the added benefit of a built-in global parameterization, as I will later discuss.

These are only a few of the many surface representations which have been devised. However, the commonality between virtually all such representations is that they are ultimately defined by three components: (1) a set of 3D point samples, (2) connectivity information, and (3) a "blending" function which generates a continuous surface based on the graph of points and edges. Note that this graph may or may not have neighbourhoods which are locally planar[2]. For example, in point set surfaces connectivity is often determined via Euclidean ϵ-balls.

This simple 3D-graph representation is what gives surface modeling its power - as long as the graph remains valid for the given representation, the artist may shape and deform it as necessary. However, the natural components of the graph - faces, vertices, and edges - are too granular and homogeneous to be useful as primitives in a procedural model. Put simply, no vertex is different from any other. Hence, to define PARTs we must somehow integrate higher-level information about larger regions of surface. As we will see, the way in which surface regions are demarcated is the main differentiator between the many different approaches to procedural surface modeling.

2.3.1 Spatial Deformation

As noted above, a drawback of primitive-based constructive modeling is that the designer can only indirectly influence the shape of the model surface. In freeform modeling domains, however, artists demand the ability to directly push or pull regions of a surface. Hence, much of the work in surface modeling is focused on *deformation*. Deformation tools generally rely on some sort of physical metaphor, allowing an artist to push, pull, bend or twist the surface into a new shape.

Because of the limited interactive systems available at the time, early work in this area focused on global deformations. The general approach is to functionally "warp" the space in which the model is defined. For example, the popular Barr warps [Barr 1984] - twist, taper, and bend - are based on affine transformations parameterized over space. Such warps have many desirable properties, such as being easily invertible, which is crucial for application in implicit modeling frameworks. However, the approach is still global, and since these warps are based on a few simple parameters, direct artistic control over the model surface is still highly limited.

In contrast, *Free-Form Deformation* (*FFD*) techniques allow the designer to embed any model (in whole or in part) in precisely-defined volumetric lattice [Sederberg and Parry 1986; MacCracken and Joy 1996]. Points on the model are then represented as convex combinations of lattice elements (usually vertices), allowing deformations of the

[2]Here I mean planar in the graph theory sense, i.e. the graph can be embedded in the plane without any of the edges being forced to intersect.

lattice to be efficiently transferred to the model. Initially FFD was controlled via indirect control-point handles, however algorithmic improvements [Hsu et al. 1992] now allow the designer to directly manipulate the surface, and the control-point deformations are automatically inferred.

Lattice-based FFDs are not particularly suitable for local surface deformations because the artist must not only construct the lattice in the unconstrained space *around* the target surface, but also predict which lattice control points will produce the desired effect. A more flexible alternative is to implicitly define a deformation volume as an offset surface from freeform geometric components. The *Wires* technique [Singh and Fiume 1998] took precisely this approach - one or more arbitrary space curves is coupled with a spatial falloff field to define a deformation volume. Manipulations of the curve can then be transferred to points embedded in the volume. This general approach naturally extends to deformations controlled by polygonal mesh patches instead of curves [Singh and Kokkevis 2000], and has recently been extended to curve-based manipulation of implicit surfaces [Sugihara et al. 2010]. These techniques are well-suited to surface deformation because the geometric curve and surface handles can be created directly on the target surface, providing an interaction that *feels* like direct surface manipulation, although it is in fact a volumetric deformation. The Wires and Wrap deformers have been implemented in the commercial system Maya [Autodesk Inc. 2010e], where they have been found to be quite intuitive to artists.

Other artist-driven volumetric deformation tools have been developed, such as the recent *Sweepers* foldover-free sculpting tool [Angelidis et al. 2004b]. An extended divergence-free formulation results in volume-preserving deformation [Angelidis et al. 2004a], a concept which has been further explored [von Funck et al. 2006]. One limitation of these techniques is that, as they are based on iterative temporal decomposition and path integration, they do not define global deformation fields that can be arbitrarily sampled or easily inverted, limiting their applicability in some problem domains.

Spatial deformations are straightforward to integrate into procedural hierarchies because they are simply maps from $\mathbb{R}^3$ to $\mathbb{R}^3$. Most work in deformation does not explicitly address Part-based interaction, however in some cases we can interpret a deformation action as describing a Part. For example, the *Wires* [Singh and Fiume 1998] and *Wrap* [Singh and Kokkevis 2000] deformers combine a spatial deformation with a bounded support field. This bounded support field defines a region on the surface, and also the desired shape of this region, which, as we will see, are the two main components of my surface Part.

Since they are composable, a sequence of bounded spatial deformations is effectively a Part-based procedural surface definition. Lewis et al. [Lewis and Jones 2004] took exactly such an approach, defining a procedural model as a sequence of simple deformations applied to artist-specified spatial volumes. The primary drawback is that the model is highly constrained - although each node is technically abstract and independent, in terms of the artistic semantics it is tightly coupled with a specific input surface.

2.3.2 Displacement Maps

Perhaps the simplest procedural surface modeling technique is the *displacement map*. To create a displacement map, we must simply define a function which takes a surface point and normal and returns a *displacement vector*, which can then be added to the surface point.

Early displacement maps were simply scalar fields over the surface which were multiplied by the surface normal field [Cook 1984], although the extension to arbitrary vector fields is straightforward. Such fields can sometimes be defined analytically, but the most common approach is to store them in a texture map, which is mapped to the surface using a pre-determined parameterization. This simple technique has evolved into powerful geometric texturing algorithms [Porumbescu et al. 2005; Elber 2005] which can be used to tile arbitrarily-complex details across any parameterized surface. Recent works have used level-set [Brodersen et al. 2007] and physical simulation [Schneider et al. 2009] techniques to dynamically generate local parameterizations for interactive geometric texturing. Commercial tools for painting displacement layers have also become popular [Pixologic, Inc. 2011; Autodesk Inc. 2010f].

Displacement maps can be layered to create a procedural model. This approach is more flexible than composing spatial deformations because the orientation-independent details can be re-applied when the 3D surface is deformed. We can even create a procedural hierarchy of Parts simply by placing the displacements that describe each Part in separate layers. However, as the displacements are specified relative to the state of the surface at the time they are created, they are again coupled with a specific input surface. This is particularly an issue if the Part involves complex protruding geometry, which can become highly deformed when the underlying normal field changes. Recent work has attempted to address this problem by finding a local spatial deformation that, when coupled with the original displacements, produces a more rigid shape [Brodersen et al. 2007]. The displacement volume technique presented by Botsch et al. [Botsch and Kobbelt 2003] has a similar goal and could perhaps be adapted to local use, although it does involve a costly non-linear optimization.

2.3.3 H-Splines and Surface Pasting

Most shape modeling tools based on displacement also "bake" each displacement operation into the final model. However, one line of work has focused on representing layered displacements in a procedural hierarchy. The first step was the Hierarchical Spline (H-Spline), proposed by Forsey and Bartels [Forsey and Bartels 1988]. The Parts in H-Splines were represented by local refinements in the control point network of an initial B-Spline surface. These local refinements, called *detail overlays* by the authors, were simply a set of displacement vectors stored in the natural parameterization of the base B-Spline surface. Hence, the detail overlays were precisely local displacement maps, applied to the control point grid.

The main limitation of H-Splines was that the detail overlays had to be aligned with the B-spline control point grid, limiting the ability of the artist to control where detail was added. This lead to the development of *Surface Pasting* [Barghiel et al. 1994], perhaps the

only existing work that focuses specifically on interactive PART-based procedural surface representation Surface Pasting allows NURBS patches to be hierarchically layered onto a base spline patch. Each layer is represented as a polygon in the parameter space of the base patch, and hence can be arbitrarily shaped and oriented. The control points of the layer are defined as displacement vectors, stored in the parameter space.

While in theory capable of procedurally representing any shape, Surface Pasting has many practical limitations. Expensive spline displacement is approximated by trimming holes in the base surface and simply positioning the "paste" surface in the trimmed hole. The result is only ϵ-*continuous*, which initially often left large gaps, although a significant amount of later work has greatly reduced the gap size [Conrad and Mann 2000; Leung and Mann 2003; Siu and Mann 2003]. This trimming approach is computationally restrictive because it involves a global integration of all the pasted areas. Hence, as the hierarchy grows, the trimming graph becomes increasingly complex and expensive to compute.

A pasted layer is represented by encoding its *Greville points* using offset-vectors defined relative to the base surface frames. Hence, as with other displacement techniques, the shape of the pasted surface may become highly deformed if the base surface is significantly curved. A variety of ad-hoc methods have been developed to try to reduce this distortion, with limited success [Leung and Mann 2003; Siu and Mann 2003].

Surface Pasting produces a representation that is clearly a graph of PART nodes, represented as layered local displacement maps. Modeling tools based on Surface Pasting demonstrated some of the power of this approach, such as operations like interactive dragging of a PART across a surface [Chan et al. 1997] and interactive surface-pulling manipulations at different levels of the hierarchy [Ma 2000]. Surface Pasting has been also applied in the commercial modeling package Houdini [Side Effects Software Inc. 2007].

Surface Pasting does share the same fundamental limitation of other displacement mapping techniques, namely the dependence on a global parameterization discussed above. Hence, as PARTs are manipulated, they and any PARTs layered on top of them inherit the (frequently un-intuitive) distortions that occur in parameter space, as demonstrated by Tsang et al. [Tsang and Mann 1998]. Similarly, the dependence on a global planar parameterization prevents topology change.

2.3.4 Multiresolution Surfaces

H-Splines provided an interesting capability which had not been previously demonstrated. Although H-Spline detail overlays were in a generic sense simply displacement maps, from the point of view of the artist, they served to increase the complexity of the spline control point grid and allow for higher levels of detail. However, because the layers were encoded differentially, it was possible to manipulate the H-Spline at any grid resolution, and have the result propagate to the higher detail levels.

An extensive amount of work has been done to extend this approach to surfaces based on meshes. The basic principle is to iteratively apply refinement and displacement steps to an initial *base mesh*. A wide variety of strategies for refinement and displacement have been explored, usually based on wavelet [Lounsbery et al. 1997] or subdivision [Zorin et al. 1997] techniques which converge to smooth surfaces under infinite iterations. The resulting *multiresolution* surfaces are powerful tools which have been applied to many

Figure 2.1: *In the top row, an initial model is subdivided 5 times using the Catmull-Clark refinement rules. The final model is simply a smoothed version of the original model. In the bottom row, the same subdivision occurs, but multiresolution details are added at each step, resulting in a final model which contains features not visible in the original surface.*

problems. In particular, multiresolution Catmull-Clark [Catmull and Clark 1978] subdivision surfaces are widely used in commercial tools [Autodesk Inc. 2010e; Pixologic, Inc. 2011; Autodesk Inc. 2010f].

Many multiresolution techniques ignore the notion of local refinement found in H-Splines, instead applying global operators, although local subdivision techniques have been developed [Pakdel and Samavati 2005]. Regardless of whether the subdivision is local or global, multiresolution techniques usually take a different approach to the displacement step. Rather than representing the displacements via texture maps and parameterizations, displacements are stored for each vertex at each level of detail.

Storing displacements at vertices allows multiresolution techniques to be applied to a mesh without needing a pre-determined parameterization. This in turn allows the topology of the mesh to change during any refinement step, as in the Hybrid Mesh [Guskov et al. 2002]. And we can again construct a procedural hierarchy of PARTs simply by having separate displacement layers for each PART. But in this case, we have traded dependency on a parameterization for a tight coupling with a given mesh topology. Essentially, the region of the PART would be the set of faces with vertices that have non-zero displacements. This is not a particularly flexible representation, so to implement higher-level PART-based interactions most works embed these face sets in local or global parameterizations.

2.3.5 Manifold Surfaces

Manifold surfaces are related to multiresolution in that they define a smooth surface based on an initial polygonal mesh. But rather than via iterative refinement, a manifold is based on analytic *transition functions* between standardized parameter spaces defined at each triangle, edge, and vertex [Grimm and Hughes 1995; Grimm 2004]. Each of these

parameterizations is called a *chart*, and the set of charts is an *atlas* for the surface. The main benefit of manifolds is that they provide an intrinsic global parameterization of the surface. As I will discuss later, this in principle makes manifolds an ideal basis for a procedural surface representation, although existing manifolds are difficult to utilize in practice.

Grimm [Grimm 2005] describes one attempt to create a layered surface representation using manifolds. The charts are all defined on the sphere, parameterized via stereographic projection. To add a local mesh feature, the mesh is first embedded into the sphere, defining the set of charts that need to be inserted into the atlas. The feature vertices can then be dragged out into 3D, but are stored as relative displacement vectors. This approach provides more flexibility than H-Splines and Surface Pasting, as the added feature mesh can have any boundary shape, although the surface is still limited to the same topology as the base parameter domain (the sphere).

Like H-Splines, Grimm's PART is described by a region in a global parameterization, in this case the feature mesh embedded in the sphere. This strategy similarly inherits the limitations of methods based on a shared global parameterization, namely that the PART region is coupled to the parametric distortion that existed at the time of its creation, and hence cannot be easily transferred to another surface.

2.3.6 Variational Modeling

The general idea behind variational modeling is that the artist works in terms of fixed geometric elements - lines, curves, and so on - and a smooth interpolating surface is automatically found by minimizing global energy functionals, primarily smoothness. Variational modeling was first proposed by Welch and Witkin [Welch and Witkin 1992] for a simplified form of H-Splines, then later adapted to triangle meshes [Welch and Witkin 1994]. Although computationally intensive, recent work has linearized several of the energy functions, allowing scalable interactive tools to be developed [Nealen et al. 2007].

As variational models define a surface via a sparse set of constraint elements, they can be considered a somewhat abstract procedural representation, however to date no aspects of hierarchy or layering have been developed. So a variational model is in one sense simply a more complex form of the 3d-graph-with-blending-function used in most "raw" surface representations. But the graph elements lack the homogeneity of those in a mesh or NURBS patch - they have meaning to the designer, and hence they do represent PARTS of the model. As the variational constraints are so tightly coupled with each other and the current surface, there does not seem to be any obvious way to utilize them as independently-composable PARTS.

2.3.7 Variational Mesh Deformation

As noted above, spatial deformations are fundamentally independent of the surfaces they are meant to deform. In addition to causing problems when used to describe PARTS, this independence also limits the ability of surface deformations to preserve surface detail. The designer must carefully construct deformations to avoid introducing unwanted distortions. This issue has lead to an increasing focus on deformation techniques which are defined

relative to the surface itself [Botsch and Sorkine 2008]. These methods adapt the energies used in variational modeling, and hence have been referred to as variational deformations. To use a variational deformation, the artist first selects a region on the surface, then defines a new shape for that region by manipulating simple geometric *handles* which constrain parts of the surface.

Taking an abstract view, we see that the artist has again defined a surface region and a new shape for that region, and hence created a PART. The shape of the PART is not defined by displacement vectors, but rather by *differential* information about the surface, combined with some global energy function which is minimized to determine the PART shape.

As we saw with multiresolution modeling, the region of a PART defined via variational deformation is a set of triangles. Furthermore, the PART shape definition generally involves handles - points, curves, and so on - which are independent elements positioned in $\mathbb{R}^3$. Hence, a variational deformation is tightly coupled with a given mesh topology *and* its 3D orientation. Although they can technically be procedurally layered, PARTs described via these mesh deformations are even less independent than displacement-map PARTs, as the intent of a given set of 3D handles is lost if an underlying layer is modified.

2.4 Surface Analysis

In the previous section I explored a variety of PART-based surface representations that have been developed, and described how other representations could be adapted to PART-based modeling. However, virtually all existing surface models are not represented as a set of PARTs, but rather as using a "flat" representation. Similarly, many 3D capture technologies as range scanners or structure-from-motion will never provide a PART decomposition of the objects they find. Hence, many works in computer graphics focus on "reverse-engineering" 3D surfaces, to discover underlying structure.

2.4.1 Mesh Segmentation

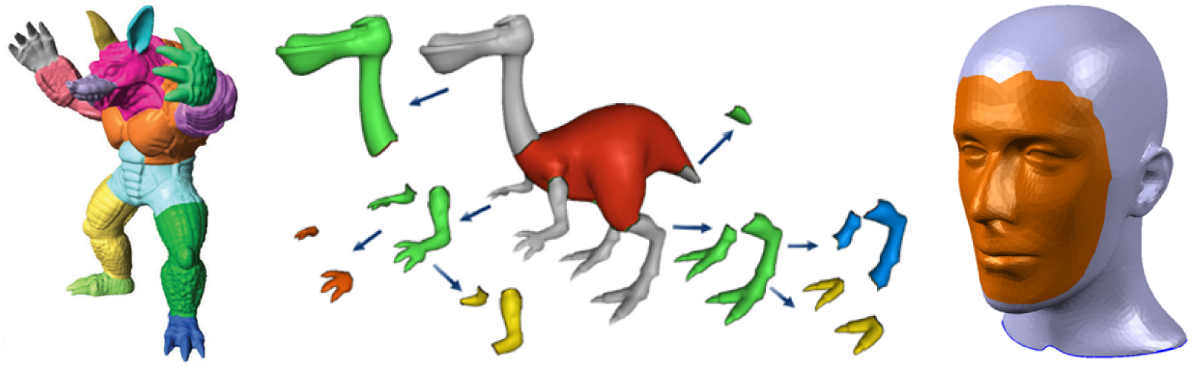
Mesh *segmentation* is the problem of dividing a given mesh into disjoint sets of faces, where each set satisfies some functional definition of a segment. Hence, the ultimate goal of most segmentation works is to construct a geometric definition of PARTs, and then automatically divide the mesh into sets of faces which best satisfy this definition. For example, one may define PARTs as being regions separated by sharp creases in the mesh, or regions that are geodesically convex, or perhaps regions that are well-approximated by simple geometric primitives [Simari and Singh 2005].

A vast number of different segmentation strategies have been developed. See Shamir [Shamir 2008] for discussion of over 50 of these algorithms, and Chen et al. [Chen et al. 2009] for a comparison of many techniques with human-authored segmentations. Some recent work has even attempted to integrate many of these different techniques using machine learning, to better-reproduce human segmentations [Kalogerakis et al. 2010].

In terms of finding PARTs, these segmentation techniques are often very effective on CAD-style models, particularly those generated via solid modeling. However, these

algorithms tend to have trouble with the smooth transitions found in organic surface models, where there is no clear boundary, but yet the algorithm insists on splitting the faces into disjoint sets. Some examples are shown on Figure 2.2. In these cases it seems more sensible that the Parts be permitted to overlap (this may be an interesting potential future direction for segmentation algorithms).

Another problem is that some relatively obvious Parts of a model may not have any salient geometric features at the boundary for an algorithm to detect. For example, the entire face in Figure 2.2c could be considered a Part, but the boundary of the face region is not demarcated by any clear changes in curvature. This face clearly has sub-Parts, so perhaps one could hierarchically define the face via grouping. Even those sub-parts, however, do not have clearly-defined boundaries - the nose smoothly blends in to the brow, for example. In general, it seems to be the case that Parts of organic surfaces blend together at all scales, and hence lack distinct geometric boundaries at which to uniquely partition the surface.



(a) (b) (c)

Figure 2.2: *Example segmentations of the standard (a) Armadillo and (b) Dino models. The boundaries between* Parts *are in some cases seemingly arbitrary, and can diverge significantly from what one might expect (for example, the tail* Part *in (b)). The boundaries of some* Parts *which have clear semantic meaning, such as the face in (c), do not appear to have any salient geometric features for a segmentation algorithm to detect.*

2.4.2 Smart Selection

Mesh segmentation techniques are generally automatic, in the sense that the user is not involved. However, many segmentation results can be improved by adding some user guidance. *Smart selection* tools take exactly this approach, with the goal of allowing the user to demarcate a Part boundary more efficiently than the brute-force strategy of selecting each face of the Part.

Funkhouser et al. [Funkhouser et al. 2004] described *intelligent scissors*, in which the user drew a stroke from a single viewpoint and a min-cut algorithm was used to extend the stroke across the back-facing triangles. Several related methods employ min-cuts, either based on strokes which roughly indicate the cut region [Lee et al. 2005], or "scribbles" which indicate the body of the desired Parts [Brown et al. 2009]. Alternately,

a hierarchical segmentation of the surface can be performed, and then the user simply has to click inside the intended Part region and roll the mouse wheel to find the desired level of segmentation granularity [Attene et al. 2008].

One limitation of these techniques is that, like segmentation, they assume that Part boundaries are demarcated by geometrically salient features. As I will show in the next chapter, this is not always the case.

2.4.3 Structure Detection

Closely related to mesh segmentation is the problem of *structure detection*, in which the focus is often on literally reverse-engineering a model to recover design intent. Generally reverse engineering tools are based on a segmentation, computed automatically or manually, followed by fitting of sets of pre-determined primitives that map to operations one would utilize in a 3D design tool [Varady et al. 1997]. Such reverse engineering tools are often manually intensive, but are widely used in industry [Singh et al. 2004; Geometry Systems Inc. 2010] to recover surface models (mainly NURBS and SubD) from point-cloud data.

Several recent works in structure detection have focused on finding symmetric portions of a surface [Simari et al. 2006; Mitra et al. 2006]. Symmetry and structural regularity are extremely strong indicators of Part structure in both man-made and natural shapes [Pauly et al. 2008]. For example, Li et al. [Li et al. 2010] do an impressive job of reverse-engineering CAD models, extracting *design intent* from a hierarchical symmetric decomposition. Another related line of work has attempted to infer model skeletons [Baran and Popović 2007] and rigid joints [Xu et al. 2009b], which are then used to better preserve model semantics during animation. In both cases the underlying algorithm must also implicitly determine a Part decomposition of the model.

Once structure in a model has been detected, many interesting procedural-style editing capabilities are available. For example, Gal et al. [Gal et al. 2009] propose an *analyze-then-edit* workflow, where symmetry and other geometric features are used to detect networks of similar feature curves. Edits to one feature curve can then be propagated to others, while maintaining salient geometric structure like circularity, right angles, and so on. Similarly, Bokeloh et al. [Bokeloh et al. 2010] automatically derive a shape grammar from an example model via symmetry detection, then can use the grammar to generate new models, a process they call *inverse procedural modeling*.

Note that to date, automated structure detection has generally only been successfully applied to CAD-style models - the sort of model one would construct with solid modeling. Organic surface models have proven significantly harder to analyze. I believe this is in large part because current attempts to decompose surface models into Partshave avoided the issue of overlapping boundaries. In the next chapter I will attempt to convince the reader that this is a critical property of parts on smooth surface models.

2.5 Part-Based Interaction Techniques

So far, I have discussed how previous works represent Parts, and how Parts can be found in existing models. However, my main interest is in how Part-based interaction can make surface modeling more efficient. In this section I will briefly review some of the Part-based interactions that have been proposed in the literature.

2.5.1 Displacement Stamping

The simple strategy of "baking" a displaced surface back into the original model is a Part-based interaction used in many interactive modeling tools [Schneider et al. 2009; Pixologic, Inc. 2011; Autodesk Inc. 2010f]. In such tools, the displacement Part is generally created independently, in some cases by editing the displacement map image itself, and then *stamped* into the target surface. Displacement stamping is also used widely in semi-interactive procedural geometric texturing tools, see Landreneau and Schaefer [Landreneau and Schaefer 2010] for a recent example and survey of related techniques. One drawback of standard displacements is that they can easily self-intersect or stretch in undesirable ways. Brodersenet al. [Brodersen et al. 2007] optimizes local volumetric parameterizations to minimize this possibility.

2.5.2 Displacement Copy-and-Paste

Existing surface-based copy-and-paste techniques involve an interaction similar to displacement stamping, however the displacement Part source is some existing surface region. For example, Biermann et al. [Biermann et al. 2002], described a tool which copies detail vectors from one multiresolution surface to another. The basic idea is that the artist selects the region containing the desired Part, and the tool automatically separates the relevant detail vectors from the base surface. Then an automatically-computed local parameterization is used to transfer the detail vectors to the new surface.

This technique was extended to general meshes by Fu et al. [Fu et al. 2004], who flattened the Part to the plane using conformal planar parameterization [Desbrun et al. 2002] and then encoded each vertex using relative displacement vectors. Note that for Parts with non-disc topology, this parameterization approach is somewhat of a hack, and there is no mathematical guarantee that it will properly function.

Local differential vectors can be seen as a generalization of displacement vectors. Sorkine et al. [Sorkine et al. 2004] describe a differential copy-and-paste technique which transfers Laplacian vectors to another mesh based on compatible local parameterizations.

2.5.3 Part Fusion

Displacement copy-and-paste techniques are generally based on copying information from one surface region to another, using local tangent-normal frames to resolve orientation changes. The main limitation of this approach is that the surface variations over the regions affects the copied shape. Another type of Part-based interaction has been pro-

posed by a variety of authors [Kanai et al. 1999; Sorkine et al. 2004; Yu et al. 2004; Funkhouser et al. 2004; Sharf et al. 2006], which I will call Part *fusion*.

In this interaction, the Part is an arbitrary mesh with an open boundary loop, and the fusion involves two steps. First, a hole with a boundary compatible with the Part boundary is created in the target mesh. This has been done manually, by linearly projecting the Part boundary onto the target surface, or by way of compatible global parameterizations. Once the target hole is known, several approaches can be used to insert the Part. Funkhouser et al. [Funkhouser et al. 2004] perform automatic rigid alignment and then generate a fillet mesh, while Sharf et al. [Sharf et al. 2006] use a Soft-ICP algorithm followed by local remeshing of the combined vertex sets. For variational approaches the user manually orients the Part in 3D, and then the deformation of a transition region [Sorkine et al. 2004] or the entire surface [Yu et al. 2004] is defined by constraining the Part boundary to the hole boundary.

Part fusion two advantages over displacement-based methods. First, arbitrary Part shapes can be represented, not just those representable by displacement or embeddable in a displaced volume. Second, only the shape of the boundary influences the deformation of the Part - curvature variations on the interior of the target Part region have no direct effect on the Part shape. The main drawbacks are that in general the target hole must be manually created and Part must be manually oriented in 3D. Sharf et al. [Sharf et al. 2006] have tried to reduce the burden of this rigid alignment by automatically *snapping* the Part into place when it is interactively dragged near the target hole, but the result still depends on the global Part orientation.

2.5.4 Part Drag-and-Drop

Displacement Copy-and-Paste and Part Fusion both allow one to re-use geometry from one model in another, but neither modifies the original model. Part Drag-and-Drop is a related interaction in which the Part is extracted from the source model, leaving a smooth surface behind, and then can be interactively dragged to a new position. Suzuki et al. [Suzuki et al. 2000] described the first Part drag-and-drop tool, in which a simple mesh feature could be rigidly dragged through the rest of the mesh. The surrounding surface was locally remeshed at each frame, limiting the drag to relatively flat, smooth surface regions. Note also that the Part cannot "jump" to another portion of the surface.

A much more advanced form of Part drag-and-drop is possible if the surface model is procedural. Chan [Chan et al. 1997] described a drag-and-drop interaction which was be applied to layered spline surface models created using Surface Pasting [Barghiel et al. 1994]. In this interface, the user can select any of the discrete Part nodes and then drag it to a new position on the base surface, as well as rotate and scale the feature. This manipulation could be extended to any other Parts that overlapped the selected Part.

2.5.5 Model Fusion

A number of works have addressed the problem of "mixing" a set of independent models. This is related to Part fusion, however the artist works with entire models instead of

Parts. Generally the result is computed completely automatically, avoiding the hole-cutting and rigid alignment steps of Part fusion. To achieve this simplification, most artistic control is sacrificed.

Kraevoy et al. [Kraevoy and Sheffer 2004] created compatible cross-parameterizations between small sets of models, allowing the global surfaces to be arbitrarily blended. An extension to this work [Kraevoy et al. 2007] assumes that a database of cross-parameterized models is available, then allows an output model to be assembled by selecting suitable Parts from the database. Hassneret al. [Hassner et al. 2005] transitioned between two models without requiring a cross-parameterization, by finding an optimal cut between the two meshes.

2.5.6 Semantic Deformation

As I have mentioned previously, once a Part decomposition is known, one can attempt to preserve semantic properties of a Part across editing operations. The iWires system of Gal et al. [Gal et al. 2009] implicitly defines Parts via automatically-detected boundary loop networks, and then preserves rigidity of geometrically similar Parts when one is deformed. Kraevoy et al. [Kraevoy et al. 2008] attacks a similar problem, trying to preserve the proportions of semantic Parts of a model during scaling operations (although in this case no discrete Parts are actually defined)

These semantic deformation techniques can be seen as a way to automate the specification of parametric modeling constraints. However, the "constraint solvers" in these works are more focused on surface properties than those in parametric modeling tools, which tend to operate on abstract parameters. These types of surface-based approaches to preserving geometric relationships may be extendable to more general surfaces, which would be very useful in a parametric surface modeling tool.

2.6 Conclusion

In this chapter I reviewed a wide range of surface modeling works, with a focus on procedural representations, Part representations, and Part-based interactions. From this overview we can draw several conclusions.

First, there is very limited support for procedural representation of surface models. While various techniques can be re-cast in this role (layered displacement maps, bounded spatial deformations, etc), the resulting procedural models are extremely rigid, and hence there are few advantages to be gained from storing the history of changes to the surface in procedural form. The only explicitly Part-based procedural representations which have been developed (H-Splines and Surface Pasting) are highly limited in the range of surfaces they can represent, compared to the complexity of surface models in use today.

Existing works do not consider the general question of how to define a surface Part. Works which do involve surface Parts generally tailor their definitions to the task at hand, such as the Part-as-NURBS-patch approach in Surface Pasting. Works which attempt to detect Parts focus either on segmenting a mesh into discrete regions, or finding structured geometric entities (embedded edge curves, symmetric regions, etc).

Part-level manipulation of surfaces is a relatively unexplored area. Some recent works attempt to preserve Part-like structure by way of geometric analysis (curvature, embedded edges, symmetry, etc) followed by constrained deformation. There has been no crossover between these works and those that impose actual Part-based representations on surfaces, such as Surface Pasting or layered displacement maps.

In terms of these truly Part-based procedural representations, current approaches are all limited to Parts represented via vector displacements. Even moderately complex Parts cannot be moved to another surface region without the risk of introducing extensive shape distortion. Integrating a Part with greater representational capability into a truly procedural model would be a clear step forward.

Finally, Part-based interactions have generally been focused on the problem of combining Parts from existing models into a new static model. While this is clearly a highly useful capability that follows naturally from Part-based representations, in a procedural model there are many other interaction problems to explore. For example, how should surface Parts be interactively positioned on a surface, and how can this positioning be maintained as the underlying surface deforms? Even more fundamentally, how can overlapping Parts be selected, or even visualized in a coherent way?

Chapter 3

PART-Based Surface Modeling

3.1 Introduction

In recent years, the level of geometric complexity evident in 3D surface models has grown very rapidly. In modern surface sculpting tools, artists now regularly work with billions of polygons [Pixologic, Inc. 2011], creating models with incredible levels of geometric detail. Figure 3.1 shows frames from a timelapse video of an artist sculpting a realistic human head. This particular model is not extraordinary; models with similar and higher levels of detail are created on a daily basis, and used to stunning effect in everything from real-time games to animated films.

After observing the work of these 3D sculptors, it is hard to imagine any 3D surfaces which would be "too hard" to at least closely approximate in modern sculpting tools. It is not only *possible*, but also *practical* for a designer to model virtually any 3D surface from scratch. A reasonable question then is, what is it that 3D artists *can't* do?

In the introductory chapter I made the case that a major omission from the current surface modeling state-of-the-art is the surface analog of the structured, PART-based representation and interaction found in solid modeling. In solid modeling we can discuss a shape in terms of its PARTs, without having to resort to geometric minutia. This is not the case with surface models, where geometric details like vertices and control points intrude on any attempts at higher-level discussion.

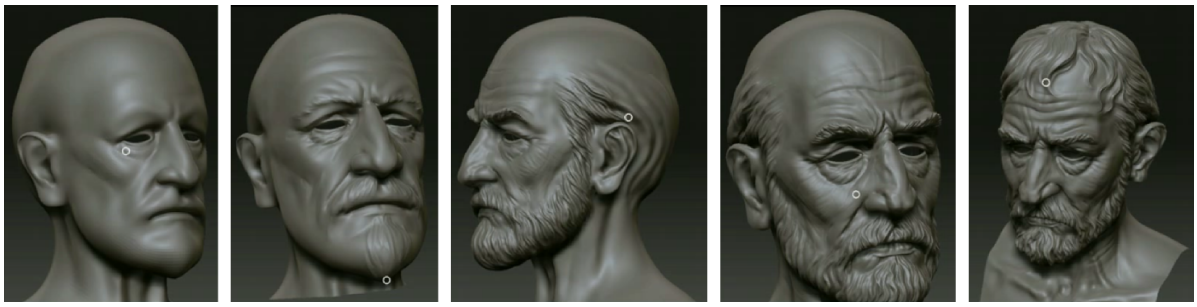


Figure 3.1: *Frames from a timelapse of a moderately detailed multiresolution surface model being sculpted in the commercial tool ZBrush.*

While there are clear theoretical benefits to having such a higher-level description, in terms of practical utility it is reasonable to ask question such as

- Would artists find any value in Part-based modeling?
- Do artists think of surface models in terms of Parts?

In this chapter, I will attempt to convince the reader that the answer to these two questions is yes. After a brief overview of the use of surface modeling practices, I will provide some evidence that artists are already embedding some Part semantics in their models. Next we will consider some of the clear limitations of these existing surface representations, by way of comparison to solid modeling. I will then develop my proposed surface Part which involves a decomposition into a Part domain and shape, and an on operator which attaches a Part to a surface.

3.2 Current Practices in Surface Modeling

In this section I will attempt to give the reader a sense of how surface modeling techniques have been put into practice by artists and designers. After a brief discussion of the evolution of surface modeling tools, I will focus on SubD mesh modeling, as this approach has become dominant in recent years.

In this discussion I will necessarily be forced to make generalizations about "what artists do". I hope the reader will understand that I am at best summarizing what I have observed. Although I am capable of using many of the standard 3D modeling tools, I am not a trained or working 3D artist. I also have not carried out any formal interviews, although because of my public development of 3D modeling software I have had many interactions with amateur and professional 3D artists (see Chapter 7). I have also spent extensive amounts of time studying 3D artists and designers in two ways:

Online Forums

I found that many CG artists are very active in online forums and message boards. There are literally thousands of such forums, in many different languages, generating vast volumes of public conversation on a daily basis. These conversations range from helping newcomers master basic techniques, to expert critiques of particular models, to commentary on current tools and practices used in the CG industry. I found that two forums in particular were frequented by many highly-regarded artists, and sometimes by developers of CG software:

- **CGTalk** (http://forums.cgsociety.org/)
- **ZBrush Central** (http://www.zbrushcentral.com/)

Online Tutorial Videos and Speed Modeling

Although highly informative, the content available in online forums is generally limited to text and images. However, many CG artists who are active in online communities also create high-quality *tutorial videos* for their peers. Such videos are easily available on public video-sharing websites like **YouTube** (http://www.youtube.com). These tutorial videos are particularly informative because they can tell us what CG artists find difficult or frustrating about current tools[1].

Another useful category of video that can be found on video sharing sites is *speed modeling*. These videos are usually just a screen-capture of a single modeling session, played back at high speed, so that a complex model may evolve before one's eyes in a matter of minutes. Although it can be difficult to see exactly what the artist is doing, speed modeling videos allow one to glean some insight into how the artist thinks about the structure of the model, which is virtually impossible to recover in any other way.

3.2.1 A Brief History of Surface Modeling Practices

Surface modeling has a long history in the computer graphics industry - nearly as long as the history of computing itself. Spline surfaces were already in wide use in early CAD/CAM systems in the 60s, and references to designing surfaces via control curves date back to Roman times, where ship ribs were produced from reusable templates [Farin et al. 2002]. In these early systems *NURBS modeling* was dominant, and much effort was devoted to providing artists with tools to manipulate NURBS patches and stitch them together with high levels of smoothness. High-end *Class A* surface design tools such as Alias StudioTools [Autodesk Inc. 2010b] became the standard way in which industrial design was communicated.

Despite decades of work, NURBS modeling is still difficult for artists, in large part because of the rigid topological structure needed to maintain smoothness. Modern NURBS modelers such as Rhino [Robert McNeel & Associates 2010] and Moment of Inspiration (MOI) [Triple Squid Software Design 2010] have made great strides in interface usability, but the fundamental difficulties of NURBS modeling remain. Hence, in the late 1990's, focus shifted to *Subdivision surfaces*, which make it easier for the artist to control the trade-off between topological flexibility and surface smoothness.

While researchers generated many different subdivision strategies, Catmull and Clark's 1978 method [Catmull and Clark 1978] remains completely dominant in commercial tools. 20 years after they were first proposed, the 1998 Pixar short Geri's Game demonstrated that Catmull-Clark subdivision surfaces were a viable replacement for spline surfaces. Today virtually every modeling tool targeting entertainment industries supports *SubD* modeling: Maya [Autodesk Inc. 2010e], Modo [Luxology Inc. 2010], Blender [The Blender Foundation 2010], 3DSMax [Autodesk Inc. 2010a], and so on. Even some engineering tools like AutoCad [Autodesk Inc. 2010c] have integrated SubD surfaces.

Like NURBS modelers, SubD modelers are based on a low-resolution control mesh

[1] For example, after I released the meshmixer software described in Chapter 5, several YouTube videos were posted demonstrating limitations of the program that individuals found frustrating.

that the artist manipulates to reshape the smooth surface. However, because subdivision is iterative, the artist is exposed to the mesh at multiple resolutions. By storing edits at different resolutions using displacement vectors, a *multiresolution* surface can be created [Zorin et al. 1997]. Multiresolution makes it much easier to manage large models, and most SubD tools incorporate multiresolution.

Even with multiresolution, though, the artist must still manipulate mesh vertices using 3D manipulation widgets. This "vertex-pulling" style of modeling is tedious and error-prone when performed via a 2D projection. A far more artist-friendly interface is *sculpting*, where the designer can specify a displacement by dragging a *virtual brush* across the 3D surface. Although Maya [Autodesk Inc. 2010e] included sculpting tools for many years, they did not gain widespread popularity until tools like ZBrush [Pixologic, Inc. 2011] and MudBox [Autodesk Inc. 2010f] made it possible to interactively sculpt multiresolution models with tens of millions of triangles (and today, billions). Sculpting interfaces also support representations that are more topologically flexible but unsuitable for vertex-pulling interaction. For example, 3D-Coat [Pilgway Inc. 2010] is based on an octree volumetric representation, and Sculptris [Pettersson 2010] uses a high resolution adaptive triangle mesh.

Due to the combination of order-of-magnitude increase in geometric detail with more intuitive interaction, digital sculpting has rapidly gained acceptance among artists, particularly in film and games where unrestricted creativity is valued more than surface quality. Recent developments in digital sculpting have focused on 'hard-surface' brushes which produce solid-modeling and high-continuity surface effects, suggesting that even those designers who require Class A surfaces may eventually adopt sculpting tools.

Parallel to the development of interactive 3D modeling tools has been the rapid growth of 3D shape capture technology. Modern laser scanners can rapidly convert a physical object into a cloud of billions of 3D points. Researchers have devoted significant effort to *point set surface* algorithms and modeling tools. However, standard practice in industry is to convert scan data to NURBS or SubD representations, usually using manual and tedious *surfacing* tools that bear close resemblance to those first proposed in 1996 by Krishnamurthy and Levoy [Krishnamurthy and Levoy 1996].

The surface models sculpted by artists today have levels of geometric detail that are much, much higher than what was possible in the 90's, when most modern 3D modeling tools and interaction techniques were developed. At the same time, 3D capture technology has advanced, to the point where physical objects are routinely converted into triangle meshes with hundreds of millions of polygons. Dealing with these massive polygon counts is an active area development in both research and industry. For example, a recent focus in shape modeling research has been techniques which mimic multiresolution-style interaction on unstructured mesh surfaces [Botsch and Sorkine 2008]. Although powerful, these methods have yet to make any headway in current interactive surface design tools.

3.2.2 Mesh Modeling and Topology Management

As I have mentioned, virtually all 3D tools in widespread use support subdivision (SubD) surfaces. As a result, the vast majority of 3D surface modeling that occurs today involves SubD surfaces. In the following I will briefly review some of the specifics of SubD modeling

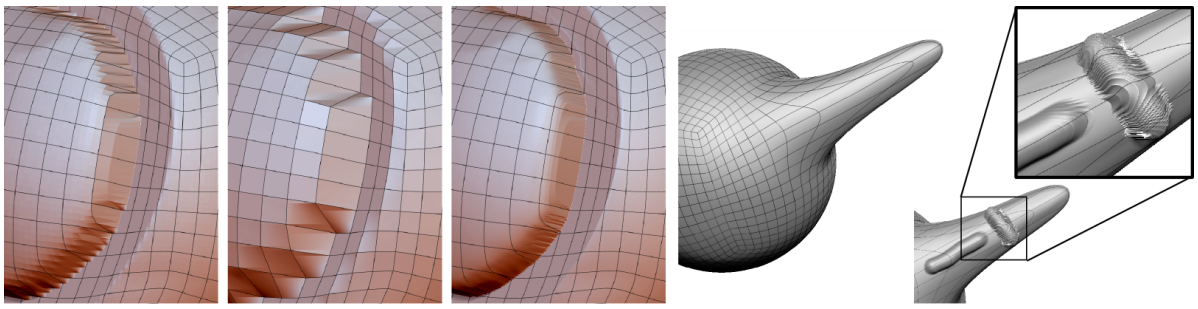


Figure 3.2: *A circular feature on a multiresolution surface (a) has "jaggy edges" because the base mesh (b) is not aligned with the feature. This aliasing effect remains even at high resolutions (c). A further problem is that when the multiresolution parameterization is very stretched (d), features aligned with the "grain" of the stretching can still be represented, but those across it will be highly aliased (e) even at extreme levels of subdivision.*

which can inform Part-based surface modeling, and will also be relevant when discussing practical applications of surface Parts in later chapters.

Multresolution SubD sculpting tools like ZBrush and MudBox provide surface interaction metaphors that are quite intuitive to artists, and one might imagine that such tools can insulate the artist from managing the 3D polygonal mesh that underlies a SubD model. Unfortunately, this is not the case. Even at extreme mesh resolutions, SubD surfaces are still restricted by the topology of the base mesh. Figure 3.2 demonstrates some of the problems that can occur when the base mesh edges are not suitably aligned with the features one wishes to create.

An even more important role for the SubD base mesh is that it is the level at which the model is animated. If the base mesh topology is poorly structured, parts of the model will pinch or distort in undesired ways. Hence, CG artists have developed an extensive theory of how to manage the topology of quad meshes so that the resulting surfaces animate properly. This general approach is called *edge loop* modeling, and was initially documented by Raitt and Minter [Raitt and Minter 1998]. The general principles in this approach are that loops of edges in the mesh are used to demarcate the boundaries of structures that will be animated; primarily musculature in character forms. For example, the sphincter muscles around the eye and mouth should be represented by closed circular loops in the mesh topology (Figure 3.3a,b).

Based on this general principle, CG artists have developed a rich set of guidelines and terminology for analyzing quad mesh topology. For example, an irregular quad mesh vertex of valence 5 is an *E-Pole*, and valence 3 is an *N-Pole*. These vertices are critical because they are locations where edge loops can change direction. Learning how to create meshes with good *edge flow* remains a significant hurdle for novice SubD artists, and an extensive amount of discussion in online forums is devoted to techniques for debugging edge flow. This largely amounts to tedious manipulation of the mesh to move E-Poles and N-Poles to more suitable locations (Figure 3.3c,d).

With the rise of high-resolution sculpting, a related workflow gaining in popularity

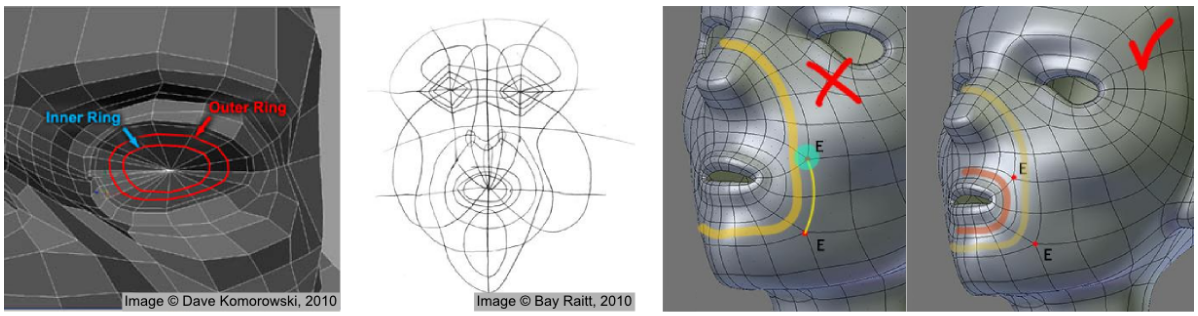


Figure 3.3: *Much of the effort in SubD modeling is devoted to carefully managing edge loops so that the surface deforms in a reliable way. Edge loops (a) represent musculature that will be animated. Standard sets of edge loops (b) simplify the creation of high-quality faces. Artists spend significant amounts of time doing tedious edge flow debugging (c,d).*

is to sculpt an initial high resolution model without worrying about good topology, and then trace a low-resolution mesh with good topology on top of the detail surface. This process is sometimes referred to as *re-topologizing* or *re-topo*[2]. Once a low-resolution mesh with good topology has been created, the multiresolution SubD hierarchy can be automatically fit ("projected") to the original sculpt. An example of this type of emerging workflow is shown in Figure 3.4. Although this example was created by a novice modeler (the author), similar but more impressive examples can be found in abundance in the forums mentioned above.

Edge loop modeling and SubD topology management are interesting because they tell us that artists have a clear sense of Part structure for a given model. Edge loops generally indicate the boundaries of important surface features. In addition, a major motivation for the development of edge loops was to allow the artist to manipulate Parts of the surface independently. As an example, the loops in Figure 3.3a separate the eye socket from the nose and brow. This allows the eye to be modified and animated as if it were a separate component. When an artist says that a model has bad edge flow, often what he or she is really saying is that there are undesirable connections between surface Parts that the artist wishes to be disjoint.

3.3 Limitations of Flat Surface Models

Model surface models are *flat* in the sense that they are represented by single 3D graph of points and edges that defines the entire surface. As a result, whether the surface being modified is a NURBS, SubD, or raw poly mesh, and whether the interactive tool is based on vertex-pulling, spatial deformations, or digital sculpting, "deforming a sphere

[2]Although extensive research has been devoted to automatic quad-mesh generation, the artists I have talked to do not find the meshes created by these algorithms suitable for practical use. This appears to be mainly because the generated base meshes do not contain the edge loops necessary to represent the underlying structures that need to be animated.

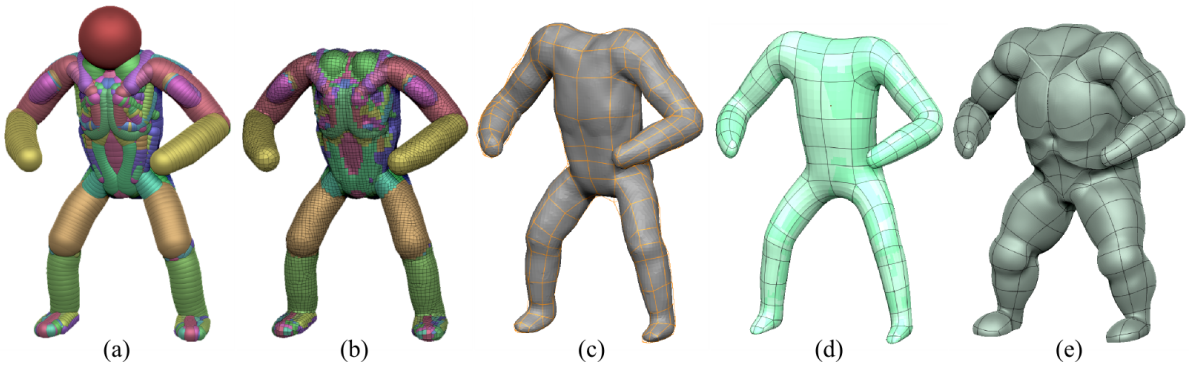


Figure 3.4: *An example of a modern surface modeling workflow. I sketched an initial shape using the Z-Spheres modeling tool of ZBrush, which was then automatically converted to a quad mesh (b). This mesh has too many quads and low quality edge flow, so I created a new topology by hand (c) to create a "cleaner" mesh (d). I then sculpted a more complex surface using displacement brushing tools (e). (Note that due to the skill limitations of the author, the mesh topology is still not particularly suited for this shape)*

or plane into a desired shape" is a reasonable high-level description of what a modern CG artist does. The amazingly complex models created using ZBrush have roughly the same intrinsic structure as a meshed sphere. The vocabulary we can use to describe such models is similarly restricted - although an artist may critique the proportion of a hand or the height of an ear, he or she must ultimately express any change to the surface in terms of vertices, edges, and faces.

Contrast this with solid modeling tools, which have rich languages for describing the structure of complex shapes. Even the simplest designs are composed of parts, assemblies, families, constraints, and so on. Designers are isolated from the tedious details of how a Boolean operation is implemented - instead they work with higher-level parts, parameters, and operators.

Because practical Part-based surface modeling tools have yet to be developed, it is hard to predict what sort of impact they might have on artists' workflows. However, I have identified several aspects of Part-based solid modeling that would be of clear value in surface modeling, but have no analog in modern tools. In this section I will describe several of these issues.

It is interesting to note that, anecdotally, artists are in some cases completely unaware of these limitations and of the comparable freedom that is available to engineers armed with solid modeling tools. It seems that the constraints of current surface modeling practice are so ingrained that it is hard for 3D designers to imagine what might be possible if they were lifted.

3.3.1 Component Assembly and Division of Labor

It is standard practice in solid modeling to decompose complex models into *assemblies*, which are collections of Parts. Often this assembly relationship can be nested, so that assemblies contain other assemblies as Parts, and so on. The process of building a

model can then be structured. After an initial conceptual stage involving a top-down division of the model into assemblies, each assembly can be created independently, and then integrated into the final model. In addition to allowing for useful organization and hiding of complexity, this approach also supports division of labor - each assembly can be created by a different designer.

In surface modeling, division of labor is very difficult because the entire surface must be created as single, continuous thin sheet. Different regions of the surface can be modeled independently, but they must eventually by stitched together. This stitching process is difficult to automate, and so requires a large amount of tedious manual intervention.

I have yet to observe any artist take this multi-piece approach to modeling a complex surface from scratch. One area where this is common is in scan data processing, where different regions of the model are often scanned at different resolutions and must be combined. In some cases this integration can be automated, but in general artists prefer a hand-stitched model with high-quality edge topology.

3.3.2 Construction History and Change Management

Many surface modeling tools do support Boolean operations, so it is possible to create surfaces in PARTs to some degree. But even these solid-modeling compositions are not procedural - PARTs are merged to create a new surface, and the inputs are discarded. The new surface can then be sculpted to 'clean up' any bad topology or sharp edges. It is not possible to return to the original PARTs, without discarding the edits that followed the merge operation.

In general, surface representations have minimal support for the tracking of *construction history*, which is a sort of generalization of procedural hierarchies. For example, the Maya modeling tools [Autodesk Inc. 2010e] support construction history on mesh editing operations, but the modifications are stored using lists of vertex indices. So if the mesh topology changes, the history becomes meaningless.

A similar issue is *change management*. Most solid modelers include rich toolsets for comparing two versions of a model to determine what has changed. This issue is completely foreign to surface modeling. If the mesh topology is static, one could in theory highlight changes to any vertices, and otherwise Booleans could potentially be used. But without a construction history, there is really no way to integrate changes from two different versions of a model anyway. So this issue does not arise today - instead, each model is 'owned' by a particular artist, who is likely to be the only one who knows how it was constructed.

3.3.3 Structured Manipulation

As mentioned above, one of the reasons to carefully manage edge loops in a SubD mesh is that it allows for some degree of *independence* in manipulating the implicit PARTs of the surface. However, even the most carefully-designed edge loop networks are at best an approximation to the sort of *structured manipulation* that is possible in solid modeling. An individual solid part can be modified in isolation and integrated back into the model

without difficultly. Similarly, an existing atomic part can be extracted and converted into multiple sub-parts, without affecting the rest of the model.

Even in multiresolution modeling tools, adding a new Part on top of another is generally only possible by deforming the surface that represents the initial part. As a result, this underlying part is effectively destroyed. This means that seemingly trivial tasks, like resizing the sphere in Figure 1.2 without affecting the bump on top of it, are virtually impossible with modern surface modeling tools. Although the artist may have a sense of the Part decomposition of the model, the Parts are in fact tightly coupled and cannot be manipulated in a structured way.

3.3.4 Asset Re-Use

One of the biggest limitations following from the lack of Part-based workflows in surface modeling is the difficulty of re-using assets. As sculpting tools have made high-quality shape modeling so much more efficient, it is becoming an increasingly common practice for artists to create large numbers of design variations for "pre-visualization". The art director or client then provides guidance about which parts of which models are desirable, and the artist refines his or her creations. In these early stages where models may undergo large changes, artists are usually creating each "refinement" from scratch. With the limited support for Part-based interaction in today's tools, it is simply more efficient for the artist to start over than to attempt to evolve an existing design.

One might argue the reason these technologies have not been developed is a lack of demand, however there are large and growing marketplaces for many other art assets, from specialized textures and shaders to full 3D worlds. Video games like *Spore* allow the player to combine parametric Parts to create their own character [Eletronic Arts Inc. 2010], but the Parts must be carefully designed to be interchangeable by the game creators. There is also a growing interest in custom human models and 'digital doubles', where a client can select body parts from a catalog, but again the difficulty of combining the Parts makes it this a time-consuming and expensive process.

3.3.5 Iterative Design

Surface modeling has very little intrinsic support for *iterative design*. The underlying assumption in both surface modeling tools and surface modeling research is that the artist has a specific "end goal" in mind, and simply needs to be able to specify the right series of modeling operations to get there. In practice, however, 3D modeling tends to not actually be very goal-oriented. The designer rarely has a fully-defined target in mind, instead following an *iterative design* process where both the current virtual model, and the desired "goal" model, are repeatedly updated as modeling proceeds. This often involves a significant amount of trial-and-error to explore the design space.

In surface modeling, trial-and-error is expensive because the designer must carry out editing operations sequentially, with operation n becoming immutable once operation $n + 1$ is initialized. The ubiquitous *undo* operation supports a limited form of trial-and-error, but operation n can only be modified by discarding all following operations, leading to a significant amount of repeated work whenever an error must be corrected.

A procedural Part-based representation would allow the artist to make a change to some previous operation, then have the change propagate forward automatically. This non-sequential editing capability makes design iterations much less costly.

3.4 Defining a Surface Part

In the previous sections I have made a case for the development of a surface-based analog to the Part-based aspects of solid modeling. However, in solid modeling a Part was easily defined as a set of points in $\mathbb{R}^3$. To develop Part-based surface modeling, we will first need to figure out what a Part of a surface is.

I emphasize that I will make no attempt to determine what makes a *good* part. In an interactive modeling tool, that is ultimately up to the artist. My goal is rather to provide a framework within which artists have the freedom to break a complex model up into Parts in the manner they see fit. I will, however, make some observations about the interaction between potential surface Parts, which do suggest that there are some Parts that will behave more predictably than others.

There is one type of Part used in surface modeling which I will discuss here to make clear the distinction between it and the surface Parts that I will focus on. It is often the case that a complex surface model will actually be composed of multiple surfaces which are simply positioned in space to intersect, giving the appearance of a single surface when rendered. For example, the buttons on a shirt may be modeled as separate polygonal meshes. These buttons are clearly Parts, but even if not technically volumetric B-reps, they are more closely related to volumetric Parts, in that they are completely independent of the shirt surface. The problems encountered in defining surface Parts do not occur with these independent surfaces, and so I will not discuss them any further.

3.4.1 Which kinds of Surfaces?

The term *surface* is used to refer to a wide variety of concepts. Before diving in our development of surface Parts, it will be useful to more precisely demarcate the types of surfaces that we are interested in. Although some of what I will describe can be applied to the continuous surfaces used in differential geometry [do Carmo 1976], my focus is the class of representations used in 3D surface modeling. As I discussed in Section 2.3, there are many such representations, including spline patches, polygonal meshes, point set surfaces, subdivision surfaces, and manifold surfaces.

Although each of these representations have different properties and capabilities, at a basic level they share some common structures. In particular, each is ultimately defined by three components: a set of 3D *point samples*, connectivity information defining the neighbourhood of each point sample, and a set of basis functions which generate a continuous surface based on the graph of points and edges. I will refer to this triplet of structures as the ***3D graph*** which defines the surface.

In much of this thesis I will focus on triangular meshes. Figure 3.5 shows an example of a triangular mesh. We have a set of point samples called *vertices* connected by a graph of *edges* to form a set of *faces*, each of which is a triangle. The linear basis functions that

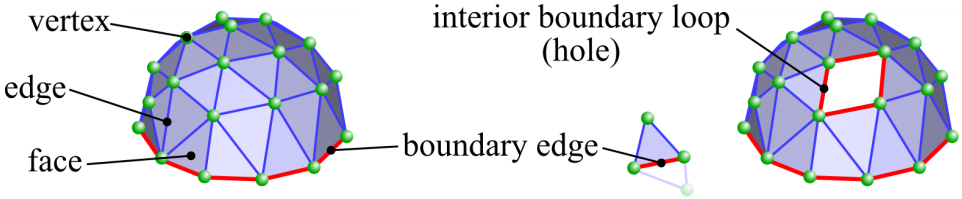


Figure 3.5: *A triangular mesh is defined by a triplet of structures I will call a* 3D graph*: a set of vertices, a graph of edges, and linear basis functions that generate triangular faces from the edges and vertices. If an edge has only one neighbouring face, it is a boundary edge. A mesh may have multiple boundary loops, which are topologically not identifiable, but in practice we will treat* interior *boundary loops differently.*

define these triangles are sometimes known as barycentric coordinates. We will assume that each edge is connected to either one or two triangles. In the literature, such a mesh is often referred to as being *manifold*, which roughly means that any small region is homeomorphic to a disc or half-disc [3]. Edges with two triangle neighbours are *interior* edges, while edges with only one triangle neighbour are *boundary* edges.

We will also assume that no vertex is connected to more than two boundary edges. In this case, the set of boundary edges can be connected up into one or more unique boundary *loops*. A mesh with a single boundary loop - a mesh *with boundary* - is homeomorphic to a disc in the plane. A mesh without any boundary loops, such as a mesh approximating a sphere, is a mesh *without boundary*. Depending on the context, a boundary loop may either treated as a poly-line of connected edges, or simply the ordered list of boundary vertices.

A mesh with two boundary loops is homeomorphic to a 2D *annulus*, the region between two concentric circles. Note, however, that there are two possible assignments of 3D boundary loops to 2D circles. In cases where this situation occurs, we will assume that extra information is available to uniquely determine which boundary loop is the *outer boundary loop*. The remaining one or more loops will be referred to as *interior boundary loops*. A similar situation is encountered when removing faces from a mesh without boundary. This creates a boundary loop, and in the strict topological sense, we simply now have a mesh with boundary. However, conceptually it will often make more sense to think of this as an interior boundary loop as well. In general, an interior boundary loop corresponds to the notion of a *hole* in surfaces both with and without boundary, and I will sometimes use this terminology.

Complicating matters is the fact that a surface may have another kind of hole, for example the hole that forms the handle of a coffee cup. This structure differs in that no boundary loops are involved. However, another acceptable term for this kind of topological structure is a *handle*, so I will reserve the term *hole* to refer to interior boundary loops.

[3]Note that this disc property is generally treated as relative to the mesh connectivity, rather than with respect to the 3D surface of points contained in the mesh. Hence, a mesh which self-intersects in 3D is still considered manifold as long as all edges have at most two neighbours in the connectivity graph.

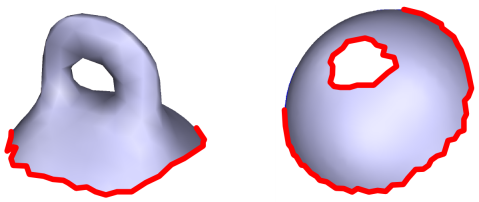


Figure 3.6: *We will encounter two kinds of "holes" - those topologically related to toroidal structures (left) and those defined by interior boundaries (right). I will refer to the structure on the left, which does not involve a boundary loop, as a "handle", and that on the right as a "hole".*

Note that the above structures - the 3D graph, boundary edges and loops, and so on - have direct analogs in most other surface representations. For example, meshes with arbitrary polygonal faces differ only in that they involve more complex basis functions. Subdivision surfaces and spline patches can be treated as polygonal meshes with higher-order basis functions. In some cases (ie NURBS patches) this restricts the topology of the connectivity graph, and depending on the basis functions the surface may not interpolate the point samples. However, the Part decomposition I will describe in the following can always be applied at the level of the base 3D graph (the control mesh).

For point set surfaces, the edge graph is generally defined based on spatial proximity (sometimes with additional criteria), and the notion of a boundary edge does not exist. Instead we can only attempt to define *boundary samples*, by examining the local neighbourhood and trying to determine if it is a full or partial disc. These boundary samples can then be grouped into boundary strips, or chained into explicit boundary loops [Bendels et al. 2006].

3.4.2 Which part is the Part?

Recall our introductory example "a sphere with a bump on it" from Figure 1.2. I made the case that in this shape, we have two parts - a sphere and a bump. If we were to model this shape using a triangle mesh, though, there will be no sphere and no bump. We will have only a single genus-0 sheet, topologically equivalent to the sphere. How can we impose a Part structure on this surface? One way to begin to answer this question is to ask "what would the surface look like if we removed the bump?".

It seems clear that we will have to decide that some region of the surface makes up the bump. But then what? How do we remove it? An obvious answer is to simply cut the Part region out of the surface, leaving a hole behind. A surface with a hole is perfectly valid. If we think of these two surface simply as being sets of 3D points, then we can combine them using the set union operator to recover the sphere-with-bump. An example of such a decomposition is shown in Figure 3.7a. One problem is immediately obvious, though - we described this shape as "a sphere with a bump on it", but our sphere now has a hole in it.

Consider a similar solid-modeling scenario, such as a sphere with an extrusion added to it. If we remove the extrusion, we are left with the sphere. In general, if we add

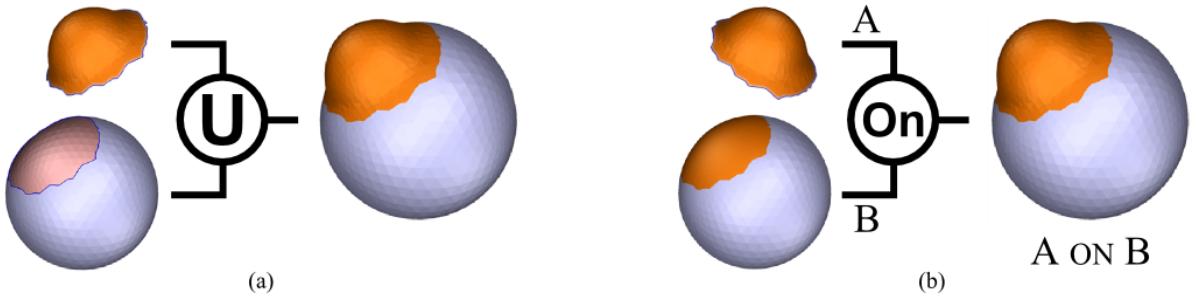


Figure 3.7: *A* PART *decomposition of "a sphere with a bump on it" could (a) be based on the set union* $\cup$ *of two independent surfaces, however in this case we do not actually have a sphere, but rather a sphere with a hole in it. To compose surfaces, I propose (b) an* ON *operator which* replaces *a region of the sphere with the bump* PART.

and then remove a PART from a solid model, the volume remains unchanged - solid composition operators are *reversible*. It is highly desirable that our surface PART have a similar property. Hence, if we remove a surface PART we should be left with the surface that was *underneath*, before it was added.

If we take this approach, then after removing the bump from "a sphere with a bump on it", we clearly should be left with a sphere. Such a decomposition is shown in Figure 3.7b. The key difference in this approach is that the bump PART *overlaps* the sphere.

Note that we can no longer apply the set union operator to combine the bump and sphere. We must *remove* the portion of the sphere that is underneath the bump, otherwise the surface will become topologically invalid. Hence, in Figure 3.7b I have introduced a new composition operator - the ON operator.

3.4.3 The ON Operator

The ON operator is an integral component of the PART structure I wish to impose on surface modeling. Essentially, the implementation of the ON operator largely determines how PARTs will be represented. Let us consider the *A* ON *B* operation in more detail.

In the sphere-with-bump example, we found that the bump PART was a surface patch with a boundary. To output the combined surface, the ON operator could take this patch and the sphere as input, cut a hole in the sphere that aligns with the bump boundary, and stitch in the bump. But this operator is extremely brittle. If the bump were to change slightly - for example, via a rigid rotation as I have applied in Figure 3.7b - it will no longer be possible to merge it with the sphere.

We have run into a problem because the bump is defined as a surface in global position. A more robust ON operator would deform the bump to fit any surface. But this introduces a new question - *where* on that surface should the bump be deformed to fit?

When we removed the bump from the sphere, we left a complete sphere behind. Hence, underneath every PART there lies another surface. I will refer to this region of surface "underneath" the bump as the *domain*, for reasons which will shortly become clear. The union of the domain and the sphere-with-hole will be called the *base surface*. Note that we now have *two* surface regions to consider - the PART surface itself, and the

PART domain on the base surface. These two regions define the PART shape and PART location, respectively.

The *where* of the PART - the location of the domain on the base surface - is largely orthogonal to the *what* of the PART (the shape). Hence, if we can define the PART shape *relative* to the part domain, for example using a displacement map, then we can move the PART to another surface simply be redefining the PART domain. Note the dependency relationship - once we know the PART domain, we can compute its shape. It is possible to reverse this dependency and define the domain based on the shape, but as I will explain below, this approach is much less flexible. Hence, I will focus on defining the *where* and *what* problems independently.

3.4.4 A PART as a Manifold Deformation

At a conceptual level, if a surface region $\mathcal{U}$ can be robustly defined on the base surface $\mathcal{S}$, then we can simply *deform* $\mathcal{U}$ into the desired PART shape $\mathcal{V}$. Let us try to formulate this process mathematically.

We can describe a surface $\mathcal{S}$ with or without boundary as the set of points that lie on it, $\mathcal{S} = \{\mathbf{p}\}$, and our PART domain $\mathcal{U}$ is simply a subset of these points $\mathcal{U} \subset \mathcal{S}$. For now we will assume that $\mathcal{U}$ is topologically equivalent to a disc. This subset can then be represented by a *characteristic* or *indicator* function X on $\mathcal{S}$:

$$X(\mathbf{p},\mathcal{U}) = \begin{cases} 1 & \text{if } \mathbf{p} \in \mathcal{U}, \\ 0 & \text{if } \mathbf{p} \notin \mathcal{U}, \end{cases} \tag{3.1}$$

Given any vector-valued function $F(\mathbf{p})$, we can now generate a *displaced* version of $\mathcal{S}$ in which only $\mathcal{U}$ has been modified:

$$D(\mathcal{S},\mathcal{U},F) = \{\ D(\mathbf{p},\mathcal{U},F)\ \} \tag{3.2}$$

$$D(\mathbf{p},\mathcal{U},F) = \mathbf{p} + X(\mathbf{p},\mathcal{U})F(\mathbf{p}) \qquad \text{where } F(\mathbf{p})\big|_{\partial\mathcal{U}} = 0 \tag{3.3}$$

Note that $\mathcal{V}$ does not explicitly appear in these equations; it can be defined as $\mathcal{V} = D(\mathcal{U},\mathcal{U},F)$. This equation is simply a locally-supported *deformation* of $\mathcal{S}$, but it is clear that it completely models the change in the surface that has occurred in Figure 3.7b. Hence, we see that if $\mathcal{U}$ and $\mathcal{V}$ are both topological discs, the ON operator can be written as a deformation:

$$\mathcal{V} \text{ ON } \mathcal{S} = D(\mathcal{S},\mathcal{U},F) \tag{3.4}$$

The boundary constraint in Equation 3.3 ensures that no new holes are introduced, so ($\mathcal{V}$ ON $\mathcal{S}$) has the same topology as $\mathcal{S}$. Similarly, the topology of both $\mathcal{U} \subset \mathcal{S}$, and any other region on $\mathcal{S}$, remain unmodified, and any parameterization of $\mathcal{S}$ is also a parameterization of ($\mathcal{V}$ ON $\mathcal{S}$).

To see how we can now append another PART let us first rename our variables, so that the base surface is $\mathcal{S}_0$, and we have $\mathcal{U}_1 \subset \mathcal{S}_0$ and $\mathcal{S}_1 = (\mathcal{V}_1 \text{ ON } \mathcal{S}_0)$. We can now append another PART by first defining a domain $\mathcal{U}_2 \subset \mathcal{S}_1$ and shape $\mathcal{V}_2$, and then applying Equation 3.4 to generate

$$\mathcal{S}_2 = (\mathcal{V}_2 \text{ ON } \mathcal{S}_1) = (\mathcal{V}_2 \text{ ON } (\mathcal{V}_1 \text{ ON } \mathcal{S}_0)) \tag{3.5}$$

Clearly we can repeat this process to add as many PARTs as desired. We can further simplify the structure by noting that since the deformation-based ON operator preserves topology, we can map $\mathcal{U}_2$ back onto the base surface $\mathcal{S}_0$. So, at least at a conceptual level, we can reduce a sequence of topology-preserving PARTs to a set of regions on the base surface, and a displacement function for each region.

Note that in the above equations we have assumed that $\mathcal{S}$ is simply a set of points. However, with a triangle mesh surface representation, $\mathcal{S}$ is generated by the 3D graph. To apply this machinery to a mesh, we limit ourselves to displacing the set of vertices and hold the connectivity graph fixed. There are several ways to define $\mathcal{U}$. The simplest is as a set of adjacent faces with disc topology, in this case $\partial\mathcal{U}$ is simply the loop of edges bordering the set of faces. In the interests of expositional efficiency I will continue with the pseudo-continuous approach, which I find more intuitive and compact. The special handing necessary to apply the results to mesh-based surface representations will be pointed out as necessary.

3.4.5 Limitations of the Deformation Form

Equations 3.3 and 3.4 are a concise mathematical model of one approach to defining surface PARTs and the ON operator. In the following I will refer to this as the ***Deformation form*** of a surface PART. The Deformation form is mathematically robust, making it easy to analyze and implement. As I discussed in Section 2.3.2, layered displacement maps correspond precisely to the Deformation form.

Unfortunately, from a practical standpoint the Deformation form has some undesirable limitations. Since both $\mathcal{U}$ and $\mathcal{V}$ are topological discs, the ON operator cannot change the topology of the base surface. So, for example, if we begin with a sphere we can never produce a coffee cup.

A more immediate limitation is that in many applications it will be the case that we are given a desired PART shape $\mathcal{V}$ a priori. To apply the Deformation form, we must construct a vector-valued displacement function F that deforms $\mathcal{U}$ into $\mathcal{V}$. In practice, even approximating an arbitrary $\mathcal{V}$ with a reasonable level of visual fidelity is a non-trivial problem, more so since on meshes we have assumed that the edge graph is held constant.

To describe ON operators capable of handling more general surface PARTs, we will have to relax the disc-topology and fixed edge-graph restrictions. I will now describe several of these alternative forms.

3.4.6 Alternative Forms of the ON Operator

Although I previously rejected set operations as a viable way to define a surface PART (Figure 3.7a), it is instructive to consider reformulating this approach as an ON operator. If our PART shape $\mathcal{V}$ is a surface with boundary, then we can write this composition as:

$$\mathcal{V} \text{ ON } \mathcal{S} = \{\mathbf{p} \in \mathcal{S} : X(\mathbf{p},\mathcal{U}) = 0\} \cup \{\mathbf{p} \in \mathcal{V}\} \qquad \text{where } \partial\mathcal{V} = \partial\mathcal{U} \tag{3.6}$$

$$= (\mathcal{S} \setminus \mathcal{U}) \cup \mathcal{V} \qquad \text{where } \partial\mathcal{V} = \partial\mathcal{U} \tag{3.7}$$

We could call this the ***Assembly form*** of surface PART composition, where the two surfaces are combined by literally cutting a hole in one and stitching in the other along a

shared boundary loop. Note that on a mesh, $\partial\mathcal{V} = \partial\mathcal{U}$ implies that the boundary edges are identical. In this case the mesh stitching involves replacing each pair of coincident boundary vertices with a single vertex, and then rewriting the edge graph.

The Assembly form is completely general - both $\mathcal{U}$ and $\mathcal{V}$ can have arbitrary topology, and can even involve disconnected components if we relax the boundary constraint. There is a clear connection to the set operations used in solid modeling; $(\mathcal{S} \setminus \mathcal{U}) \cup \mathcal{V}$ precisely defines a tree of Boolean set operations, with 3 primitives and 2 operators. However, in solid modeling, any primitive or operator can be independently manipulated. The above formulation of the ON operator enforces a condition on the boundary of $\mathcal{V}$ which can only be satisfied in very restricted cases. Any perturbation of $\partial\mathcal{V}$ or $\partial\mathcal{U}$ which is not coupled with a compatible manipulation of the other nodes will create holes in the resulting surface. Similarly, if not perfectly aligned, $\mathcal{V}$ may intersect $(\mathcal{S} \setminus \mathcal{U})$. I will refer to both of these results as *invalid*, a term which I will now attempt to define.

In the paragraphs below I will attempt to re-formulate Equation 3.6 in ways that are more robust to changes in the PART domain and shape. Ideally, the ON operator would automatically preserve the *validity* of PART composition, in the same way that regularized Boolean operators do for CSG. First we need a working definition of a *valid* surface modification. Clearly we want the result of $(\mathcal{V}$ ON $\mathcal{S})$ to respect the mesh properties we described above, namely that each output edge has no more than two neighbouring faces, and each vertex connected to either zero or two boundary edges. The other relevant issue is the number of boundary loops inserted or removed. Say that $\mathcal{S}$ has N boundary loops, $\mathcal{U}$ has K, and $\mathcal{V}$ has M. Then we will only consider the PART insertion *valid* if after replacing $\mathcal{U}$ with $\mathcal{V}$, the output surface has at most $(N + M - K)$ boundary loops. Essentially, this means that the operation can add the holes in $\mathcal{V}$, and consume any holes in $\mathcal{S}$ which lie fully within $\mathcal{U}$ (see Figure 3.8).

In the previous section we defined $\mathcal{V}$ as a deformation of a disc-shaped $\mathcal{U}$, with fixed mesh topology. We can clearly rewrite the Deformation form using the assembly notation above. However, deformations can be applied to non-disc-shaped regions. It is also commonplace in mesh manipulation to combine deformation with *refinement*. By refinement, I refer to a sequence of incremental changes to the mesh edge graph, such that each step the topology (number of handles and holes) of the 3D surface defined by the graph is equivalent. For example, edge rotations replace two triangles with two triangles, edge splits replace two triangles with four triangles by inserting a vertex into an edge, and edge collapses remove two triangles and one vertex. I will refer to the combination of deformation and refinement applied to a connected component as an *Edit*.

Given an Edit E, we can write $\mathcal{V} = \mathsf{E}(\mathcal{U})$, and then express the ***Edit form*** of ON as:

$$\mathcal{V} \text{ ON } \mathcal{S} = (\mathcal{S} \setminus \mathcal{U}) \cup \mathsf{E}(\mathcal{U}) \qquad \text{where } \mathsf{E}(\mathbf{p})\big|_{\partial\mathcal{U}} = \mathbf{p} \tag{3.8}$$

Note that this form now has three operators ($\setminus$,$\cup$, and the unary E) and two primitives, however $\mathcal{U}$ has multiple parents and hence the result is no longer strictly a tree. Nevertheless, the Edit form is more flexible than the Deformation form, while being more robust than the Assembly form. As long as the boundary condition is preserved, we can arbitrarily and independently modify E and $\mathcal{U}$ without producing an invalid surface. Note that to apply the Edit form to a mesh, we also must also constrain the refinements of E such that they do not modify the boundary loop $\partial\mathcal{U}$.

The Edit form is quite general, and in my Surface Tree implementation (Chapter 6) most of the procedural PART nodes will be based on it. However, in the Edit form $\mathcal{V}$ and $\mathcal{U}$ must still be topologically equivalent. Similarly, like the Deformation form, in practice it is very difficult to construct an E which generates a given PART shape. To support an initial PART shape $\mathcal{V}$ we define the ***Insertion form***:

$$\mathcal{V} \text{ ON } \mathcal{S} = (\mathcal{S} \setminus \mathcal{U}) \cup \mathsf{E}(\mathcal{V}) \qquad \text{where } \mathsf{E}(\mathcal{V})\big|_{\partial\mathcal{V}} = \partial\mathcal{U} \tag{3.9}$$

Note that E is applied to $\mathcal{V}$ rather than $\mathcal{U}$. The interior of $\mathcal{U}$ is not deformed, but rather discarded. Hence, on a mesh we perform edge graph *replacement*, rather than refinement. As a result, the topology of $\mathcal{V}$ does not have to be the same as $\mathcal{U}$. Similarly, $\mathcal{U}$ has more flexibility, and can even include multiple connected components, if $\mathcal{V}$ is to be attached along multiple boundary loops (for example to create the handle of a coffee mug). Such an operation will be used to create topological handles in Chapter 6. Another advantage of the Insertion form over the previous alternatives is that it is much easier to formulate E operations that simply enforce the boundary constraint in a generic way. I will describe precisely such an operation in Chapter 5.

Note that in Equation 3.9 we again have a tree-like structure, as $\mathcal{U}$ only appears in a single node. However, in all of the above formulations, we have additional boundary conditions that must be enforced, which implicitly couple $\mathcal{U}$ and $\mathcal{V}$. So in fact *none* of the above structures can be represented by a fully independent hierarchy of nodes. As I will discuss further in Chapter 6, this coupling appears to be an inherent property of surface PARTs, and a fundamental difference when compared to volumetric compositions.

3.4.7 Valid Combinations of $\mathcal{U}$ and $\mathcal{V}$

In the previous sections I have described three different forms of the ON operator, and associated restrictions on the PART domain $\mathcal{U}$ and shape $\mathcal{V}$. The table below summarizes the capabilities of these alternatives:

ON form	Topology of $\mathcal{U}$	Topology of $\mathcal{V}$	Edge Graph Change
Deformation	disc	disc	none
Edit	connected component	same as $\mathcal{U}$	refinement
Insertion	multiple components	connected component	replacement

Figure 3.8 displays the different possible combinations of domain and shape topology in graphical form, along with the versions of the ON operator that can support the pairing. From this table it is clear that the Deformation form is the most limited. The Edit form is slightly more capable, although still quite restricted. This is in part because I limited refinements to those that do not change the surface topology. If we allow single edges or faces to be added, it would then be possible to place a disc over a hole. The Insertion form is most general, supporting all the different topological combinations.

One shared limitation of the presented forms of the ON operator is that they assume $\mathcal{U}$ to be a subset of $\mathcal{S}$. This is not strictly necessary. If we allow $F(\mathbf{p})$ to be non-zero where $\partial\mathcal{U} = \partial\mathcal{S}$, then we can grow a boundary of $\mathcal{S}$ outwards. Figure 3.9 shows how we can use a Deformation operation to create the necessary surface region to contain a desired Ω.

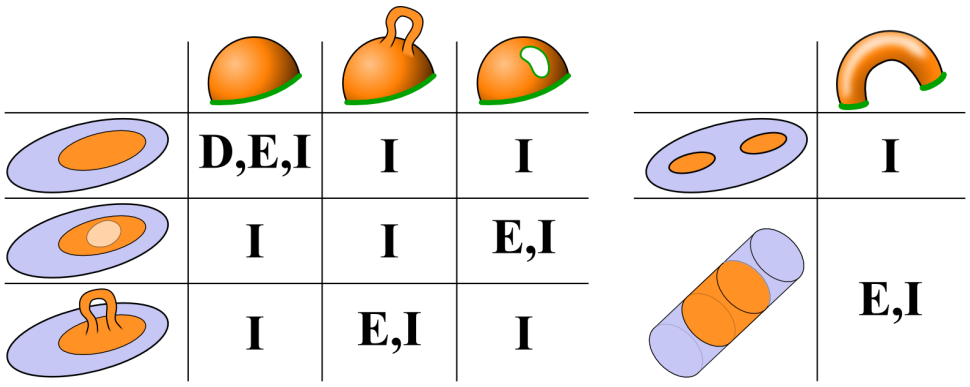


Figure 3.8: *Valid combinations of* PART *domain* $\mathcal{U}$ *(rows) and shape* $\mathcal{V}$ *(columns) for the (**D**)eformation, (**E**)dit, and (**I**)nsertion forms of the* ON *operator.*

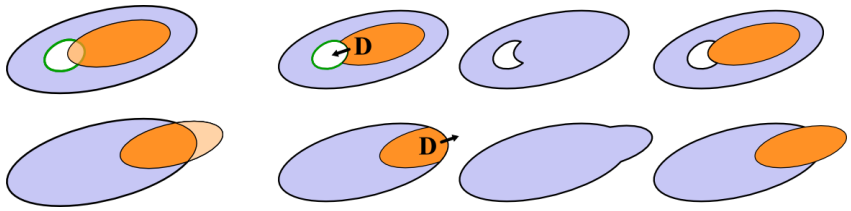


Figure 3.9: *The* PART *domain* $\mathcal{U}$ *must lie within the base surface, so the cases on the left cannot be directly handled. Deformation operations that expand outward from the base surface boundary can "make space" for the desired* Ω*.*

More problematic cases are shown in Figure 3.10a,b. Here the desired PART domain covers two disconnected portions of boundary loop. These cannot be joined by a deformation or E, as doing so requires modifications to the edge graph which change surface topology. Note that it is possible to join the two disconnected regions of $\mathcal{U}$ that intersect with $\mathcal{S}$ using a tube, however this will also increase the genus of the output surface.

Figure 3.10c is another interesting case. One way to look at this region is simply as a cylinder, in which case we can apply an Edit or Insertion PART. However, another potential operation we might wish to consider is shown in d-f, where we "break" the handle and reduce the genus of the surface by one. To do so we must either leave a hole on the remaining side of the handle, or allow a PART composed of two disconnected components. In the preceding text I have declared both of these invalid.

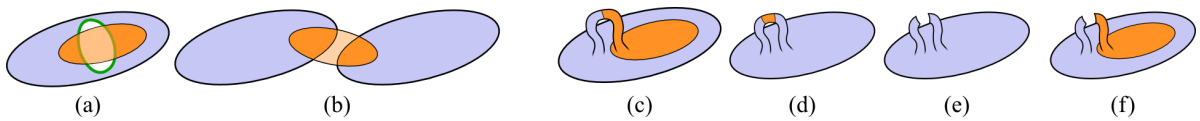


Figure 3.10: PART *domains* $\mathcal{U}$ *that fall outside the bounds of what I have described as permissible. Topology-changing operations are required to create domains (a) and (b). Although (c) could be a cylinder, we may also wish to "break" the handle. Representing this topological modification as shown in (d-f) is not possible within my current framework.*

3.4.8 Desirable Properties for the Representation of $\mathcal{U}$

So far we have considered the different approaches to PART composition largely as abstractions. In each case we essentially have the same two components - the PART domain $\mathcal{U}$ and the PART shape $\mathcal{V}$ - But we were unconcerned with how to practically represent these elements. However, to implement PART-based tools we must now turn to these issues. The definition of these two components is also where most existing works vary. So for the remainder of this chapter I will focus on the location and shape independently. Before continuing I will briefly describe some properties that we can use to evaluate the capabilities of different representations for $\mathcal{U}$ and $\mathcal{V}$.

To simplify the discussion, we again assume that our arbitrary region $\mathcal{U} \subset \mathcal{S}$ is a topological disc, so $\partial\mathcal{U}$ is a single boundary loop. Clearly if $\mathcal{S}$ is static, there are a multitude of ways to describe a fixed region. The challenge is to define $\mathcal{U}$ in such a way that it can be recomputed if $\mathcal{S}$ is modified. This modification could be a simple deformation, or could involve replacing $\mathcal{S}$ with an entirely different surface. The general problem to be solved is a *mapping* of $\mathcal{U}$ from one surface to another. I will broadly classify these mapping problems into four different types, of increasing difficulty.

Deformation A deformation is perhaps best thought of as a functional mapping $\mathcal{D} : \mathcal{S} \rightarrow \mathcal{S}'$, where $\mathcal{S}'$ is a differently-shaped surface. If $\mathcal{D}$ is a mesh deformation that does not change the mesh topology, then clearly $\mathcal{D}$ can be used to map $\mathcal{U}$ to $\mathcal{S}'$.

Resampling As discussed above, surface representations are based on a 3D graph, usually in the form of a mesh topology. Many procedural operations will dynamically generate this topology, so when $\mathcal{S}$ changes it may also be *resampled*, meaning that the number and location of the samples and graph edges will change. Clearly it would be desirable if $\mathcal{U}$ were as invariant to resampling as possible.

Surface Transport In an interactive tool, it is desirable to be able to *move* $\mathcal{U}$ to a different location on the current surface. The simplest way to accomplish this is to *slide* $\mathcal{U}$ a very short distance along $\mathcal{S}$, which can be expressed as allowing $\mathcal{U}$ to *flow* through a surface vector field for a small timestep. I will refer to this problem as *surface transport*.

Generalized Transport If Surface Transport can be solved, repeated application can be used to drag $\mathcal{U}$ along continuous paths. A harder problem is to move $\mathcal{U}$ discontinuously, such as to a disjoint part of $\mathcal{S}$ or another surface entirely, where no deformation $\mathcal{D}$ is explicitly given. This is a key capability for supporting arbitrary procedural composition of PARTs. I will call this problem *generalized transport*.

As we will see below, there are some trivial solutions to these problems in certain cases. For example, if a parameterization of $\mathcal{S}$ is available, we can define $\mathcal{U}$ in parameter space, and easily move it to any other parameter space. However, unless $\mathcal{S}$ is developable, any global parameterization will contain unintuitive and often extreme distortion, which will be transferred to $\mathcal{U}$ if transport operations are applied in parameter-space. Similarly,

when a global parameterization is recomputed it is entirely possible that the parameter-space coordinates will "shift" across the surface, which would then cause $\mathcal{U}$ to move. This is also undesirable. Hence, we must consider the following somewhat aesthetic property of any approach to solving the problems listed above:

$\mathcal{U}$-Rigidity How well does the method preserve the affine properties of $\mathcal{U}$ on the surface? By affine properties, I mean location, orientation, and scale of $\mathcal{U}$ relative to $\mathcal{S}$. Closely related is how well the method preserves the intrinsic shape properties of $\mathcal{U}$, such as the local curvature of $\partial\mathcal{U}$.

Essentially, the representation of $\mathcal{U}$ should be as independent of the surface geometry - current 3D configuration, mesh topology, and so on - as possible, while also making it as simple as possible to solve the four mapping problems.

3.4.9 Desirable Properties for the Representation of $\mathcal{V}$

We have defined the Part shape as some modification E applied to either the Part region $\mathcal{U}$ or the Part shape $\mathcal{V}$. In some sense, this is the easier problem - most of the major approaches to defining a surface deformation can be trivially adapted to formulate a suitable E. We will see, however, that most of these approaches fail to preserve the aesthetic properties of the initial Part shape if $\mathcal{U}$ changes. Similarly, some approaches which can more robustly represent shape can only do so for limited classes of shapes (displacement maps are a prime example).

E-Capacity As noted, some approaches to defining a 3D surface shape have limited their representational capability. We would prefer an E that can handle as broad a range of Part shapes as possible. I will refer to this property as the *capacity* of E.

E-Rigidity If $\mathcal{V}$ is the initial part shape, how well does E preserve the shape properties of $\mathcal{V}$ as $\mathcal{U}$ varies? Of interest are both mathematical rigidity - for example the deviation in local curvature across $\mathcal{V}$ - and how rigid the part "feels" to the artist.

3.5 Existing Approaches to Implementing Parts

Many existing works in computer graphics have either implicitly or explicitly defined Parts that can be explained in terms of my the domain/shape decomposition. In fact, this decomposition makes clear the commonalities and differences between many methods. I will now briefly review the different alternatives.

3.5.1 Part Domain

I have identified five different ways of describing the part domain. Four largely follow my approach of defining the domain independently of the shape, and one takes the inverse approach of determining the domain from a rigid surface patch in global position.

Spatial Volume

An region on a surface can be defined by the intersection of the surface with a bounded 3D volume. Bounded spatial deformations such as *Wires* [Singh and Fiume 1998] take exactly this approach, and Lewiset al. [Lewis and Jones 2004] created complex procedural models using deformations of spatially-bounded mesh regions. In purely technical terms, this is an excellent representation - since it is not tied to the surface, even *Generalized Transport* is trivial. However, this same independence from the surface means that the shape of the surface domain - $\mathcal{U}$-Rigidity - is likely to be completely lost under all mappings except *Resampling*.

Projection

If we are given a PART shape as a rigid surface-with-boundary oriented in 3D, then one way to find the PART domain is to *project* the boundary of the part onto the surface. A simple type of projection is intersect a ray through each boundary point with the target surface. This sort of projection has been used in several "cut-and-paste" tools [Sorkine et al. 2004; Yu et al. 2004]. A related variant is to actually intersect the part with the target surface. Issues such as some rays missing the surface or the intersection not forming a closed loop can be handled with minimal-cut post-processing [Hassner et al. 2005]. As with the spatial volume, though, projection-based region definition involves a geometrically independent oriented 3D shape which is in fact tightly coupled in an aesthetic sense, making it extremely difficult to maintain $\mathcal{U}$-Rigidity.

Explicit Region on Surface

Perhaps the most obvious way to define a region on a discrete surface such as a polygonal mesh is as a list of adjacent faces or vertices. A straightforward generalization is to store the region boundary $\partial\mathcal{U}$ as a poly-line embedded in the surface, for example as vertices stored in mesh faces using barycentric coordinates. This approach has been put into use in some mesh cut-and-paste works, where the artist must define a suitable pasting boundary on the target surface [Yu et al. 2004; Funkhouser et al. 2004; Sharf et al. 2006; Huang et al. 2007].

The region-on-surface representation is appealing in its simplicity. However, because the region must be embedded in the 3D graph of the current surface representation, it is completely dependent on the current graph. *Deformation* can be trivially handled if the PART domain can be permitted to deform with the mesh. This may be artistically acceptable, but if not, there is no clear way to improve $\mathcal{U}$-Rigidity. *Resampling* can be accomplished by projecting the region boundary onto the new surface, and *Surface Transport* is possible by advecting the part boundary along the mesh, though it can be difficult to preserve $\mathcal{U}$-Rigidity [Pedersen 1995; Suzuki et al. 2000].

The key drawback of storing an explicit surface region is that there is no straightforward way to transfer a subset of one mesh topology to another completely different surface. Hence, all existing techniques which have performed *Generalized Transport* operations on these surface region representations have done so by first converting to a representation based on a region in a parameterization.

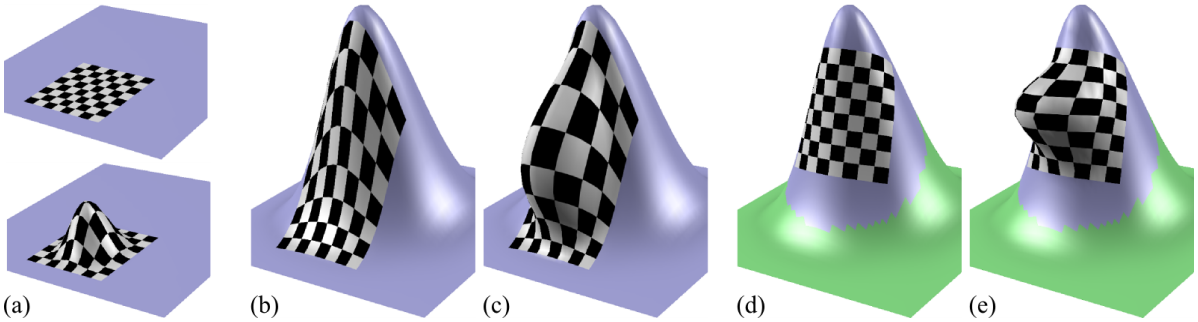


Figure 3.11: *In (a), a displacement "bump" is layered onto a planar NURBS patch by mapping displacements using the intrinsic NURBS parameterization. This parameterization is stretched as the planar patch is deformed (b), resulting in a deformed displacement (c). Dynamically computing a new local parameterization (d) reduces distortion in the layered displacement (e).*

Region in Parameterization

If $\mathcal{S}$ has a global *parameterization*, ie a mapping $\mathcal{P} : \mathcal{S} \rightarrow \mathbb{R}^2$, then $\mathcal{U}$ can be defined as a map from a 2D domain in this parameterization, or $\mathcal{U} = \mathcal{P}(\mathcal{U}_{2D})$. This representation is technically ideal, in that each mapping problem can be solved simply by copying 2D regions from one parameterization to another. However, in practice $\mathcal{P}$ inevitably involves *distortion* [Floater and Hormann 2005], meaning that regions in parameter space are deformed when they are mapped to the surface. So if we follow the strategy above, copying a region from one parameterization to another will likely result in very poor $\mathcal{U}$-Rigidity[4].

Hence, like in the case of spatial volumes, the parametric-region representation appears to be independent but in fact is closely coupled with the relationship between the parameterization and the surface. This relationship is most problematic when $\mathcal{P}$ is determined a priori for the entire surface. For example, in Surface Pasting the natural planar parameterization of the base NURBS surface is used to store the domains of overlapping PARTs [Barghiel et al. 1994]. This has unintuitive effects on the PART shapes, particularly during animation [Tsang and Mann 1998]. See Figure 3.11 for an example.

$\mathcal{U}$-Rigidity can be greatly improved by dynamically computing a *local parameterization* for the PART domain. This strategy has been used by many works that attempt to transfer PART-like geometric information from one surface to another [Pedersen 1995; Kanai et al. 1999; Biermann et al. 2002; Fu et al. 2004; Sorkine et al. 2004; Brodersen et al. 2007]. Note, however, that the region over which to locally parameterize must still be determined using one of the above methods. One exception is Biermannet al. [Biermann et al. 2002], where the parameterization region was defined by a geodesic polygon around a single point on the surface. This further decomposition of the PART domain into a single point and a function on the surface provides significant advantages, and I

[4]A potential strategy here would be to optimize $\mathcal{U}$ in parameter space to preserve its 3D shape, but this has yet to be explored.

will adopt a similar approach.

Region in Manifold

An extension of the region-in-parameterization approach is to impose a *manifold* structure onto the surface. A manifold provides a continuous global parameterization constructed out of a set of overlapping *charts*. *Transition functions* connect the charts, so one can move transparently from one chart to any overlapping neighbours.

Ideally, the charts individually have low distortion, and so Part domains can be stored in the charts without penalizing $\mathcal{U}$-Rigidity. However, existing manifold implementations for spheres [Grimm 2005] and polygonal meshes [Grimm and Hughes 1995; Ying and Zorin 2004] do not have isometric charts, so any embedded region will still inherit significant distortion. However, the manifold formalism is very powerful and I will base my approach on it.

3.5.2 Part Shape

Although vastly more work has been done on defining 3D shape than on describing a surface region, I believe the alternatives can be adequately characterized by three categories.

Spatial Deformation

Spatial deformations can be used to deform any Part domain into virtually any shape, so the E-Capacity of this approach to shape description is very high. The standard approach of applying a deformation to each vertex independently does preclude any changes to the mesh edge graph, and likewise to surface topology. A larger drawback, though, is that the shape of the Part is again tightly coupled to a particular Part domain. Hence, E-Rigidity is very limited - transferring a deformation from one surface to another will generally result in a meaningless shape.

Relative Displacements

In contrast to spatial deformations, shapes defined via relative displacements can be easily transferred to another Part domain. One issue is that E-Rigidity may suffer if the surface curvature significantly varies. It is not difficult to construct scenarios where a Part that protrudes significantly from the surface will self-intersect (see [Brodersen et al. 2007] for one approach to reducing the normal distortion in displacement mapping).

The main limitation of displacement representations is that they have very restricted E-Capacity. The part shape must be representable by a vector function over a domain. In particular, this means that the Part cannot have topological handles. Geometric texture [Porumbescu et al. 2005; Elber 2005; Brodersen et al. 2007] circumvents this by embedding an arbitrary surface in a simple rectangular *displacement volume*, however the part surface must be fully contained in this volume. Another issue is resolution - if we copy a displacement from one surface to another, the target needs to have a sufficiently-dense tessellation to reproduce the sharp edges of fine details in the displacement map.

Surface Pasting [Barghiel et al. 1994] avoided this problem by trimming the target surface and directly inserting the pasted spline geometry, although at the cost of actual geometric continuity.

Differential Representation

Recent work in variational shape modeling has demonstrated the power of *differential representations* of meshed surfaces. In this case each mesh vertex is replaced with some higher-order differential information about its local surface derivatives. Once boundary constraints are fixed, the internal surface can be reconstructed from this differential information using global energy minimizations. This approach is very powerful, as configurable energy functions allow us to control the trade-off between E-Rigidity and computational efficiency. The internal shape is completely defined by the boundary and a fixed mesh topology on the interior, so PARTs with arbitrary genus can be represented.

The most efficient approaches are based on *linear variational methods*, meaning that the energies are quadratic. This can limit the ability of the methods to handle some kinds of changes in $\mathcal{U}$. For example, differential *Laplacian* vectors are defined in global coordinates, and must somehow be rotated when the orientation of the PART domain changes [Sorkine et al. 2004]. Since the PART interior is essentially defined as a smooth interpolant of the boundary values, interior boundaries (holes) require special handling. Despite these issues, differential approaches present the most compelling option for describing PART shape. Hence, I will adopt a differential approach, although one that does not depend on variational minimization.

3.6 Manifold-Based PART Representation

In the preceding sections I have considered a variety of different approaches to implementing a surface PART. There are two problems to solve: *where* the part is on the surface (a domain), and *what* the part looks like (a shape). I will now describe a new approach to implementing PARTs which draws from these existing methods, but is capable of supporting an arbitrary procedural hierarchy.

3.6.1 Where: Parameterized Geodesic Disc

Grimm [Grimm 2005] presents a compelling approach to representing an region $\mathcal{U}$ on a surface, namely by embedding it in the atlas of a manifold. The advantage of the manifold embedding over a standard global parameterization is that, in theory, the manifold charts can be nearly isometric (ie undistorted). Hence, the mapping of $\mathcal{U}$ to the surface will be as rigid as possible. But two problems arise. First, an actual implementation of a nearly-isometric manifold has yet to be developed - existing manifold techniques may have extensive distortion in the charts [Grimm and Hughes 1995; Ying and Zorin 2004; Grimm and Zorin 2006; Gu et al. 2006; Siqueira et al. 2009]. Even if the initial manifold has low distortion, new charts would need to be inserted to maintain this low distortion as the manifold is modified by editing operations. We would then need to map $\mathcal{U}$ to this

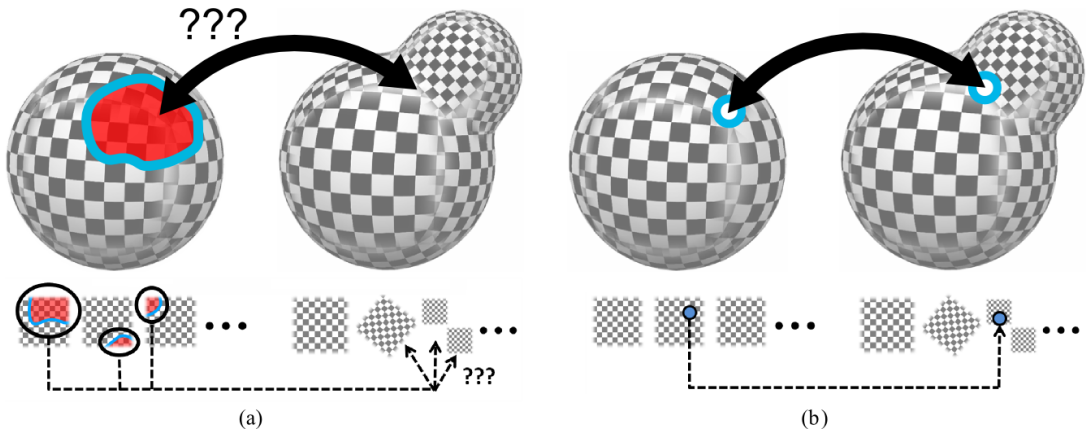


Figure 3.12: *An region (a) embedded in one manifold, which overlaps several charts, is non-trivial to map onto another manifold with a different set of charts, both in 3D and in the parameter space. In contrast, a single point (b) embedded in a manifold can trivially be mapped to a point in another manifold, either via 2D or 3D correspondences.*

new atlas (Figure 3.12), a problem which is completely unaddressed by existing work, and clearly very difficult.

Biermann's multiresolution pasting [Biermann et al. 2002] suggests a way to sidestep the domain encoding issue, by encoding $\mathcal{U}$ relative to a single point. Then only this point need be mapped between atlases, which is a much easier problem (Figure 3.12). And because the $\mathcal{U}$ is computed directly on the surface, we need not worry about distortion in the manifold. In fact, we will see in chapter 6 that we do not even need a rigorously-defined manifold atlas.

The question remains, how to encode $\mathcal{U}$ relative to a point? Biermann's scheme involved encoding a source $\mathcal{U}$ as a geodesic polygon, which was reconstituted on the target surface by tracing geodesics. I propose a simpler alternative, namely to embed $\mathcal{U}$ within a parameterized geodesic disc that contains it. Then to reconstruct $\mathcal{U}$ at a point on some other surface, we simply compute another parameterized geodesic disc (Figure 3.13). The advantage of this approach is that it introduces a further level of abstraction - once the geodesic is known, *any* Part can be inserted into it. We have hence reduced the part definition from $(\mathcal{U}, \mathsf{E})$, where $\mathcal{U}$ is a surface region, to $(\mathbf{u}, \mathsf{E})$, where $\mathbf{u}$ is a 2D region. To find the 3D surface region, we evaluate $\mathcal{U} = \mathcal{P}(\mathbf{u})$. One caveat about this approach that should be mentioned is that as $\mathcal{U}$ is now a function of a 2D region, certain Part domains can no longer be handled. In particular, the bottom rows of Figure 3.8, where the domain contains a topological handle or is cylindrical, will not be supported. Note, however, that the domain can still contain holes.

As I have mentioned several times, parameterization algorithms can introduce unexpected distortion. A key driver for this geodesic-disc decomposition is the local parameterization algorithm I will present in Chapter 4. This algorithm efficiently parameterizes a geodesic disc using *normal coordinates*, a type of intrinsic local coordinate system on surfaces which has a variety of very desirable properties for part-based interaction.

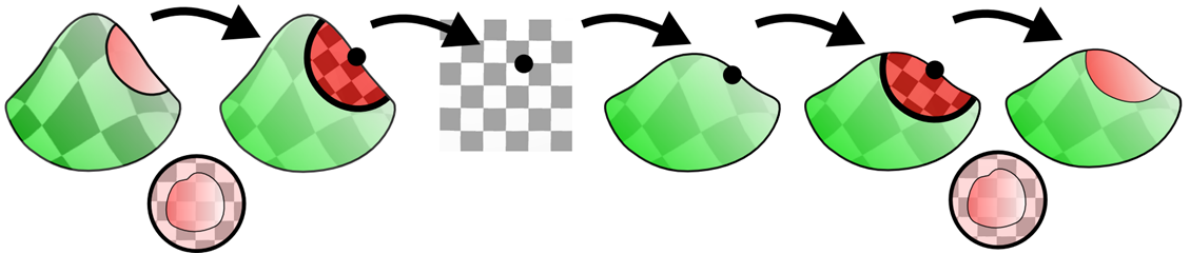


Figure 3.13: *A region on an initial surface can be embedded within a parameterized geodesic disc. Then only the center point of this geodesic disc needs to be embedded in the atlas of the underlying surface manifold, and can be mapped to a different manifold without difficulty. Then a new parameterized geodesic disc is computed, and the part domain is mapped from its parameter space to the new surface.*

3.6.2 What: Geometric Differential Deformation

I have described a conceptually sound approach to representing a Part domain, which can be easily mapped to any surface. Once the domain is known, we can easily support a variety of Part shape representation strategies. Displacement-map representations are straightforward, and I will consider procedural displacement maps in Chapter 6.

However, it would be desirable to avoid the E-Capacity limitations of displacement maps. One way to do this is to encode the Part shape $\mathcal{V}$ *relative* to its 3D boundary loop $\partial\mathcal{V}$. This is easily possible with differential representations, which can deconstruct a surface-with-boundary into a set of abstract *differential coordinates* at each internal point, and a set of fixed boundary constraints. We can then take $\partial\mathcal{U}$ to be the desired boundary of the Part on the target surface, and use a deformation to enforce the boundary constraint $\partial\mathcal{V} = \partial\mathcal{U}$.

A further advantage of the boundary-based approach is that the Part shape is largely isolated from any internal surface variations within $\mathcal{U}$ - only the boundary matters. One difficulty, though, is that variational differential techniques have some practical limitations. Hence, in Chapter 5 I will describe a novel *geometric* differential deformation, which is designed specifically to solve the problem posed here in a robust and efficient manner.

3.7 Towards A Theory of Surface Parts

In the preceding sections my goal has been to present my construction of surface Parts, and the associated on operator, as concisely as possible. An interesting question, though, is how these components fit into the larger universe of Part-based surface editing. For example, there may be types of surface operations that on does not adequately represent. Or perhaps there are surface Parts that would be better described by something other than replacement of a fixed region with a new shape.

The remainder of this chapter is meant to document some of the initial explorations I have made in these directions, which will perhaps inform future work on Part-based

surface modeling.

3.7.1 ON is not Symmetric

Throughout this chapter I have focused on the question of removing the bump PART from the sphere PART. But what if we removed the sphere instead? Remember that we have defined the bump PART as a region of the sphere-with-bump surface. Thus the sphere PART could be described as the complement of this region. If we follow the procedure I have developed, then removing this other PART will result in a sphere-with-hole surface, and leave some surface on the "back-side" of the bump.

This symmetry is appealing, but the results are nonsensical - we again have lost our sphere, and the bump has become a solid shape that would be difficult to now compose with some other surface. Hence, we must accept that there is an *ordering* to surface PARTs. This ordering is even implied in the textual description I have used - the bump is *on* the sphere.

In general, a PART must be a surface patch with boundary to be placed on another surface. A surface without a boundary, such as a sphere, cannot be placed on another surface. Hence, if A ON B is the composition operator between two PARTs, then only B can be a surface without boundary. Note, however, that both A and B can be surfaces with boundaries.

3.7.2 Other PART Operators

In this thesis I will focus on the ON operator. An interesting question is whether there are other PART composition operators we might wish to consider. The ON operator is highly general, and can express virtually any local operation that involves attaching one surface patch to another. So, unlike in the solid modeling case, ON can be used to both add and remove (apparent) volume from a surface. Similarly, inserting a boundary loop into a surface can be expressed as an ON operation.

One assumption I have made for simplicity is that the ON operator involves a disc-shaped region $\mathcal{U}$. This is not strictly necessary - most of the equations in Section 3.4 are still applicable when $\mathcal{U}$ involves disconnected components. So, for example, attaching a tube at two locations to create a topological handle can be written as an ON operation (as will be done in Chapter 6).

One requirement of the A ON B operation is that A be an surface patch with a boundary, leading to an asymmetry and imposing an ordering on the compositions. However, one can easily imagine examples in which we wish to combine two surfaces without boundaries. As such surfaces are technically volumes, we could resort to solid modeling compositions here, but there are advantages to limiting ourselves to surface-based operations, which can explicitly preserve surface-specific properties like parameterizations.

In topology, the composition of two surfaces is known as the *connected sum*, and is written $A \# B$. The connected sum involves cutting compatible holes in both A and B and joining the two surfaces along the hole boundaries. This operation, which we could perhaps call AND, could be re-cast as a pair of ON operations - one to insert a boundary loop in A, and then the second to attach this newly-created surface-with-boundary to B.

However, there is a significant conceptual difference to the *A* AND *B* operation, namely that it is symmetric.

Although I will not discuss AND any further, there have been some works which do in fact implement AND compositions for surfaces, such as Hassner et al. [Hassner et al. 2005]. In this work, the composition result is derived from the global position of the two input surfaces, and hence the result is very much like a Boolean composition. One question is whether AND can be generalized to provide more flexibility, for example to incorporate the sort of deformations we integrated into ON. Further exploration of AND, and perhaps other PART-based composition operators, are interesting potential directions for future work.

3.7.3 What is a PART?

I motivated my development of surface PARTs with the simple example of "a sphere with a bump on it", a statement which I decomposed into two objects - a *sphere* and a *bump* - and an *on* relationship between them. From this decomposition I have derived a sort of grammar for surface PARTs, albeit one with a very small vocabulary. A reasonable question to ask is whether or not I would have arrived at a different result if I had begun with a different statement. Or, in other words, how dependent on language is my notion of PARTs?

One assumption I have made is that the reader will parse "a sphere with a bump on it" into the same two PARTs that I have. One might argue that this is reasonable, *sphere* and *bump* being the two nouns in the statement. But the entire statement itself is also a sort of noun, in that it refers to an object. It is entirely possible that one who is less mathematically inclined might describe my object as "a ball with a bump on it", or even "a bumpy ball". But these two statements differ significantly in their linguistic structure, in that the latter has only one noun, and no "on"-like word that represents the composition. Of course this will not surprise any reader who has studied philosophy of language, linguistics, or natural-language processing. It is well-known that there is not a unique mapping between objects and words. But the question remains: if I had started with "a bumpy ball", would I still have arrived at *A* ON *B*?

I argue that in fact my development does not depend in any way on the linguistic parsing of "a sphere with a bump on it". Recall that this statement is meant to be a description of a 3D model. My formulation is meant to capture the part semantics of the shape, not the linguistic part semantics of the statement describing the shape. I have taken advantage of the fact that this model can be described by a relatively unambiguous sentence, but this is simply an expositional convenience. To see this, consider that "a surface with a local deformation" is a completely reasonable alternative, and can be encoded using the same *A* ON *B* operation.

One might argue that ON seems to imply a spatial relationship that is not inherent in "a surface with a deformation". But again, the use of "on" follows from the need to express an idea concisely in words. This is not the spatial "on" of "a ball on a table" that one might use in a natural language description of a 3D scene [Coyne and Sproat 2001], but rather a token indicating the mathematical constructs described above. It would perhaps be more precise to call the operator "into", because as we move forward

we will ultimately be inserting one manifold into another. And in strict mathematically terminology, the operation that ON expresses is a *fiber sum*, related to the connected sum I described above. I will continue to use "on", though, because it agrees with my intuition about how surface PARTs are most often used.

As I stated at the beginning of this section, I am not concerned with what may or may not be the "right" part decomposition of a surface. I would go even further and claim that there is no intrinsically "wrong" part decomposition. Rather, my PARTs and ON operator are a framework in which one can represent changes to surfaces in a structured way. Within this framework, the answer to the question "what is a part" is simply that a PART is any change to a surface, and ON is the means for applying that change to another surface.

3.8 Conclusions

In this chapter I explored current practices in 3D surface modeling, finding that modern tools allow artists to efficiently create incredibly detailed 3D surface models. Looking deeper, I found that artists already embed some notions of surface PARTs in their models, in the form of structured sets of quad edge loops. I then illustrated some of the limitations of the current state-of-the-art in surface modeling, by way of comparison to the capabilities of procedural solid modeling.

These findings provide a reasonably strong motivation for the development of a procedural PART-based surface representation. However, to date I have found no clear definition of a surface PART in the literature. I took the first step towards this goal, by considering several possibilities and selecting an approach that segments the PART into an on-surface *domain* and a 3D *shape*, with a boundary loop connecting the two. This decomposition naturally led us to the ON operator, which attaches a PART to another surface. I then considered some properties which we would like our PARTdomain and shape representations to have, considered existing approaches in this respect, and then synthesized a new technique based on a parameterized geodesic disc and a surface deformation. This PART is specifically designed to support *generalized transport*, the capability to move a PART from one surface to any other without requiring a known deformation.

Chapter 4

Representing Regions on Surfaces

Some material in this chapter is derived from the article "Interactive Decal Compositing with Discrete Exponential Maps" authored by Ryan Schmidt, Cindy Grimm, and Brian Wyvill [Schmidt et al. 2006]. The text and images reproduced here are used with permission from my co-authors.

4.1 Introduction

In the previous chapter I showed that a region on a surface can be efficiently represented by embedding it inside a parameterized geodesic disc. As the geodesic disc is defined by a single 3D point, it can be efficiently mapped between parameterizations without inheriting any of the distortions inherent in parameterizations.

The next question to answer is, how should we parameterize this geodesic disc? Global planar parameterization methods [Sheffer et al. 2006] provide a wide range of options, however as I will later discuss, the global optimizations used in most of these techniques have some drawbacks. Looking to differential geometry, we find that geodesic discs have a natural parameterization known as geodesic radial coordinates, an "on-surface" version of the well-known 2D radial coordinates which exist in the *tangent space*, a linear 2D space centered at point $\mathbf{p}$ on a surface. Within this space, we can convert the radial coordinates to Euclidean *normal coordinates*.

In this chapter I will provide a brief development of normal coordinates. After reviewing existing techniques which can compute normal coordinates, I will present the *Discrete Exponential Map* (DEM), a new algorithm which efficiently computes normal coordinates using Dijkstra's algorithm. The basic version of this algorithm [Schmidt et al. 2006] has some instabilities, so I will present some simple techniques to significantly improve robustness. I will then evaluate the DEM with respect to a variety of geometric considerations, like sampling and geodesic discontinuities, followed by an empirical exploration of error and convergence and a comparison to global mesh parameterization methods. I will then present my first PART-based interactive tool, a simple drag-and-drop texture compositing interface. In this context, textures are simply an alternative way to represent the PARTs of a surface which do not require discrete geometry. However, the problem is simplified compared to procedural geometric PARTs because all texture

Parts lie on a single shared surface.

4.2 Normal Coordinates

Looking to differential geometry [do Carmo 1976], we find that on a suitably-smooth continuous surface $\mathcal{S} \in \mathbb{R}^3$, a local 2D coordinate system (x^1, x^2) can be defined around any point $\mathbf{p} \in \mathcal{S}$ using the *log map*. This map takes nearby points $\mathbf{q}$ into the *tangent space* $T_\mathbf{p}$, a 2D vector space centered at $\mathbf{p}$. Assuming $\mathcal{S}$ is a Riemannian manifold, these coordinates in the tangent space are *Riemann normal coordinates*.

The log map can be defined via curves on the surface. For any neighbour $\mathbf{q}$ within a sufficiently small neighbourhood around $\mathbf{p}$, there is a *geodesic* curve γ which begins at $\mathbf{p}$ and passes through $\mathbf{q}$. At $\mathbf{p}$, the tangent vector $\gamma'(\mathbf{p}) = v$ is also tangent to M, hence $v \in T_\mathbf{p}$. As $T_\mathbf{p}$ is a vector space, we can assign a basis $(\mathbf{e}^1, \mathbf{e}^2)$ and compute *geodesic polar coordinates* (r_g, θ_g), where r_g is the arc-length from $\gamma(\mathbf{p})$ to $\gamma(\mathbf{q})$, and θ_g is the angle between $\mathbf{e}^1$ and v. We can then define the *normal coordinates* $(u^1, u^2) = (r_g \cos\theta_g, r_g \sin\theta_g)$. We will denote the normal coordinate of $\mathbf{q}$ in the tangent space at $\mathbf{p}$ as $T_\mathbf{p}\mathbf{q}$, and write $T_\mathbf{p}\mathbf{q} = \log_\mathbf{p}(\mathbf{q})$. We will also refer to the inverse of the log map, the *exponential map*, which takes points $T_\mathbf{p}\mathbf{q}$ to $\mathbf{q} \in \mathcal{S}$, hence $\mathbf{q} = \exp_\mathbf{p}(T_\mathbf{p}\mathbf{q})$.

Conceptually, then, the normal coordinates at $\mathbf{q}$ are defined by the shortest curve "in the surface" which shoots outwards from $\mathbf{p}$ (Figure 4.2a). The inverse function theorem ensures that for any differentiable point $\mathbf{p}$, $\log_\mathbf{p}$ is defined and invertible in some neighbourhood around $\mathbf{p}$, meaning that each $\mathbf{q}$ in that neighbourhood has a unique normal coordinate [do Carmo 1976]. Furthermore, on a smooth manifold the Hopf-Rinow theorem [Cheeger and Ebin 1975] states that $\log_\mathbf{p}$ is guaranteed to be *defined* on the entire surface. However, the map is still only diffeomorphic within a local star-shaped neighbourhood in $T_\mathbf{p}$, which is bounded either by the surface boundaries or the *cut locus* [do Carmo 1976; Pennec 2006].

At points on the cut locus two or more minimizing geodesics intersect, hence the normal coordinate depends on which geodesic path is taken and the log map is no longer unique. Rustamov [Rustamov 2010; Kendall 1990] states that if the radius of a geodesic disc satisfies $2r\sqrt{\max(0, K)} < \pi$, where K is the maximum Gaussian curvature within the geodesic ball (disc), then the ball is called *regular* and any two points within the ball are connected by a unique geodesic. This provides a theoretical bound on the size of our parameterized geodesic disc.

One useful property of normal coordinates is that at $\mathbf{p}$, the metric tensor g is the identity matrix [Deturck and Kazdan 1981]. Intuitively, this means that the log map is a local isometry, so distances in normal coordinates correspond to distances on the manifold, although only in the limit. As one moves away from $\mathbf{p}$, the deviation from isometry depends on the variation in Gaussian curvature, so the flatter the surface, the lower the distortion in the map between $\mathcal{S}$ and $T_\mathbf{p}$. On a *developable* surface, where Gaussian curvature is everywhere 0, the entire map is an isometry.

In Figure 4.1 a sphere is a texture-mapped using analytic normal coordinates. The checkerboard texture exhibits low distortion near $\mathbf{p}$, where the normal coordinates essentially preserve the intuitive "squareness" of the 2D texture. Note, however, that the

mapping is not conformal - angles are not preserved, and distortion in the mapping increases to catastrophic levels as the distance to $\mathbf{p}$ grows.

4.2.1 Computing Normal Coordinates

If $\mathcal{S}$ is defined analytically, it is possible to compute analytic normal coordinate expansions. This approach has been pursued primarily in the mathematical physics literature, where normal coordinates are of great utility when analyzing the curved spaces of general relativity. As an early example, Dolgov and Khriplovich [Dolgov and Khriplovich 1983] provide a fourth-order expansion of normal coordinates along a geodesic. Modern computational algebra and tensor software allows for expansions of arbitrary complexity, see [Brewin 2009] for examples and a survey of recent work.

The focus of this thesis is *discrete* surfaces, where the surface is approximated by a set of samples, possibly with mesh connectivity. In this latter case, if $\mathbf{p}$ is a mesh vertex, projection of neighbouring vertices $\mathbf{q}$ onto the tangent plane at $\mathbf{p}$ is a commonly-used local approximation to $\log_{\mathbf{p}}(\mathbf{q})$. This mapping is dependent on the normal vector and can degenerate, so Welch and Witkin [Welch and Witkin 1994] more robustly approximate $\log_{\mathbf{p}}$ on the *one-ring* neighbourhood of $\mathbf{p}$ by scaling the interior angles such that they sum to 2π. However, this method does not have an obvious extension to larger neighbourhoods.

In some sense, any parameterization algorithm [Sheffer et al. 2006] applied to a local patch around $\mathbf{p}$ will produce an approximation to $\log_{\mathbf{p}}$. Pedersen [Pedersen 1995] first defines a "geodesic quadrilateral", then generates a vector field within the interior and parameterizes the patch using iso-parametric curves of this vector field. Schneider et al [Schneider et al. 2009] apply surface-constrained mass-spring relaxation, while Shapira and Shamir [Shapira and Shamir 2009] constrain vertices at a fixed geodesic radius to a circle and then flatten the interior using mean-value coordinates. Such algorithms can make no guarantees about radial geodesic distances or angles, and hence most of the desirable properties of normal coordinates are lost.

A direct approach to computing $\log_{\mathbf{p}}$ is to trace a radial geodesic from $\mathbf{p}$ to each $\mathbf{q}$. Dijkstra's algorithm [Dijkstra 1959] is perhaps the best-known technique for approximating geodesic distances. However the piecewise linear geodesics produced by Dijkstra's algorithm always lie on graph edges and hence provide a very poor estimation of θ_g. Recent work in geodesic distance approximation can be applied, as the geodesic curve can be found by "back-tracing" through the geodesic distance gradient field [Peyré and Cohen 2005]. Mitchell [Mitchell 2000] surveys a variety of such algorithms, and Surazhsky et al. [Surazhsky et al. 2005] describe an algorithm which computes exact and approximate distances on triangle meshes, although with a high memory cost. Fast-marching solutions to the eikonal equation front propagation methods can also be applied to compute geodesic distances on meshes and point clouds [Kimmel and Sethian 1998].

Although these geodesic distance computations are often $O(N \log N)$, the back-tracing step is expensive. Brun [Brun 2008] describes a more efficient alternative, which exploits the fact that the log map is related to the gradient of the squared geodesic distance field:

$$\log_{\mathbf{p}}(\mathbf{q}) = -\frac{1}{2}\nabla_{\mathbf{y}} d_g^2(\mathbf{y}, \mathbf{q})\Big|_{\mathbf{y}=\mathbf{p}}. \tag{4.1}$$

Here $d_g(\mathbf{y}, \mathbf{q})$ is the geodesic distance field generated from point $\mathbf{q}$ and $\nabla_{\mathbf{y}}$ is the gradient at $\mathbf{y}$. Essentially, this formula says that the normal coordinate at $\mathbf{q}$ can be found by first computing the geodesic distance field emanating from $\mathbf{q}$, then finding the gradient of this field at $\mathbf{p}$. Since it would be inefficient to compute a geodesic distance field for each $\mathbf{q}$, Brun instead computes the geodesic distance field at several points surrounding $\mathbf{p}$. Using these samples he discretely approximates the gradient at $\mathbf{p}$ for each $\mathbf{q}$. This approach provides a highly accurate approximation to $\log_{\mathbf{p}}$ which reproduces any discontinuities in the geodesic distance field. Unfortunately, for our purposes this exact reproduction of the cut loci is undesirable. The methods I propose later in this chapter will ultimately smooth out such discontinuities, creating more desirable parameterizations for graphics applications.

Although not explicitly used to assign local coordinates, Zelinka and Garland's *geodesic fans* [Zelinka and Garland 2004] consist of a set of piecewise-linear radial geodesic curves traced outwards from $\mathbf{p}$. Approximate normal coordinates could be found by projecting vertices $\mathbf{q}$ onto the geodesic fan, however this is computationally intensive and may result in local foldovers if the fan is too coarse relative to the underlying surface.

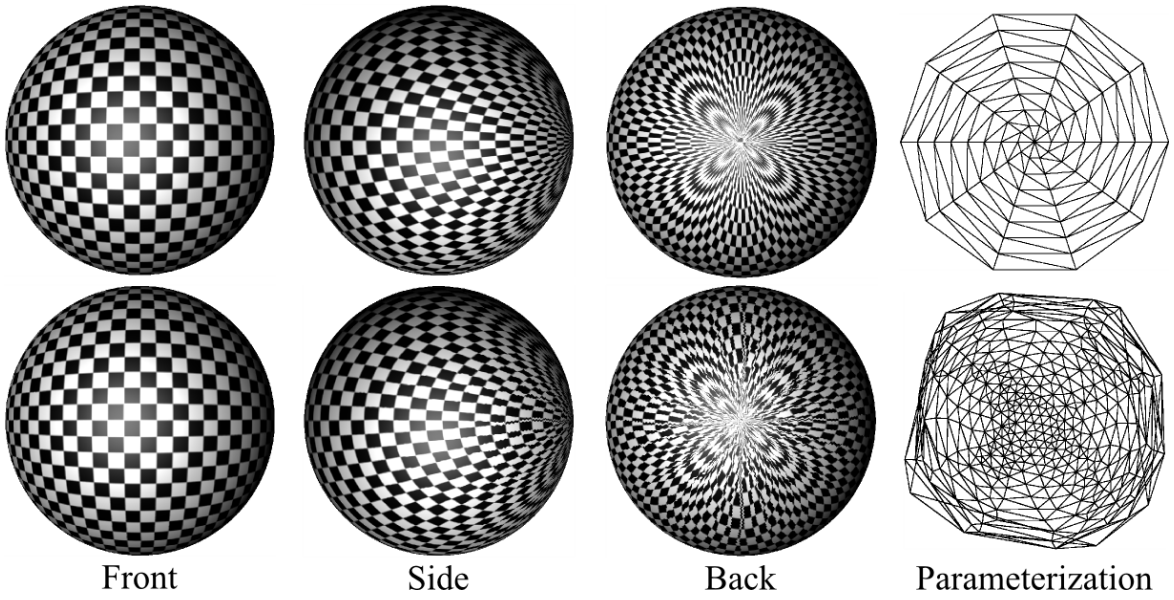


Figure 4.1: *Checkerboard texture applied to sphere using analytic parameterization (top) and discrete approximation (bottom). Front views show very high correspondence. Overall patterns in side and back views are similar, however the approximation exhibits errors accumulated during propagation. Rightmost images show 2D parameter space for lower-resolution triangulated spheres.*

4.3 The Discrete Exponential Map

Assume we are given a set of samples $(\mathbf{p}_i, \mathbf{n}_i)$ taken from some underlying smooth surface $\mathcal{S}$, where $\mathbf{p}_i$ is a point in 3D and $\mathbf{n}_i$ is the surface normal. Given a point $\mathbf{p}_i$, our goal is to assign normal coordinates to the samples within some geodesic neighbourhood of $\mathbf{p}_i$.

Rather than attempting to find the geodesic curve from $\mathbf{p}$ to each neighbouring point $\mathbf{q}$, which would then determine the geodesic polar coordinates, the DEM DEM incrementally "lifts" nearby points into $T_\mathbf{p}$ by propagating normal coordinates outwards from $\mathbf{p}$. This computation is performed directly in the tangent space at $\mathbf{p}$. The resulting algorithm requires only a simple vector addition along the piecewise-linear curves produced by Dijkstra's algorithm.

Before we begin, a note about naming. The algorithm in this section will be referred to as the *Discrete Exponential Map* or DEM. Brun [Brun 2008] pointed out that as we are mapping from surface to normal coordinates, the DEM algorithm in fact approximates the log map. However, as the algorithm projects each sample of $\mathcal{S}$ into $T_\mathbf{p}$, it effectively finds a pointwise definition of both $\log_\mathbf{p}$ and $\exp_\mathbf{p}$. Hence, for consistency with the original publication [Schmidt et al. 2006] and works which refer to it, I will retain the original name.

4.3.1 DEM Algorithm

To find the tangent-space coordinate $T_\mathbf{p}\mathbf{q}$, assume that we have a piecewise-linear path $\{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_n\}$ from $\mathbf{p}$ to $\mathbf{q}_n$, where for notational convenience, $\mathbf{q}_0 = \mathbf{p}$ and $\mathbf{q}_n = \mathbf{q}$. Then $T_\mathbf{p}\mathbf{q}$ can be defined recursively as

$$T_{\mathbf{q}_0}\mathbf{q}_n = T_{\mathbf{q}_0}\mathbf{q}_{n-1} + P_{\mathbf{q}_{n-1}} \cdot T_{\mathbf{q}_{n-1}}\mathbf{q}_n \tag{4.2}$$

where $P_\mathbf{x}$ is an operator that projects vectors from the tangent space $T_\mathbf{x}$ into $T_\mathbf{p}$. The Discrete Exponential Map (DEM) simply implements this recursion on a point-sampled surface. To do so, we must define approximations for the projection operator $P_\mathbf{x}$ and the "local" tangent-space coordinates $T_\mathbf{q}$ in the neighbourhood of any point $\mathbf{q}$.

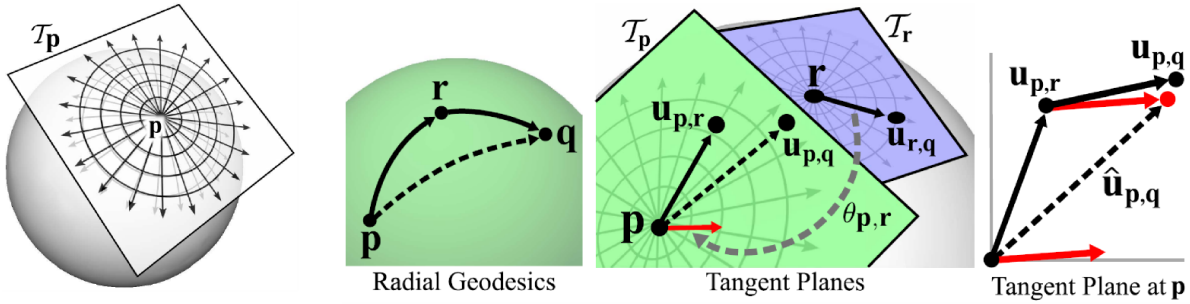


Figure 4.2: *The* exp / log *map at a point* $\mathbf{p}$ *takes geodesic curves originating at* $\mathbf{p}$ *to straight vectors emanating from the origin of the tangent space* $T_\mathbf{p}$ *(a). Iso-contours of the geodesic distance function are mapped to circles about the origin of* $T_\mathbf{p}$*. The normal coordinates* $\mathbf{u}_{p,q}$ *of the unknown radial geodesic from* $\mathbf{p}$ *to* $\mathbf{q}$ *in (b) can be approximated using the known geodesics from* $\mathbf{p}$ *to* $\mathbf{r}$ *and* $\mathbf{r}$ *to* $\mathbf{q}$*. The vector to* $\mathbf{u}_{r,q}$ *(in normal coordinates at* $\mathbf{r}$*) is transferred to the tangent plane at* $\mathbf{p}$ *using a 2D rotation with angle* $\theta_{p,r}$*, producing the red vector in (c). This vector is an approximation to* ($\mathbf{u}_{p,q}$-$\mathbf{u}_{p,r}$) *and can be added to* $\mathbf{u}_{p,r}$ *(d) to get the approximate result* $\mathbf{u}_{p,q}$*.* In this figure, $\mathbf{u}_{p,q} = T_\mathbf{p}\mathbf{q}$.

Tangent Space Alignment

I will defer the problem of local tangent-space coordinates and focus on approximating $P_{\mathbf{x}}$. Consider the simplest case: 3 points $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{q}$. Assume the local geodesics from $\mathbf{p}$ to $\mathbf{r}$ and $\mathbf{r}$ to $\mathbf{q}$ are known, but the geodesic from $\mathbf{p}$ to $\mathbf{q}$ is not (Figure 4.2b). Our goal is not to find this geodesic but rather to compute the 2D normal coordinate $T_{\mathbf{p}}\mathbf{q}$. Linearity allows us to write $T_{\mathbf{p}}\mathbf{q} = \mathbf{v} + T_{\mathbf{p}}\mathbf{q} - \mathbf{v}$, where $\mathbf{v}$ is any vector in $T_{\mathbf{p}}$. Let $\mathbf{v} = \log_{\mathbf{p}}(\mathbf{r}) = T_{\mathbf{p}}\mathbf{r}$. This is the known geodesic from $\mathbf{p}$ to $\mathbf{r}$. We now have

$$T_{\mathbf{p}}\mathbf{q} = T_{\mathbf{p}}\mathbf{r} + (T_{\mathbf{p}}\mathbf{q} - T_{\mathbf{p}}\mathbf{r}) \tag{4.3}$$

The 2D vector $(T_{\mathbf{p}}\mathbf{q} - T_{\mathbf{p}}\mathbf{r})$ corresponds to some unknown curve on the surface from $\mathbf{r}$ to $\mathbf{q}$, which in the general case is not a geodesic. Regardless, we will **approximate** this curve with the known geodesic from $\mathbf{r}$ to $\mathbf{q}$, which defines the normal coordinate of $\mathbf{q}$ in the tangent space at $\mathbf{r}$, namely $T_{\mathbf{r}}\mathbf{q}$. This is the necessary vector, however it is defined in the tangent space $T_{\mathbf{r}}$, which in general will not have the same basis vectors as $T_{\mathbf{p}}$.

To resolve this conflict, consider our 2D tangent-space vectors in their respective 3D tangent planes, which are defined by triplets of orthogonal basis vectors $(\mathbf{e}^1, \mathbf{e}^2, \mathbf{n})$, where $\mathbf{n}$ is the surface normal. We then have $(\mathbf{e}^1_r, \mathbf{e}^2_r, \mathbf{n}_r)$ at $\mathbf{r}$ and $(\mathbf{e}^1_p, \mathbf{e}^2_p, \mathbf{n}_p)$ at $\mathbf{p}$. First construct the 3D rotation $M_{\mathbf{n}}$ that aligns $\mathbf{n}_r$ with $\mathbf{n}_p$, then a second rotation $M_{\mathbf{e}^1}$ that aligns $M_{\mathbf{n}}\mathbf{e}^1_r$ with $\mathbf{e}^1_p$. By applying the combined rotation $(M_{\mathbf{e}^1}M_{\mathbf{n}})$, we bring the two tangent spaces into alignment. We can now perform the vector summation in the 3D tangent plane at $\mathbf{p}$, and project back into the 2D tangent space.

Note that $M_{\mathbf{e}^1}$ simply rotates vectors in the tangent plane around the normal $\mathbf{n}_p$ by some angle $\theta_{p,r}$. Since our goal is to transform $T_{\mathbf{r}}\mathbf{q}$, which is a 2D vector embedded in the tangent plane, we can simply apply a 2D rotation by $\theta_{p,r}$ to the tangent space. Hence, $T_{\mathbf{p}}\mathbf{q}$ can be approximated by

$$T_{\mathbf{p}}\mathbf{q} = T_{\mathbf{p}}\mathbf{r} + \text{Rot2D}\,[\,\theta_{r,p}\,] \cdot T_{\mathbf{r}}\mathbf{q} \tag{4.4}$$

as shown in Figure 4.2d (the red vector is $\text{Rot2D}\,[\,\theta_{p,r}\,] \cdot T_{\mathbf{r}}\mathbf{q}$). Note that the in-plane basis vectors $\mathbf{e}^1$ and $\mathbf{e}^2$ are completely arbitrary, the only requirement is that they be orthogonal.

Equation 4.4 involves two approximations. First, we used the geodesic from $\mathbf{r}$ to $\mathbf{q}$ to approximate the curve defined by applying exp to the line from $T_{\mathbf{p}}\mathbf{r}$ to $T_{\mathbf{p}}\mathbf{q}$. This curve is only a geodesic on developable regions, where Gaussian curvature K is everywhere 0. As the surface becomes more curved, this approximation suffers. This is easy to see in the analytic normal coordinates on the sphere in Figure 4.1 - most of the straight edges in the checkerboard clearly do not map to geodesics as we move away from $\mathbf{p}$. However, note that we use this approximation only locally.

Second, we used a simple affine transformation to map between the tangent spaces $T_{\mathbf{r}}$ and $T_{\mathbf{p}}$. Minding's theorem states that this map is an isometry only between surfaces where K is constant [do Carmo 1976]. So this map is accurate on a sphere or developable surface, but it otherwise introduces additional error.

Local Tangent-Space Coordinates

Equation 4.2 applies the projection operator which we have just defined to the "local" tangent-space coordinate $T_{\mathbf{r}}\mathbf{q}$. On a discrete surface, these local neighbourhoods will either be one-rings of triangle meshes, Euclidean ϵ-balls, or k-nearest neighbours ($k-NN$). Lacking any additional information, we can only assume that Euclidean distance is an adequate approximation of the geodesic distance [1]. Hence, we need only to estimate the geodesic polar angle θ_g.

We have the sample $(\mathbf{p}, \mathbf{n})$, as well as a set of neighbours $\{(\mathbf{q}_i, \mathbf{n}_i)\}$. First we define the tangent-plane orthogonal to $\mathbf{n}$, with orthogonal basis $(\mathbf{e}^1, \mathbf{e}^2, \mathbf{n})$. The standard approach to estimating θ_g is then to compute the angle between $\mathbf{e}^1$ and the projection of the vector $\overrightarrow{\mathbf{pq}_i}$ onto the tangent plane.

One can easily construct cases where this projection fails. For example, if $\mathbf{q}_i$ lies in a plane, the projection of $\mathbf{p}$ onto this plane falls outside the convex hull of $\mathbf{q}_i$. We can make a small improvement using the normals at $\mathbf{p}$ and $\mathbf{q}_i$. We replace the projection in the description above with a rotation around the vector $\mathbf{n} \times \overrightarrow{\mathbf{pq}_i}$ by the angle θ_q between $\overrightarrow{\mathbf{pq}_i}$ and the tangent plane. In well-conditioned cases, this gives the same result as the projection. However, if $\mathbf{n} \cdot \mathbf{n}_i < 0$, we can assume that the neighbourhood covers a region where the surface turns back over itself, for example at the rim of a cup. Then $\mathbf{q}_i$ is on the 'far side' of such a surface, and hence we should rotate in the opposite direction, by angle $\pi - \theta_q$.

This rotation approach is an improvement, but still fails to properly unfold some neighbourhoods. Generally in such cases the underlying surface is inadequately or noisily sampled, and we cannot expect the algorithm to perform accurately without additional processing of the point set. Note, however, that these projection failures are not necessarily catastrophic. In most cases they simply introduce a small local foldover for one-ring neighbourhoods. One possibility in this case would be to adapt the angle-spreading technique of Welch and Witkin [Welch and Witkin 1994]. For point set neighbourhoods, such cases usually occur at points which do not have a well-defined local manifold, and are perhaps best discarded.

DEM Front Propagation

We have defined the two mathematical steps necessary to evaluate the recursive Equation 4.2. In the notation of Equation 4.4, pseudocode for this expression is

$$T_{\mathbf{r}}\mathbf{q} = \|\mathbf{q} - \mathbf{r}\| \frac{(\ \mathbf{e}_r^1 \cdot (\mathbf{q}\text{-}\mathbf{r})\ ,\ \mathbf{e}_r^2 \cdot (\mathbf{q}\text{-}\mathbf{r})\ )}{\|\ (\ \mathbf{e}_r^1 \cdot (\mathbf{q}\text{-}\mathbf{r})\ ,\ \mathbf{e}_r^2 \cdot (\mathbf{q}\text{-}\mathbf{r})\ )\ \|} \tag{4.5}$$

$$M_{\mathbf{n}} = \text{Rot3D}\ [\ \mathbf{n}_r \times \mathbf{n}_p\ ,\ \text{acos}(\mathbf{n}_r \cdot \mathbf{n}_p)\ ] \tag{4.6}$$

$$T_{\mathbf{p}}\mathbf{q} = \text{Rot2D}\ \big[\ \text{acos}(\mathbf{e}_p^1, M_{\mathbf{n}}\,\mathbf{e}_r^1)\ \big]\ T_{\mathbf{r}}\mathbf{q} \tag{4.7}$$

where Rot3D is an axis-angle rotation matrix, and Rot2D is a 2D rotation matrix. Note, however, that care must be taken to properly handle mathematical and numerical degen-

[1] Belkin & Niyogi [Belkin and Niyogi 2008] note that such Euclidean distances approximate the geodesic distance up to order three.

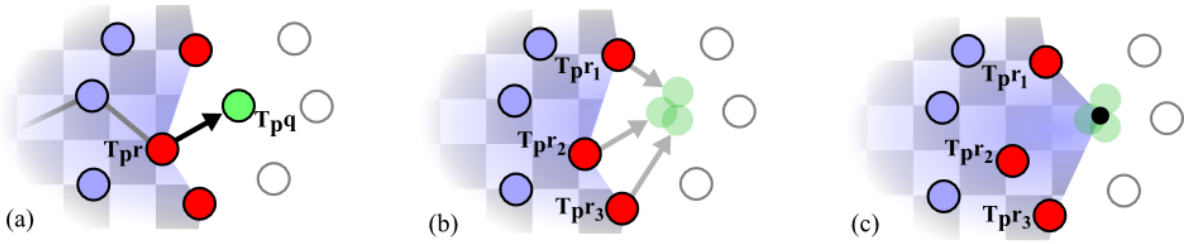


Figure 4.3: *Although the Discrete Exponential Map estimates the normal coordinate* $\mathbf{u}_p$ *from a single upwind sample (a), other nearby points on the front provide equally likely estimates (b) which can be averaged to enhance DEM robustness (c).*

eracies arising from cases such as near-parallel normals.

The only remaining question is how to define the piecewise-linear path from $\mathbf{p}$ to $\mathbf{q}$. To minimize error, this path should be as close of an approximation to the actual geodesic as possible. While there are many possible approaches, the one I propose below is efficient and simple to implement.

If each $\mathbf{p}$ is connected to a local neighbourhood of points using one of the methods described above, then we have a connected graph over the point set. Dijkstra's algorithm can be applied to find the minimum graph distance from $\mathbf{p}$ to each other point. If we set the edge costs equal to their Euclidean distances, Dijkstra's algorithm then approximates the geodesic distance. Furthermore, in computing this distance Dijkstra's algorithm also defines a piecewise-linear path from $\mathbf{p}$ to each $\mathbf{q}_i$.

The algorithm is a simple front propagation Again consider three points $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{q}$, where the geodesic distance from $\mathbf{p}$ to $\mathbf{r}$ is already known, and $\mathbf{q}$ is a neighbour of $\mathbf{r}$. The distance $d_g(\mathbf{p}, \mathbf{q})$ can then be defined as

$$d_g(\mathbf{p}, \mathbf{q}) = d_g(\mathbf{p}, \mathbf{r}) + \|\mathbf{r} - \mathbf{q}\|. \tag{4.8}$$

This equation has the same form as Equation 4.2. Hence, as Dijkstra's algorithm propagates distances outwards from $\mathbf{p}$, we can simply append another step which propagates normal coordinates. Equation 4.2 is then evaluated in-line, and as it takes constant time, the $O(N \log N)$ runtime complexity of the priority-queue implementation of Dijkstra's algorithm is unchanged.

Note that each local tangent-space vector is mapped directly into the tangent space at $\mathbf{p}$, rather than being incrementally mapped through each previous tangent plane along its path. The conditions described above on the tangent plane mapping hold regardless of the distance between $\mathbf{p}$ and $\mathbf{p}_i$. Transforming the vector through each previous tangent plane results in much higher total error.

4.3.2 Upwind Averaging

One limitation of the DEM is that any error introduced at $\mathbf{q}_i$ will be propagated to downwind points whose path passes through $\mathbf{q}_i$, potentially leading to catastrophic failures. Similarly, as the paths are completely independent the error can vary wildly between two neighbouring points. This often occurs at discontinuities in the geodesic distance field,

and will appear as 'tears' in a texture pattern mapped onto the surface using the normal coordinate parameterization. Several authors have noted that these problems tend to limit the use of the DEM to relatively flat and smooth regions [Cipriano and Gleicher 2007; Schmidt and Singh 2008].

Since the DEM sums vectors rather than scalars, $T_{\mathbf{p}}\mathbf{q}$ could be estimated from *any* nearby point whose tangent-space coordinate is already known. In the terminology of front propagation, such points are referred to as being *upwind* from $\mathbf{q}$. As the normal coordinate generated using each upwind point will be slightly different due to geometric and numerical errors, we re-define $T_{\mathbf{p}}\mathbf{q}$ as a weighted average of these estimates:

$$T_{\mathbf{p}}\mathbf{q} = \sum_i w(\mathbf{q}, \mathbf{r}_i) \left(T_{\mathbf{p}}\mathbf{r}_i + P_{\mathbf{r}_i} T_{\mathbf{r}_i}\mathbf{q} \right) \tag{4.9}$$

where $\mathbf{r}_i$ are nearby upwind neighbours (Figure 4.3) and w is the inverse distance weight

$$w(\mathbf{x}, \mathbf{y}) = (\|\mathbf{x} - \mathbf{y}\|^2 + \epsilon)^{-1}. \tag{4.10}$$

As shown in Figures 4.4 and 4.5, upwind averaging significantly improves DEM robustness, with a small 5-10% increase in runtime cost.

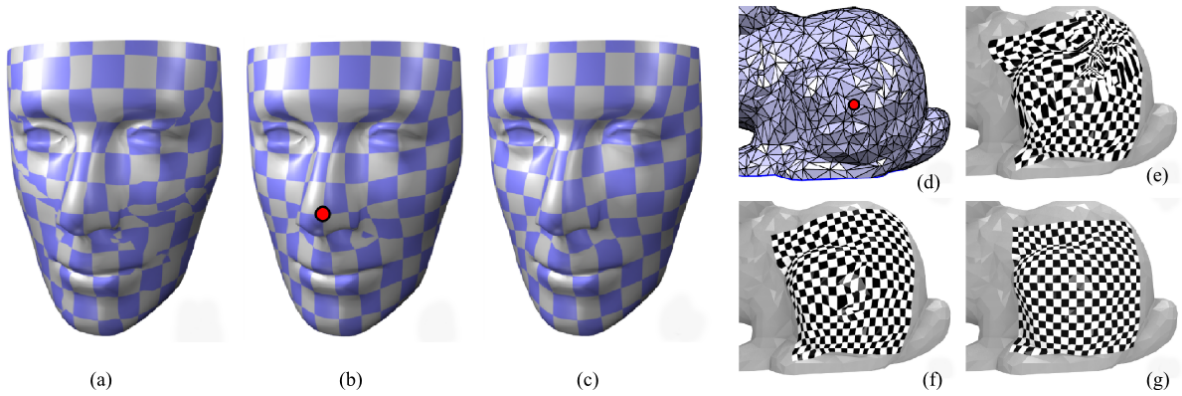


Figure 4.4: *The DEM is prone to "tearing" and foldovers when crossing regions with wide variation in curvature (a). The upwind average DEM (b) has only a few small foldovers near very high-frequency changes, which are relaxed by the addition of normal smoothing (c), resulting in a much more robust parameterization. Similarly, for certain seed points (red dot) on a highly irregular bunny mesh (d), the original DEM fails catastrophically (e). The upwind-average DEM is more robust (f), and with the addition of local normal smoothing (g) a low-distortion parameterization is produced.*

4.3.3 Normal Smoothing

As we have seen, variation in surface curvature is the root of most of the fundamental mathematical limitations of the DEM. Intuitively, if we were to apply smoothing operations to the set of surface samples, curvature variation in the surface would be decreased, so on the smoothed surface the DEM would be more robust.

Considering the equations we have presented so far, it is clear that the DEM is essentially a function of surface normals and distances between points. Local smoothing operations have a much larger effect on the surface normals than they do on the inter-point distances. Furthermore, on most discrete surfaces the normals at samples are not given, but must be estimated from the sample positions. This estimation is often noisy at best. As a result of these two observations, a reasonable strategy is to apply smoothing directly to the normal field, rather than smoothing the surface and estimating new normal vectors. The effect of doing so is essentially to relax the normal coordinate parameterization in regions of higher curvature.

Replacing each normal with a distance-weighted average of k-neighbourhood normals results in a significant improvement and can be efficiently evaluated in-line with the DEM. Figure 4.4 shows that, combined with upwind averaging, normal smoothing results in much more stable maps (Figure 4.4). Figure 4.5 compares the individual and combined effect of each of these modifications in two different cases. In the first, a spurious flipped normal introduces major failures, which upwind averaging can only smooth out. Normal smoothing repairs the flip. This sort of problem is very common when estimating normals from poorly-sampled surfaces, and hence normal smoothing has the larger visible effect. In the second row of Figure 4.5, however, the visible tearing in the parameterization is caused by geodesic discontinuities. Normal smoothing has little effect in such cases. Upwind averaging, however, integrates information from paths which have gone "around" the discontinuity, and hence after a short distance the distortion introduced by the ear has dissipated.

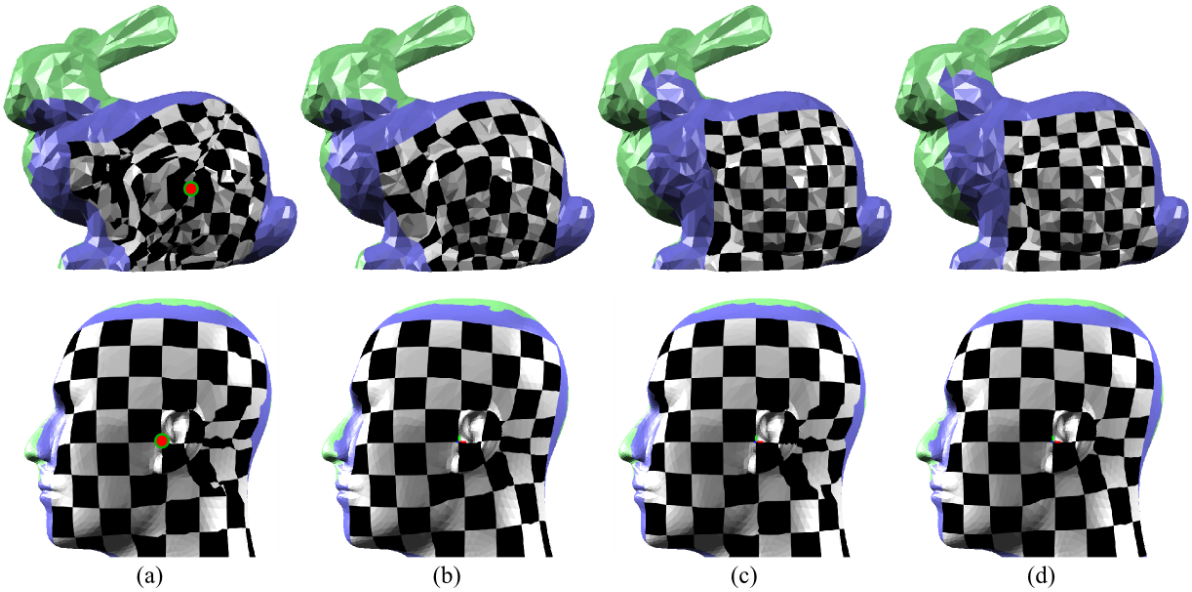


Figure 4.5: *Results computed using (a) the original DEM, (b) the upwind-average DEM, (c) the original DEM with local normal smoothing, and (d) the upwind-average DEM with local normal smoothing.*

Multiple rounds of normal smoothing will generally increase the smoothness of the parameterization, at the cost of increasing local metric distortion. If we consider the ex-

treme case of replacing all normals with a constant vector, then no frame alignments are necessary. If we also remove the re-scaling step in Equation 4.5 which preserves the local vector lengths, then the DEM is exactly a planar projection. Hence, when we smooth normals, we are effectively projecting the local tangent vectors into the tangent space of the surface described by those smoothed normals. As the normal field becomes increasingly smooth, the parameterization locally approaches that which would be computed using tangent-plane projection, while preserving sense of the surface curvature at larger scales.

This property has an interesting application for generating flattenings of local features such as holes and handles, as shown in Figure 4.6. In this case one can imagine a smooth base surface that the handles are lying on. We can attempt to infer the normal field of this smoothed surface by using many rounds of normal smoothing. Observing the normal field, we see that the portions of the handles protruding from the surface are now perpendicular to their normals. Hence, during the DEM projection, the vector sum of these components will largely cancel out, and the result is that the handles are effectively projected onto the base surface. If we remove the re-scaling step in Equation 4.5, then after projection these near-perpendicular vectors have minimal contribution, leading to a smoother result. This more general DEM has interesting capabilities, and we are currently exploring potential applications.

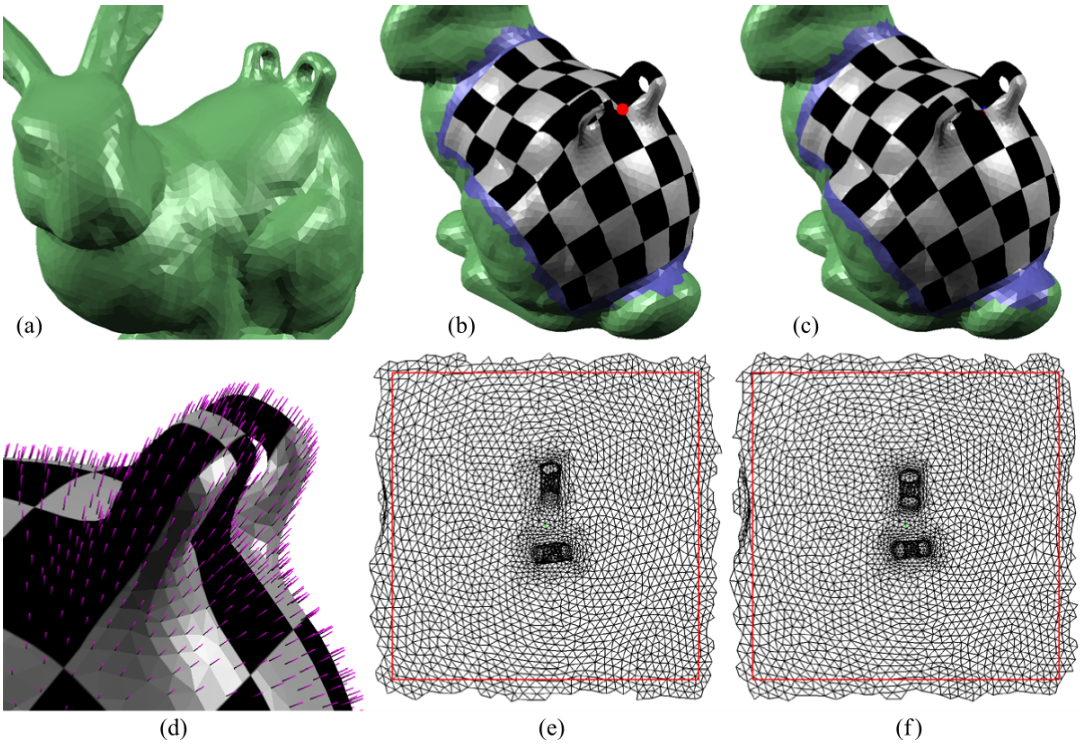


Figure 4.6: *Generally, surfaces with (a) non-planar topology cannot be parameterized to the plane. However, by using (b) a highly smoothed normal field, the DEM will essentially (c,e) parameterize the topological handles "onto" the base surface. Skipping the scale-preservation step of the local tangent-space vectors (d,f) generates smoother results.*

4.4 Properties of the DEM

In the previous section I introduced the Discrete Exponential Map (DEM) and several variants, which generate approximate normal coordinates on a point-sampled surface. There are a variety of questions one might ask about these algorithms. For example, do they reproduce properties of the analytic log map? Do they converge to analytic normal coordinates? When do they fail, and what causes these failures? In this section we will consider these and other aspects of the DEM.

4.4.1 Sampling Considerations

The sampling rate of the underlying point set largely determines the accuracy in the DEM. The first requirement on sampling rate is that local geodesic neighbourhoods must be computable. We use a k-nearest geodesic neighbour scheme, with $k = 10$. If mesh connectivity is unavailable, nearest Euclidean neighbours are assumed to be geodesic neighbours. This assumption is valid only if global sampling criteria hold [Dey and Goswami 2004]. In some cases, such as subdivision and implicit surfaces, additional samples can be generated automatically to resolve undersampling. Otherwise, algorithms that take a global approach to the neighbour-determination problem are necessary [Fleishman et al. 2005]. Note also that k-neighbourhoods have some issues on highly irregular sampling, Bendels et al. [Bendels et al. 2006] describe a symmetric $k\epsilon$-neighbourhood which robustly handles these cases.

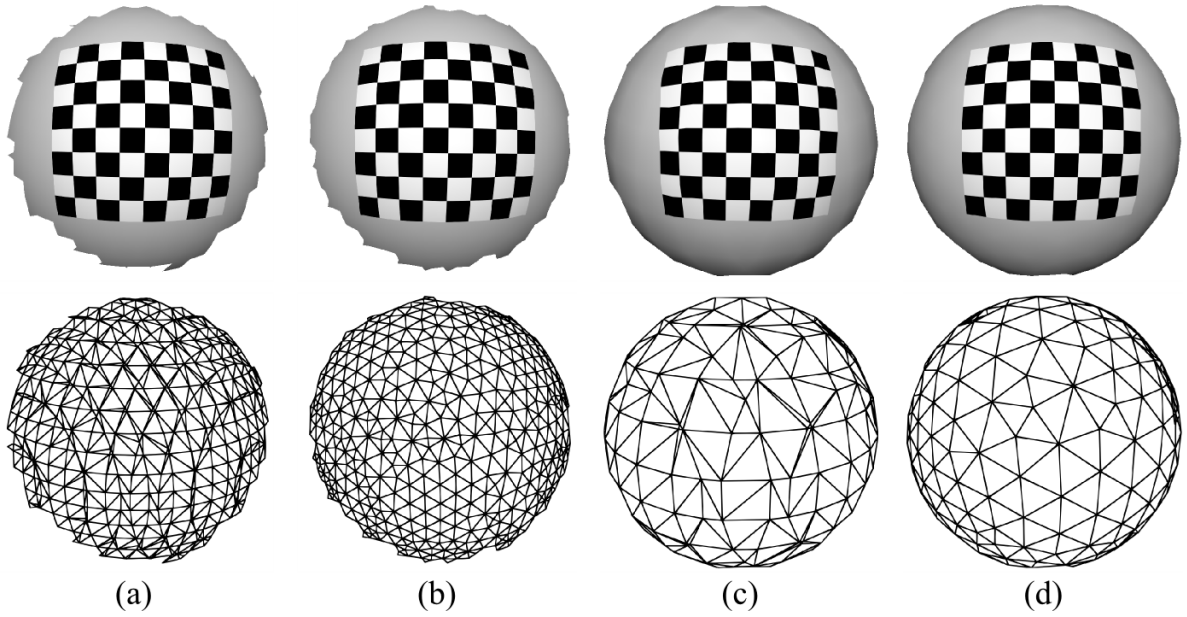


Figure 4.7: *Sample density/distribution has minimal effect on the DEM, even in cases with very irregular distributions (a,c) which would be problematic for most mesh parameterization methods due to the sliver triangles. Mesh edges are shown here to convey the sample distribution; the parameterization is computed directly from the vertex point set.*

Given adequate sampling, we have found the DEM to be very robust in practice. In particular, there is no requirement that the surface sampling be regular. The tool used to create most of the figures in this section was based on implicit surfaces visualized using marching cubes. The marching cubes implementation did not generate mesh connectivity data, so we take the vertex set as our surface sampling and generate Euclidean k-NN neighbourhoods. Marching cubes is known to produce very irregular vertex distributions, resulting in a worst-case sampling density much lower than the equivalent number of points evenly distributed. The resulting map is visually indistinguishable from that computed with a more regular sample distribution, even at moderate tessellation resolutions (Figure 4.7).

The DEM does assume that accurate surface normals are available. As shown in Figure 4.8, noise in the surface normals introduces local distortions, but the general structure of the output remains stable. This property explains why normal smoothing can have such a beneficial effect - it relaxes these small local distortions. Noise in the sample positions, however, affects the geodesic distances and is much more problematic. The unmodified DEM will quickly degenerate in the presence of surface noise, although with upwind averaging and normal smoothing performance is somewhat improved.

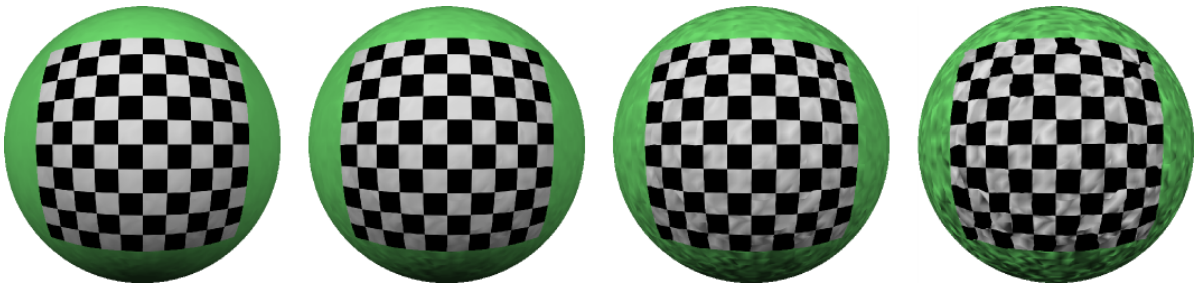


Figure 4.8: *Texture mapping from DEM normal coordinates on point sets with noise in the normal vectors. The underlying point set is sampled from an analytic sphere, however uniformly-distributed random noise has been added to the normal vectors. The maximum noise is (left to right) 5%, 10%, 25%, and 50%. Inaccurate normal vectors introduce very local distortion, however the overall structure is preserved.*

One final aspect to consider is the sample connectivity graph itself. In many applications of the DEM, mesh connectivity will be available. However, in many cases mesh connectivity is a poor proxy for geodesic connectivity. For example, consider a standard algorithm for tessellating a plane. We first dice it up into quads, and then split each quad into two triangles. In this case each vertex is connected to its 4 nearest neighbours, and 2 of the four next-nearest neighbours. This irregular pattern can lead to artifacts in the approximate normal coordinates. Hence, I have found that even when mesh connectivity is available it is preferable to use Euclidean k-NN or ϵ-ball neighbourhoods. Even these neighbourhoods can have problems near boundaries or on surfaces with highly irregular sampling. Bendels et al. [Bendels et al. 2006] discuss these problems and describe a mixed $k\epsilon$ neighbourhood that robustly handles such cases.

4.4.2 Developable Surfaces

One property of analytic normal coordinates is that on fully developable surfaces, where Gaussian curvature K is everywhere 0, there is no *metric distortion*, so the metric tensor g is the identity matrix. Intuitively, this means that the 3D surface can be unfolded into a 2D plane without any stretching, or alternately, that we could construct the 3D surface by smoothly bending a stiff piece of paper.

Some simple types of developable surfaces include cylindrical surfaces generated by extruding a curve along an infinite axis, and cones created by revolving a line around an infinite axis. As shown in Figure 4.9, the DEM reproduces this no-distortion property on fully developable cylindrical surfaces, up to numerical and discretization limits.

On the cone, however, the DEM generates artifacts at points on the "back half" of the cone. These artifacts occur because at the tip of the cone $K = \infty$, so the surface is not fully developable. Although the patch being parameterized may be developable, because the DEM is a vector summation in the tangent space, the paths to points on the back side of the cone are influenced by the non-developable apex. I will discuss this issue at length in Section 4.4.4.

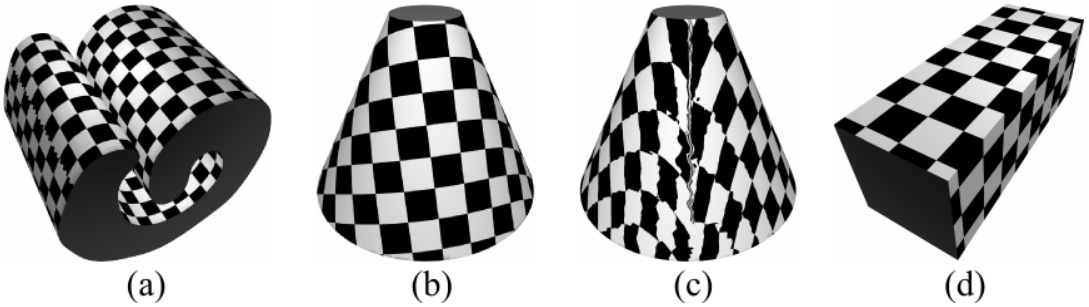


Figure 4.9: *Normal coordinate maps generated for developable surface patches - (a) swiss roll, (b)/(c) cone front/back, and (d) box. Artifacts occur on the cone because it is not developable at the apex.*

4.4.3 Geodesic Discontinuities

As we noted in Section 4.2, the log map is only diffeomorphic within a local neighbourhood. This is because on most surfaces, variations in curvature will cause the minimizing geodesics emanating from $\mathbf{p}$ to intersect at certain points on the surface, the set of which is known as the *cut locus*. If we were to visualize the scalar field of geodesic distances to $\mathbf{p}$, the cut locus would be visible as a set of discontinuities in this field. Hence, the cut locus is the set of points $\mathbf{q}$ at which there are multiple "shortest path" curves to $\mathbf{p}$ with the same length.

A simple example of the problems introduced by the cut locus can be constructed by smoothly deforming a small bump out from a plane, and computing the tangent space at some point on one side of the bump, as in Figure 4.10a-b. In this case the cut locus is a line on the surface which begins at the tip of the bump. As the DEM grows a geodesic disc outward from $\mathbf{p}$, in the tangent space this geodesic disc will be mapped to a 2D

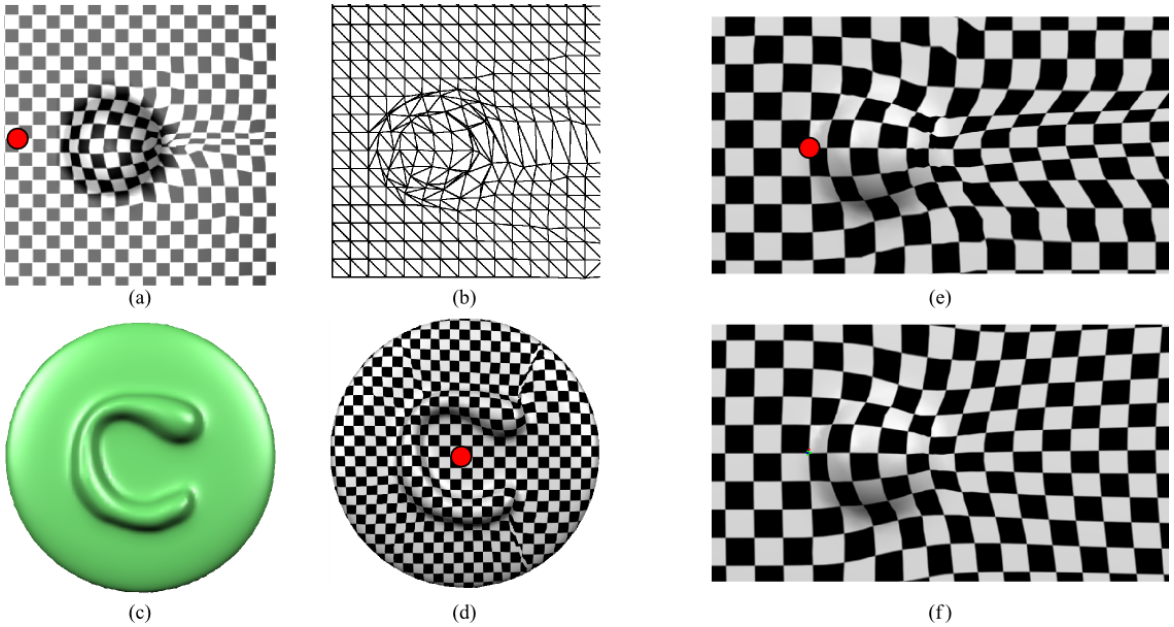


Figure 4.10: *Surface curvature leads to discontinuities in the geodesic distance field, causing stretching in the tangent space (a-b), which is visible as distortion and tearing in mapped textures (a,c,d). With upwind averaging these discontinuities are not only smoothed, but often dissipate at some distance (e,f).*

disc with a wedge cut from it (like pac-man). On a low-resolution triangular mesh, this discontinuity manifests itself as stretching of the triangles in the tangent space, which in turn leads to compression of the checkers in the texture-mapped surface.

If we increase the resolution of the surface, as in Figure 4.10c-d, the impact of the cut locus becomes more localized, and begins to appear as 'tears' in the texture mapping. Although these effects are accurate in terms of the analytic mathematical properties of normal coordinates [Brun 2008], they are problematic for many of the computer graphics applications of normal coordinates, where a smooth and connected local coordinate system is more important.

In the previous two cases, if we slightly move $\mathbf{p}$ we may observe large and seemingly random changes in this distortion. This chaotic behavior is due to the fact that in the original DEM the coordinates assigned depend on which path is taken to each vertex by Dijkstra's algorithm. At points on the cut locus, a choice must be made between two entirely different paths.

The addition of the upwind averaging described in Section 4.3.2 instead averages these paths, significantly reducing the undesirable effects of the cut locus. Figure 4.10e-f shows that not only does upwind averaging reduce the irregular variations and make the result more symmetric, it also damps out the distortion as the parameterization moves past the point where the cut locus would begin. Essentially, this is because at each point additional information from neighbours off the cut locus is being integrated, which will eventually overcome the geometric 'shock' introduced when the cut locus was first encountered. Hence, upwind averaging is in some sense analogous to the *viscosity solutions* used in front propagation, which prevent discontinuities from developing in

distance fields and other advancing fronts.

4.4.4 Gaps and Holes

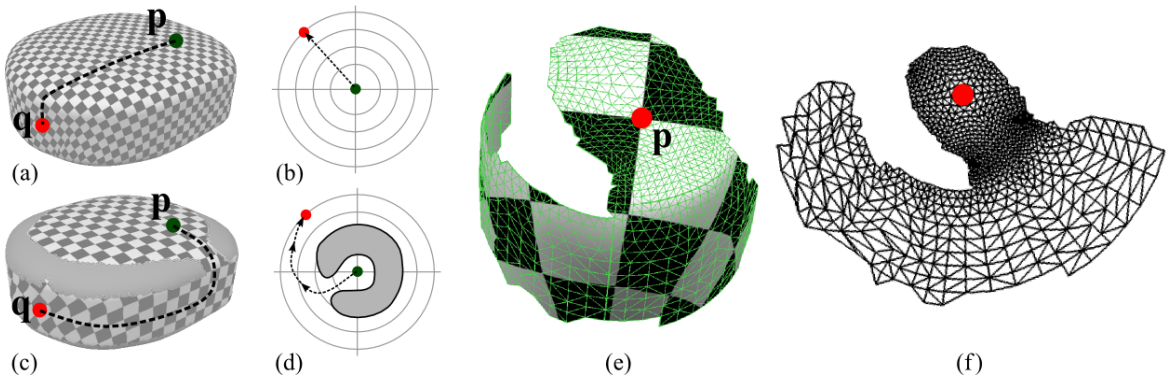


Figure 4.11: *When crossing a geodesically convex region, the path taken from* $\mathbf{p}$ *to* $\mathbf{q}$ *(a) roughly corresponds to the radial geodesic defined by* $T_{\mathbf{p}}\mathbf{q}$ *(b). When taking a path (c) around a "geodesic" hole, however, the vector sum in the tangent space produces roughly the same normal coordinate (d), and hence a radial geodesic that passes over the hole. Another example is shown in (e-f).*

Several times I have stated that the DEM propagates normal coordinates outwards from the point $\mathbf{p}$. While a geodesic propagation on the surface is used to determine the propagation *order*, the actual normal coordinate propagation is performed in the tangent space. In many cases this difference is inconsequential, but there is a specific case where it has a major impact, and that is on surfaces where the disc growing on the surface is geodesically non-convex - i.e., it has holes in it.

A simple example is shown in Figure 4.11a. On this lozenge surface, the geodesic path takes a direct route from $\mathbf{p}$ to $\mathbf{q}$. In the tangent space, this path is mapped to a straight line outwards from the origin. Now we cut away a portion of the lozenge, which is shown in grey in Figure 4.11c. The geodesic path on the surface now must travel "around" the missing region. If we consider the surface underneath this path, we see that it is close to developable, so we should expect low distortion. However, the mapped texture looks roughly the same as in Figure 4.11a.

To understand this effect, we must consider what is happening in the tangent space. A sketch of the tangent-space path taken by this geodesic is shown in Figure 4.11d. Note that the path is not shown as a straight line from the origin - ie it is not a radial geodesic. This is because the DEM sums *vectors* in 2D. We can arbitrarily perturb points along the straight-line path to $T_{\mathbf{p}}\mathbf{q}$, but the sum of the vector differences will add up to the same location, as in Equation 4.3. This same property holds, up to the errors introduced by our approximations, when we compute the differential vectors in different tangent spaces and map them into $T_{\mathbf{p}}$.

What this means is that if we remove a portion of the geodesic disc, the normal coordinates generated by the DEM will appear to "pass over" the resulting hole. Another example is shown in Figure 4.11e-f. Here we have a piece of surface taken from a rounded

cylinder. The patch is near-developable, with only a small amount of curvature along the rim. Global parameterization algorithms can flatten this surface with very little distortion. The normal coordinate map, though, has high metric distortion and the cylindrical portion is strangely curved. If we consider what the normal coordinates on the full rounded cylinder would be, however, we see that Figure 4.11f is very close to what one would expect.

The problem with the cone that was observed in Figure 4.9 is now more apparent. Even if the tip is missing, the tangent-space paths in the DEM pass through the non-developable region around the tip of the cone. This is an interesting difference with the LogMap algorithm of Brun [Brun 2008], which computes normal coordinates based on the gradient of the geodesic distance field. As this is a completely local calculation on the surface, it will not pass around holes like the DEM, and will correctly recover the developable normal-coordinate flattening of the cone. While in some cases it may be desirable that the normal coordinates take holes into account, I will show in Section 4.6.3 that we can put the DEM's handling of holes to practical use.

4.4.5 Empirical Error Analysis

The DEM assigns a normal coordinate to each 3D neighbour $\mathbf{q}$ of $\mathbf{p}$ by projecting a set of displacement vectors into the 2D tangent space $T_{\mathbf{p}}$. Since normal coordinates have an analytic definition, an obvious question is whether the normal coordinates output by the DEM *converge* to analytic normal coordinates as the sampling rate increases. One difficulty in performing such an analysis is that there are very few surfaces for which analytic normal coordinates can be computed in closed form. My results are limited to comparisons on a sphere, although I will briefly return to the question of more rigorous analysis of the DEM at the end of the section.

Geodesic Distance Error As I noted in Section 4.2, the magnitude of the normal coordinate $\|T_{\mathbf{p}}\mathbf{q}\|$ is the minimal geodesic distances from $\mathbf{p}$ to $\mathbf{q}$. In Figure 4.12 I have plotted the deviation from the true geodesic distance for each point on a sphere near-regularly sampled with 5000 points. The DEM was evaluated at the north pole, and the algorithm was implemented in C++ using 32-bit floating point arithmetic. We can immediately observe that the approximation error is quite low for the first half of the sphere (ie the northern hemisphere), but grows rapidly once we pass that point. This is in part because after passing the equator the geodesics between to "compress" as they move towards the south pole. This compression is problematic for Dijkstra's algorithm, which is forced to assign each point to one path.

Remember that one of the approximations we have made is two replace "real" local geodesics with a linear approximation (Euclidean distance). Figure 4.12 also compares using analytic and approximate local geodesics in the DEM. We see that this approximation has little effect - the accuracy with real geodesics is at best slightly improved. This agrees with Belkin & Niyogi's [Belkin and Niyogi 2008] observation that Euclidean distance approximates geodesic distance up to order three.

One important question is whether the DEM approximates the geodesic distance with more accuracy than the graph distances generated by Dijkstra's algorithm. The latter

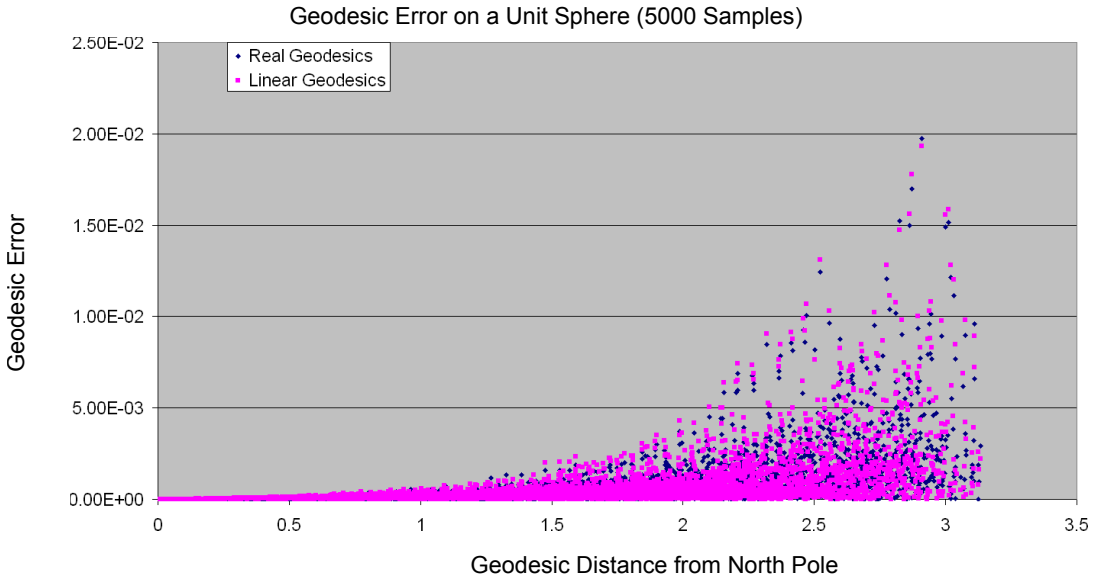


Figure 4.12: *Error in geodesic distance of the DEM on a sphere, with analytic and approximate (Euclidean) local geodesic distances.*

are known to not converge to geodesic distances under increasing sampling. As we see in Figure 4.13, the DEM is significantly more accurate. The effect of the non-convergence of the Dijkstra graph distances is visible in Figure 4.14, where we compute a normal coordinate based on the DEM geodesic polar angle, but the Dijkstra graph distances. The "wiggliness" of the Dijkstra is virtually as the sampling rate increases.

This improvement in approximation accuracy is reminiscent of the improvement in Euclidean distance approximation observed when using *vector distance transforms*, which sum vectors, instead of *chamfer distance transforms*, which sum only distances [Satherley and Jones 2001]. In some sense, the DEM can be thought of as a vector distance transform "in the surface".

Geodesic Polar Angle Error In addition to the geodesic distance r_g, the DEM also defines the geodesic polar angle θ_g (Section 4.2). In Figure 4.15 we plot the angular error computed using the same conditions as in Figure 4.12. Again, we see that the error remains very low until we pass the equator, in this case less than half a degree.

Convergence While the DEM does appear to visual converge as the sampling rate is increased, analyzing convergence more rigorously is challenging because there are variety of ways that the piecewise-linear path from $\mathbf{p}$ to $\mathbf{q}$ can vary. In Figure 4.16 I took the approach of finding the analytic geodesic, placing samples along it, randomly perturbing each sample, and then projecting it back onto the unit sphere. The values in this figure were computed in Mathematica, which uses an arbitrary-precision numerical representation.

In Figure 4.16a we consider the simplest case, where there are only 3 points. As the distance between the points increases, the geodesic error increases roughly quadratically. In the following two plots I have constructed similar cases with 50 and 100 samples per-

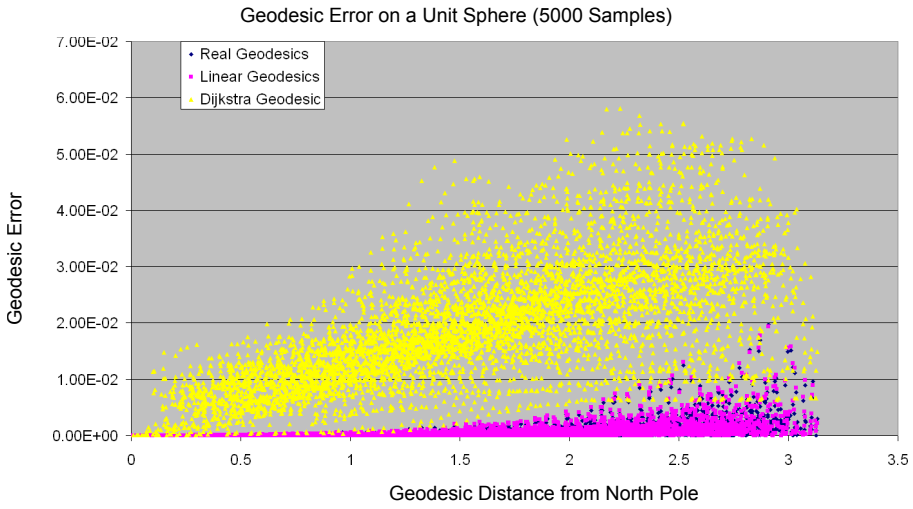


Figure 4.13: *Error in geodesic distance of the DEM on a sphere, with analytic and approximate (Euclidean) local geodesic distances, compared with the Euclidean Graph Distance calculated by Dijkstra's Algorithm.*

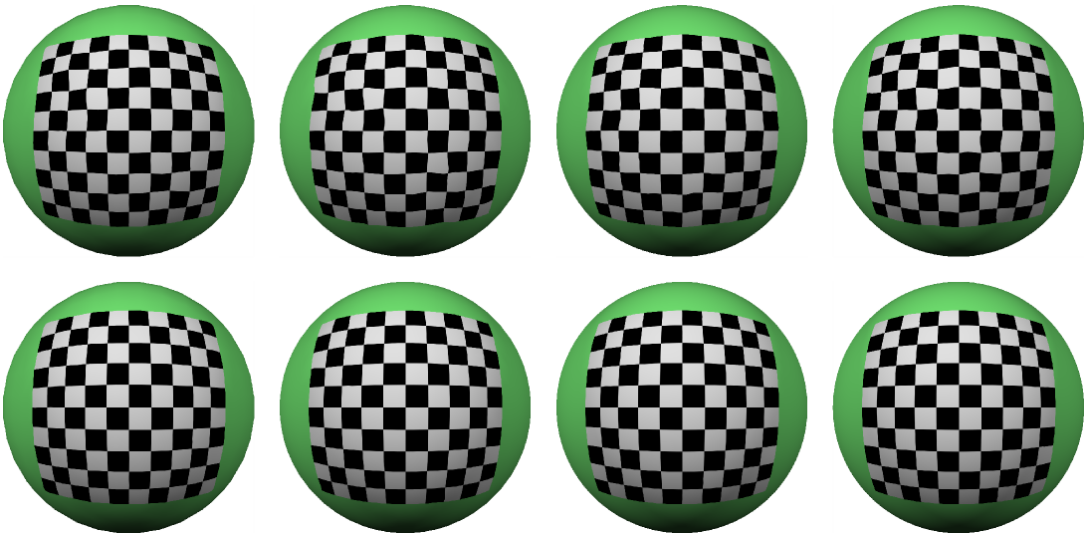


Figure 4.14: *DEM normal-coordinate parameterizations based on radial geodesic distances approximated using Dijkstra's Algorithm (top) and Discrete Exponential Map (bottom). The underlying point set is extracted from a marching cubes mesh at voxel-resolution-per-edge of 30, 50, 100, 200 (left to right).*

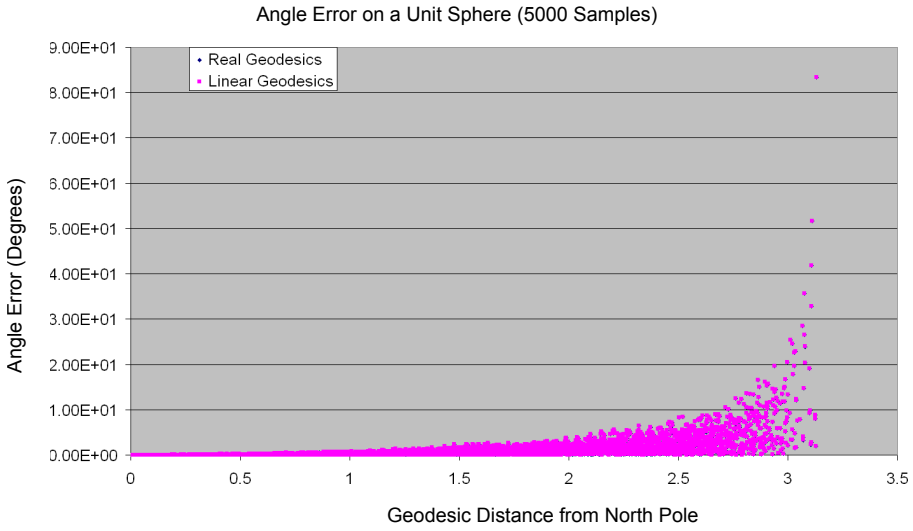


Figure 4.15: *Error in geodesic polar angle of the DEM on a sphere.*

turbed from the analytic geodesic (connectivity is fixed by the initial sampling). Again, we observe convergence as the average inter-sample distance decreases, and the convergence appears to be mildly super-linear. Note also that adding more samples does improve the approximation accuracy.

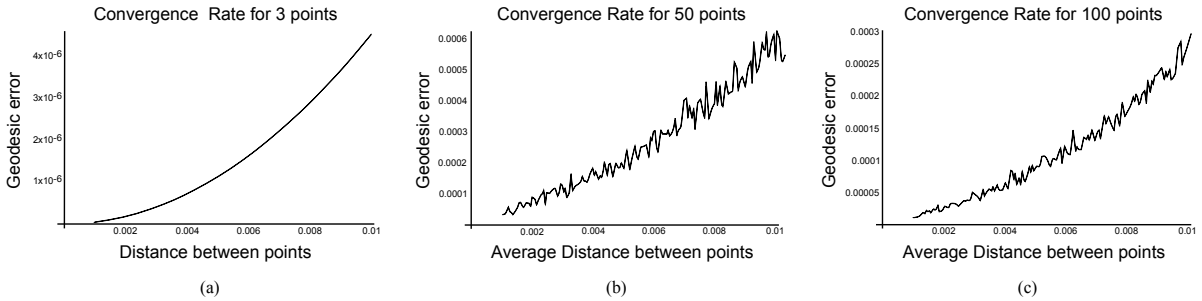


Figure 4.16: *Geodesic distances computed with the DEM are compared with true geodesic distances on a sphere, while varying the distance between the samples used in the approximation. In (a), the base-case of 3 points is tested, while (b) and (c) use piecewise-linear paths with 50 and 100 segments, respectively. The intermediate points are initialized on the geodesic and then randomly offset along the surface by up to a maximum geodesic distance (X axis). Each data point is the average of 10 runs.*

So far we have focused on the original DEM algorithm. Figure 4.17 compares convergence of the original DEM with that of the upwind-average DEM (UA-DEM). The three cases considered are the error in geodesic distance, geodesic polar angle, and the L_2 error in the 2D normal coordinates. Both the mean and max error are plotted as the sampling rate of the sphere increases. The values are computed on the first geodesic quarter of the sphere, ie points within a geodesic radius of $pi/4$ from the north pole. In this case the computation is performed in double-precision (64-bit) floating point.

We immediately observe some of the properties we have already seen, such as that the geodesic error in the DEM appears to decrease quadratically. As we might expect,

the geodesic error in the upwind DEM is significantly higher. This is easily explained if we consider that on a sphere, the 'best' distance approximation is always defined by the nearest sample. Averaging this value with others will always reduce the accuracy, causing a slight 'drag' on the accumulation of geodesic distance.

In the geodesic polar angle, however, we see that the UA-DEM does markedly better than the original DEM. These two effects seem to average out to roughly the same mean behavior in the L_2 error, which is ultimately a combination of the distance and angular errors. However, there is a visible improvement in the maximum (L_∞) error with the UA-DEM.

These results suggest an interesting possibility. The DEM clearly better-approximates geodesic distance, while the UA-DEM is more accurate on the angular error. Nothing prevents us from combining the two. Repeating the experiment in Figure 4.17, I found that this 'mixed' DEM produces an order-of-magnitude improvement in mean L_2 error, with the maximum error of the UA-DEM. Since the geodesic error in the UA-DEM is in part caused by the convexity of the sphere, this improvement may not hold on more complex surfaces. However, it is a possible avenue for future consideration.

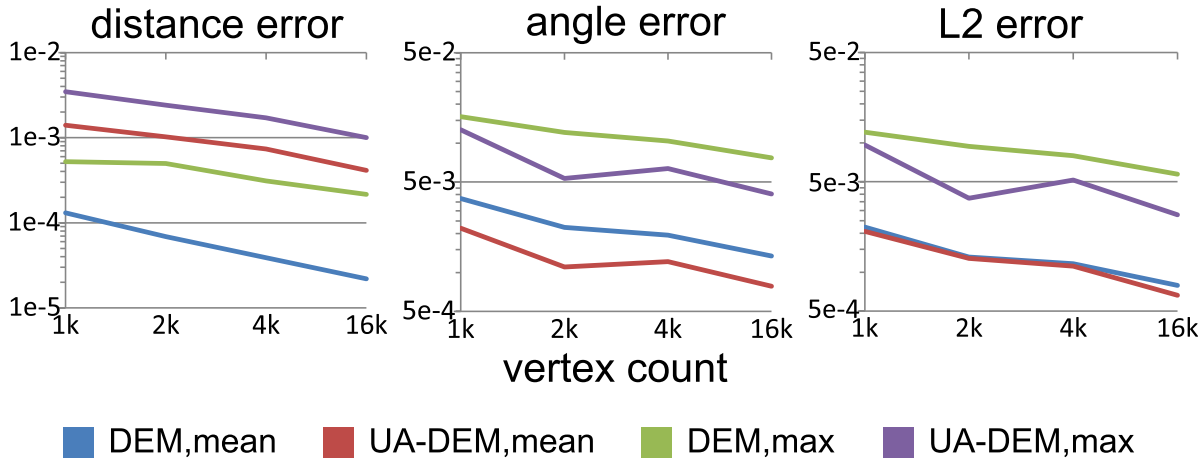


Figure 4.17: *Convergence behavior for absolute error in geodesic distance and geodesic polar angle, and L_2 error in normal coordinates, for the original DEM and upwind-average DEM (UA-DEM). Mean and maximum curves are plotted. The values are computed on samples within a geodesic disc of radius $\pi/4$ from the north pole of a sphere, sampled with the numbers of points shown. In each case the vertical axis is the error magnitude, on a log scale.*

Towards Convergence and Approximation Proofs

The empirical results above indicate that the DEM does converge on a sphere. A rigorous analytic proof of this property seems plausible, and may be of some value, particularly if it can be extended to more general surfaces. Even more desirable would be an analytic bound on the approximation error, which would allow us to predict the quality of DEM normal coordinates for a given sampling.

The main difficulty in constructing such proofs is that analytic normal coordinates themselves are extremely difficult to work with. The higher-order analytic expansions

provided by Brewin [Brewin 2009] involve several pages of formulas, with hundreds of terms. Although some proofs take advantage of the properties of normal coordinates 'in the limit', they are rarely used explicitly as local coordinate systems, even in numerical computations. Hence, existing literature provides little guidance on how to analyze the DEM, beyond the bound on local geodesic approximation provided by Belkin & Niyogi [Belkin and Niyogi 2008].

Rigorous analysis on a sphere may be considerably more straightforward, but is also of limited value because the transformation between $T_{\mathbf{r}}$ and $T_{\mathbf{p}}$ is analytically a rigid-body rotation. Minding's theorem tells us that this is not the case for surfaces with non-constant Gaussian curvature [do Carmo 1976]. On these surfaces we must take the metric distortion into account, which again is very difficult to analyze. This mapping between tangent spaces also appears to be rarely used in practice, although some recent works have done so in the case of the group of rotations, which happens to lie on a sphere [Kobilarov et al. 2009]. Crane et al. [Crane et al. 2010] have also recently provided a theory of *discrete connections*, which support parallel transport of a vector from one tangent space to another and hence may lead to more accurate computation of normal coordinates.

4.5 Comparison to Mesh Parameterization Methods

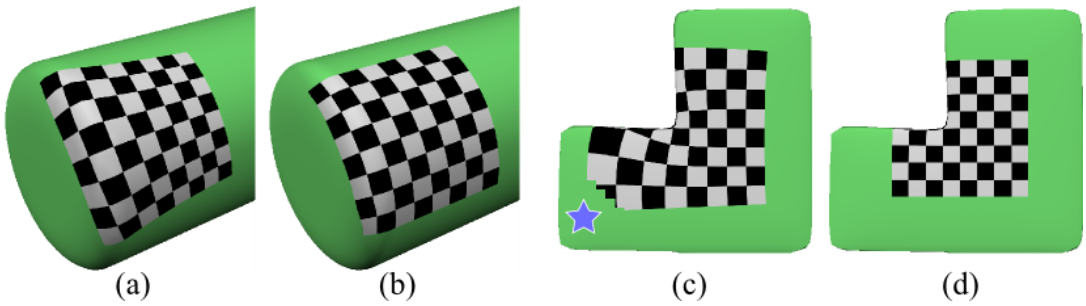


Figure 4.18: *Patches parameterized using global Conformal optimization (a,c) are less "square" than the DEM results (b,d). In (c), the conformal parameterization extends beyond the boundary of the underlying geodesic disc (blue star).*

In the context of this thesis, my goal is not to explicitly develop techniques for approximating normal coordinates, but rather to define a local coordinate system with desirable properties. The wide variety of techniques for flattening a mesh-with-boundary to the plane, known as *global parameterization*, *global* parameterization techniques, are obvious alternatives to normal coordinates.

One drawback of global parameterization techniques is that they are often computationally intensive, making their use as a component of an interactive tool impractical. Another issue is that most algorithms assume that mapping from 3D to 2D on the boundary of the mesh is known a priori. Since this is not the case in our applications, we are limited to *free-boundary* techniques.

The free-boundary technique which has undergone the most extensive study is known as global *conformal* parameterization. A conformal flattening is one which minimizes the distortion of *angles* in the map from 3D to 2D. Desbrun et al [Desbrun et al. 2002] showed that on triangular meshes, a conformal parameterization can be formulated in terms of the minimization of a simple quadratic energy, based on the interior angles of each triangle. This *discrete conformal energy* can be minimized via a sparse linear system, so it is relatively efficient, and also has a *natural* boundary condition, so only two boundary vertices must be fixed to determine the rotation and scale of the parameterization. Hence, to compute a local parameterization with discrete natural conformal parameterization (DNCP) [Desbrun et al. 2002], I first determine an approximate geodesic disc, where the geodesic radius is determined via Dijkstra's algorithm on the graph of mesh edges. The triangles within this disc are segmented and parameterized.

Figure 4.18 compares the DEM with DNCP. My main observation in these and many other cases has been that the DEM provides more predictable behavior. This often takes the form of an apparent rigidity in the DEM map. For example, in Figure 4.18a, the conformal map stretches outwards as it passes onto the cap of the cylinder, while the DEM result remains roughly square. In fact, the distortion in the DNCP map begins to occur even before the checkerboard passes over the border, which is inexplicable unless one is aware of the extends of the underlying geodesic disc.

This intuitive sense of 'squareness' generated by the DEM is perhaps due to the fact that normal coordinates preserve geodesic distances from the center point, while DNCP distorts geodesic distances to achieve lower angular distortion. Another effect of unconstrained distortion in geodesic distances is that the portion of the conformal parameterization lying in the unit-square can also expand beyond the boundary of the geodesic disc. This leads to clipping when the unit square is used for texture mapping, as in Figure 4.18c.

This inherent rigidity is a consequence of the DEM algorithm; as the normal coordinates grow outwards, they do not "communicate" with far-away portions of the surface. In contrast, DNCP and most other parameterization techniques use global optimization methods, which generally minimize a squared error. This approach is known to be sensitive to large-magnitude 'outliers', so each vertex within the geodesic disc affects all others. A practical consequence is that as the boundary or interior of the geodesic disc is modified, the parameterization can completely change. This is particularly problematic when temporal coherence is desired, see Figure 4.19.

Conformal coordinates do have some desirable properties, particularly with respect to distortion, which, thanks to the global least-squares optimization, is usually much smoother than with the DEM. As we have seen, in regions of high distortion or near the cut locus, the "greedy" nature of the DEM generates many artifacts. These artifacts can be repaired by applying conformal parameterization as a post-process. Highly-distorted regions in the DEM normal coordinates are discarded and re-parameterized using natural conformal parameterization, with the remaining DEM values used as constraints [Desbrun et al. 2002]. The result is a combined DEM / Conformal map, or *Hybrid* map. As can be seen in Figure 4.20, these hybrid maps can preserve some of the desirable properties of the DEM, such as relatively 'square' parameterizations with lower metric distortion. It should be mentioned, however, that due to the large number of constraint points they can

(a) (b) (c) (d)

Figure 4.19: *Final frames from a simple animation of a bump rising out of a flat surface with DEM (a) and conformal (b) local coordinates. The DEM result is affected only locally by the mesh deformation (c), while the conformal map deforms globally (d). This deformation is not frame coherent, but rather rapidly oscillates as the shape changes.*

be more expensive to compute, and may also be overconstrained, leading to catastrophic foldovers.

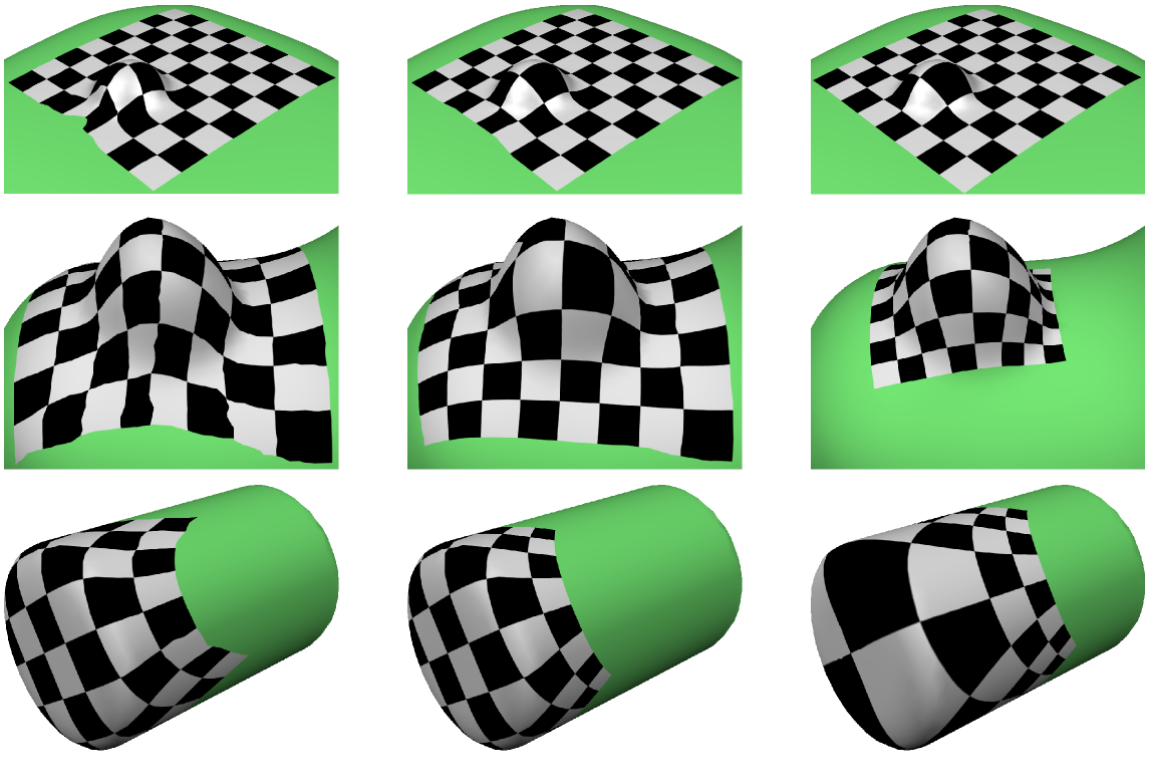
4.6 Application: An Interactive Surface Texturing Tool

Later in this thesis we will explore applications of normal coordinates to problems in geometric modeling and surface representation. However, as we have already seen in many examples, the DEM can be put to immediate use as a method to create local parameterizations suitable for texture mapping. In this section I will describe an interactive texture-design tool for procedurally compositing local texture layers defined via the DEM.

In many cases, these local textures will represent Parts of a model which could otherwise be represented via geometry, such as an eye, nose, wrinkles, fur, and so on. Without a tool based on procedural composition, the artist is forced to paint the final composition as a single holistic image. Hence, the same problems I have noted with surface modeling arise, namely how to represent the texture as a set of Parts which are straightforward for the artist to interact with. The problem is simplified because, as all the texture Parts "live" on the same surface, they can be trivially layered via painting order. But overall, the texture-Part problem can be seen as a simplified version of the more general surface-Part question I am exploring.

4.6.1 Background

Texture mapping [Blinn and Newell 1976] is one of the major stages in the modeling and animation pipeline. Texture design is generally a manual process and consumes a significant amount of the effort in most animation projects. Constrained parameterization [Lévy 2001] can provide some relief if suitable images are available, however constraint placement is tedious and may need to be repeated if the surface or texture is modified. Painting tools, particularly 3D painting systems [Hanrahan and Haeberli 1990], are the real workhorses of interactive texture design. Unfortunately these tools



(a) DEM (b) Hybrid (c) Conformal

Figure 4.20: *Comparison between DEM (a), Hybrid (b), and Conformal (c) local coordinates. The Hybrid map has less distortion but maintains a geodesic radius closer to the DEM than the conformal map.*

are relatively inflexible. Useful operations such as copy-and-paste are unavailable and there is no provision for re-using existing textures.

A third style of texturing interface, which combines aspects of both painting and constraint tools, is the *decaling* interface, introduced by Pedersen [1996]. In this approach the metaphor is that of decals, or 'stickers', which are 2D images affixed to the surface. Decals are treated as independent scene elements which are simply constrained to lie on surfaces, but may otherwise be interactively manipulated. Because a simple mapping exists between the image and the surface, 2D image processing tools can be trivially implemented. Decals are composited in real-time, mimicking 2D image compositing [Porter and Duff 1984] and vector graphics interfaces. This approach allows artists to interact with surface texture directly, using familiar 2D methods and tools.

Pedersen's [1996] pioneering work on decaling interfaces had many practical limitations. In addition to a global base parameterization, the artist had to define the four corners of each decal, making interactions like dragging cumbersome. The focus of his work was on this decal creation, rather than the interactions a decal texturing interface can make possible. Similarly, other works with decal-like approaches such as lapped textures [Praun et al. 2000] and texture sprites [Lefebvre et al. 2005] have focused on technical aspects.

I have developed a decaling interface based on the DEM, which, due to its simple non-parametric definition as a point on a surface with local radius and tangent-normal frame, is much more straightforward to manipulate than Pedersen's decals. Such decals can also be generated automatically and controlled algorithmically. Below I will consider a variety of interaction techniques, including a deformation tool and surface vector graphics, as well as decals with interior holes to reduce distortion, and experiments with decal textures on animated (implicit) surfaces, including those undergoing topological change.

4.6.2 Decal Parameterizations

In my interactive texturing system, a *decal* is defined by a square region in parameter space surrounding a *seed point* $\mathbf{p}$ on the surface. Unless otherwise noted, the parameterization in use is approximate normal coordinates generated via the DEM or one of the DEM variants we have considered. However, I will refer to the parameterization in generic terms as there is no requirement that it be derived from normal coordinates.

The decal support region is the set of vertices within an approximate geodesic distance of $r+\delta$, where δ is the largest inter-neighbour distance or mesh edge length. This ensures that the disc of radius r is fully contained within the decal, and hence all vertices of any face overlapping the decal are parameterized, avoiding any clipping. To transform the decal to the standard normalized coordinate system used for texture mapping, the parameter values are scaled by $1/\sqrt{2}r$ and translated by $(0.5, 0.5)$, so that the "geodesic square" inscribed in the disc lies at $[0,1] \times [0,1]$ in parameter space.

Since my prototype interface is based on triangular meshes, the decals are stored as local parameterizations of portions of the mesh, similar to [Praun et al. 2000]. This approach is highly interactive. Decal update times scale roughly linearly with the number of point samples, and are independent of other decals. On a 1.6 Ghz laptop, the ExpMap parameterizations in Figures 4.7a and 4.7c are computed at 121 and 362 fps (4.7a has 4.4

times as many points). User interface overhead reduces visual update times to 31 and 78 fps, respectively.

All decals are dynamically composited at each frame using the alpha blending and texturing mapping hardware found on commodity graphics hardware (Figure 4.21a). More complex compositing models can be implemented with programmable GPUs. Note that dynamic compositing may involve the storage and rendering of hundreds of decals, which is acceptable for interactive editing but less so if many decaled objects are part of a complex scene. To increase rendering efficiency, or for compatibility with existing rendering pipelines, decals could be baked into a global parameterization or octree texture.

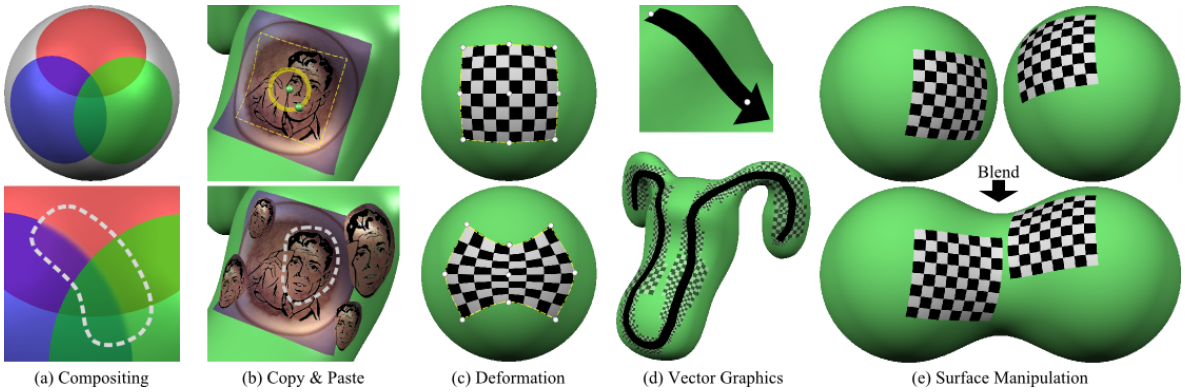


Figure 4.21: *Three overlapping semitransparent decals are composited in (a, top). A selection region is used to blur the blue decal (a, bottom) while leaving the red and green decals unchanged. In (b), the two decals inside the selection region are baked into a new decal which has been pasted back onto the surface several times. In (c), a lattice tool is used to deform the decal. A surface vector line with an endcap is created between the control points in (d,top) by rendering a 2D line into an automatically-generated decal. Surface curves can be rendered using a set of vector lines (d,bottom). The underlying decals are shown as checkerboards. In (e), the decals on two separate implicit surfaces are automatically updated when the surfaces are blended.*

4.6.3 Interaction Techniques

The goal of my decaling interface is to make 3D surface texture design as fluid and straightforward as 2D vector graphics and image compositing is with software such as Adobe Illustrator. This is accomplished by hiding all aspects of surface parameterization in the texturing interface. Each decal is an independent scene object in the system, with a simple layer order to determine the decal compositing sequence. The artist interacts with decals, rather than the underlying parameterizations.

The artist manipulates decals in our interface with a simple 3D widget (Figure 4.21b,top). The decal is moved by dragging the center point across the surface, which is implemented via ray-surface intersections. The decal orientation can be controlled by dragging on or near the yellow circle, and the radius modified by dragging the outer green point towards or away from the decal center.

Curves drawn in screen space can also be projected into decals and used to control 2D image processing operations such as blurring (Figure 4.21a). To map the vertices of the curve 'lasso' into the decal, we must between the 3D surface and 2D parameter space for points not in the initial point set. A 2D parameter $\mathbf{u}$ can be mapped to 3D $\mathbf{q}$ using barycentric interpolation in the 2D triangle containing $\mathbf{u}$ (a Delauney triangulation must first be computed for point sets). Mapping from 3D $\mathbf{q}$ to 2D $\mathbf{u}$ would ideally be done by adding $\mathbf{q}$ to the point set and recomputing the parameterization, however this is relatively expensive. A quick approximation is to propagate the parameterization from the nearest neighbour to $\mathbf{q}$.

I also experimented with dynamically compositing multiple decals into a new decal, allowing operations like copy-and-paste to be implemented (Figure 4.21b). The above curve-mapping is used to project the lasso into each underlying decal, but we also must generate a decal that fully contains the lasso, which is used as the target of the compositing operation. In general, we can generate a decal which covers a set of points $\{\mathbf{p}_i\}$ by growing outwards from a seed point until all $\mathbf{p}_i$ have been reached. To contain the lasso, we determine the seed point by projecting the center of the 2D lasso bounding box.

Decal Deformation

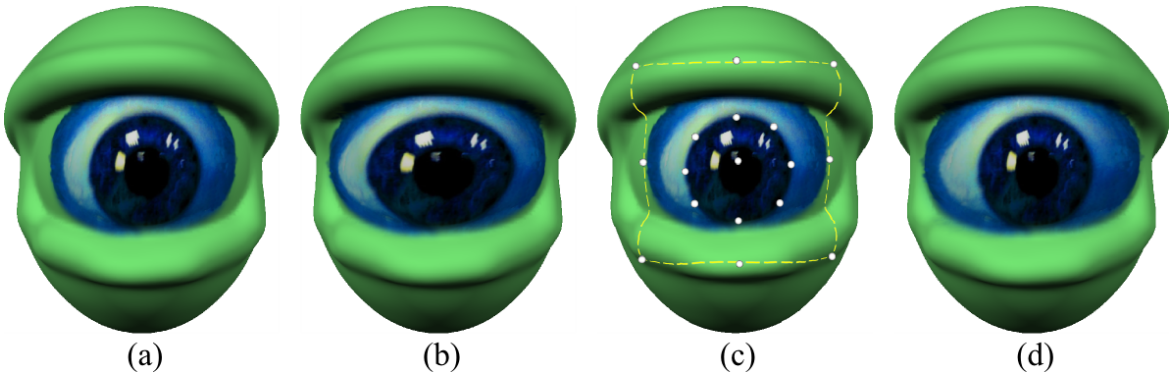


Figure 4.22: *Decal deformation. In (a), the decal image is too small to cover the eyeball. Using the lattice deformation shown in Figure 4.21 results in a stretched pupil (b). Placing extra constraint points around the pupil (c) produces the desired result (d).*

It is often useful to be able to deform a decal, either to match image features to surface features, or to correct for some unwanted distortion. Image deformations can be applied to the texture, however this approach is non-interactive for high-resolution images and limits texture re-use. Instead, we deform the decal parameterization. If the image deformation ω is desired, then the inverse deformation ω^{-1} must be applied to the parameterization. Since deformations based on the point samples may exhibit frame-incoherence, a functional space deformation is used. To reduce the likelihood of foldovers, we construct a deformation with infinite support.

Our deformation tool is based on C^2 variational scattered-data interpolation, also known as *thin-plate splines* or *radial basis functions*. A set of point constraints between sources (u, v) and destinations (u', v') is created by projecting the 3D handles into 2D. Two standard 2D thin-plate splines are fitted [Turk and O'Brien 1999], w_u which maps

(u', v') to the corresponding u values, and w_v which maps to the v values. The warp function $w = (w_u, w_v)$ is then evaluated for each point in the parameterization. Since the variational solution is based on point constraints, both lattice-style deformation (Figure 4.21) and feature alignment (Figure 4.22) interfaces can be implemented using this technique.

Surface Vector Graphics

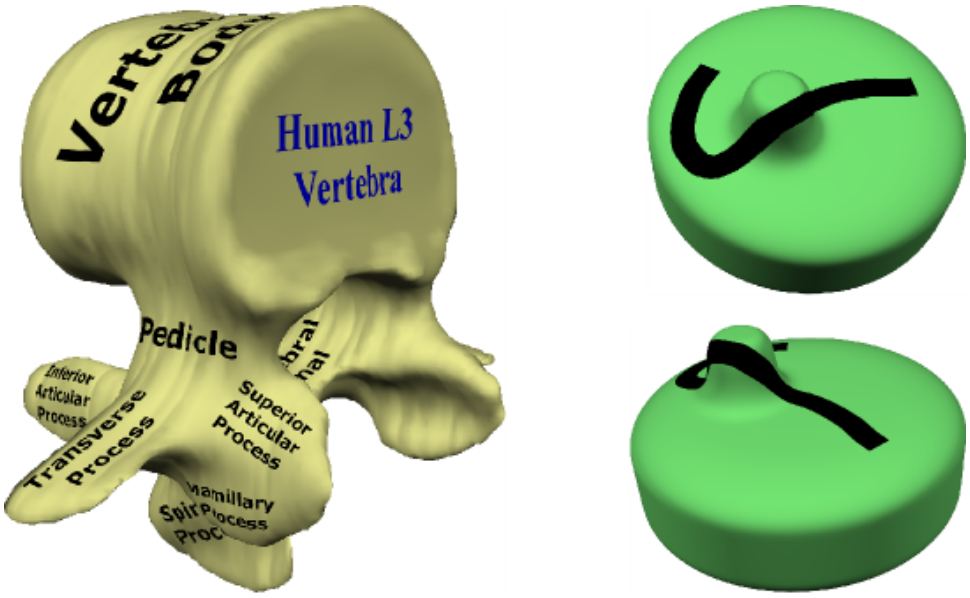


Figure 4.23: *In (a), a vertebra model is extracted from a volume dataset and annotated using decals. Though the surface is quite rough, there is little distortion in the text. In (b), the vertices of a screen-space stroke (top) are projected onto the surface and connected with vector lines. The stroke is continuous across occluded regions (bottom).*

Decals can be used as canvasses for standard 2D vector graphic elements. For example, a geodesic line element can be created between two points by generating a decal originating at one point with a radius equal to the geodesic distance to the second point (Figure 4.21d). If the decal is relatively un-distorted, uniform line width can be maintained by scaling linearly based on the decal radius. Lines across regions of widely varying curvature can be subdivided into multiple decals to reduce distortion.

These geodesic line elements can be used as a building block for other surface vector elements. As an example, arbitrary screen-space curves can be 'painted' onto the surface by projecting the 2D vertices onto the surface via ray-intersection and then connecting them with line elements (Figure 4.23b). One advantage of this approach is that portions of the surface which are occluded can still be painted, unlike the projective texturing found in 3D painting tools.

Decals can also be used to efficiently map text onto 3D surfaces, as shown in Figure 4.23a. In this case the entire string of text is rasterized into a single decal. Cipriano et al. [Cipriano and Gleicher 2007] alternately render each character into a separate

decal, and then layout the characters along 3D space curves. That same work also uses DEM decals to implement a variety of iconic labeling strategies on molecular surfaces, each of which is essentially a surface vector graphic.

Partial Decals

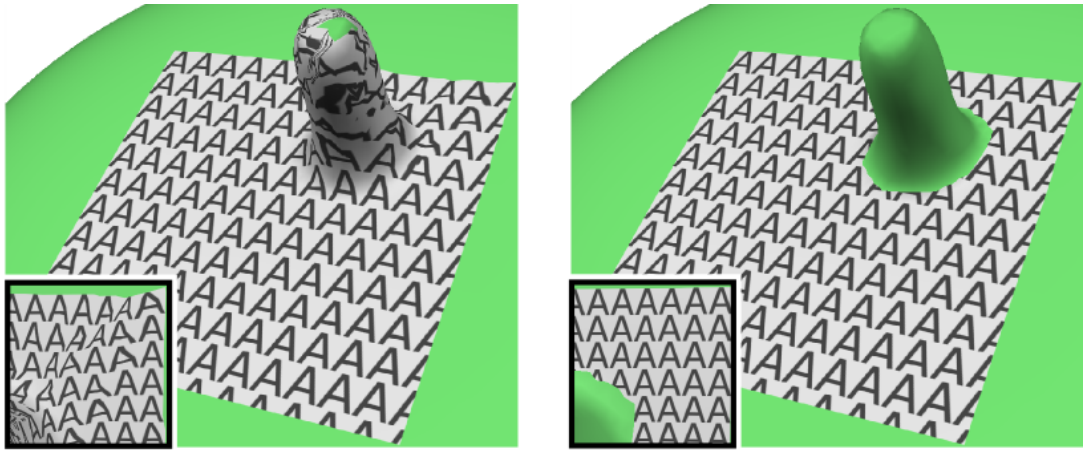


Figure 4.24: *In the case of a relatively smooth surface with a high-frequency feature, the decal is significantly distorted (a). By halting the vector propagation based on curvature magnitude, a hole is created as the decal passes around the feature (b). The distortion behind the feature is also avoided (insets).*

Like patchinos [Pedersen 1996], DEM-based decals become highly distorted when passing over large protrusions on an otherwise relatively flat surface (Figure 4.24a). Pedersen noted that this situation could be avoided if the patchino contained a hole which passed around the feature. His system could not support this because the patchino optimization procedure required a complete mass-spring mesh, however it is trivial to implement with DEM decals.

To create partial decals, the unwanted points are removed from the neighbourhood graph used in the Dijkstra-based front-propagation algorithms, and hence are simply left unparameterized. The DEM then simply passes around the "hole" in tangent space, as was discussed at length in Section 4.4.4. In the interactive tool we simply halt the front propagation at points whose absolute Gaussian curvature is larger than a user-defined threshold (Figure 4.24b). Other alternatives could include halting at creases, or based on boundaries painted by a user.

Surface Deformation and Animation

Most texture-mapping schemes assume that the underlying surface is either completely static, or if deforming, has a fixed mesh topology. Significant surface deformation or topology change requires re-assignment of parameter values, at which point the current texture must be projected from the old to new parameter-spaces. Both these steps - re-parameterization and texture projection - are non-trivial, and it is particularly difficult

to transfer texture between parameterizations without introducing frame incoherence. Even handling simple remeshing can be problematic.

Our DEM decals are relatively stable under remeshing (Figure 4.7), and as the decal parameterization is automatically generated, can easily be recomputed after the surface deforms or changes topology. The only complication is that the seed points must 'track' the changing surface. This can be implemented in a variety of ways, the simplest being to project each seed point onto the nearest surface point at each frame. Using this strategy, decals tend to flow onto the new surface in a consistent and predictable manner (Figure 4.25).

If the surface has underlying structure, for example if it is generated based on a hierarchical volumetric model, we can also 'bake' the seed point into the local coordinate frame of a particular node. This approach is particularly effective for texturing hierarchical implicit surfaces [Wyvill et al. 1999; Schmidt et al. 2006]. Of course, an artist could also manually keyframe the parameters of a moving decal.

High distortion in the decal parameterization can introduce noticeable frame incoherence, as the parameterization is likely to "pop" from frame to frame. The popping occurs because the sampling rate is too low in the distorted regions, so denser and more regular sampling can help to some extent. Upwind-averaging and normal smoothing 4.3.3 can also significantly improve continuity between frames.

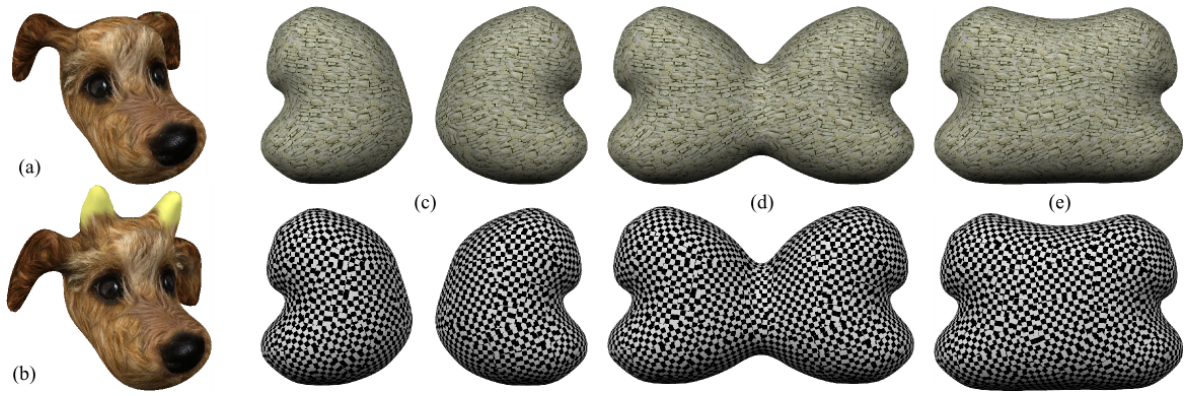


Figure 4.25: *The existing set of decals used to texture a dog model (a) are automatically updated when the ears are rotated and horns added (b). No manual adjustments of the decals were performed. In (c-e), frames are shown from an animation of two implicit surfaces blending, where each surface is textured with lapped decals. The top row shows the lapped textures, the bottom shows the same decals with a checkerboard texture. The decals smoothly flow into the blending region, and appear to slide together in the video. Only the decals covering the portion of the changing surface are affected, decals on the far sides of each object are stationary.*

4.6.4 Results

Combined with a digital camera, the decaling tool described above provides a quick and easy-to-use interface for texturing 3D models. Using an existing model and a few snapshots, the dog in Figure 4.26 was composited in a few minutes. The base fur texture

was generated by distributing fur decals across the surface, essentially creating a lapped texture [Praun et al. 2000].

The textures for the gremlin model (Figure 4.26) were taken from a variety of sources, including photographs of a frog, a painting, and the author. The images were segmented and re-colored using image-processing software, and then pasted onto the surface. Many of the textures are re-used multiple times. For example, each finger and toe uses the same set of decals, although they are deformed to better fit the particular geometry.

Figure 4.26: *This gremlin model (left) was textured using 19 different images cut out of various photographs and then manipulated using 2D image editing software. The texture consists of 392 decals, although only 78 were placed manually. The 6 images used to texture the hand are shown. For the dog model (right), textures were taken from the two source images shown. Generating the lapped base fur texture takes from a few seconds to a few minutes, depending on the number of decals and the sampling rate of the underlying surface. The eyes, nose, and extra fur textures take only a minute to position. The extra fur textures (and the fur on the edge of the eyes) help break up the semi-regularity of the base texture, creating a much more compelling result.*

An example of modeling a real-world object is shown in Figure 4.27. A rough 3D model was generated using sketch-based modeling software, and then 22 snapshots of the clay statue were taken. A base texture was generated using the lapping technique, and then relevant features were cut out of each image and attached to the model. Texture placement was very quick, the total time including taking photographs and cutting out the relevant features was approximately an hour. However, low lighting conditions and "automatic color balancing" hardware on the digital camera resulted in each photograph (and its feature textures) having a slightly different color balance from the base texture and eachother. Manual color balancing took an additional two to three hours.

4.7 Conclusions

In this chapter I presented the Discrete Exponential Map, a novel method for constructing an intrinsic coordinate system on a 3D surface. Although the original DEM algorithm had some limitations in the types of surfaces it could handle, the upwind-averaging and normal smoothing extensions I presented significantly improved DEM robustness. I evaluated

Figure 4.27: *A clay elephant statue (left) was modeled using sketch-based implicit-surface modeling software. Then, a lapped base texture and 25 feature textures were extracted from 22 images taken with a digital camera and composited on the surface. Photography, image creation, and texture positioning was completed in under an hour.*

several properties of the DEM, and performed some initial empirical explorations which indicate that at least on the sphere, the DEM appears to converge to analytic results. I also compared to conformal parameterization and found that the geometric rigidity of the DEM is likely to be preferable for Part-based modeling.

Given this geometric component, I described a surface texture design interface based on decals, which can be interpreted as texture Parts. This interface provides designers with high-level tools to composite and manipulate decals, hiding all aspects of the underlying parameterization. This is a fluid and efficient way to texture surfaces, particularly for those who are not skilled at texture painting. Even for texture painters, the ability to easily mix-and-match from existing textures makes decaling a useful addition to the texture design toolbox.

While I will focus on using the DEM as a basis for 3D surface Parts, there are a wide range of other applications that have yet to be explored. The local 2D coordinate system provided by the DEM has many potential applications in geometry processing, as a generalization of the venerable vertex one-ring. For example, I have recently shown how to use the DEM to compute approximate conformal parameterizations of point sets [Schmidt and Singh 2010a].

Clearly a more sound theoretical basis for the DEM would be useful if it is to be used as a component in other algorithms. In particular, more robust bounds on the radius of a DEM would be useful. In Section 4.2 I did mention that on continuous surfaces, a bound is known. However, upwind averaging and normal smoothing essentially sacrifice some normal coordinate accuracy for increased robustness across curvature variation. For most applications of parameterized local regions, this is a good trade-off - the method of Brun [Brun 2008] reproduces the cut loci much more accurately, and it is clear from figures there that these discontinuities would be problematic for our application. So it would be useful to know how these intrinsic smoothing techniques affect the location of cut loci.

Another interesting question is whether the DEM can be extended to arbitrary dimensions. This is non-trivial, as the rotations used to align tangent spaces are much more complex to compute in higher dimensions [Mortari 2001]. But robust local parameterizations of low-dimensional manifolds embedded in high-dimensional spaces would

have a host of applications not only in computer graphics, but also in computer vision and machine learning [Brun 2008].

Chapter 5

A Geometric Differential Representation of PART Shape

Some material in this chapter is derived from the articles "Drag-And-Drop Surface Composition" and "Drag, Drop, and Clone: An Interactive Interface for Surface Composition" authored by Ryan Schmidt and Karan Singh [Schmidt and Singh 2010b; Schmidt and Singh 2010c]. The text and images reproduced here are used with permission from my co-author.

5.1 Introduction

Recall that in Chapter 3 I separated the description of a PART into two components - a surface region $\mathcal{U}$ and a shape representation relative to that region, $\mathcal{V} = \mathsf{E}(\mathcal{U})$. Assuming we are given an initial PART shape $\mathcal{V}^0$, our goal is then to *encode* $\mathcal{V}^0$ such that it can be reconstructed by E relative to any $\mathcal{U}$. To accomplish this decoupling, we have three potential geometric entities which can be utilized - the anchor point $\mathbf{p}$, the boundary curve $\partial\mathcal{U}$, and the surface patch $\mathcal{U}$. For height-field features $\mathcal{U}$ is a natural choice, but most features of interested are not representable by 2D functions. If the feature is to be attached by some sort of world-space geometry blending, the frame transformation defined by $\mathbf{p}$ may be suitable. However, as I explained in Section 3.6.2, I will build my representation on $\partial\mathcal{U}$ as I believe it supports the most suitable trade-off between E-Capacity and E-Rigidity.

Hence, my goal is is to encode $\mathcal{V}$ relative to the boundary loop $\partial\mathcal{V}$, so that given a new boundary curve we can decode $\mathcal{V}$ in a way that naturally leads to $\partial\mathcal{V} = \partial\mathcal{U}$. Recent variational approaches to shape deformation [Alexa 2003; Sorkine et al. 2004; Lipman et al. 2005; Botsch and Sorkine 2008] are an obvious way to approach this problem. However, as I will discuss in Section 5.8, variational techniques introduce several constraints on both the types of shape that can be represented, and what controls can be provided to artists to manipulate those shapes.

Many of the limitations of these variational methods can be tied back to their dependency on global energy minimizations. An alternative is to represent the deformation geometrically. for example, the WIRES method [Singh and Fiume 1998] and vari-

ants [Milliron et al. 2002] can be easily adapted to this problem. However, as I will show, straightforward application of existing geometric deformations leads to undesirable shape distortions. Hence, in this chapter I will generalize WIRES-style relative-deformation approaches, and then show how a specific type of incremental relative deformation greatly improves the preservation of Part shape. This deformation is in some sense a single long WIRE wrapping around the surface, so I will call it a *COIL*. I will show that COILS produces results similar to the Rotation-Invariant Coordinates linear variational deformation [Lipman et al. 2005], but trades some smoothness for a variety of practical advantages.

Once the COILS deformer is defined, it can be combined with the DEM from the previous chapter to realize my general Part definition. In Section 5.9 I will put this Part to use, in a simple *mesh drag-and-drop* tool that essentially allows an artist to first extract a Part from one mesh, then procedurally move it to another. Although this tool simply "bakes" the Part into the second surface, and hence does not build a procedural hierarchy, it already demonstrates many of the possibilities of Part-based surface modeling.

5.2 Relative Deformations

My deformation technique is motivated by the simple intuition that if a rigid transformation maintains relative positions between points $\mathbf{p}$ and $\mathbf{q}$, then a deformation that is "as rigid as possible" should attempt to do the same. Since some flexibility is desired, the goal is to maintain rigidity at each point with respect to some region Ω. Many such deformation techniques take Ω to be a local neighbourhood. In this section I will explore the use of more general domains.

The first step is to construct a representation of $\mathbf{p}$ which is invariant to rigid transformations. Given some other point $\mathbf{q}$ with arbitrary orthonormal coordinate frame $\mathcal{F}_\mathbf{q}$, $\mathbf{p}$ can be expressed as $\mathbf{q} + \mathcal{F}_\mathbf{q}\mathbf{v}(\mathbf{p},\mathbf{q})$, where $\mathbf{v}(\mathbf{p},\mathbf{q}) = \mathcal{F}_\mathbf{q}^{-1}(\mathbf{p}\,\text{-}\,\mathbf{q})$ is a vector in the frame $\mathcal{F}_\mathbf{q}$ (here multiplication by a frame implies a 3D rotation with the frame vectors as rows). Applying a rigid transformation $\mathcal{M}$ results in a new point $\widehat{\mathbf{q}} = \mathcal{M}\mathbf{q}$ and frame $\mathcal{F}_{\widehat{\mathbf{q}}} = \mathcal{M}\mathcal{F}_\mathbf{q}$, from which we can compute a new position $\widehat{\mathbf{p}}_\mathbf{q}$:

$$\widehat{\mathbf{p}}_\mathbf{q} = \widehat{\mathbf{q}} + \mathcal{F}_{\widehat{\mathbf{q}}}\mathbf{v}(\mathbf{p},\mathbf{q}) \tag{5.1}$$

We can now consider integrating Equation 5.1 over a spatial region Ω. Given an arbitrary weighting function $w(\mathbf{p},\mathbf{q})$, we can construct a differential representation of $\mathbf{p}$:

$$\widehat{\mathbf{p}} = \int_{\mathbf{q}\in\Omega} \widetilde{w}(\mathbf{p},\mathbf{q})\widehat{\mathbf{p}}_\mathbf{q}\, d\Omega \qquad \widetilde{w}(\mathbf{p},\mathbf{q}) = \frac{w(\mathbf{p},\mathbf{q})}{\int_{\mathbf{q}\in\Omega} w(\mathbf{p},\mathbf{q}) d\Omega} \tag{5.2}$$

Since $(\widehat{\mathbf{p}}\,\text{-}\,\widehat{\mathbf{q}}) = \mathcal{M}(\mathbf{p}\,\text{-}\,\mathbf{q})$, this representation of $\mathbf{p}$ is invariant to rigid transformation. In the case of non-rigid deformation of Ω, each $\mathbf{q}$ undergoes a unique rigid transformation and "predicts" a different point $\widehat{\mathbf{p}}_\mathbf{q}$. Equation 5.2 defines $\widehat{\mathbf{p}}$ as the weighted superposition of these predictions. If $w(\mathbf{p},\mathbf{q})$ is continuous, a smooth deformation of Ω will result in a

smooth spatial deformation as $\mathbf{p}$ varies. Note that although Equation 5.2 can be interpreted as defining a spatial deformation, the weight and displacement-vector functions also effectively define an *encoding* of $\mathbf{p}$, which can be reconstructed relative to any Ω.

Many relative deformation techniques can be expressed in this framework. For example, if $\Omega = \{\mathbf{q}_i\}$ is the one-ring neighbourhood of mesh vertex $\mathbf{p}$ and $\delta_p = \sum w_i(\mathbf{p} - \mathbf{q}_i)$ is the Laplacian coordinate, then the Laplacian deformation framework [Botsch and Sorkine 2008] defines $\mathbf{p}$ as

$$\widehat{\mathbf{p}} = \sum w_i \widehat{\mathbf{q}}_i + T_p \delta_p = \sum w_i (\widehat{\mathbf{q}}_i + T_p(\mathbf{p} - \mathbf{q}_i)) \tag{5.3}$$

where w_i can be any of a wide range of weights, the most popular being the *cotangent* weights [Desbrun et al. 2002]. Equation 5.3 is simply Equation 5.2 integrated over a finite domain, with $\mathbf{v}(\mathbf{p}, \mathbf{q}) = \mathbf{p} - \mathbf{q}$ and $\mathcal{F}_{\widehat{\mathbf{q}}_i} = T_p$, a constant rotation which must somehow be estimated. Of course, since most $\mathbf{q}_i$ are unknown, a variational approach is necessary to solve for all vertices simultaneously. To support direct evaluation, Ω must be known a priori. For example, a Bezier patch is a deformation of a plane, with Ω the given control points, $\mathbf{v} = 0$, and w the Bernstein polynomials.

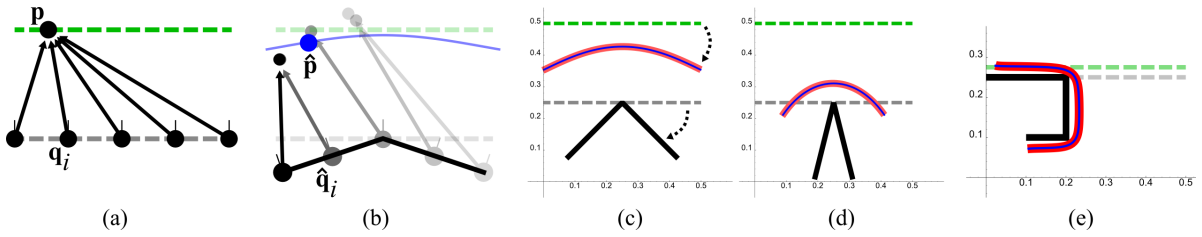


Figure 5.1: *In (a), a point* $\mathbf{p}$ *is represented by a set of displacement vectors relative to points* $\mathbf{q}_i \in \Omega$*. When* Ω *is deformed (b), the new point* $\widehat{\mathbf{p}}$ *is defined by a weighted sum of transformed displacement vectors. Examples c-e compare closed-form integral solutions (thin green lines) with discrete solutions (thick red lines) based on regular samplings of* Ω *(black line).*

Curves are widely recognized as an efficient and intuitive control handle for deformation [Singh and Fiume 1998]. To deform $\mathbf{p}$ relative to a curve $\mathcal{C}$, we set $\Omega = \mathcal{C}$ and use an inverse-distance weight:

$$w(\mathbf{p}, \mathbf{q}) = \frac{1}{d(\mathbf{p}, \mathbf{q})^k + \epsilon} \tag{5.4}$$

where d is either a Euclidean or geodesic distance function. If $\mathcal{C}$ is composed of linear segments, a closed form solution to Equation 5.2 can be computed for $k = 2$, but in general an analytic solution will not be available, so we must approximate $\mathcal{C}$ with a discrete sampling $\Omega = \{\mathbf{q}_i\}$. Equation 5.2 is then rewritten as

$$\widehat{\mathbf{p}} = \sum_{\Omega} \frac{w(\mathbf{p}, \mathbf{q}_i)}{\sum_{\Omega} w(\mathbf{p}, \mathbf{q}_i)} (\widehat{\mathbf{q}}_i + \mathcal{F}_{\widehat{\mathbf{q}}_i} \mathbf{v}(\mathbf{p}, \mathbf{q}_i)) \tag{5.5}$$

which generalizes the WIRES deformer [Singh and Fiume 1998] and alternative recent formulations [Kanai et al. 1999; Milliron et al. 2002; Yu et al. 2004]. Equation 5.5 can also

be cast as a Monte-Carlo solution to Equation 5.2, which aids in interpreting the effects of different sampling and weighting strategies. For example, random sampling exhibits the expected reduction in variance, while regular sampling provides a smooth approximation (Figure 5.1). Generally we are concerned with fixed point-sampled geometry, and and hence must counteract sampling bias by modulating our weighting scheme. For example, to correct for non-uniform sampling of $\mathcal{C}$, we scale $w(p, \mathbf{q}_i)$ by $\sum_{q_j \in N(q_i)} |\mathbf{q}_i - \mathbf{q}_j|$, where where $N(\mathbf{q}_i)$ is the connected neighbourhood of $\mathbf{q}_i$.

We can apply Equation 5.5 to deform a sampled surface $\mathcal{S} = \{\mathbf{p}_i\}$ based on a 3D deformation of the open boundary curve $\partial\mathcal{S}$. Figure 5.2b demonstrate the limitation of this approach, namely that points distant from $\partial\mathcal{S}$ undergo significant distortion. This is easily understood by re-factoring Equation 5.5 into the sum of a weighted centroid and an average displacement vector. For points near $\partial\mathcal{S}$, weight is concentrated near the closest point on the boundary, and hence the centroid is relatively static. For samples further from the boundary, however, weight is distributed more evenly over $\partial\mathcal{S}$, pulling the centroid downwards and causing vertical squashing.

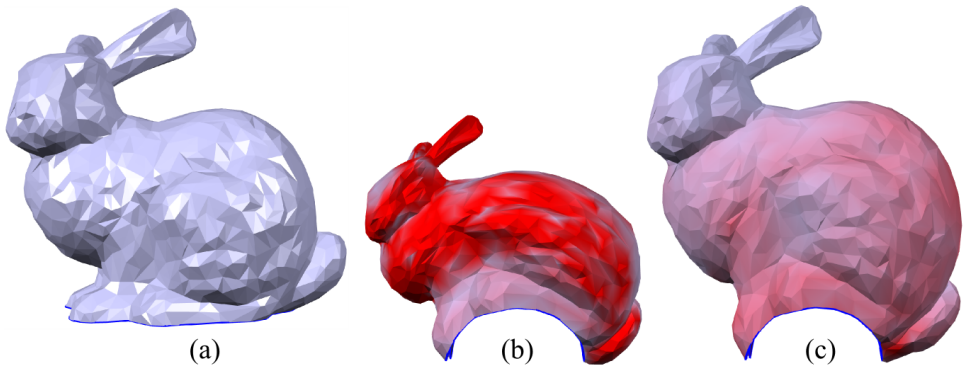


Figure 5.2: *Embedding the boundary loop at the base of the bunny mesh (a) in a cylinder results in shrinkage using boundary-relative deformation (b). The COILS deformation smoothly distributes edge distortion (mapped to red) away from the boundary (c).*

5.3 COILS: A Geometric Differential Deformation

As is evident in Figure 5.2, the WIRES-like deformer preserved shape only in regions near the boundary curve. The issue with that approach was that as $\mathbf{p}$ gets further from the handle, the distance-based weights become more uniform, increasing the amount of "pull" towards the center-of-mass of the boundary curve. This observation suggests that we can better preserve the global shape by deforming each point relative to some spatially-adjacent surface region $\Omega(\mathbf{p})$, such that the weighted centroid remains close to $\mathbf{p}$.

Remember that the deformation must still be driven by the boundary curve. If we are to take a geometric approach, then intuitively the surface must "grow" out from the boundary, as that is the only information that is initially available. Hence, conceptually my approach is to slice the surface into thin layers propagating away from the boundary,

and reconstruct each layer relative to the last. If we consider a front propagating along the surface away from the open boundary, then each timestep defines a layer which can be deformed relative to some *upwind* region(Figure 5.3), supporting a forward evaluation from the deformed boundary. Since at each layer we are effectively applying the WIRES-like deformer, I will call the resulting deformation a *COIL*.

An explicit layer segmentation would be cumbersome on arbitrary point sets, so I will use the weight function $w(\mathbf{p}, \mathbf{q})$ to implicitly represent the front propagating away from Ω. The first step is to determine an *arrival time* at each vertex on the mesh. Assuming again that $\mathcal{S}$ will be deformed relative to its open boundary $\partial\mathcal{S}$, arrival time can be defined as the geodesic distance $g_{\mathbf{p}_i}$ from $\partial\mathcal{S}$ to $\mathbf{p}_i$, approximated using Dijkstra's algorithm (Figure 5.3).

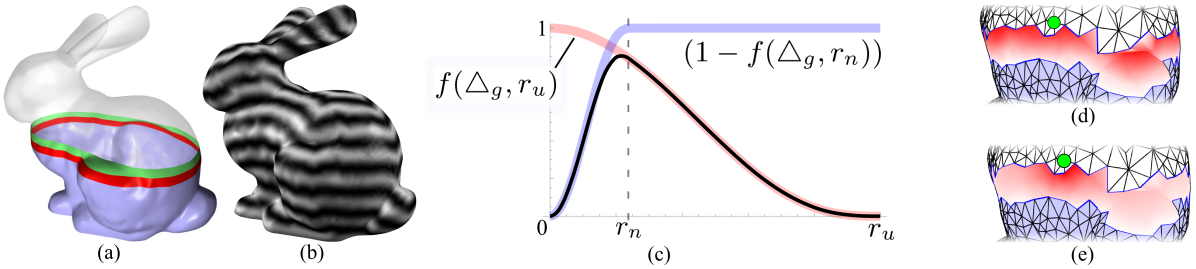


Figure 5.3: *Conceptually, the bunny is sliced into layers defined by a front propagating across the surface, away from the open base (a). The layer on the front (green) is defined by the previous upwind layer (red). In practice the approximate geodesic distance from the boundary (b) and a weight function (c) are combined to implicitly define a "smeared-out" front at the green point (d). The product of this function on the mesh is then combined with the inverse distance weight to define a combined weight function for the green point (e).*

One an arrival time is known at each vertex, we must select a set of upwind points to represent the current front. To provide maximum rigidity the upwind region should form a closed loop around the surface (see Section 5.7). Given an arrival time radius r_u, taken to be a small multiple (usually 2.1) of the average edge length, the front at $\mathbf{p}_i$ is approximated by the upwind band $\Omega(\mathbf{p}_i) = \left\{\mathbf{p}_j : g_{\mathbf{p}_i} - r_u < g_{\mathbf{p}_j} < g_{\mathbf{p}_i}\right\}$. Because we are operating on a discrete set of points, sampling variation will result in a discrete front that changes from vertex to vertex. The effect of this 'popping' is clearly visible in the deformation. To mitigate this effect we can modulate the weights using a smooth falloff field. This smoothing is effectively an adaptation of the "smeared-out" Heaviside functions used in level set front propagation [Osher and Fedkiw 2003].

Given a function which smoothly falls off from 1 to 0, such as $f(x, r) = \max((1 - x^2/r^2)^3, 0)$, we define the upwind weight:

$$\triangle_g = g_{\mathbf{p}_i} - g_{\mathbf{p}_j} \qquad r_n = \min_{i \neq j} |\mathbf{p}_i - \mathbf{p}_j| \tag{5.6}$$

$$w_{arr}(\mathbf{p}_i, \mathbf{p}_j) = f(\triangle_g, r_u)\,(1 - f(\triangle_g, r_n)) \tag{5.7}$$

where the first term falls off away from the front and the second reduces the weight on points whose arrival time is nearly the same as at $\mathbf{p}_i$ (Figure 5.3c,d). This second term

is necessary because $g_{\mathbf{p}}$ may vary slightly between points which ideally would have the same arrival time. This can cause biased sampling of the upwind region, which leads to asymmetric deformation (Figure 5.4a-d).

As in all sampling solutions to integral problems, we must address the issue of sampling irregularity. In the weighting formulation we have constructed so far, the deformation will be pulled towards regions of relatively higher sample density (Figure 5.4). To counteract this, the weight can be modulated with a regularization factor which effectively represents the surface area represented by $\mathbf{p}$:

$$w^{\triangle}_{reg}(\mathbf{p}_j) = \sum_{T_k \in N(\mathbf{p}_j)} \text{Area}(T_k) \qquad w^{\bullet}_{reg}(\mathbf{p}_j) = \min_{\mathbf{p}_k \in N(\mathbf{p}_j)} |\mathbf{p}_k - \mathbf{p}_j|^2 \tag{5.8}$$

The one-ring area weight $w^{\triangle}_{reg}$ works well on triangle meshes (Figure 5.4e-h), and $w^{\bullet}_{reg}$ produces reasonable results when topology is unavailable. Hence, our final weighting scheme is

$$w_{up}(\mathbf{p}_i, \mathbf{p}_j) = w_{arr}(\mathbf{p}_i, \mathbf{p}_j) w_{reg}(\mathbf{p}_j) w(\mathbf{p}_i, \mathbf{p}_j) \tag{5.9}$$

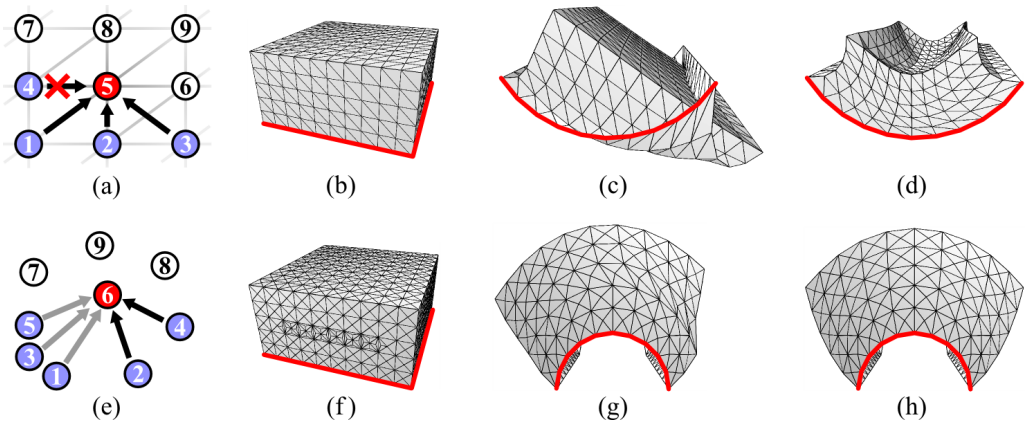


Figure 5.4: *In (a), point 4 is considered upwind to point 5 due to slight numerical differences common on regular meshes (b), leading to highly biased deformation (c) that is mitigated by the ramp-up term in w_{arr} (d). Irregular vertex density (e,f) cause similar problems (g), largely corrected by our regularization weight w_{reg} (h).*

Note that $\mathbf{p}_i$ is now deformed relative to other internal points, but transformed frames are only provided at boundary samples. If we restrict $\Omega(\mathbf{p}_i)$ to neighbours $\mathbf{p}_j$ whose full one-ring neighbourhood is upwind, we will be able to estimate the necessary tangent frames from the current mesh. However, I found that when taking this approach small estimation errors tend to accumulate, resulting in catastrophic distortions. Instead I normalize a linear blend of upwind-relative frames, $\mathcal{F}_{\widehat{\mathbf{p}}} = \sum_{\mathbf{q}_i \in \Omega(\mathbf{p})} w_i \mathcal{F}_{\widehat{\mathbf{q}}_i} \mathcal{F}^{-1}_{\mathbf{q}_i} \mathcal{F}_{\mathbf{p}}$. This approach is known to be sub-optimal, although an evaluation of different rotation composition techniques by Gramkow [Gramkow 2001] has shown that for small rotations, the error introduced by this re-normalized linear blending is in fact very minor. I experimented with quaternion-based rotation blending and logexp matrix blending [Alexa 2002], and found that qualitatively the results were only slightly "stiffer" than the much faster linear blend, so in my implementation I use the latter.

On a related note, I found that when approximating geodesic distances using Dijkstra's algorithm, better results were achieved when the edge graph was determined using Euclidean ϵ-balls, even if mesh connectivity is available. A graph based on one-ring connectivity works well on regular meshes, but will generate geodesically out-of-order arrival times on the semi-regular tilings in Figure 5.4. Likewise, initializing boundary neighbours with exact distances to the loop polyline will avoid many pathological cases introduced by irregular mesh topology.

5.4 Parameters and COILS

Since the COILS deformation is based on a geometric forward evaluation, the functions $w(\mathbf{p}, \mathbf{q})$ and $\mathbf{v}(\mathbf{p}, \mathbf{q})$ can be arbitrarily parameterized with procedural or hand-tuned factors (such as painted weights), providing artists with extensive real-time control. Figure 5.5 demonstrates how parameterized deformation can be used in a variety of modeling tasks.

One useful parameter is a rigidity factor which controls how strongly the shape of the feature is preserved. There is an inverse relationship between desired rigidity and the power k in the inverse-distance weight (Equation 5.4). A uniform rigidity multiplier provides a simple control over how the boundary deformation propagates out into the global shape (Figure 5.5a-c). Varying rigidity based on the distance to the boundary enhances feature preservation (Figure 5.5d), while an initialization based on Gaussian curvature allows stretching to be absorbed by high-curvature regions (Figure 5.5m-p).

Scaling is supported by adding a multiplier to the offset vectors $\mathbf{v}$, which can also be varied based on boundary distance, or any other factor (Figure 5.5j-l). We can also tune how the shape is pulled towards or pushed away from the boundary, which roughly translates into manipulation of the shape volume, by scaling r_u, the upwind falloff radius from Equation 5.7. Larger r_u values mean that weight is distributed more uniformly over the upwind region, causing pulling towards the boundary as in Figure 5.2b (although much less extreme).

Transformation of boundary tangent-normal frames directly affects the local shape. We include a *bias frame* $\mathcal{F}_B$, and a parameter α that controls interpolation between the original frame and $\mathcal{F}_B$ (Figure 5.5e-i,q-s). Boundary rotation is also useful for controlling continuity when using the COILS deformer to attach a Part to another surface, as we will see in later sections. A useful extension would be to allow direct manipulation of some boundary frames, followed by smooth blending of the manipulated curves along the frame, perhaps via a *peeling* interface.

5.5 Multiresolution and Hierarchical Extensions

While the complexity of the COILS deformer is less than $O(N^2)$, it is significantly higher than linear. Experimentally, the size of the upwind ring is roughly $O(\log^2 N)$, so the total cost is $O(N \log^2 N)$. Interactivity is greatly enhanced by pre-computing offset vectors v and constant terms in w_{up} at each point, and caching normalized weights as they

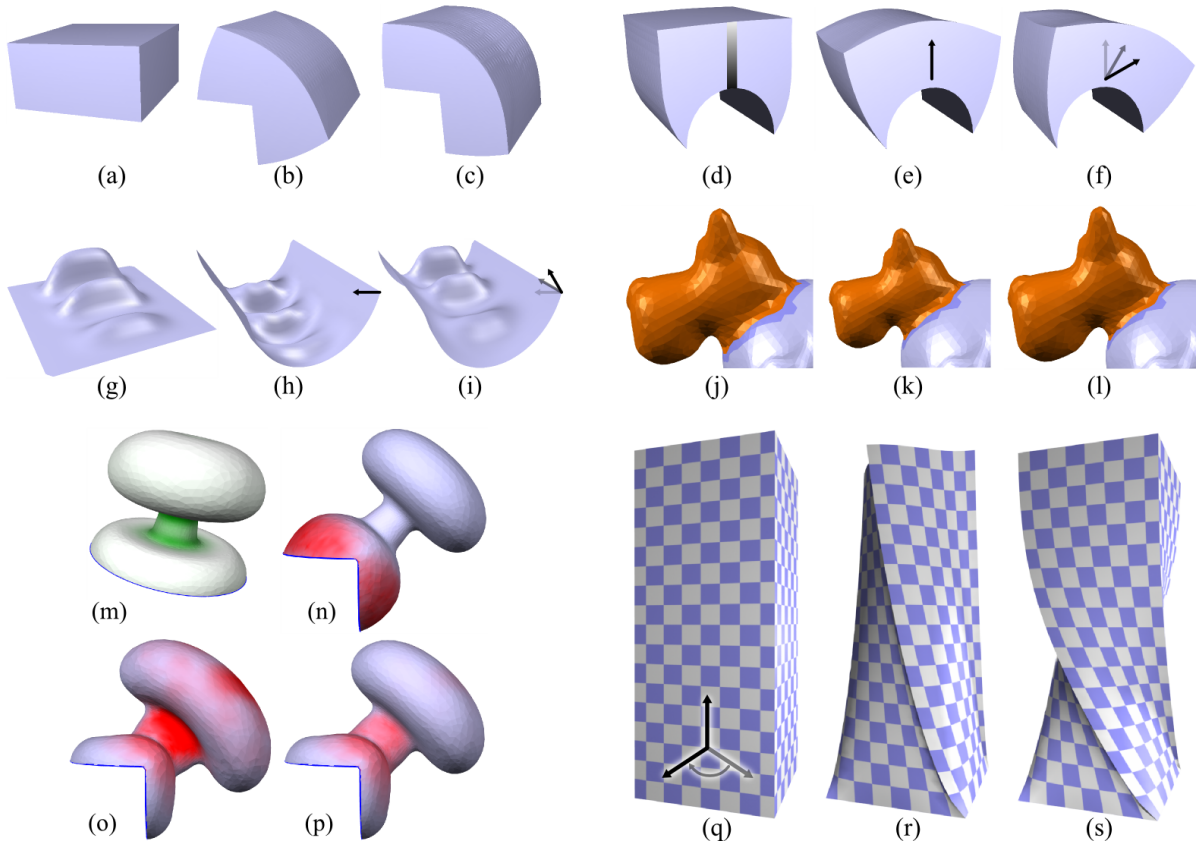


Figure 5.5: *Increasing the rigidity power k on the inverse-distance weight from* 2 *(b) to* 3 *(c) reduces the stiffness of a box bent across a* 90° *corner. Varying rigidity based on the vertical height produces a flexible base with a stiff top (d), while applying a constant rotation to the boundary tangent-normal frames can be used to create asymmetric shapes (e,f). Interior bumps on a plane (g) squash together after an inverted bend (h), but do not foldover, and rotating boundary normals in the vertical direction lifts them back up (i). In a later example we will need to fit three dog heads onto a surface, but the neck will be too large (j). Uniform scaling (k) followed by tuning of a blend between scaling factors based on boundary distance (l) allows us to squeeze on all three. The base of a mushroom shape (m) becomes distorted when bent (n), but increasing k from* 2 *to* 3 *flattens it out (o), and varying k based on a curvature map (green in m) preserves the cap as well (p). In another boundary frame manipulation, varying k with height controls the propagation of a* 90° *boundary twist applied to a tall box (q-s).*

are generated. This requires $O(N \log^2 N)$ storage, further limiting scalability. Hence, I propose the following multiresolution scheme.

The first step is to generate a smooth, low-resolution base surface $\mathcal{B}_\mathcal{S}$ by simplifying $\mathcal{S}$. We iterate rounds of edge-collapses and non-shrinking Laplacian smoothing [Taubin 1995], without any detail-preserving metric. Then for each $\mathbf{p} \in \mathcal{S}$ the nearest point $\mathbf{q} \in \mathcal{B}_\mathcal{S}$ is found, $\Omega(\mathbf{p})$ defined as the k nearest connected base samples to $\mathbf{q}$, and $\mathbf{p}$ deformed using Equation 5.5. Essentially, $\mathcal{S}$ is an offset surface from $\mathcal{B}_\mathcal{S}$, where the displacement is defined by (approximately) integrating rotation-invariant offsets over a local region of $\mathcal{B}_\mathcal{S}$.

The above represents a detail surface relative to a lower-resolution base surface. This approach can be seen as a generalization of the way in which multiresolution surfaces represent the difference between two different resolution levels. There, a single *detail* vector is stored on the base mesh for each high-resolution vertex [Zorin et al. 1997; Kobbelt et al. 1998]. By (approximately) integrating over a local region, we improve shape rigidity and reduce the likelihood of foldovers, while avoiding the computational expense of more complex techniques such as displacement volumes [Botsch and Kobbelt 2003].

Some multiresolution examples are shown in Figure 5.6. Note that this approach can be extended to multiple levels of detail, supporting multiresolution on arbitrary meshes. However in the application described in Section 5.9 I only use one detail layer, automatically applied whenever $N > 2000$.

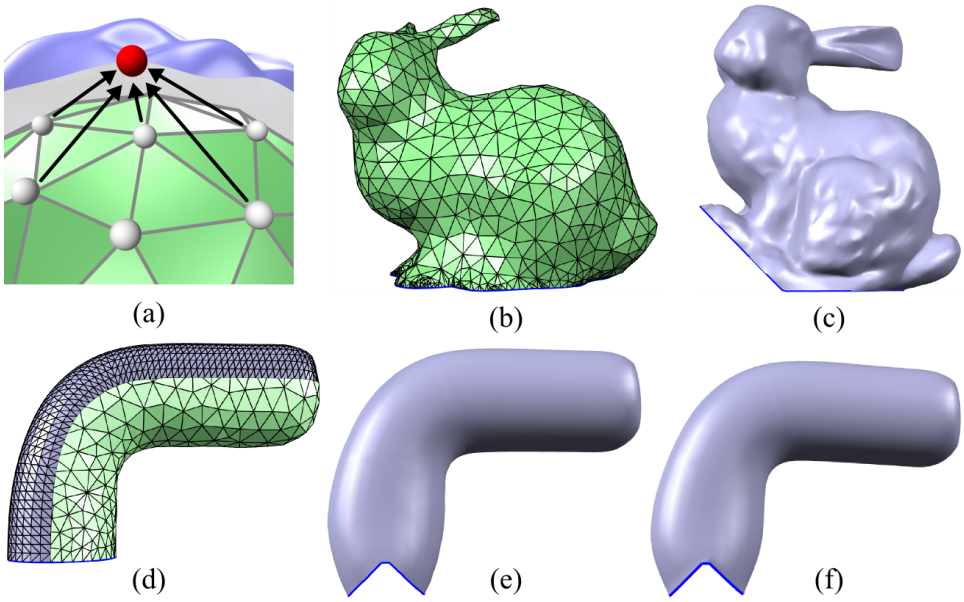


Figure 5.6: *By averaging multiple base-surface displacement vectors (a), an 11k bunny mesh can be reduced to 1k vertices without a detail-preserving simplification scheme (b), and the resulting deformation (c) is still of high quality. The deformation induced by bending the base of a simplified tube mesh (d) remains smooth (e) and visually similar to the full solution (f).*

Another issue with the COILS deformer, which I will discuss further in the next section, is that the deformation is influenced by the shape of the geodesic iso-contours (the "slices") growing inwards from the boundary. This can be problematic in some cases, such as in Figure 5.7, where the shape of the iso-contours causes the protruding features to pull together. This situation could possibly be avoided by curvature-dependent front propagation speed. However, since the COILS deformer can handle non-manifold interiors, a more flexible approach is to segment the feature into discrete components and reconstruct them in sequence, from the boundary inward.

In addition to reducing dependencies on the outermost boundary shape, this hierarchical approach offers higher internal rigidity, more control to the user, and opportunities for parallelization. While Part determination could be driven by automatic segmentation algorithms, some user guidance will likely be desirable. While this hierarchical deformation does have significant practical benefits, I have not explored this direction in any great detail, and it has not been utilized to create any of the other results in this section.

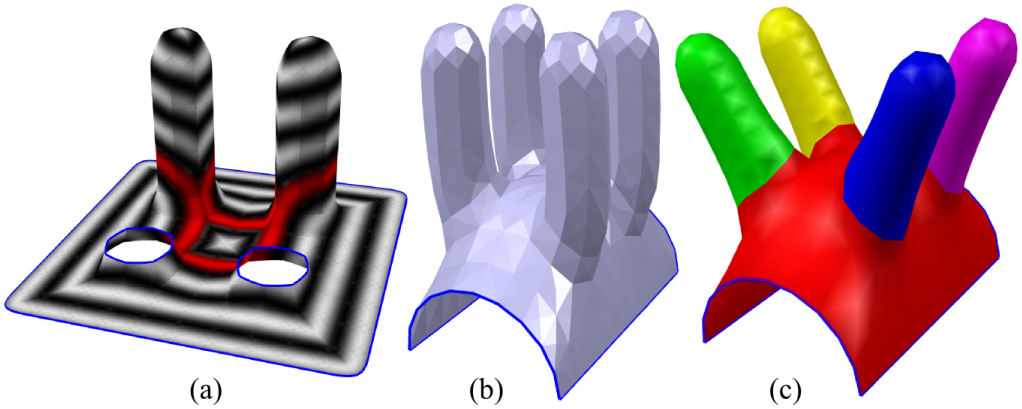


Figure 5.7: *The shape of the upwind region is influenced by the boundary shape. In (a) the iso-contours of the geodesic distance field are connected part-way up the bumps, causing them to stick together as the base deforms (b). Segmentation followed by hierarchical deformation produces a more intuitive result (c).*

5.6 Limitations

As noted above, one limitation of the COILS deformer is that the geodesic distance field is neither smooth nor continuous. This causes a variety of problems. The most common problem this introduces is non-smoothness in the deformation. Figure 5.8a,b shows such an example, where the irregularity of the boundary is clearly reflected in the geodesic distance field, and hence in the deformation. Another issue visible in this figure is that the geodesic isocontours again inadvertently connect semantically-disjoint internal shape features, leading to a somewhat unintuitive deformed shape. In general, there is no guarantee that the geodesic isocontours will be aligned with any internal Part boundaries - instead they are largely determined by the shape of the outer boundary loop.

Another problem occurs if the interior surface contains holes, of both the topological and manifold variety. In these cases the geodesic distance will branch when the feature is first encountered, and then later merge together. On either side of this boundary, the upwind neighbourhoods are completely disjoint, so the surface can *tear* at this boundary (Figure 5.8c,d). One way to reduce this effect is to encode each point relative to *all* upwind fronts at the given arrival time, even those which are disjoint. This resolves tearing problems (Figure 5.8e) but can also lead to connections between semantically-disjoint Parts, where a significant deformation of the Part is in fact desired[1].

One way to resolve most of these problems would be to generate a smoother field from which to derive arrival times. Curvature-dependent Level-set *viscosity solutions* are often used to smooth out discontinuities in the front propagation literature, and could possibly be adapted here. Another possibility would be to attempt to incorporate geometric smoothing. An interesting way to approach this is to note that, similar to Rotation-Invariant Coordinates [Lipman et al. 2005], the COILS reconstruction of tangent-normal frmaes does not depend on the positions. Hence, the frame reconstruction can be pre-computed and then post-processed to reduce discontinuity, for example via local smoothing. As we saw in the normal-smoothed DEM (Section 4.3.3), even simple neighbourhood averaging can have a significant impact. However, neither of these approaches will resolve the issues with topological branching. A more robust alternative may be to compute a variational harmonic interpolant [Xu et al. 2009a] using the geodesic gradients as a guidance field, or perhaps a MLS-based scalar field [Jin et al. 2009] with geodesic-distance constraints.

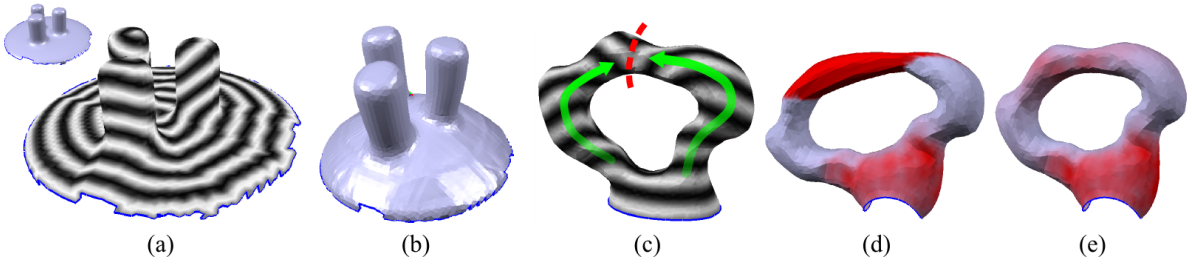


Figure 5.8: *An irregular boundary leads to high-frequency variation in the geodesic distance field (a), which is reflected as non-smoothness in the deformed surface (b). Topogical branching and merging (c) can lead to tearing along the merge boundaries (d). These tears can be resolved by encoding relative to all disjoint upwind neighbourhoods (e).*

5.7 Other Upwind Deformation Domains

The COILS deformer involves a relatively complex choice of upwind neighbourhood. A reasonable question is whether there are simpler alternatives that might be more effective.

[1]In my experience, this rarely occurs in Partbased modeling scenarios. Generally rigid Part shapes seem to be much more preferable, and when extreme deformation is desired, the artist would rather apply it as a controllable post-process than to have it determined by the boundary deformation

In addition to the WIRES-style boundary-relative deformation, I have explored three other alternatives, which are compared visually in Figure 5.9.

Local Upwind Neighbourhood An obvious choice would be to simply encode **p** relative to the upwind members of its mesh one-ring. This approach leads to extreme discontinuities, but is is essentially an approximation to the upwind portion of a geodesic disc. Sampling a larger disc improves smoothness, but the boundary deformation is clearly represented in the deformed Part (Figure 5.9b). Essentially, each point can only "see" a portion of the boundary loop, and so is primarily determined by that region of the boundary, with the overlap between neighbourhoods serving mainly to smooth the result.

All-Upwind Surface Another simple strategy is to encode **p** relative to the entire upwind surface. If we consider the centroid/displacement-vector factorization, then we see that the centroid is now the center-of-mass of the currently reconstructed surface. Hence, as in the boundary-relative deformation, some squashing of the shape is apparent. Otherwise the All-Upwind method is very rigid, even more so than COILS, but a practical drawback is that it is $O(N^2)$.

Random Sampling of All-Upwind If the All-Upwind scheme is meant to approximate an integration over the upwind surface, then we can always approximate the approximation by using fewer samples. In the terms of Monte-Carlo integration, this should increase the variance or 'noise' in the approximation. This is exactly what we observe - in Figure 5.9d the deformed shape is visually similar to Figure 5.9c at larger scales, however there are higher-frequency variations in the shape.

5.8 Comparison with Variational Techniques

The underlying principle behind many recent works in variational shape modeling is that the initial surface-with-boundary is converted into a differential representation, with fixed boundary constraints. My COILS deformer takes much the same approach. However, the difference with variational methods is that they then define the surface as the global minimum of some *energy function* which describes how the variational information and boundary constraints should be integrated to recover the 3D surface.

The main complexity of variational deformation methods is that finding the global minimum of an energy function requires complex numerical optimization techniques, which are often very computationally intensive. Simple quadratic least-square energies lead to linear solutions, which can then be used in interactive scenarios by utilizing pre-computation techniques such as factorizing fixed system matrices [Botsch and Sorkine 2008]. These linearized deformations can have difficulty handling large rotations, so more robust non-linear techniques have been developed [Botsch et al. 2007; Lipman et al. 2007], however they are significantly more expensive.

One drawback of energy-minimization approaches in general is that, compared to geometric methods, they are much more difficult to *parameterize*, as any additional pa-

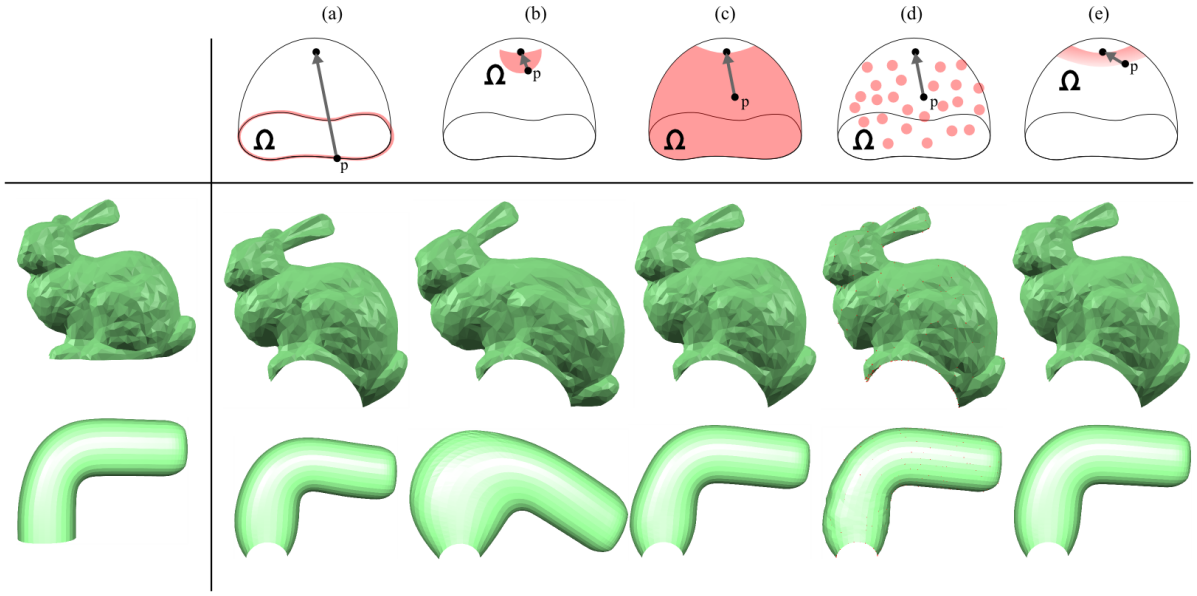


Figure 5.9: *A point* **p** *can be reconstructed relative to a variety of upwind deformation domains* Ω*, including (a) the surface boundary, (b) a local upwind neighbourhood, (c) the entire upwind surface, (d) random sampling of the upwind surface, and (d) a thin upwind band (the approach taken in my COILS deformer).*

rameters must be expressed as changes to the energy function. It is of course desirable that any algorithm produce plausible results with the default parameter settings, and with the COILS deformer we have found that this is generally the case. However, Zimmermann et al. [Zimmermann et al. 2008] found that for the same handle configuration, subjects in a study disagreed on which of two deformations was expected. This suggests that there is probably no 'best' energy function or set of parameters for a particular deformation scenario; artists will inevitably want to tweak the results.

In general, parameterizing an energy function is non-trivial. In production environments, complex procedural controls are often authored using visual programming tools. Such parameters may not even be differentiable, so integrating them into an energy minimization means resorting to expensive and non-smooth gradient-free methods. Even if the parameters can be linearized, they are problematic for linear variational techniques because these methods achieve interactivity via pre-computation techniques such as factorizing fixed system matrices. Tuning a parameterized energy function would generally require expensive pre-computations to be repeated, which would be very detrimental to interactivity. It should also be noted that even determining *how* to change an energy function to provide a specific artistic control is more difficult than in geometric approaches. For an example of integrating material stiffness controls into a variational deformation, see Popa et al. [Popa et al. 2006].

Most variational techniques also require relatively "clean" mesh topology, and could not handle the non-manifold polygon soups that are common in many practical applications. A notable exception is Sumner et al. [Sumner et al. 2007], who bridge the gap between surface and spatial deformation by constructing a spatial 3D graph from an

arbitrary point set, deforming the graph based on a nonlinear optimization, and then generating a spatial deformation by blending transformations of the graph nodes. Because COILS is geometric and based only on differences between points, it can be implemented to be independent of mesh topology without requiring additional analysis of the point set, and hence even applied to surfaces made of disjoint components.

A final assumption of most variational techniques is that the domain of the deformation - the mesh graph - has no interior holes. This is because variational approaches are essentially interpolations of the boundary constraints across the interior, modulated by the differential information. So, for example, the boundaries of the eye cut-outs common in head meshes must also be fixed in 3D[2]. This is not possible in our Part shape representation - until the shape has been reconstructed, we have no way of pre-determining any 3D boundaries except those lying in $\mathcal{U}$. As we will see, the COILS deformer handles interior holes much more effectively.

5.8.1 Comparison with Rotation-Invariant Coordinates

A variety of linear variational mesh representations have been developed, see Sorkine & Botsch [Botsch and Sorkine 2008] for a recent survey. One popular approach is based on the *vertex Laplacian*, defined in Equation5.3. One issue with the Laplacian vector, though, is that it is defined in world coordinates and hence must be rotated using a tertiary algorithm. Lipman et al. [Lipman et al. 2005] provided a suitable rotation technique, based on encoding the tangent-normal frame at each vertex relative to those of its neighbours. This approach, known as *Rotation-Invariant Coordinates* or RIC, encodes the surface such that it can be reconstructed from a set of boundary positions and normals, exactly as in the COILS deformer. Although I will not go into details here, RIC involves two linear solves. First the boundary normals are used to solve for an orientation field that smoothly varies over the surface. Then the differential Laplacian vectors are rotated using this field, and the Laplacian reconstruction of Sorkine et al. [Sorkine et al. 2004] is applied to find the deformed shape.

I implemented RIC and compared it with COILS in the context of the Part-based geometry drag-and-drop tool I will discuss in Section 5.9. In this tool, the Part shape is deformed to fit any new Part region, and in addition the boundary frames are often rotated either in towards or out away from the surface, to preserve continuity across the Part boundary. In cases where the boundary is significantly deformed from its original configuration, or the boundary rotation is significant, I have observed that RIC can break down, producing results which are much less desirable than the COILS deformer (Figure 5.10a).

In these case the new mesh normals do not deviate significantly from the desired RIC-estimated normals, but rather the mesh is simply too deformed. This is a difficult situation to handle, but I did find that one option was to manipulate the boundary orientation constraints in the following way. Consider the axis formed by the line from

[2]Rather than fixing internal boundaries in space, it may be possible to constrain the edge lengths around the hole boundary. This still does not guarantee the sort of rigidity we would desire, but it may at least avoid catastrophic failures

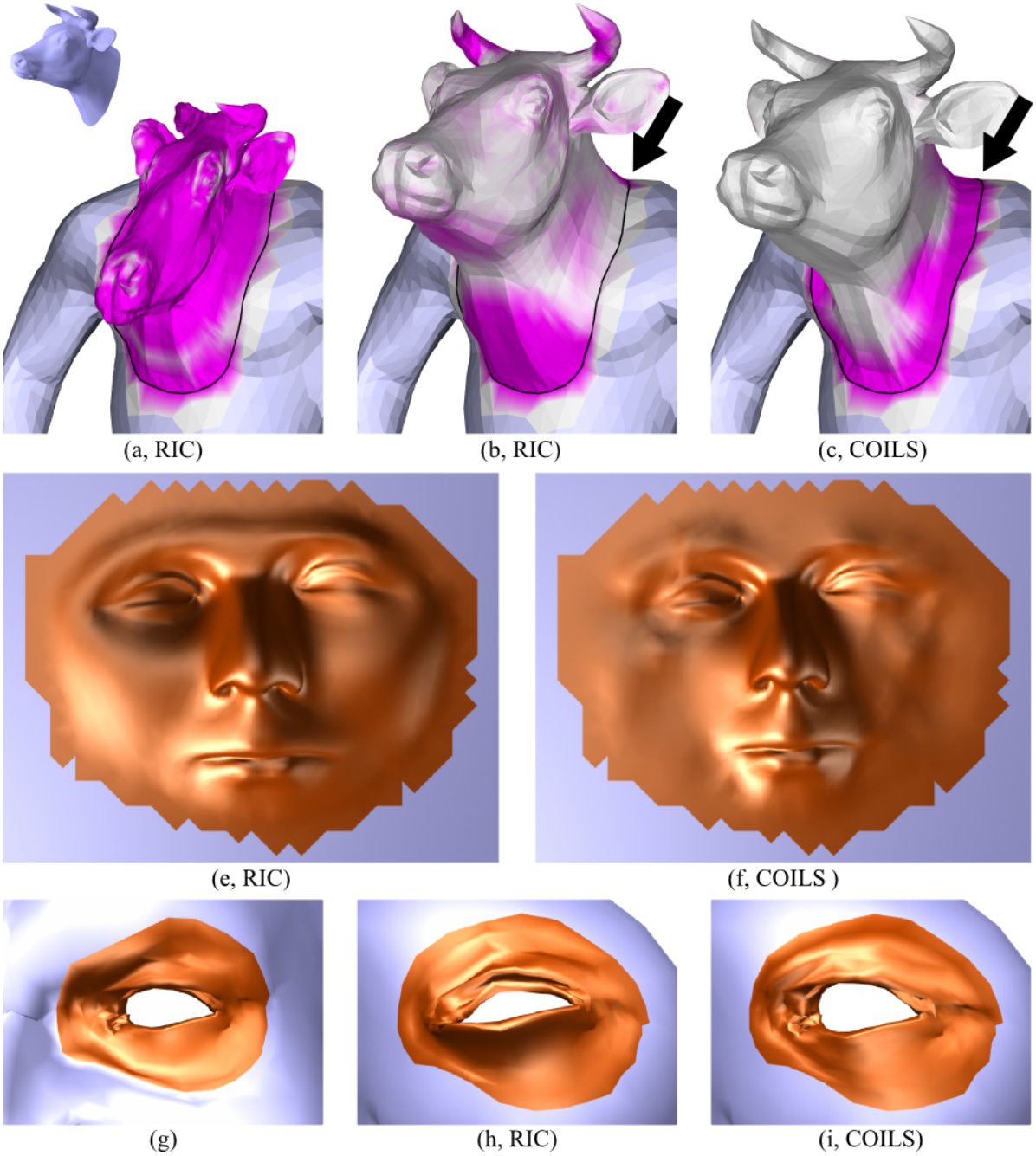


Figure 5.10: *Dropping a cow* Part *onto a torso while maintaining boundary continuity involves too severe of a bend for rotation-invariant coordinates (a). Optimizing boundary frames to minimize distortion (purple) reduces boundary continuity (black arrow) but preserves interior shape (b). The COILS deformer concentrates distortion near the boundary, allowing for continuity with more interior rigidity than RIC (c). The least-squares optimality properties of RIC allow it to outperform the non-smooth COILS deformer on high-resolution surfaces (e,f), while the COILS deformer is more robust at preserving the shape of interior holes (g-i).*

vertex $\mathbf{p}_{i+1}$ to $\mathbf{p}_{i-1}$. We can rotate the constraint frame at boundary point $\mathbf{p}_i$, and efficiently find a new solution via back-substitution. The optimal global rotation angle θ is found by minimizing the total area deformation of the mesh triangles:

$$\arg\min_{\theta} \sum_i \left[\left(A_i^{\text{orig}} / A_i^{\text{def}(\theta)} - 1 \right)^2 A_i^{\text{orig}} \right] \tag{5.10}$$

where A_i is the area of triangle T_i in the original or deformed configuration. Note that a more refined solution can be found by optimizing for per-vertex angles, but this approach is too expensive for interactive use. The net result of this boundary angle optimization is a significant improvement in the deformed shape, but at the cost of a smooth transition at the shape boundary (Figure 5.10b). Note that the COILS deformer has no problem maintaining this boundary continuity while also actually appearing to provide a higher level of internal shape rigidity (Figure 5.10c).

One significant benefit of the RIC deformer is that it produces smooth deformations. This is particularly evident on large, high-resolution smooth surface regions, as in Figure 5.10d,e. As we have seen, the COILS deformer lacks this smoothness. Similarly, because it globally solves for a smooth orientation field, RIC transparently handles the internal topological holes and handles that introduce tears into the geodesic distance field used in COILS. As I discussed in Section 5.6, this is a major problem for COILS, although it can be addressed at the cost of coupling the deformation of geodesically-disconnected regions.

Another difference I noted between geometric and variational methods is their handling of internal boundaries. Figure 5.10f shows a standard case - an eye cutout on a face mesh. In the RIC deformation the interior hole can be arbitrarily distorted, while the COILS deformer maintains rigidity (Figure 5.10g,h). The issue here is that the local weights used in RIC (and most other variational techniques), which are meant to approximate the mean-curvature normal, are mathematically invalid at boundary vertices [Schneider and Kobbelt 2001]. In variational deformations these internal boundaries would generally be fixed, but when using RIC to reconstruct a Part at an arbitrary location, the location of internal boundaries is unknown. One possibility would be to introduce very stiff edge-length constraints on the internal boundary edges, however it is unclear that this would sufficiently constrain the solution.

5.9 A Geometry Drag-And-Drop Tool

Re-use of existing 3D modeling assets is a challenging problem in freeform shape design, in large part because existing 3D surface models lack any sort of Part decomposition. An interesting question is whether or not we can impose Part structure on these existing models a priori. If the Parts of a surface could be reverse-engineered, they could easily be edited and re-combined in novel ways.

In this section I will explore this problem by developing a Part-based modeling interface for geometric drag-and-drop. This tool allows an artist to manually separate a region of surface from an existing model and re-use it as a procedural Part. To accomplish this, we will first need to *infer* the portion of surface that could have been

"underneath" the Part. Once the underlying surface is known, the selected patch can be decomposed into a Part region and shape. This Part can then be interactively repositioned on any other surface region, much like the texture Parts of Section 4.6. To the artist, this simple interface allows Parts of unstructured mesh surfaces to be dragged-and-dropped in the same way as one would with a 2D pixel editing tool like Photoshop [Adobe Systems Inc. 2007b].

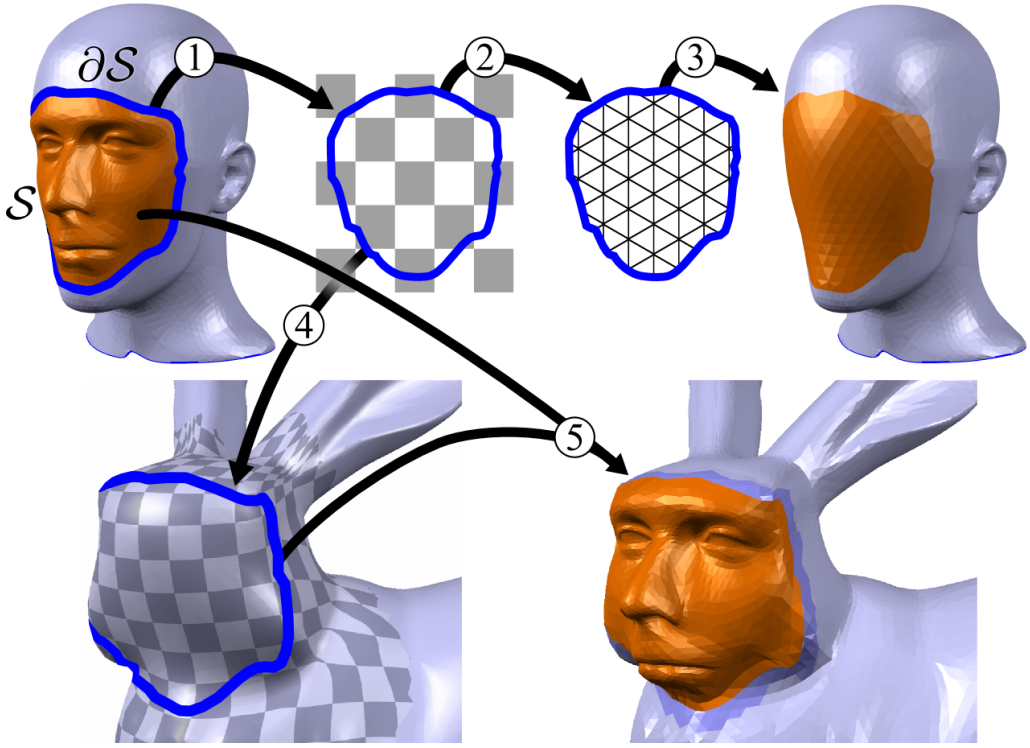


Figure 5.11: *To drag the mannequin face to a (flattened) bunny, a region-of-interest is first selected. The boundary loop is embedded in the plane (1), then a planar mesh is generated (2) and deformed to smoothly fill the hole (3). The drag is completed by mapping the boundary loop to the target region via a local parameterization (4), and then the ROI is deformed relative to the new boundary (5).*

Figure 5.11 provides an illustrative overview of my approach. The artist first demarcates the Part surface by enclosing it within a boundary loop, created by sketching sequential curve segments on the surface. The Part surface $\mathcal{V}$ is then removed from the mesh, leaving a hole with boundary $\partial\mathcal{V}$. The boundary $\partial\mathcal{V}$ is now embedded in the plane and triangulated, creating a planar mesh which is then deformed to precisely fit $\partial\mathcal{V}$, thus spanning the hole. Note that this procedure is independent of the interior topology of $\mathcal{S}$. Next this fill surface is parameterized using the DEM and $\partial\mathcal{V}$ is embedded in it, defining the 2D Part region. The Part shape is then defined by generating a COILS encoding of $\mathcal{V}$ relative to $\partial\mathcal{V}$. The Part region can now be transferred to another surface by first finding a local DEM parameterization, and then mapping the 2D Part boundary into it. Finally the COILS deformer is applied to deform the 3D shape to conform to the new Part region.

5.9.1 User Interface

In this section I will describe the drag-and-drop interface from the perspective of the artist. The technical details underpinning each technique are described in the following section.

Selection and Part Extraction

The first step of a drag-and-drop operation is to select a region-of-interest on the 3D surface. My prototype interface supports two selection tools. In the first the artist can draw one or more connected strokes on the surface, from multiple viewpoints, to select an interior set of connected faces. Alternately a lasso stroke beginning on the scene background cuts through the model and selects the enclosed faces. *Intelligent scissor* techniques [Funkhouser et al. 2004] for efficient Part selection could be applied to here, although in practice we observe that our selection boundaries often lie some distance away from geometrically-salient features (Figure5.11).

Once a selection is made and a drag operation initiated, the selection interior is separated from the base mesh into a Part, and the resulting hole is filled with a patch that smoothly blends with the hole boundary. The artist may use simple controls to manipulate the shape of the fill region (Figure 5.12a-c), and once satisfied, accept the current result and move on to the drag-and-drop phase.

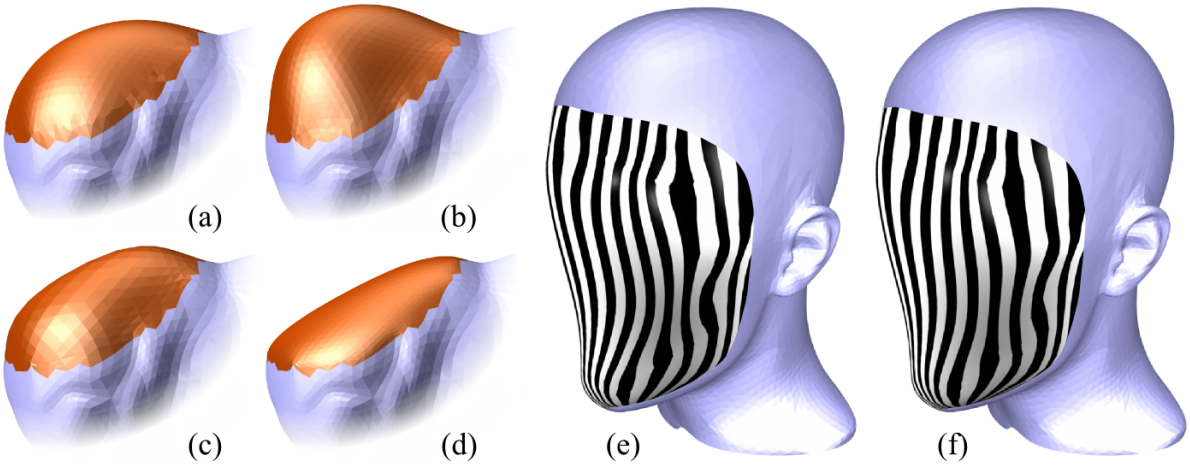


Figure 5.12: *Given an initial smooth fill surface (a), deformation parameters can be tuned to create a wide range of alternatives (b-d). In (e) we set parameters to visually approximate the result (f) of a nonlinear fairing technique [Schneider and Kobbelt 2001]. The reflection lines vary on the interior due to subtle shape differences, but the boundary continuity is visually quite similar.*

Geometry Drag-And-Drop

While in the drag-and-drop state, the artist can interactively drag the active Part across the surface, uniformly scale it, and rotate its boundary "in the surface". These con-

strained interactions do away with the challenges of 3D manipulation, allowing for efficient Part placement. The Part mesh is deformed such that its boundary is embedded in the target surface, and a seamless connection between the target surface and Part is displayed, providing a live preview of the result of the drop operation. If the Part contains too much geometric detail to preview interactively, a reduced-resolution mesh can be shown during interaction, and the full-resolution result is computed when the drop is completed.

To support additional re-use of Parts, the interactive tool includes a Part *library*, a scrolling list attached to the modeling window. The artist can drag the active Part into the Part library, storing it for later use. When a Part from the library is dragged onto the active model, the extraction state is skipped and drag-and-drop mode is entered directly.

Variable Part Rigidity

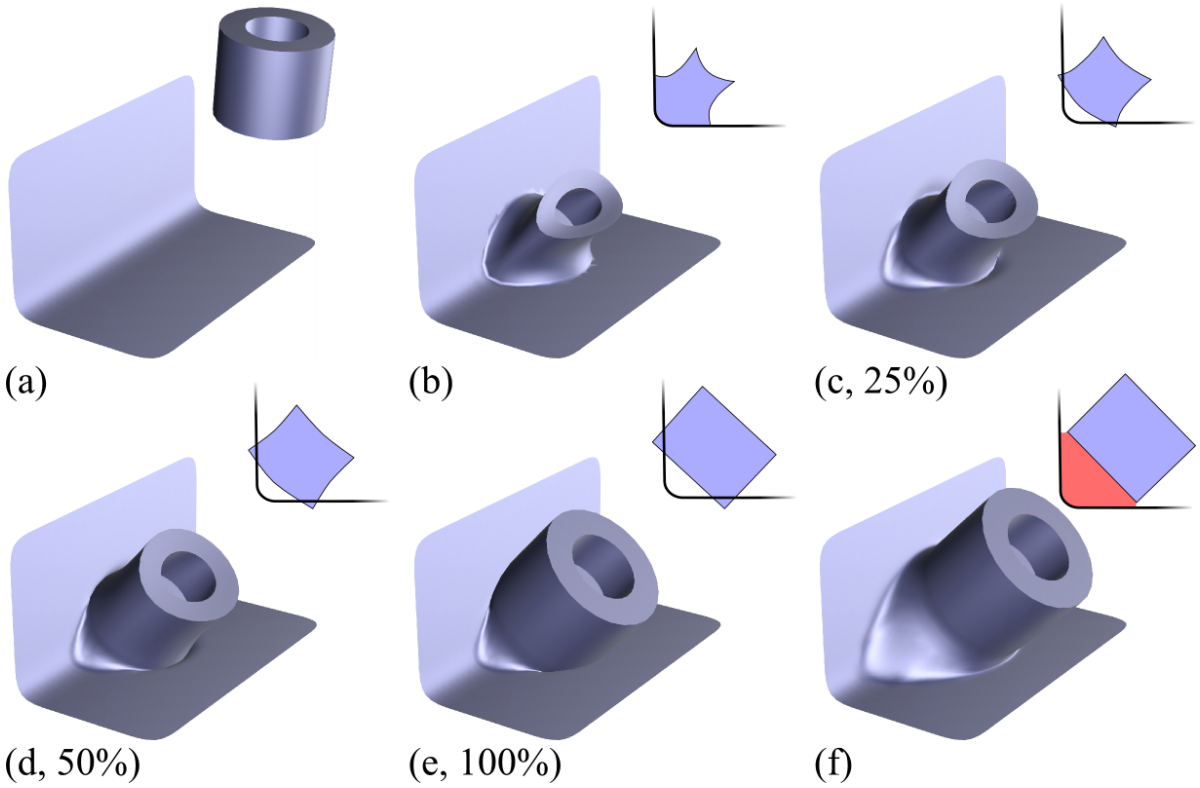


Figure 5.13: *Embedding the boundary of a cylindrical* Part *on a corner surface (a) results in extreme distortion (b), which can be reduced by increasing boundary rigidity (c-e). Offsetting the boundary from the surface and increasing the radius of the fair transition region produces a useful socket (f).*

By default, the drag-and-drop operation deforms the Part to conform to the rigid target surface. An alternative is to smoothly deform the target region to fit the Part.

One can then easily imagine a continuum of solutions between these two extremes, with a partial deformation of both the Part and target region.

To control this blending of deformations, the artist is given a *rigidity* control over the Part boundary. As rigidity is increased above 0, the Part boundary smoothly blends from the embedding in the target surface to its initial rigid configuration, oriented with respect to the target location. Once the boundary is detached from the target surface, a smooth, variable-radius transition surface is added between the target region and the Part. This smooth connection also allows the artist to offset the Part into or away from the target surface, increasing the expressive capability of the tool. Natural extensions would be to allow for further rigid transformation of the Part boundary, or even interactive deformation of the loop vertices.

5.9.2 Algorithms

As mentioned, mesh drag-and-drop involves not only pasting a feature in a new location, but also filling the hole left behind. I have applied my DEM parameterization and COILS deformer to implement these geometric editing operations. Note that in most cases other parameterizations and deformations could be also be used (for example, the Rotation Invariant Coordinates deformation described above). However, it is convenient that only these two simple geometric algorithms need to be implemented to reproduce the system.

5.9.3 Filling Holes

To implement Part extraction we must infer the surface underneath the Part, which in effect means filling the hole left behind when the Part is removed. As it could include interior holes or topological handles, the Part mesh is of no use in solving this problem. Although excellent techniques for completely automatic hole filling have been developed [Schneider and Kobbelt 2001; Davis et al. 2002; Liepa 2003; Bischoff et al. 2005], which can even copy local context [Sharf et al. 2004], I took a deformation-based approach, because it more easily provides the artist with control over the result.

Figure 5.14 displays the steps of my hole-filling algorithm. A *hole* is considered to be a piecewise-linear boundary loop with normals, and without an initial interior surface. The boundary loop is embedded in a 2D circle, such that the distances between vertices are preserved. Additional interior vertices are generated on a regular triangular grid and meshed using Delaunay triangulation [Shewchuk 1996]. The planar boundary is then mapped to the hole boundary, and the interior deformed using COILS, filling the hole.

Unfortunately, if the 3D boundary is significantly non-circular, the fill surface will be highly distorted and may contain foldovers (Figure 5.14b). Hence, the next step is to find a surface which spans the hole, map it to the plane via parameterization, and re-apply the deformation. Although the DEM parameterization produces satisfactory results when the boundary is relatively convex, for more complex cases free-boundary conformal parameterization [Desbrun et al. 2002] is more robust. Ideally, this initial preliminary fill surface would be continuous across the hole boundary. In most cases, however, a minimal mean-curvature mesh suffices, found either by iterative Laplacian

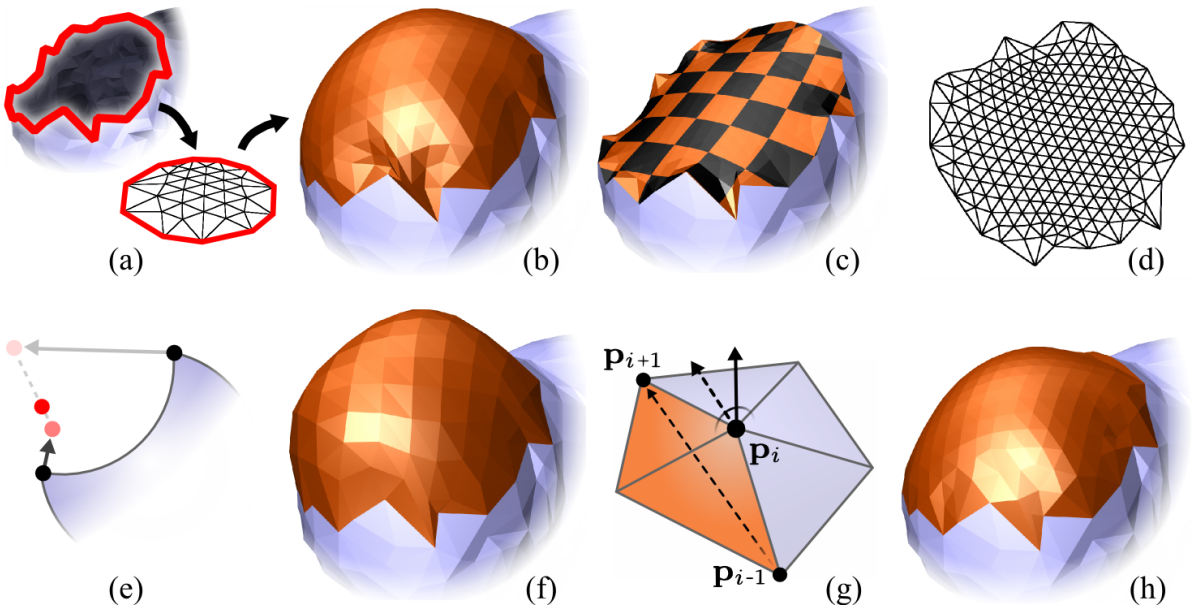


Figure 5.14: *A hole can be filled by embedding the boundary loop in a planar disc (a) and then deforming the disc, but this leads to undesirable distortion for irregular boundaries (b). Instead we first create a membrane surface (c) and find a free-boundary parameterization of this mesh (d). Distant points can pull the fill away from continuity in the transition region (e,f). To better approximate the target normals, we find an optimal rotation around a fixed axis at each boundary point (g), resulting in a smooth fill (h).*

smoothing of the circular fill mesh, or directly by solving Equation 5.3 with the Laplacian vectors of the circular fill mesh set to zero.

Since the optimized planar mesh has a boundary shape very similar to that of the hole, the fill surface is smooth. However, vertices on the "far side" of the hole have non-zero weight, producing an undesirable bulge in the fill surface (Figure 5.14e). To correct this, observe that the (estimated) normals on the hole boundary should be preserved after the fill. This can be accomplished by defining a rotation M_i for each boundary vertex frame, and then finding the set of transformations that minimizes total normal deviation:

$$\arg\min_{M_i} \sum_i |1 - N(i, M_i) \cdot \mathbf{n}_i| \tag{5.11}$$

where $\mathbf{n}_i$ is the target boundary normal at point $\mathbf{p}_i$ and N is a procedure which applies the transformations M_i to the boundary frames, reconstructs the relevant interior region, and estimates the output normal at $\mathbf{p}_i$.

This is a rather complex non-linear optimization problem, but I have found it sufficient to constraint M_i to a 1D rotation around the axis $(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$. Since N is a complex procedure, I apply numerical gradient descent in two stages. First the rotation angles θ_i are tied to a single global angle, resulting in a fast 1D search that removes the largest error. Next the vector θ_i is tuned, which generally converges after 1-5 line searches, where convergence is determined by a total angular error improvement of less than 2° between steps, or when a time budget elapses. Although optimization should be repeated after

each parameter change, this reduces interactivity and can cause some frame incoherence. Instead I optimize once and then let the artist tune parameters based on the initial smooth fill.

This approach produces smooth, boundary-continuous fill surfaces that are of a quality similar to much more complex techniques while also allowing the fill mesh to be interactively re-shaped via deformation parameters (Figure 5.12e-f). Hence, it is also a useful tool for "erasing" surface features and geometry repair.

The main limitation of this approach is that for hole boundaries that deviate significantly from the plane or are highly non-convex, the mean-curvature membrane will not be continuous at the boundaries, and may have very large bending angles. When this occurs, the DNCP [Desbrun et al. 2002] parameterization will have large distortions, and in particular the 2D boundary may not conform to the hole boundary shape, reducing the quality of the fill mesh. In such cases, Schneider and Kobbelt's [Schneider and Kobbelt 2001] fairing method gives good results, but is quite expensive.

5.9.4 Part Insertion

After the hole surface is filled, I find the (approximate) geodesic center using Dijkstra's algorithm and compute a DEM parameterization that contains the entire Part boundary. The boundary is then embedded in the parameterization, as are the relative tangent-normal frames at each boundary vertex.

The embedded boundary can be projected onto any 3D surface via a DEM parameterization of the target region, after which the relative frames are transformed back into global coordinates using the new surface tangent-normal frames. Having determined the necessary boundary constraints for the COILS deformer, the Part mesh is deformed form its original shape and orientation to a configuration which precisely fits the new location. Once the Part is deformed, it is stitched into the target mesh by inserting the boundary within the DEM parameter space with a constrained Delaunay triangulation [Shewchuk 1996]. Rotating and scaling the parameterization transforms the Part, although the COILS displacement vectors must also be scaled.

With this machinery available, the artist can transport a Part across the current surface, or to another surface, simply by positioning, orienting, and scaling a DEM-based decal parameterization, as in the texturing tool of the previous chapter. However, the artist never sees the parameterization. Because the deformation and stitching are performed interactively, to the user it appears that they are directly manipulating a Part which is "on top" of the base surface.

This approach to Part drag-and-drop is quite expensive. Feedback rates can be made more consistent by simplifying the Part interior via edge collapses until it has been reduced to a few thousand vertices, and then representing the full-resolution Part as an offset surface, using the multiresolution approach (Section 5.5) As an illustrative example, with this approach I could drag a 22k-vertex part over a target region containing 1k vertices at several frames per second on a single core of an 2.8Ghz Intel Core2 Duo, while a Part with 10k vertices runs at 10-15 frames per second. To support Parts with more extreme levels of detail, a multiresolution hierarchy [Kobbelt et al. 1998] could be created, and the non-interactive detail levels only reconstructed after the Part is

dropped.

As with hole filling, the boundary optimization described above is usually necessary, as even for features with a sharp edge the designer may include a surrounding buffer region which should be smoothly pasted. To improve interactivity while dragging the feature across the surface, only the first single-angle step of the optimization is performed; per-vertex tuning is deferred until the mouse button is released.

5.9.5 Variable Boundary Rigidity

Deforming the Part boundary to conform to the target surface can result in extensive deviation from the initial shape. Hence, I also explored the option of a boundary with variable rigidity, implemented using a linear variational curve deformation.

I adapted the Laplacian mesh optimization technique described by Nealen et al [Nealen et al. 2006] to curves. In this method, the differential vectors are set to 0 and a weighted "soft" positional constraint is applied to each vertex, with the weight effectively controlling an interpolation between complete fairness and the original shape. We would instead like to blend between the original boundary loop B_o, and the 3D embedding on the target surface, B_t. Hence, we compute the differential vectors using B_o and constrain the vertex positions using B_t. Note that as the differential vectors are not rotation-invariant, we must compute a suitable transformation which aligns B_o with B_t [Horn 1987].

With a very low weight, these positional constraints only fix the global position of the solution, which otherwise looks like B_o. As the weight increases, the curve vertices are smoothly deformed towards B_t. The artist is provided with a rigidity parameter which is simply the inverse of this weight. With this semi-rigid boundary, the deformed part will be disconnected from the target surface, so we find a fair transition surface by solving a linear variational thin-plate problem [Botsch and Sorkine 2008] within a variable-radius cylindrical region of the target surface. Given this flexible connective surface, it is straightforward to allow the boundary to be shifted in to or out of the surface.

5.9.6 Results

To evaluate the drag-and-drop interface, I made a prototype surface compositing tool called *meshmixer* available for free on the internet[3]. A wide range of interesting feedback was collected on this tool, which I will discuss in more detail in Chapter 7. Here I will only briefly demonstrate some of capabilities of the tool.

Perhaps the most prevalent use of surface modeling is for creating characters for the entertainment industries (film, video games, and so on). Figure 5.15 includes various mythical creatures generated using the drag-and-drop tool, each in just a few minutes.

Surface models are also used extensively in industrial and automotive design. In these fields, the actual designers often prefer to work with sketches and physical models, rather than digital models, because of the comparative ease of iterative design. Simplified Part-based interfaces like mesh drag-and-drop have the potential to make digital tools much

[3]http://www.meshmixer.com

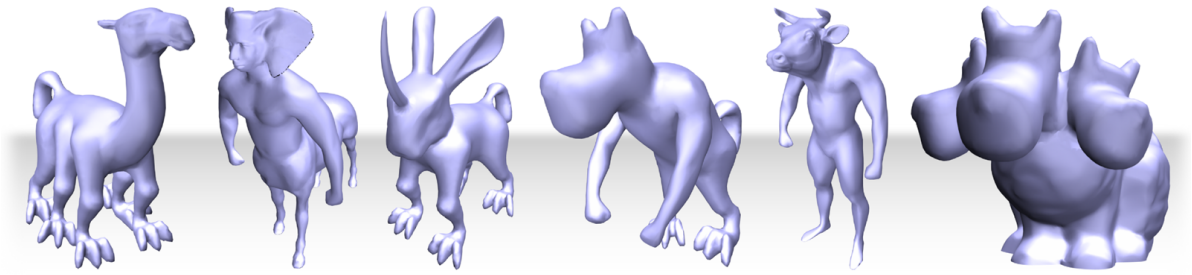


Figure 5.15: *A gallery of mythical creatures including (left to right) the elusive 6-legged Allocamelus, Elephant-Eared Centaur, Al-mi'raj, Cynocephalus, Minotaur, and the terrifying Cerebunny. Each of these examples was created in just a few minutes using my interactive drag-and-drop* Part *compositing tool.*

more usable by traditional designers. In Figure 5.16 I experimented with two example scenarios - satisfying engineering constraints, and integrating styling features from two models. This latter examples shows one of the benefits of the COILS deformer, which can handle the large interior hole.

The biggest complication with the meshmixer prototype was the Part selection technique. My simple tool only cuts on existing edges, and in some cases the source meshes had inconvenient tessellations. The ability to cut across faces, combined with one of the *Intelligent scissor* techniques described in the literature [Funkhouser et al. 2004; Sharf et al. 2006], would be a welcome addition.

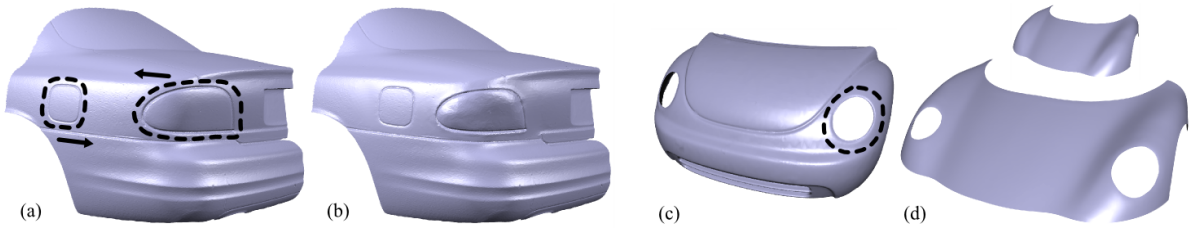


Figure 5.16: *Features of a scanned car surface (a) are dragged-and-dropped to (b) satisfy hypothetical engineering or design constraints. Since the COILS deformer can be applied to non-manifold features, it can be used to transfer headlight cut-outs (c) to the hood of another car (d).*

The simplicity of the meshmixer interface provides a level of expressive freedom not available in previous composition tools, or even in many other "intuitive" and "easy-to-use" shape modeling tools. Some of the most positive feedback I received came from artists who wish to utilize 3D modeling in their physical works. The main impact of meshmixer seemed to be that it transformed tasks which had previously been tedious, error-prone, and best avoided if at all possible, turning them into enjoyable creative experiences. This expands the range of models which are practical to create. Figure 5.17 shows a variety of creative models produced by people who have experimented with the meshmixer tool. Note that in most of these examples, the artists are unlikely to ever

have invested the time necessary to create the results in a traditional tool.

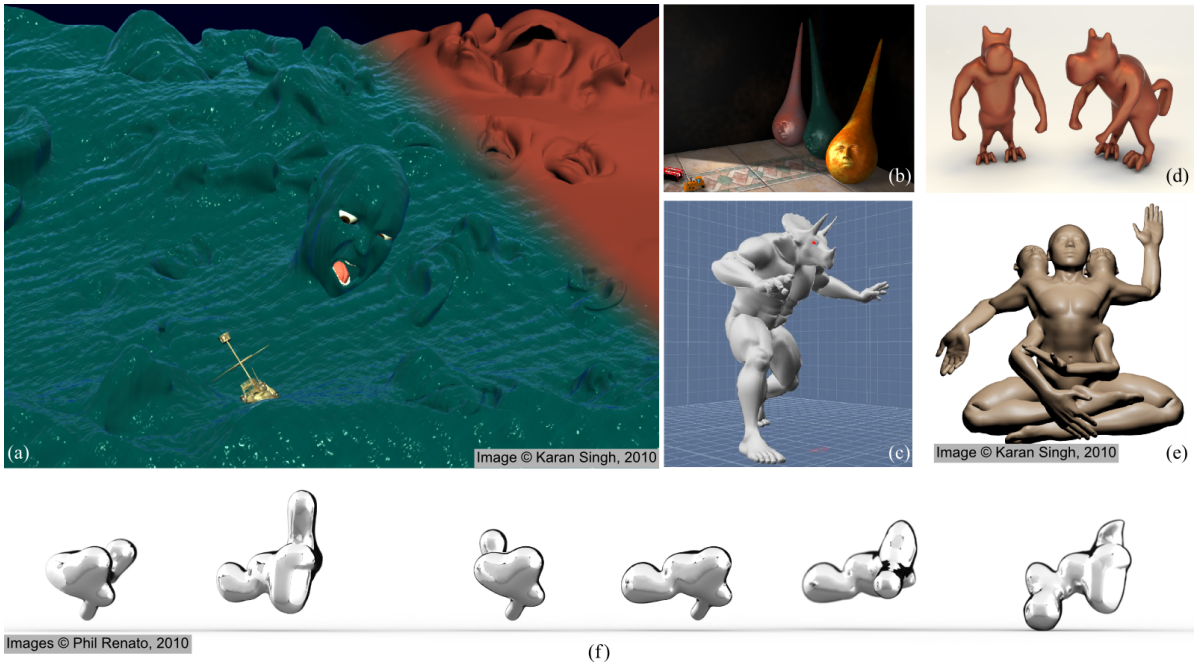


Figure 5.17: *My drag-and-drop interface makes it simple and even enjoyable to create interesting 3D compositions.* "Sea of Faces" *(a) is composed of 42 parts and was created in under an hour by a graphics researcher, as was* "Bodhi" *(e).* "Toys for my Grandchild" *(b) and* "Tricerasapien" *(c) were contributed by testers of meshmixer. A professional artist and jewelry designer sent us the results of a form exploration (f), using our tool to mix fluid simulation "spatters".*

5.9.7 Limitations

The use of the DEM leads to a major limitation of meshmixer, namely that Part boundaries need to be roughly circular and the surfaces onto which Parts are pasted need to be relatively smooth. A fundamental property of the DEM is that it will distort and even fold-over when passing across regions with widely varying curvature. This can cause visible tearing and even catastrophic failure in the local remeshing operations. A more robust local parameterization, or perhaps a scheme for local transport of the boundary across the surface, could improve the capabilities and robustness of the tool. The parallel transport algorithm of Crane et al. [Crane et al. 2010] may help with the latter problem, however it will do nothing to help with *generalized transport* involving arbitrarily complex target surfaces. In these cases, parallel projection will be more robust.

Two other similar problems were raised by professional artists, most of whom use rely almost exclusively on quad meshes for production models. The *edge loops* carefully designed into their edge loops to support high-quality re-posing and animation (Section 3.2.1) were generally destroyed by the triangles we inserted to fill holes or stitch the Part to the target mesh. These regions must be manually "re-topologized". Global

quad-meshing algorithms do not appear to address these hole filling and stitching problems, as it is imperative that the existing quad mesh structure not be modified. Based on the feedback I collected, this appears to be the most pressing issue for artists and designers who have tested meshmixer.

5.10 Conclusions

In this chapter I presented a new approach to shape-preserving surface deformation, which I called the COILS deformer. This deformer is driven by a geometric front propagation, so in some sense it solves an initial-value problem, in contrast to the boundary-value problems solved by variational and energy-minimizing techniques. The COILS deformer allows the designer to control the deformation of any point-sampled geometry via arbitrarily-complex parameters, at interactive rates. The popularity of geometric deformation techniques in commercial modeling tools suggests that these are significant practical benefits.

In comparison with the variational Rotation Invariant Coordinate (RIC) deformation, COILS is neither smooth nor topologically robust. However, I did show that it was more rigid under large deformations, and in particular is able to preserve relatively extreme boundary conditions that cause RIC to break down. In general, I found that with the COILS deformer it was easier to provide the artist with tools to control how deformation propagated from the boundary into the Part surface.

An interesting question is whether COILS and variational techniques like RIC can be integrated. The smoothness and topological limitations of COILS are largely due to the use of a distance field to determine upwind ordering. One possibility is to replace the distance field with a smooth harmonic field, found via variational interpolation. Another is to use the result of the COILS deformer to determine per-vertex frame transformations, and apply these transformations to the Laplacian vectors (essentially replacing the rotation-estimation step of RIC). The resulting least-squares solve may diffuse the discontinuities in the orientation field.

Another recent work which provides a geometric approximation to a variational problem is Farbman et al. [Farbman et al. 2009]. This work factorizes the 2D Poisson equation such that the problem reduces to finding a smooth thin-plate (Laplacian) interpolant. This variational interpolant is then replaced with a geometric solution, found via generalized mean-value coordinates [Schaefer et al. 2007]. The simple boundary-only deformation I described in Section 5.2 takes a roughly similar approach, although in 3D, using inverse-distance weights, and with the addition of displacement vectors. It is possible that the COILS deformer may have a similar sort of variational interpretation, although I have yet to be able to make any connection to existing techniques.

Based on the COILS deformer, I completed my Part definition and described a novel artist-oriented Part drag-and-drop tool for polygonal meshes. This tool allows a 3D designer to quickly compose a complex 3D model by pulling in Parts from a pre-existing library, as well as create new library entries simply by selecting and dragging a region of any existing mesh. As was my intent, this tool is highly interactive, allowing the designer to drag features across a surfaces and manipulate parameters at real-time

rates.

While the meshmixer tool described in this chapter focuses on creating static meshes, there are no technical restrictions preventing it from transparently building a Surface Tree which represents the construction history of the assembly. Hence, the meshmixer tool is essentially an interface for interactively assembling Surface Trees from existing PARTs. Furthermore, as it can be used to select and extract PARTs as well as infer the surfaces "underneath" these PARTs, meshmixer could be extended to support reverse-engineering of Surface Trees from static models.

Chapter 6

The Surface Tree: A Hierarchical PART-Based Representation

Some material in this chapter is derived from the article "Sketch-Based Procedural Surface Modeling and Compositing with Surface Trees" authored by Ryan Schmidt and Karan Singh [Schmidt and Singh 2008]. The text and images reproduced here are used with permission from my co-author.

6.1 Introduction

In the previous two chapters I have described one approach to implementing the domain/shape surface PARTdefinition of Chapter 3. The Discrete Exponential Map (DEM) allows the PART domain to be encoded relative to a single surface point, and if we embed the part boundary in the DEM parameter space, the COILS deformer can reconstruct the PART shape.

The next question is, can a hierarchical, procedural surface representation be constructed out of these parts? The drag-and-drop tool described in the previous chapter is in some sense a very simple 2-node procedural hierarchy. In this chapter I will explore how to *layer* these PARTs in a procedural manner. Once a sequence of PARTs can be layered, we will be able to take the output of the layering and insert it into another layer sequence. The result is a structured tree-like hierarchy which procedurally represents a complex surface model, which I will call a *Surface Tree*.

Recall that in Chapter 2 I described several properties that we would like to have in a procedural model - abstraction, independence, hierarchy, and composability of nodes, and suitability for interactive use. We will see that my Surface Tree does better on these axes than any previous work. However, as each PART does depend on a single point embedded in some other manifold, Surface Tree nodes cannot be completely independent in the same way that a solid modeling node can be.

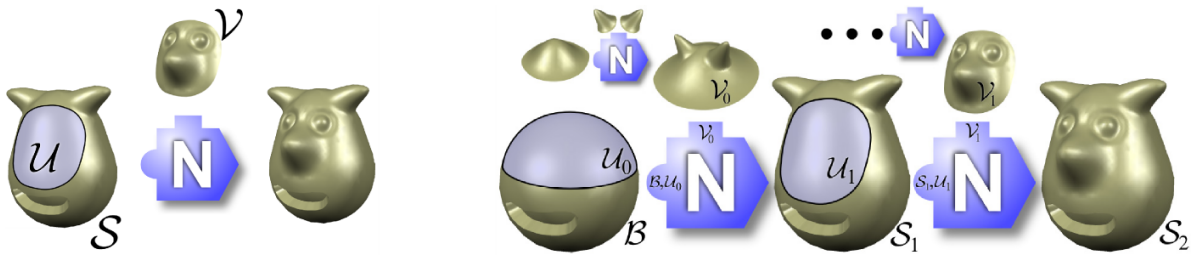


Figure 6.1: *A surface editing operation locally replaces some region $\mathcal{U}$ of a surface with a new patch $\mathcal{V}$ (left). In a Surface Tree, operations can be applied hierarchically, although secondary branches must output surfaces with boundaries to be properly merged (right).*

6.2 The Surface Tree

The first step in constructing the Surface Tree is to define a *Node*. In a CSG Tree, a node is typically either a *primitive*, a *unary* operator, or a *composition* operator. At a more conceptual, however, each of these different types of nodes simply modifies the existing inside/outside classification for each point inside a fixed volume. Hence, a general solid modeling node replaces some 3D volume with another. Since we are operating on surfaces, we can define a similar general surface modeling node as one which takes an input surface $\mathcal{S}$, some region $\mathcal{U} \in \mathcal{S}$, and replaces $\mathcal{U}$ with a new surface $\mathcal{V}$:

$$N(\mathcal{S},\mathcal{U},\mathcal{V}) = (\mathcal{S} \setminus \mathcal{U}) \cup \mathcal{V} \tag{6.1}$$

Note the symbols here are exactly the same as those used in my PART domain ($\mathcal{U}$) / shape ($\mathcal{V}$) decomposition, and Equation 6.1 is exactly the Assembly form from Section 3.4. This is not accidental - the process if attaching a PART using the ON operator provides most of the key components of a Surface Tree node. We will see below that we only need to add one more piece of information to combine PARTs into a procedural hierarchy. At an abstract level, though, a node simply replaces $\mathcal{U} \in \mathcal{S}$ with $\mathcal{V}$.

$\mathcal{N}$ can be thought of as a surface *compositing* operation, and hence a complex surface can be recursively defined by applying nodes to a *base surface* $\mathcal{B}$:

$$\mathcal{S}_{i+1} = \mathcal{N}_i(\mathcal{S}_i,\mathcal{U}_i,\mathcal{V}_i) \tag{6.2}$$
$$\mathcal{S}_1 = \mathcal{N}_0(\mathcal{B},\mathcal{U}_0,\mathcal{V}_0) \tag{6.3}$$

Intuitively, the final output surface is defined by incrementally *layering* a series of surface patches $\mathcal{V}_i$ onto $\mathcal{B}$. Although the recursion above is sequential in nature, any $\mathcal{V}_i$ can be defined by another series of compositions. Hence, a Surface Tree is a structured binary tree of composition nodes $\mathcal{N}$, with a *primary branch* that contains $\mathcal{B}$ as the initial leaf node, and a series of nested *secondary branches* which feed into the $\mathcal{V}_i$'s of the primary branch (Figure 6.1).

Note that the arguments to $\mathcal{N}$ are not independent. $\mathcal{S}$ can be any surface, but it is necessary that $\mathcal{U} \subseteq \mathcal{S}$, and $\mathcal{V}$ must always have boundaries compatible with $\mathcal{U}$. This implies that only $\mathcal{B}$ can be a surface without boundary - all $\mathcal{V}_i$'s must have a boundary loop (and hence so must all secondary branches).

6.2.1 Expanding the Definition

Equation 6.3 provides the most abstract definition of a Surface Tree, as a tree of nodes which replace surface regions with others. However, unlike in the case of solid modeling where any volume replacement is valid, certain constraints must hold for Equation 6.1 to be evaluated. We can expand Equation 6.1 to make these constraints more explicit, and also satisfied by definition.

A major assumption in the equations above is that the boundary of the replacement surface $\mathcal{V}$ precisely aligns with the region $\mathcal{U}$ on the input surface, $\partial\mathcal{V} = \partial\mathcal{U}$. As we saw in Section 3.4, one way to guarantee that this property holds is to define $\mathcal{V} = \mathsf{E}(\mathcal{U})$, where E is a boundary-preserving editing operation (Figure 6.2). Ideally it will also be the case that E can be recomputed for any $\mathcal{U}$.

We now can assume that $\mathcal{V}$ can be be procedurally re-computed if $\mathcal{U}$ changes. However, if the designer changes $\mathcal{U}_0$ in Figure 6.1, $\mathcal{S}_1$ will change, and $\mathcal{U}_1$ will need to be procedurally re-computed as well.

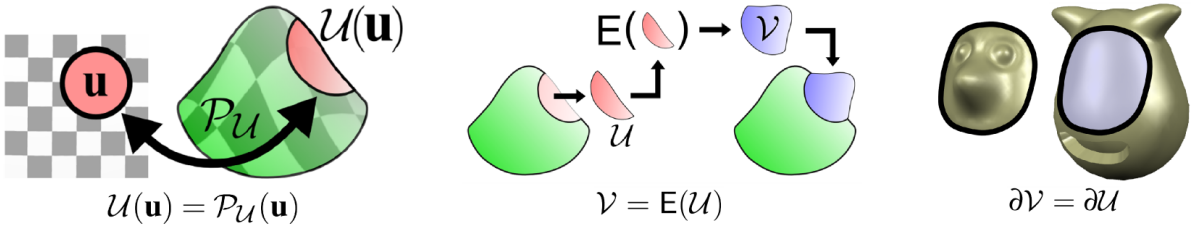


Figure 6.2: *To support a dynamically-changing input surface $\mathcal{S}$, the editing region $\mathcal{U}$ is stored in a parameterization of $\mathcal{S}$ which can be recomputed if $\mathcal{S}$ changes (left). Similarly, and replacement patch $\mathcal{V}$ is represented as an editing operator applied to $\mathcal{U}$ (center) which preserves the boundary $\partial\mathcal{U}$ (right).*

As discussed in Section 3.4.8, to ensure maximum flexibility it is desirable that the re-computation of $\mathcal{U}$ be as independent of $\mathcal{S}$ as possible. One way to accomplish this is to use the tools of Riemannian geometry [do Carmo 1994]. We restrict $\mathcal{S}$ to 2-manifolds embedded in $\mathbb{R}^3$, guaranteeing that any point on $\mathcal{S}$ has a local neighbourhood with disc-like topology. $\mathcal{S}$ is then covered with a finite *atlas* of topological discs, referred to as *coordinate patches*. A mapping $\mathcal{P}$ known as a *planar parameterization* exists between each 3D patch and $\mathbb{R}^2$. Given an atlas on $\mathcal{S}$, we can now *encode* $\mathcal{U}$ as a mapping from some 2D region $\mathbf{u}$ of the atlas parameter-space to the 3D surface. To simplify the following exposition, assume that $\mathcal{U}$ is contained within a single coordinate patch with parameterization $\mathcal{P}_\mathcal{U}$. Then we can write $\mathcal{U}(\mathbf{u}) = \mathcal{P}_\mathcal{U}(\mathbf{u})$ (Figure 6.2), and rewrite Equation 6.1 as

$$\mathcal{N}(\mathcal{S}, \mathbf{u}, \mathsf{E}) \;=\; (\mathcal{S} \setminus \mathcal{U}(\mathbf{u})) \;\cup\; \mathsf{E}\,(\mathcal{U}(\mathbf{u})) \tag{6.4}$$

With this formulation, Surface Tree nodes can be procedurally re-computed because the editing region $\mathcal{U}$ is encoded independent of the current 3D embedding of the input surface $\mathcal{S}$. Instead, it depends on the parameterization. As I have mentioned, available manifold representations for arbitrary surfaces have a variety of limitations (Section 3.6.1). Even if a manifold were available, mapping areas between manifolds is highly

non-trivial (Figure 3.12). Hence, to build a practical system using mesh or point set representations, we must avoid assuming the existence of a global, consistent manifold parameterization. This is problematic because we now lack a global embedding space in which to fix the location and shape of each layer. In the next section, I will describe how we deal with this issue and implement Surface Trees in an interactive system.

6.3 Implementing a Surface Tree

In Section 6.2 I described a mathematical formulation of a Surface Tree. In this section I describe one practical implementation of the Surface Tree, focusing on how the tree is constructed and updated. To begin, I will limit surfaces $\mathcal{S}$ to triangular meshes, although the general approach is also applicable to point sets and arbitrary polygonal meshes.

6.3.1 Constructing a Global Parameterization

Since a global manifold will not be available, we need to devise a way to represent $\mathcal{U}$ and map it between surfaces. Recall that in Section 3.6.1 I explained that even if a manifold were available, these problems would be more easily solved by encoding $\mathcal{U}$ within a parameterized geodesic disc. In this case, the 2D manifold region $\mathbf{u}$ shrinks down to a single point, and we must also store the geodesic radius r. Note that this geodesic disc may not be specified directly, as a node can be initially created by specifying $\mathcal{U}$ directly on the surface. In this case we must determine a $\mathbf{p} \in \mathcal{S}$ and geodesic radius r, such that the geodesic disc $d_g(\mathbf{q}, \mathbf{p}) < r$ contains $\mathcal{U}$. We can then map $\mathcal{U}$ into the DEM parameter space of this geodesic disc, resulting in a sort of canonical 2D representation of $\mathcal{U}$. To find $\mathcal{U}$ on some other surface, we need only to compute and parameterize a geodesic disc, and then $\mathcal{U}$ can be mapped to the new surface.

With this geodesic-disc decomposition in place, we can rewrite Equation 6.1 as

$$\mathcal{S}_{out} = \mathcal{N}(\mathcal{S}_{in}, \mathbf{u}, r, \mathsf{E}) \tag{6.5}$$

where $\mathbf{u}$ is now a single point embedded in some parameterization, r is the radius of a geodesic disc which contains $\mathcal{U}$, and E is a mesh editing operation. The primary branch of a Surface Tree built with these nodes begins with a base mesh $\mathcal{B}$ and incrementally applies procedural mesh editing operations:

$$\begin{aligned} \mathcal{S}_{i+1} &= \mathcal{N}_i(\mathcal{S}_i, \mathbf{u}_i, r_i, \mathsf{E}_i) \\ \mathcal{S}_1 &= \mathcal{N}_0(\mathcal{B}, \mathbf{u}_0, r_0, \mathsf{E}_0) \end{aligned}$$

Note that $\mathbf{u}$ must be mapped to a 3D location on the surface before the geodesic disc can be computed. I will refer to both this 3D point, which I will denote $\mathbf{p}$, and the 2D point $\mathbf{u}$ as the *seed point* of the node, with the embedding of the point determined by context.

We have simplified the problem to embedding the $\mathbf{u}$ in some underlying parameterization. However, we still do not have a manifold parameterization. To move forward, I will assume that the mesh edit E maintains the disc-like topology of $\mathcal{U}$. If this is the case, then $\mathcal{V}$ can be parameterized, and a one-to-one map exists between both the surfaces

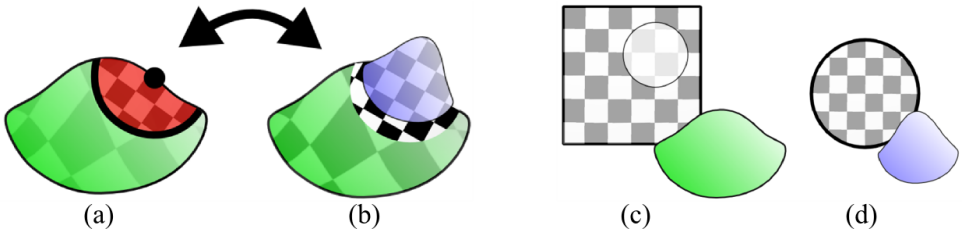


Figure 6.3: *If the edit applied by a node is an invertible deformation of the manifold, then the parameterization provides a map between* $\mathcal{U}$ *(a) and the deformed surface (b). A global manifold-like parameterization of the final surface is now provided by the composition of (c) the masked parameter space for the green surface and (d) the parameter space for the edit.*

and parameterizations of $\mathcal{U}$ and $\mathcal{V}$ (Figure 6.3a-b). Note that in some sense, any E which maintains disc-like topology can be cast as a *deformation* of the manifold, in which case it becomes clear that the parameterization of $\mathcal{U}$ *is* a parameterization of $\mathcal{V}$.

If we restrict E to deformations, then we have effectively created a global parameterization which covers the surface. This structure is not technically a manifold, but it serves much the same purpose for embedding individual points. To understand this global parameterization, note that evaluation of a Node (Equation 6.5) not only creates a hole in $\mathcal{S}_{in}$ and replaces it with $\mathcal{V}$, but also creates a hole in the parameterization of $\mathcal{S}_{in}$ and replaces it with the parameterization of $\mathcal{V}$. The union of these two parameter spaces - the parameterization of $\mathcal{S}$ with a hole cut out of it, and the parameter space of the inserted PART - precisely cover the composite surface, and hence a global parameterization is maintained at the output of each node (Figure 6.3c-d).

Note that if the parameterization of $\mathcal{V}$ is taken from $\mathcal{U}$, as is straightforward if E is actually a deformation, the above statements are somewhat obvious. In fact this is exactly the process that occurs in $H-Splines$ and Surface Pasting - each node is simply deforming the manifold, and the global parameter space is never modified. However, in a more general node based on my PART definition, E will involve stitching an arbitrary surface into $\mathcal{U}$. The resulting mesh will have a completely new parameterization, but as long as it still has disc topology, no changes need to be made to the above discussion. One must simply take care to realize that the global covering is actually a disjoint set of parameter spaces, with maps between the boundaries.

6.3.2 Node Anchoring

For the purposes of storing individual points on a surface, the global covering parameterization I have just presented serves much the same purpose as a proper manifold parameterization. We can embed a point in the composite parameter space, and map that point back to the surface. The main difference with a proper manifold is that we have no transition functions between the disjoint regions of the parameterization. However, for our purposes the computations for which one might desire transition functions can be computed directly on the 3D surface.

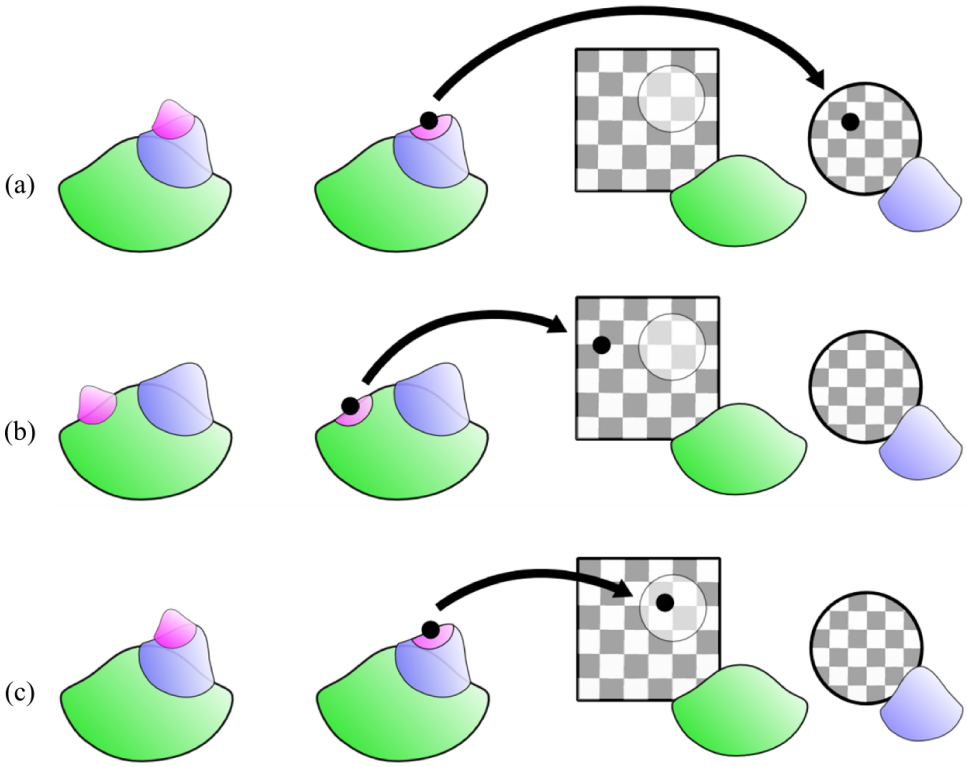


Figure 6.4: *Assuming we have the atlas from Figure 6.3, a layered node can be anchored in three different ways. If the 3D position of the anchor is (a) within the blue node, then a sensible choice is to store it in the parameter space of the blue node, and similarly (b) if the node is on top of the green surface. Note, however, that a node layered over the blue surface can also be anchored in the parameter space of the green node, as in (c). In this case the 3D surface position can only be found by mapping through the blue node.*

Given this global parameterization, the center point $\mathbf{p}_i$ of the geodesic-disc support region for a new node can be embedded as an *anchor point* $\mathbf{u}_i$ in the parameter space of some upstream node $\mathcal{N}_{j<i}$. Figure 6.4 considers the three different cases of embedding $\mathbf{p}_i$ in a surface composed of two parameter spaces. We can imagine this surface being covered by a sort of patch-quilt of parameter spaces, in which case the straightforward approach is to embed $\mathbf{p}_i$ in the parameter space of the patch which contains it. However, technically there may be additional parameter spaces "underneath" that of the uppermost node, and it is also possible to embed $\mathbf{p}_i$ in any of these parameter spaces (Figure 6.4c). This possibility complicates the process of recovering $\mathbf{p}_i$ from $\mathbf{u}_i$, as I will discuss below.

Figure 6.5 shows some additional examples of anchoring nodes in simple Surface Trees. Figure 6.5b makes clear one of the benefits of the anchor point approach - it cleanly handles the ambiguous cases where an editing region $\mathcal{U}$ overlaps the boundaries of several underlying layers. Since the anchor is a single point, it can be embedded in the parameter space of a single input node. The boundary overlap still occurs once the geodesic disc is created, but at that point we are operating directly on the composite output mesh and the composite global parameterization is no longer in use.

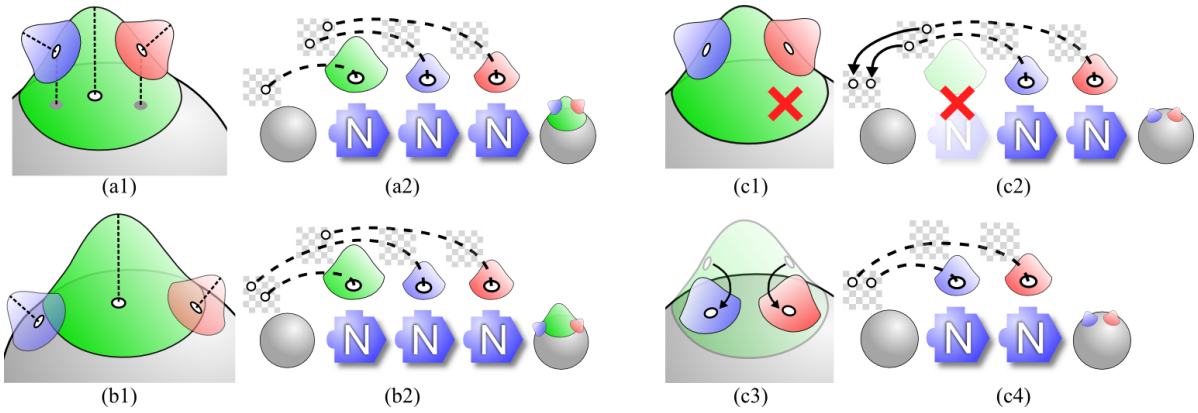


Figure 6.5: *Three bumps are layered onto a sphere In (a1-a2), the red and blue bumps are anchored to the parameter space of the green bump, while the green bump is anchored to the sphere. In (b1-b2), the blue bump has moved off the green bump, and so is also anchored to the sphere. In (c1-c4), the green bump is deleted. In this case the anchors for the red and blue bumps are mapped down to the sphere before the bump is deleted. Then the bumps can be recomputed on the sphere.*

6.3.3 Node Evaluation

With these encoding and anchoring schemes in place, we can now discuss how Equation 6.5 can be implemented. Psueodocode for my evaluation algorithm is shown in Algorithm 6.3.3. The process is relatively straightforward, once the surface anchor is known, we compute the parameterized geodesic disc using the DEM, find the new PART shape, and then insert the PART mesh into the target surface. The main complication is the *ProjectToSurface* function, which I will now describe.

$$
\begin{aligned}
&\mathcal{S}_{i+1} \leftarrow \mathcal{N}(\mathcal{S}_i, \mathbf{u}_i, r_i, \mathsf{E}_i) \\
&\quad \mathbf{p} = ProjectToSurface(\mathbf{u}_i) \\
&\quad \mathcal{U} = GeodesicDisc(\mathbf{p}, r_i) \\
&\quad \mathcal{P}_{\mathcal{U}} = ExpMapParameterization(\mathcal{U}) \\
&\quad \mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \mathcal{U}) \quad \cup \quad \mathsf{E}_i(\mathcal{U}, \mathcal{P}_{\mathcal{U}})
\end{aligned}
$$

Algorithm 1: *Computing the output of a Surface Tree node.*

Recall from the previous section that each node i is anchored in the parameter space of some upstream node $\mathcal{N}_{j<i}$. This embedded anchor can be mapped forward through the intervening nodes to unambiguously fix the position of the seed point on the current surface using the algorithm listed in Algorithm 6.3.3. Two examples of this process are visually depicted in Figure 6.6. Note that as a parameterized point $\mathbf{u}_i$ has both 2D and 3D coordinates, here usage is determined by context.

With the *ProjectToSurface* algorithm in place, we now know how to evaluate Equation 6.3.3. Figure 6.7 graphically depicts this evaluation process. This figure also makes clear a significant aspect of my Surface Tree implementation, namely that the anchor point approach creates a dependency between $\mathcal{N}_i$ and $\mathcal{N}_j$ (Figure 6.5). As a result, the

$\mathbf{p} \leftarrow$ ProjectToSurface($\mathbf{u}_i$)
    $k = j : \mathbf{u}_i \subset \mathcal{N}_j$
    while k != i
        $\mathbf{u} = \mathsf{E}_k(\mathbf{u})$
        $k = k + 1$
    $\mathbf{p} = \mathbf{u}$

Algorithm 2: *Projecting a seed point to a surface.*

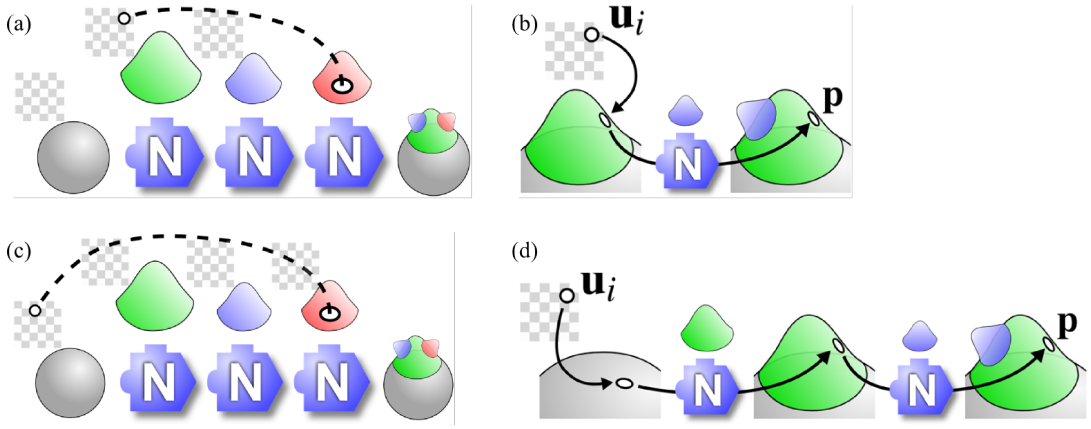


Figure 6.6: *In (a), the red bump is anchored in the parameter space of the green bump. To map this anchor to the surface (algorithm* ProjectToSurface*), we (b) must first project it to the surface of the green bump, and then apply the edit operation of the blue bump, as the blue-bump node is "below" the red bump. Similarly, in (c) the red bump is anchored to the sphere, so it must pass through (d) both the green and blue bumps to reach the surface.*

Surface Tree is not strictly a tree, but rather a highly structured dependency graph [Haeberli 1988].

I should note that it is not clear how one could possibly avoid this dependency structure. Even if a full manifold parameterization were available at each node output, node i would still need to be dependent on node $i-1$ to maintain consistent positioning on the surface if the manifold were to change. I believe this is upstream dependency is unavoidable because the "space" over which each node is defined is ultimately another surface. This is a key difference with CSG nodes, which are each defined over a consistent space, namely $\mathbb{R}^3$. The Surface Tree dependency structure does complicate tree manipulation operations, as I will discuss in the next section.

Secondary Branches The above discussion computes node i based on node $i-1$, and hence only explicitly handles the primary branch. Secondary branches generate arbitrary meshes with boundaries, which must be merged into the primary branch. This is accomplished by defining E to be the merging operation. I described precisely such an operation in the previous chapter, in Section 5.9. The basic idea is to embed the boundary of the mesh in a parameter space, insert this boundary into the PART domain

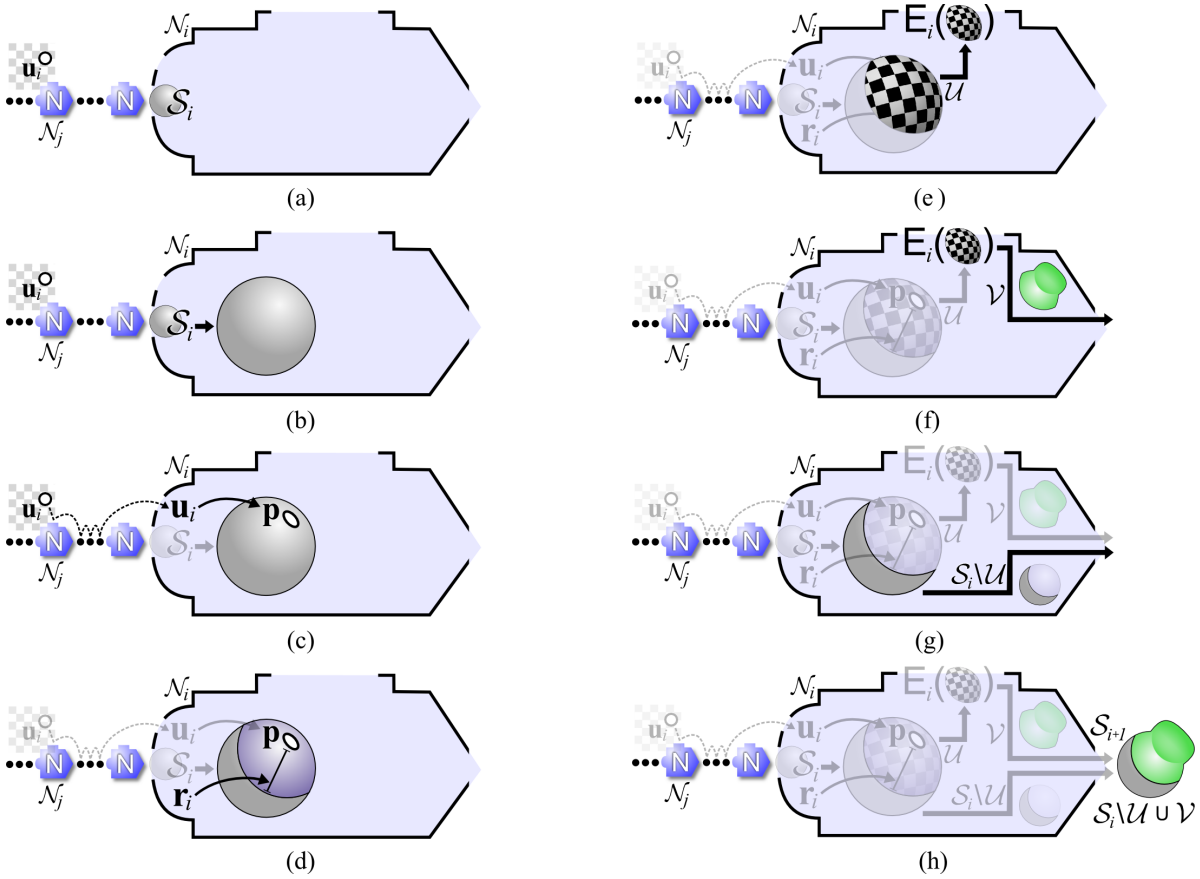


Figure 6.7: *To evaluate node $\mathcal{N}_i$ for (b) the input surface $\mathcal{S}_i$, we (c) first take the anchor point $\mathbf{u}_i$, which is embedded in some upstream node $\mathcal{N}_{j<i}$, and map it forward to the point $\mathbf{p}$ on the input surface $\mathcal{S}_i$. Next we compute the editing region $\mathcal{U}$ by (d) segmenting from $\mathcal{S}_i$ an approximate geodesic disc with radius r_i around $\mathbf{p}$, (e) parameterizing this disc using the DEM, and then (f) passing it to the shape operation E_i to generate the edited region $\mathcal{V}$. Finally $\mathcal{V}$ is combined with (g) $\mathcal{S}_i \setminus \mathcal{U}$ to produce (h) the output surface $\mathcal{S}_{i+1}$.*

$\mathcal{U}$, and then use the COILS deformer to deform the PART shape, in this case the output of the secondary branch. If the part is simple enough, a local geometric texturing [Elber 2005] approach can also be taken - an example is shown in Figure 6.1.

6.3.4 Surface Tree Manipulation

One advantage of the Surface Tree construction I have described is that nodes can be dynamically manipulated simply by interacting with the anchor points using the surface-constrained handle described in Section 4.6. This 3D widget, which also incorporates components for rotating and scaling, allows edits to be quickly "dragged-and-dropped", re-ordered, and deleted.

Recall that at $\mathcal{N}_i$, the input surface is covered by a composite parameterization formed from a set of disjoint patches defined by the upstream nodes $\mathcal{N}_{j<i}$. If we assume that $\mathcal{N}_i$ is initially anchored in $\mathcal{N}_j$, then as the user drags the 3D anchor point $\mathbf{p}_i$ across the surface, it may pass out of the parameter-space region of $\mathcal{N}_j$ and into the space of $\mathcal{N}_{k\neq j}$. In this case we must dynamically re-anchor the node.

In Figure 6.4 I showed that at any point on the current surface, there may be multiple underlying parameter spaces. Hence, when dynamically re-anchoring a node we may have a choice of which parameter space to embed it in. In my interactive tool, I took perhaps the simplest approach: each time node $\mathcal{N}$ is explicitly moved by the designer, I automatically re-anchor its seed point in the "nearest" input node using Algorithm 6.3.4. Note that in this algorithm the inverse mapping E^{-1} is required. To avoid requiring E to be explicitly invertible, I implement the inverse mapping by finding the triangle containing $\mathbf{p}$ and interpolating $\mathbf{u}$ from the triangle uv-coordinates.

$\mathbf{u} \leftarrow$ FindAnchorPoint($\mathbf{p} \in \mathcal{N}_i$)
    $k = i - 1$
    $\mathbf{u} = \mathsf{E}_k^{-1}(\mathbf{p})$
    while $\mathbf{u} \notin \mathcal{U}_k$
        $\mathbf{u} = \mathsf{E}_k^{-1}(\mathbf{u})$
        $k = k - 1$

Algorithm 3: *Finding the anchor point for a surface point.*

Algorithm 6.3.4 can also be used to handle anchor point management during explicit tree manipulation. There are two main operations which must be handled, namely removing and inserting a node. Before removing node $\mathcal{N}_i$, Algorithm 6.3.4 is used to transfer any seed points anchored in $\mathcal{N}_i$ to some input node $\mathcal{N}_{j<i}$. Once this is done, there are no longer any dependencies on $\mathcal{N}_i$, and it can be can be safely removed by connecting its input $\mathcal{N}_{i-1}$ to its output $\mathcal{N}_{i+1}$. Similarly, to insert $\mathcal{N}_k$ between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, any seed points anchored to $\mathcal{N}_i$ lying within the new support of $\mathcal{N}_k$ are "pushed forward" to $\mathcal{N}_k$.

Although it is not possible to perfectly anticipate the user's intent in all cases, I have found that Algorithm 6.3.4 produces the expected behavior most of the time. Essentially, if node A is anchored to node B, then when B is dragged across the surface, A will "stick" to it. This emulates the object grouping support found in vector graphics tools such as Adobe Illustrator [Adobe Systems Inc. 2007a].

It is important to note that automatic anchoring will only link A to B if the user explicitly drags A on to B - anchoring does not change if B is dragged underneath A. This prevents accidental sticking when a node has larger support than is clearly visible, but can be unintuitive if the artist later forgets the tree structure, attempts to drag B, and expects A to come with it.

The node layer order implicitly defined by the dependency structure of the Surface Tree can also be somewhat unintuitive if node $\mathcal{N}_{i+1}$ involves a large change to the surface, and is dragged on top of a small change at node $\mathcal{N}_i$, which will then be "underneath" it. Conceivably these cases could be detected and the tree automatically restructured, but this would involve significant and costly interrogation of the surfaces themselves, and inference would inevitably be prone to error. Some of the Surface Pasting did in fact try to address this situation by automatically re-ordering layers [Barghiel et al. 1994], but there are many cases where this behavior can produce un-expected results, and it diverges from 2D layer-based interfaces [Adobe Systems Inc. 2007a] where layer ordering is explicitly under user control. Hence, in my interface the user must also explicitly re-order the nodes.

6.3.5 Parts with Arbitrary Topology

In the discussion above, I have assumed that the Part inserted at each node has disc topology. In this case, each node conceptually defines an invertible deformation of the underlying manifold, and hence we can map any point between the input and output surfaces of each node. However, this assumption means that no node can modify the topology of the base surface. Hence, it is useful to consider how we might do away with this assumption.

There are two distinct types of non-disc topology that a Part may have. The first is if the Part has more than one boundary loop, but is otherwise isomorphic to a cylinder. In this case, there are multiple disjoint $\mathcal{U}$'s which can be handled independently. Two holes are cut in the base surface parameterization, and a cylindrical parameterization is used along the Part. I will present an example of this class of Part in Section 6.6.1.

The second case, which can be combined with the first, occurs if the Part internally has topological holes and handles, and hence is not isomorphic to a disc or cylinder. This does not affect the Part insertion, but if the Part surface needs to be parameterized, then we must either make cuts to create a disc topology, or construct a piecewise atlas of local maps.

In both these cases, the main problem is that there is no intrinsic mapping from the base surface to the Part surface. If node $\mathcal{N}_j$ has non-disc topology, then we cannot evaluate *ProjectToSurface* within $\mathcal{U}_j$. Similarly, E_j will not be invertible, so *FindAnchorPoint* must terminate on $\mathcal{V}_j$.

These restrictions are not catastrophic, but they do introduce two constraints. First, we cannot evaluate any node whose anchor would need to map through $\mathcal{U}_j$. In practice, this means that no downstream node $\mathcal{N}_{k>j}$ can be anchored "underneath" $\mathcal{N}_j$, or in other words, $\mathcal{N}_j$ must be "on top" of everything else that overlaps $\mathcal{U}_j$.

Similarly, if we were to remove $\mathcal{N}_j$, we cannot map any anchors attached to $\mathcal{N}_j$ back to the input surface $\mathcal{S}_j$. These nodes can only be discarded. In an interactive system,

an alternative is to force the user to move any anchor points off of $\mathcal{N}_j$ before it can be removed.

Of course, this discussion assumes that we insist on parameter-space maps for all our anchor points. It is also possible to transfer anchor points directly in 3D. For example, after removing $\mathcal{N}_j$ we could find the closets point on $\mathcal{S}_j$ for each 3D anchor position lying on $\mathcal{N}_j$.

6.4 Comparison to Previous Approaches

Many previous procedural surface representations can be expressed as specializations of the Surface Tree. In this section I will consider three of the most expressive alternatives - H-Splines, Multiresolution meshes, and Surface Pasting. In each case I will consider the properties of procedural models that I described in Section 2.2 - abstraction, independence, hierarchy, and composability - and then describe how the operations in these methods map to my Surface Tree notation.

H-Splines

Forsey and Bartel's seminal Hierarchical Splines or H-Splines [Forsey and Bartels 1988] offered the first truly procedural approach to interactive surface editing. An H-Spline begins with an initial B-Spline patch. The regular control point grid can then be locally refined using knot-insertion rules to increase the level of detail. These *detail overlays* are blended into the base patch, and their 3D control points encoded using relative offset vectors. The relative encoding allowed the detail surfaces to naturally deform as underlying layers were interactively modified, producing a truly procedural and hierarchical surface representation. The overlays are resolution-specific and cannot be arbitrarily re-ordered, so abstraction is lacking. However, the uniformity of B-Spline control point grids does confer a reasonable level of independence. Similarly, two detail overlays can be functionally composed only if their underlying topologies are compatible.

In terms of the Surface Tree, each detail overlay is a node or Part, and $\mathcal{U}$ can be defined as a box in the intrinsic B-Spline parameterization. This parameterization is shared between all surfaces in the Tree, ie $\mathcal{P}_\mathcal{S} = \mathcal{P}_\mathcal{B}$. The operator E is a normal-frame-encoded vector displacement of the refined control point network. Note that Parts must be expressed in terms of the control-point grid. In particular, Part boundary edges must all be axis-aligned.

Multiresolution Meshes

Multiresolution surfaces [Zorin et al. 1997; Lounsbery et al. 1997; Guskov et al. 2002] adapt the H-Spline concept to mesh representations. Note that in most multiresolution methods, the notion of local refinement has been discarded, and replaced with global refinement followed by smoothing and displacement steps. This representation is procedural, and technically a hierarchy of displacement layers can be formed, although I am not aware of any system to do so - the standard layout is a flat list. To avoid dependence on global parameterizations, multiresolution methods encode displacements relative to

the base mesh topology. As a result, independence of displacement layers is extremely limited. Assuming topological compatibility, we do have relatively high levels of abstraction and composability, as displacement layers can easily be re-ordered or functionally combined.

To write a multiresolution surface, each refinement/displacement layer becomes a node. No parameterizations are used, instead $\mathcal{U}$ is instead the entire input surface, ie $\mathcal{U} = \mathcal{S}$. The modified surface $\mathcal{V}$ is constructed by refining and displacing $\mathcal{U}$. Hence, any procedural manipulation must be expressed in terms of the refinement and displacement rules, and the displacement representation is tightly coupled with the vertices of the base surface $\mathcal{B}$.

Surface Pasting

Surface Pasting [Bartels and Forsey 1993; Barghiel et al. 1994] was developed to address one of the key limitations of H-Splines, namely that the boundaries of detail overlays were constrained to lie along iso-curves of the underlying parametric patch. Motivated by the desire for a fast approximation to layered spline displacement [Bartels and Forsey 1993], Surface Pasting allows NURBS patches to be hierarchically pasted to a base spline patch, with no restrictions on the patch shape or its orientation. To accomplish this, each patch is actually an independent surface - at each node, the underlying surface is trimmed to create a suitable hole, and the patch simply positioned in the hole, with the boundaries optimized so that the composite surface appears as continuous as possible.

The pasted overlay patches are represented using an offset-vector representation, as in H-Splines. The set of overlays is represented as a DAG, and any overlay in the hierarchy can be interactively manipulated, after which overlapping patches are re-applied. The result is a truly procedural and hierarchical representation. All the patches are stored as a tree of polygons in the base NURBS parameterization, which can be arbitrarily re-ordered, and in that sense abstraction and independence are well-satisfied. However, because the pasted surfaces are not actually integrated into a new surface, but rather must be processed independently, pasted patches cannot really be composed in the complexity-hiding sense.

As a Surface Tree, each pasted surface is a node, and as in H-Splines, the parameterization at each input surface is the shared global NURBS parameterization $\mathcal{P}_\mathcal{S} = \mathcal{P}_\mathcal{B}$. For each node, $\mathcal{U}$ is a 2D polygon in this parameterization. The operator E is again a normal-frame-encoded vector displacement of the patch control points, followed by a trimming of the base surface and an optimization of the pasted surface boundary. Note that in Surface Pasting, the operation $(\mathcal{S} \setminus \mathcal{U}) \cup \mathcal{V}$ does not produce a continuous output surface, but rather two disjoint surfaces.

6.5 A Procedural Mesh Data Structure

So far I have presented the major algorithmic aspects of my Surface Tree implementation. In terms of an interactive tool, another major challenge to realizing a Surface Tree editing interface is that the evaluation of each node involves considerable computational expense.

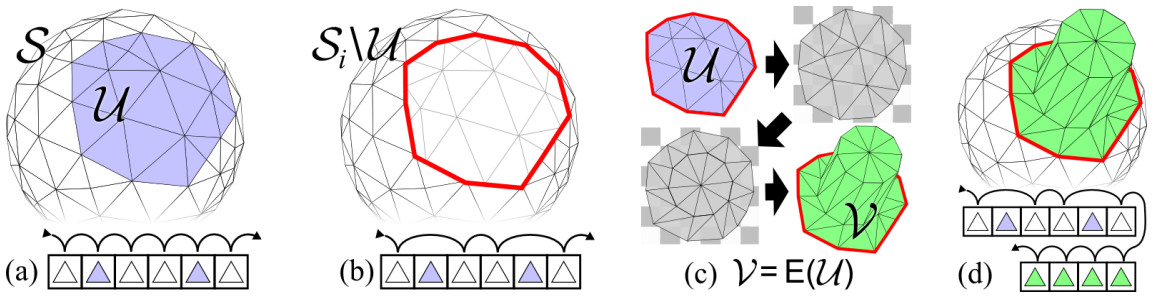


Figure 6.8: *The* Mask *operator creates a hole in* $\mathcal{S}$ *by hiding triangles in the editing region* $\mathcal{U}$ *(a) during mesh iterations (b). Edits* E *generate* $\mathcal{V}$ *by copying and modifying* $\mathcal{U}$ *(c), which often involves mapping to uv-space for re-meshing. Finally,* Weld *synthesizes a manifold mesh by transparently combining* Mask($\mathcal{S}$,$\mathcal{U}$) *and* $\mathcal{V}$ *during iteration (d).*

When a node is modified, its parents are invalidated, so a modification deep in the tree can cause a cascade of re-computation. In this section I will describe how to reduce the cost of this recomputation.

I will focus on triangular meshes. To simplify notation, assume that a mesh is a flat list of triangles. Each node in the Surface Tree then takes an input mesh $\mathcal{S}_{in}$, modifies it, and outputs a new mesh $\mathcal{S}_{out}$. As the surface region $\mathcal{U}$ is defined with respect to $\mathcal{S}_{in}$, each node is dependent on its input mesh, making a full tree update $O(N)$ in the number of nodes. This cost can be reduced by storing all intermediate meshes - then if the designer alters node i, only nodes $j \geq i$ need be evaluated. Unfortunately, the overhead of copying and storing the mesh at each node is overwhelming.

Since tree nodes are locally supported, it is possible to construct a more efficient data structure for representing intermediate meshes. First we define two abstract mesh editing operations - *Mask* and *Weld*. *Mask* simply removes the triangle subset $\mathcal{U} \subset \mathcal{S}$ from $\mathcal{S}$ (Figure 6.8b), and, recalling our previous notation $\mathcal{V} = \mathsf{E}(\mathcal{U})$, *Weld* inserts $\mathcal{V}$ into the hole created by *Mask* (Figure 6.8d):

$$\mathit{Mask}(\mathcal{S},\mathcal{U}) = \mathcal{S} \setminus \mathcal{U} \qquad \mathit{Weld}(\mathcal{S},\mathcal{V}) = \mathcal{S} \ \cup \ \mathcal{V}$$

As previously noted, a procedural edit E must preserve the boundary of $\mathcal{U}$, to avoid introducing "cracks" between $\mathcal{S}$ and $\mathcal{V}$. Hence, assuming that $\mathcal{U}$ and $\mathcal{V}$ have been computed, these operators can be chained together:

$$\mathcal{S}_{out} = \mathit{Weld}(\ \mathit{Mask}(\mathcal{S}_{in},\mathcal{U})\ ,\ \mathcal{V}\ )$$

One way to realize these operators is to access the geometry of any $\mathcal{S}$ through iterators. Then *Mask* and *Weld* can be implemented as iterators which either skip certain triangles (*Mask*) or iterate over multiple meshes (*Weld*). With this approach, a node never modifies $\mathcal{S}_{in}$, but rather generates an iterator which masquerades as a manifold mesh by transparently hiding the triangles in $\mathcal{U}$ and inserting $\mathcal{V}$. Clearly, *Mask* and *Weld* can be applied recursively, creating a procedural mesh data structure.

In practice, our mesh is not simply a list of triangles, but an efficient vertex/edge/face manifold representation. This complicates the implementation of *Weld*, as it must transparently re-write the incoming and outgoing indices of boundary vertices and edges, to

preserve the outward appearance of a manifold mesh. Note that *Mask* and *Weld* can also be applied to point set surfaces, where the implementation is simplified because the explicit boundary re-writing is unnecessary.

Since *Mask* and *Weld* do not copy or modify $\mathcal{S}_{in}$, they are highly efficient even when applied to large meshes. They do, however, add overhead to mesh iterations, which can limit interactivity as the Surface Tree grows. Hence, in practice we have found it useful to occasionally cache a full copy of $\mathcal{S}_{in}$ at a node. This cache simply copies the geometry produced by the incoming *Mask*/*Weld* iterators into a single manifold mesh, which is much more efficient to iterate over. In particular, if the user selects $\mathcal{N}_i$ for editing, we cache the output of $\mathcal{N}_{i-1}$ to ensure that interactive feedback is as fast as possible.

6.6 A Surface Tree Modeling Tool

To explore how Surface Trees could be used in an interactive 3D design tool, I implemented a Surface Tree editor as an extension to the ShapeShop 3D modeling system [Schmidt et al. 2005]. Similar to other sketch-based modeling tools, ShapeShop's workflow involves a mixture of sketch-based and traditional interactions. A suggestion-list interface [Igarashi and Hughes 2001] allows the user to create and modify a variety of edits based on sketched curves. Other parameters are controlled using 2D and 3D widgets.

The Surface Tree tool I will describe here was designed to support "from scratch" modeling, driven by sketch-based interactions. The development of this modeling tool also pre-dates the COILS deformer [Schmidt and Singh 2008]. As a result, the Part shapes created by the various sketch-based editing tools were limited to those that could be represented as vector displacements. I will discuss the integration of COILS-based Parts later in this section.

6.6.1 Sketch-Based Surface Tree Modeling Tools

Each tool in the Surface Tree editor takes one or more user-sketched strokes as input parameters, and then appends a node to the Surface Tree. Within this node, E is a displacement operation that also may involve local remeshing in the DEM parameter space. These editing operations are designed to be procedural themselves, and hence can be computed at arbitrary resolution, allowing high-quality surfaces to be generated.

Sketched Displacement

Displacement mapping [1984] is one of the simplest types of procedural surface manipulation, and simple displacement-painting tools [Lawrence and Funkhouser 2003] have become popular in commercial systems [Pixologic, Inc. 2011]. Although here a sketch-based approach is taken, painted displacements could also be captured and stored in a procedural decal.

The tool supports displacement of arbitrary regions by allowing the user to directly sketch a closed loop on the surface (Figure 6.9a). The displacement function has a variety

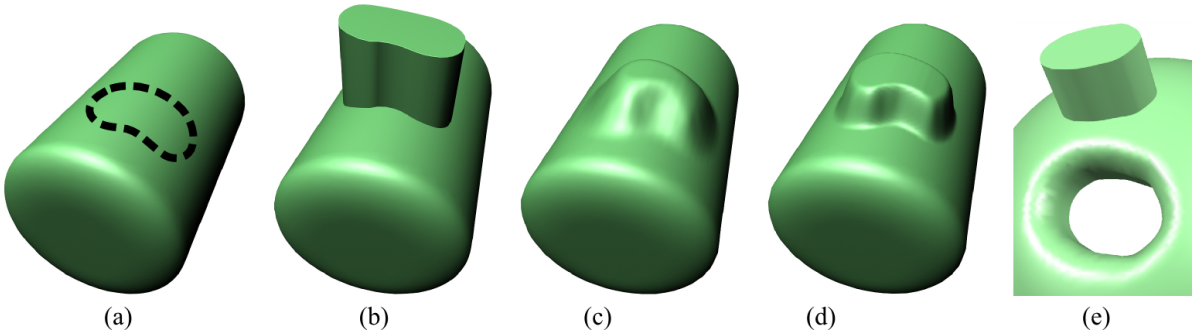


Figure 6.9: *Contours sketched on the surface (a) can be used to create sharp extrusion (b) and soft displacement (c). These edits have various parameters, such as blending radius (d), and can also be inset into the surface (e).*

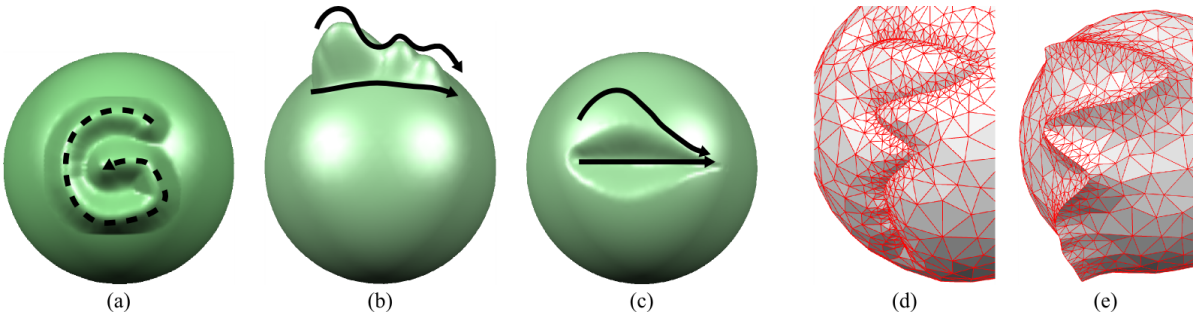


Figure 6.10: *Displacement along a sketched curve on the surface can either have uniform height and width (a), varying height defined by a profile curve (b), and a width which varies with the profile height (c). A sharp crease can be produced by sampling a sketched spline and inserting the resulting polyline directly into the uv-space mesh. A fall-off region blends the crease into the surrounding surface. The crease can be offset (d) into or (e) away from the surface.*

of parameters, including smooth and sharp transition (with the latter being referred to as *extrusion*), displacement height, and blending radius. Boundary edges of extrusions are preserved by inserting the sketched polyline directly into the mesh using constrained Delaunay triangulation [Shewchuk 1996]. To create the soft transitions, a smoothed approximation to the 2D distance field of the boundary contour is generated [Peng et al. 2004; Schmidt et al. 2005], and smoothed distance is then mapped to displacement height.

Also supported is displacement along a user-sketched surface curve. Given a single curve, we create a uniform-width "tube" on the surface (Figure 6.10a). A second curve can be drawn which modulates the displacement height (Figure 6.10b), with an option to tie displacement radius to height (Figure 6.10c). This last option produces a variable-width displacement reminiscent of the area-proportional inflation used in the Teddy system [Igarashi et al. 1999]. These displacements are also defined by a scalar field, generated by accumulating radial 2D fields placed at regular intervals along the sketched curve. The combined field is defined as $\max_i(h_i * max(1-(d_i/r_i)^2)^3, 0))$, where d_i is the distance to the origin of the ith field, r_i is its radius, and h_i the height.

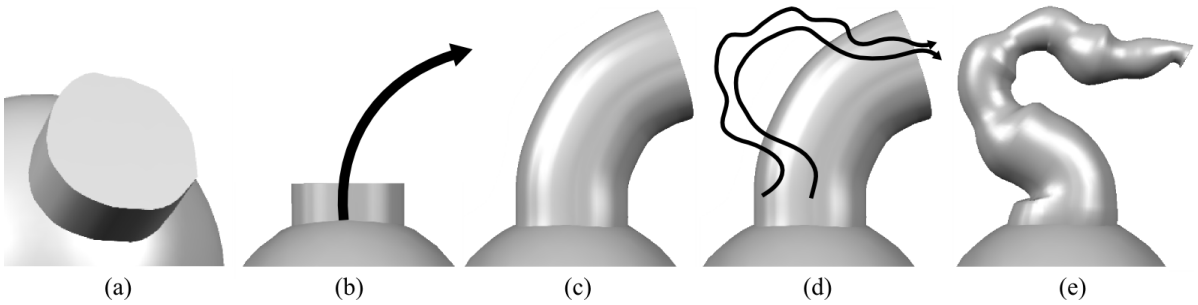


Figure 6.11: *A linear extrusion (a) can be converted to a path extrusion by sketching the path beginning on the extrusion surface (b). The extrusion profile is then swept along the path (c). The extrusion can be over-sketched to modify the path, and an optional profile curve (d) can also be added to describe an arbitrary procedural generalized cylinder (e).*

Curves can also be used to insert sharp creases into the mesh. To do so, we remove the square on the $(d/r)^2$ term in the equation above, producing a field with a sharp, "inverted" falloff region. The crease is defined by a spline curve, but sampled into a poly-line which is inserted directly into the mesh, again using Delaunay triangulation in parameter space. This ensures that a sharp edge is produced (Figure 6.10d,e).

Generalized Displacement

Although the techniques described above provide a wide range of modeling functionality, they only displace the surface along linear vectors (surface normal, constant vector, and so on). More powerful procedural editing can be formulated in terms of displacement along 3D curves.

In the Surface Tree modeler, a linear extrusion can be extended along a curved path by drawing the path and selecting the resulting suggestion icon. The path can be edited by oversketching from the current viewpoint. A tapering parameter allows the user to linearly vary the scale of the 2D template polygon along the length of the extrusion. A second curve can optionally be drawn alongside the first to define a profile function, turning a simple extrusion into a flexible generalized cylinder (Figure 6.11).

Holes and Handles

As I have previously discussed, the Surface Tree can support the creation of topological holes and handles. In the editing tool I have implemented support for creating hole and handle PARTs that are isomorphic to cylinders, ie with two disjoint boundary loops. This involves two separate decals, each of which create an opening in the manifold (Figure 6.12b-c). These 2D holes are then connected together with a generalized cylinder, resulting in a 3D topological hole or handle (Figure 6.12d-e). Unlike volumetric approaches to topological modification, which require consistent inside/outside spatial partitioning, this manifold-stitching approach can also be applied to surface patches with boundaries (Figure 6.12f).

The hole/handle creation interface is similar to path extrusion - a linear extrusion is

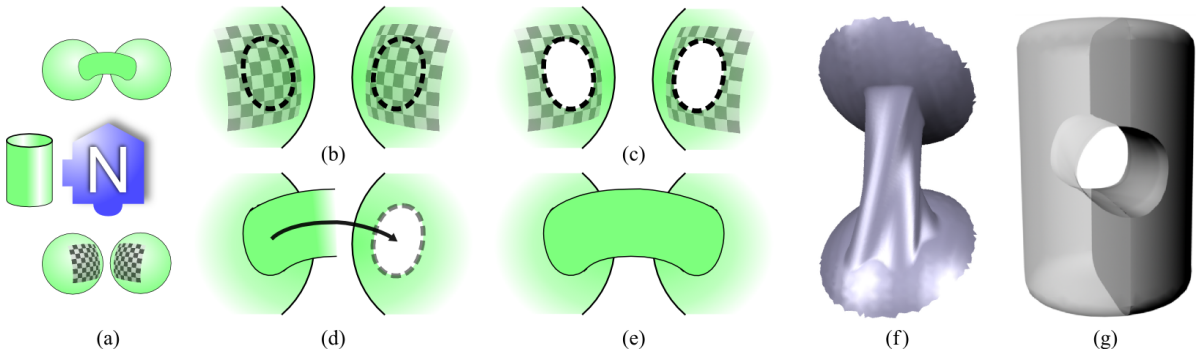


Figure 6.12: *A topogical hole or handle node (a) takes two parameterized regions as input. Within these parameterized regions are the boundary curves (b), which are used to cut holes in the mesh (c). A generalized cylinder then attaches one side of the handle to the other (d-e). Each side of the handle can have a different profile curve (f). A topological hole (g) is simply a handle that appears to be "inside" the surface.*

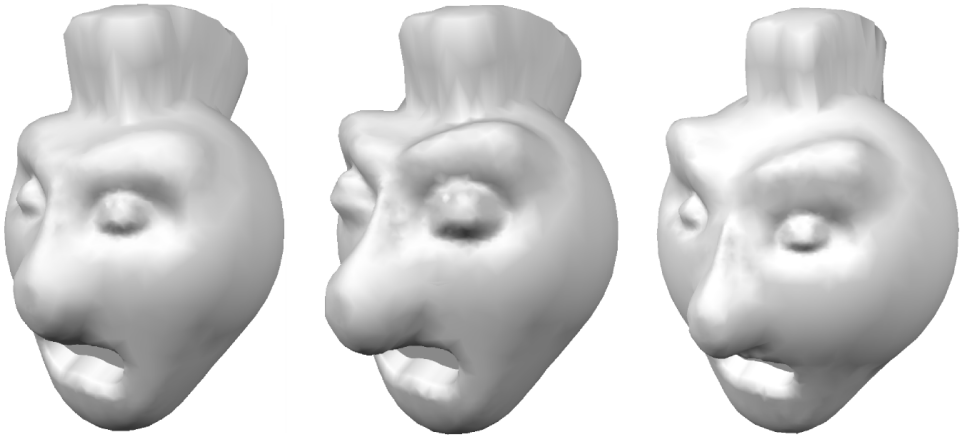


Figure 6.13: *The model on the left was created by adding features to an initial sphere. In the next two images, these procedural* PARTs *are independently manipulated.*

first created, and then a line drawn from it to another point on the surface. Selecting the resulting suggestion icon creates a hole or handle with the same contour at both ends. Alternately, the user can draw a second contour, in which case we interpolate between the two along the handle path (Figure 6.12f).

6.6.2 Procedural Interactions

As the artist creates features using the procedural tools described above, nodes are transparently added to a Surface Tree, which effectively catalogs the construction history of the model. The main reason to build and store this tree is to support the types of operations described in Section 3.3, such structured manipulation.

Some basic structured manipulations are shown in Figure 6.13. This face model was created by adding a series of PARTs to an initial sphere. As the PARTs are represented

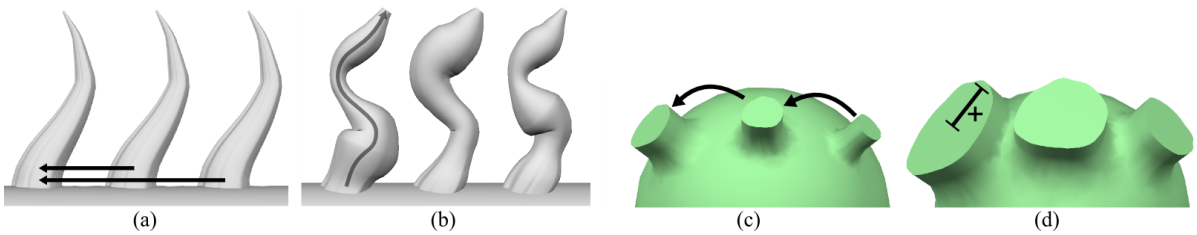


Figure 6.14: *In (a), two copies of a curve extrusion are linked to the original, such that even if their profile curves are specified independently, a change to the original path propagates to the copies (b). In (c) the copies are sequentially linked, with the radius functionally defined to be half the size of the parent (d).*

procedurally, we can manipulate them independently and arbitrarily. In the middle image the eye, nose, and brow PARTs are exaggerated in scale, while in right image the underlying sphere is made smaller. In this latter case, all of the overlapping PARTs must be procedurally recomputed. In a traditional global mesh representation, it would simply not be possible to express this change without essentially rebuilding the entire model. With a Surface Tree, it takes only a few seconds.

Once a Surface Tree structure structure is available, we can easily implement higher-level operations like copy-and-paste of PART nodes, even if the desired PARTs lie "underneath" overlapping layers. In contrast to the mesh drag-and-drop tool I previously presented, these copies become new Surface Tree nodes. Even more significant is that these copied nodes can be *linked*, such that when the parameters of one node are modified, linked copies are automatically updated (Figure 6.14). This linking can be selective, such that some parameters are linked but not others.

Procedural manipulation of the Surface Tree introduces a new set of visualization and interaction problems that have yet to be explored. In my tool, the user can select an existing node for manipulation by clicking on its support region. If multiple nodes lie under the cursor, repeated clicking cycles through them. The selected node is highlighted, and all overlapping layers are rendered transparently (Figure 6.15a-b). This gives the user some sense of the Surface Tree structure, but the precise ordering and branch structure is still opaque. An independent tree view could help, but it would be more desirable if the structure could be directly visualized in the 3D view. In other work I have made some progress towards visualizing overlapping internal structures in 3D models, by rendering solid model PARTs in a "sketchy" style [Schmidt et al. 2007]. These *visual scaffolds* should be adaptable to the overlapping hierarchical PARTs in Surface Trees.

As I previously discussed, the computational cost of recomputing the output of a Surface Tree increases with the depth of the node being modified. To provide interactive feedback, overlapping nodes must be dynamically re-evaluated and rendered as the user manipulates a selected node. To guarantee rapid feedback, in my interactive modeling tool I limited re-evaluation of overlapping layers. Essentially, at each frame a timer is started, and tree re-evaluation is halted when a quarter of a second has elapsed. The full tree update is deferred until the user completes the manipulation (Figure 6.15c-d). I do note that this can be problematic if one wishes to make changes relative to other visual

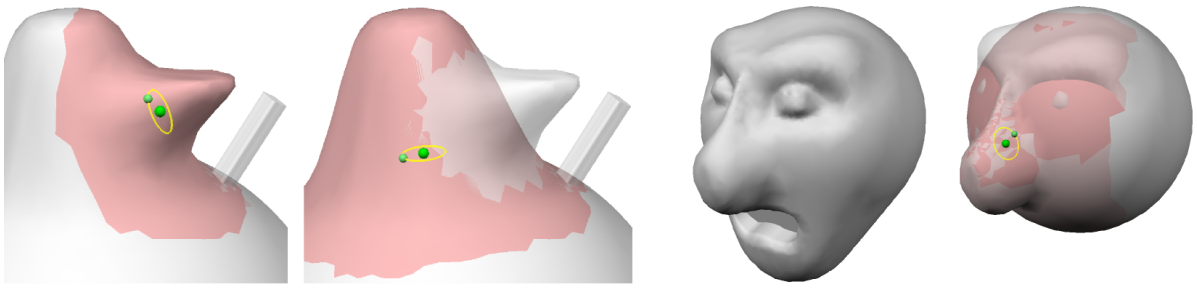


Figure 6.15: *The support region of an edit node is highlighted when it is selected (a), and all overlapping layers are rendered as transparent (b). Computation of layers in (c) can be deferred as the user manipulates an underlying layer (d) to maintain a minimum interactive frame-rate.*

elements, which may not be visible once the manipulation begins.

6.6.3 Results and Discussion

Figure 6.16 displays several completely procedural surface models created with the Surface Tree editor I have described in this section. It is important to note that in this tool, PARTs are created using simple sketch-based operations that can only express limited geometric complexity. Hence, we cannot expect the model complexity to reach that of the results created by dragging-and-dropping existing mesh PARTs. However, each feature of these surfaces is a procedural PART and hence can be independently manipulated.

One may note that in the discussion of this system, I have neglected to mention secondary branches. This is due to limitations of the ShapeShop interface, which does not have support for editing independent PART assemblies. Hence, each Surface Tree tool simply appends a node to a sequential list. I did perform one simple experiment to demonstrate that secondary branches were possible. If the current tree output has a boundary, it can be encapsulated as a secondary branch and inserted back into itself using a local geometric texturing operation. An example is shown in Figure 6.17. This subtree is not a copy, but rather a procedural instancing, so changes to the original portion also propagate to the duplicate.

One limitation of the system is that the response time of the system varies depending on which edit is being manipulated, as the computational cost of a tree update depends on the layer depth of the modified node. The cost of updating a node is highly variable - major factors include the resolution of the input mesh, the size of the node support region, and the level of refinement necessary to faithfully represent the edit. Generally, the 3-5 edits at the top of the tree are interactive at moderate resolutions. Beyond that depth, the partial-update scheme described above (Figure 6.15) comes into effect, reducing visual fidelity.

As I noted above, the Surface Tree implementation in ShapeShop is based on displacement-style edits. I have also developed a stand-alone Surface Tree viewer, which can load a sequence of arbitrarily-complex layered PARTs and attach them using COILS-based de-

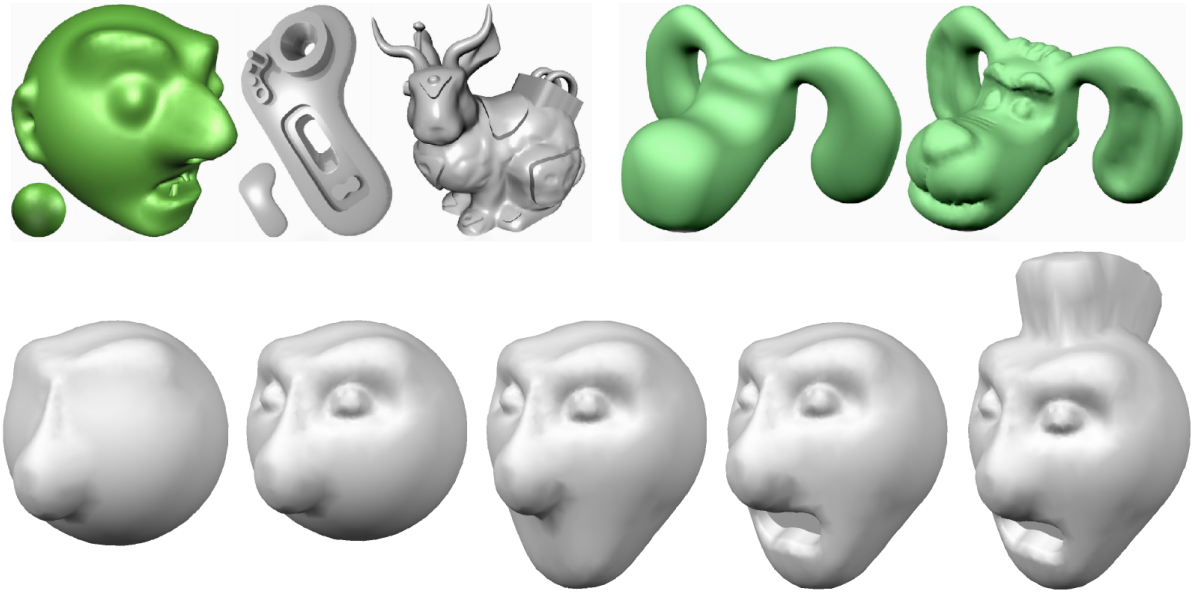


Figure 6.16: *Results created with our Surface Tree-based 3D modeling tool. The top row shows several completed examples, while the bottom row demonstrates the creation sequence for a simple head model.*

formation nodes. A basic example is shown in Figure 6.18, where I have used the drag-and-drop tool (Section 5.9) to "reverse-engineer" several PARTs of the Stanford Bunny. Once these parts are layered in a Surface Tree, they can be independently manipulated. It would be straightforward to allow other PARTs to be imported via drag-and-drop, although this has not been implemented.

6.7 Conclusion

In this chapter I have constructed a hierarchical, procedural representation of a 3D surface, the Surface Tree. This representation has the potential to greatly increase the power of surface modeling tools available to designers, allowing them to "go back in time" and non-destructively modify any modeling decisions made in the past. In contrast to a static surface mesh, this procedural hierarchy makes iterative design much more efficient. Designers can easily explore variations without having to discard existing work.

After developing both the Surface Tree data structures and a strategy for implementing them, I described a prototype Surface Tree modeling tool. Although my current system has many limitations, it does allow one to create a 3D model which is completely procedural - each surface modification can be interactively modified, and the rest of the tree is updated without user intervention. Sketched holes and handles enable procedural topology change without resorting to a volumetric approach, and procedurally-linked nodes allow for complex dynamic surface modifications to be controlled by simple parameters.

The main practical limitation of the Surface Tree is that it is very computationally

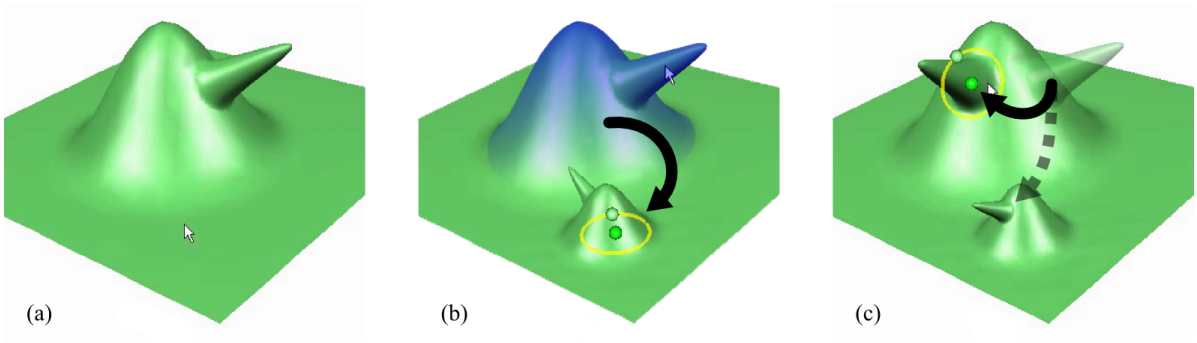


Figure 6.17: *The three-node Surface Tree in (a) contains a base plane and two layered bumps. This entire tree can be instanced as a secondary branch, which is then (b) appended onto the main tree. This instancing is "live", so (c) modifying any of the original nodes propagates to the copy.*

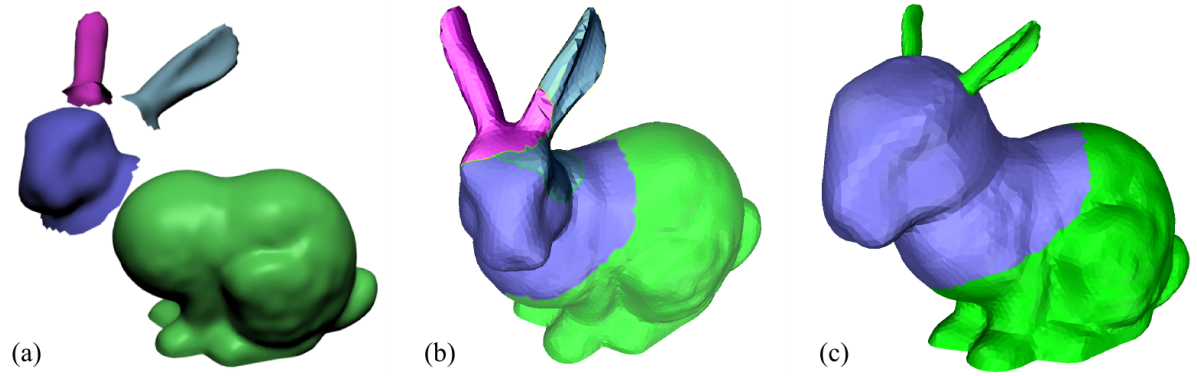


Figure 6.18: *A surface is segmented into a set of* PARTS*(a), which are then layered in Surface Tree Viewer (b). The tree nodes can be manipulated to make a new model (c). Here the head has been rotated and scaled up, and the ears scaled down and repositioned.*

intensive. As in complex solid models, "rebuilds" of significant portions of the tree could easily take minutes or even hours. The cost of tree updates is exacerbated by the dependency structure of the tree. If node A is an input to node B, it will *always* be applied before B. Hence, evaluation of the primary branch is inherently sequential. However, if the input regions of A and B are disjoint, then B is not actually dependent on A, and they could be computed in parallel. For example, the ears on a head model are unlikely to interfere with eachother.

One way to achieve this parallel computation would be to modify the interface, so that the user can create *layers* of nodes that cannot overlap. The actual Surface Tree could then be automatically generated from the layer structure. Artists may find this sort of layer metaphor more intuitive, as it is very similar to the layering components of vector-graphic and image editing tools. In this layer-based approach, node anchoring could also be made more explicit, which would also make the model structure more transparent.

Another alternative to enable parallel node evaluation would be to dynamically partition the input surface as the tree is created. The partitions would only need to be

integrated if some later node overlapped a partition boundary. Determining the partitioning could be quite expensive, although as it would not actually change the visible surface, it could be done in idle time.

Another interesting problem is whether a sequence of edits can be efficiently collapsed into a single node without losing too much fidelity. For example, if a sequence of nodes all lie within some bounded region, and we wish to modify the underlying mesh, then it is possible to pre-compute the difference between the output of this set of nodes and the underlying patch. We could then approximate the output of these nodes by rewriting them as a single node which deforms the pre-computed surface using the COILS deformer. This is essentially a dynamic cache node, such as those proposed for use in hierarchical implicit modeling [Schmidt et al. 2005]. As in that case, some visual fidelity would be lost, but from the user point-of-view it would likely be a reasonable trade-off if the same order-of-magnitude speedup were to result.

Chapter 7

Evaluation

Some material in this chapter is derived from the article "Drag, Drop, and Clone: An Interactive Interface for Surface Composition" authored by Ryan Schmidt and Karan Singh [Schmidt and Singh 2010c]. The text and images reproduced here are used with permission from my co-author.

7.1 Introduction

In the preceding chapters I have developed a hierarchical, procedural PART-based surface representation, and applied this infrastructure in three novel artist-oriented 3D design interfaces. In each chapter I considered the mathematical properties of each novel algorithm or technique, and compared its merits relative to existing works. However, in the context of this thesis, the larger question is whether PART-based modeling could enhance interactive 3D surface design workflows. Essentially, the pertinent question here whether or not the PART-based approach is *useful*.

An obvious approach to demonstrating the utility of PART-based interactions would be through some sort of "user study", as is used when evaluating new interface widgets in the HCI community. After extensive consideration, I have come to the conclusion that these sorts of very narrow controlled experiments are not appropriate for the interfaces I have proposed. The cornerstone of such user studies is the carefully-measured and controlled variation of a small set of parameters. However, the novel modeling tools I have described are still prototypes, involving many complex interactions and user interface elements that combine to provide a compelling modeling experience across a wide range of tasks. Results obtained by a reduction to a simplified, measurable task cannot be assumed to generalize to this overall experience. In addition, comparisons to other modeling tools would ultimately be apples-to-oranges; no interfaces with even remotely similar capabilities is currently available.

So how are we to determine whether or not the tools I have developed can be applied to any useful effect in 3D design? In the field of HCI, Greenberg and Buxton [Greenberg and Buxton 2008] have considered the difficulty of evaluating novel interface prototypes, and proposed some alternatives to user studies. They note that in design fields, *design critiques* are a widely accepted way for designers to evaluate eachothers work. Essentially,

this approach involves collecting the opinions of experts. As I will explain below, I have collected a wide range of feedback on the utility of PART-based modeling, focusing on the simple drag-and-drop PART composition of Chapter 5.

Another important higher-level issue that has yet to be discussed is the *representational capability* of my PART definition. The approach I have presented cannot represent *any* PART. Later in this chapter I will consider what classes of surfaces work well as PARTs in my system, and what classes do not.

7.2 Utility of Part-Based Modeling

As I have previously explained, there are theoretical and philosophical reasons for devising PART-based surface representations. It is somehow aesthetically incongruous that we can talk about the PARTs of a surface, but not be able to turn our words into a concrete mathematical form. But is there a practical reason to do so? In Chapter 3 I made the case that there ought to be, by considering some of the current limitations of surface modeling when compared to solid modeling. But claiming that artists *should* want PART-based interaction is not the same as showing that they *do*.

To explore whether artists would actually find my tools useful, I released each of them publicly on the internet, free of charge. The decal-based texturing tool (Section 4.6) and sketch-based Surface Tree modeling tool (Section 6.6) were integrated into the ShapeShop modeling system [Schmidt et al. 2005]. The PART drag-and-drop interface of Section 5.9 was released as part of stand-alone tool called *meshmixer*.

Although the decal texturing and surface tree editors were released much earlier (2006 and 2008, respectively), they remain largely undocumented and unsupported. As a result, there is little I can report about these techniques. I have been contacted by 3D artists who have seen videos of the tools and wish to use them, but to date I have received no significant feedback from these users. The most limiting factor is that neither of these tools can be integrated into an artist's daily workflow. The Surface Tree models one can create within ShapeShop must be "baked" into a static mesh to be exported, at which point all the benefits of PART-based design are lost. Decal texturing has a similar problem - since no other tool supports this type of texture representation, there is again no "downstream support" in professional graphics pipelines.

Meshmixer, on the other hand, is designed to work with existing meshes, of which there is a vast abundance. Similar "easy-to-use" tools for re-using these models simply do not exist. As a result, meshmixer was widely publicized in the graphics industry press, blogs, and message boards, has been downloaded thousands of times, and appears to be in active use on a daily basis. This broad distribution has generated a large volume of user feedback, which can inform future work on PART-based interaction.

Before I begin, I must mention two important caveats about artist feedback. The first is that most artists spend the majority of their time working with a small set tools with highly similar, almost standardized interfaces. This appears to lead to a natural tendency to evaluate new tools solely with respect to their familiar workflows. Second, because industry workflows are also so regimented, novel modeling prototypes generally cannot be used in day-to-day work, and hence most judgments (both positive and negative) are

heavily colored by early experiences. The reader should keep this in mind when reading the following section.

7.2.1 meshmixer Distribution and Usage

I distributed a basic version of my mesh drag-and-drop tool *meshmixer* (Section 5.9) to a wide community of computer graphics hobbyists and professionals, by making it freely available on a public website (http://www.meshmixer.com). This distribution began in early December of 2009 and is currently ongoing. Initially minimal instructional material was provided (a 2-minute demonstration video and a 4-minute tutorial video with text captions) An update to the software (meshmixer02) was released in March 2010, and at the same time a set of tutorials was added to the website. Although I offered technical support via e-mail and an online forum, most requests involved advanced uses of our tool or software bugs. I have yet to encounter a user who was confused about the basic operation of the drag-and-drop interface.

I will now discuss some usage statistics on meshmixer and its website. This data was collected using Google Analytics [Google Inc. 2010]. At the time of writing (August 1st, 2010) the meshmixer website has received approximately 40,000 visits, of which 68% were "new" visitors who had not previously viewed the website[1].

The combined total number of downloads for the two released versions from the meshmixer website is 4,528 (3,582 for meshmixer01 and 946 for meshmixer02). However, these numbers only count clicks on the "download" page, not direct links to the installer executables (which have also been mirrored by several sources).

To communicate with potential meshmixer users, a feature was added to the tool which downloads a "news" webpage from the meshmixer website each time it is run. The purpose of this page is to inform users about new versions, tutorials, and so on, but also has the serendipitous effect of allowing me to determine how often the meshmixer binary is executed[2]. To date, this page has been downloaded 7,557 times. More importantly, though, 5,448 (72%) of these downloads are from "returning visitors". This statistic implies that the majority of executions come from users who have executed the software more than once.

The *returning visitors* statistic is significant, because it is often the case that artists who download this type of prototype tool will run it only once, to "try it out" before deciding that it is not worth investing the time to learn. That a majority of executions are by repeat users suggest that at least some are finding meshmixer useful. It should be noted, however, that my web-statistics methodology cannot differentiate between whether a very small percentage of downloaders are very frequent users, or whether a larger pool are occasional users. Hence, the size and proportion of this returning-user pool remains unknown.

[1] Google Analytics uses "cookies" to track website traffic, and it is generally the case that cookies are not shared between web browsers or accounts on different systems. So it is possible for a unique individual to be counted as a visitor multiple times.

[2] This webpage can only be accessed if user is connected to the internet, so the statistics here are conservative.

Month	Downloads	Executions	Repeat Executions (%)
April	194	569	62%
May	166	460	74%
June	283	802	72%
July	284	880	75%

Table 7.1: Meshmixer download and usage statistics for four months in 2010

As I mentioned, meshmixer received a large amount of initial publicity, which has naturally tapered off over time. So it is reasonable to wonder whether the above statistics still hold. In Table 7.1 I provide some statistics for four recent months in 2010 We can draw two important conclusions from this data. First, repeat use of meshmixer remains relatively consistent, and second, the number of executions is significantly larger than the number of downloads. This again indicates that artists are indeed using the tool.

7.2.2 Artist Feedback

The release of meshmixer generated a considerable amount of public discussion in online message boards, blogs, and industry news websites. As of writing (August 1, 2010), a Google query for the term "meshmixer" returns 19,500 results. I browsed the first 200 of these results, which mostly are web forum discussions, to get a sense for how the meshmixer interface is viewed by artists. I note that over 95% of these 200 pages do actually pertain to my meshmixer project, and that at least half are in languages other than English (and hence were ignored). Many of the pages include a single forum post, but several contain extensive discussion of meshmixer. In this section I will discuss this body of feedback, first by attempting to extract some overall themes from the data, and then by examining specific comments.

Analysis of Forum Comments

Attempting to draw conclusions from written human feedback is largely a subjective task. Grounded Theory Methods (GTM) [Glaser and Strauss 1967] are widely accepted in the social sciences as a method of making subjective data exploration more rigorous. GTM are not meant to test any specific hypothesis, but rather are a set of practices that are meant to help a researcher construct theories from the data. These theories are then said to *emerge from the data*, and be *grounded in the data*.

Grounded Theory Methods are complex, and there are various schools of thought about when and how they should be used. Since I am not really attempting to formulate a theory of how artists feel about meshmixer, but simply provide the reader with a sense of the data, a full GTM process seems unnecessary. However, the early stages of GTM involve *coding*, in which the data is repeatedly abstracted until key concepts emerge [Muller and Kogan 2010]. Coding is relatively accessible even to GTM novices (such as the author), and it is this process that I have attempted to apply.

My data set consists of 131 comments, extracted from web forums and comment threads. These comments range from a single word ("Brilliant.") to short paragraphs

(100 words). Because the data set is relatively small, for the initial *open coding* step, I simply extracted all the words and phrases that seemed to be possibly relevant to each authors' opinion or experience of meshmixer. After grouping very similar words and phrases, I was left with 69 codes. The *axial coding* step then involves grouping these codes into higher-level categories. In this step I found 9 main themes, shown in the table below and ordered by frequency

Axial Code	Example Words and Phrases
General Impression	cool, awesome, great, wow, excellent, gorgeous, insane
Purpose	combine meshes, merging, mash up, assembling, library of parts
Novelty	innovative, interesting, unique, brilliant, amazing, nice concept
Utility	useful, not useful, practical, handy, save time
Usability	easy, simple, intuitive, usable, interactive, crashed
Topology	topology, quads, triangulation, edge flow, UVs, clean-up
Integration	workflow, integrate, sculptris, blender, ZBrush, 3DSMax
Enjoyment	fun, hilarious
Cost	free, freeware

The most frequently occurring axial code is the *General Impression*, where the comment author simply presents an opinion about meshmixer. These comments were overwhelmingly positive - overall, 76% of the comments were explicitly positive, and 64 specifically used terms like those mentioned in the table ("cool", "awesome", and so on). Of the remaining 24%, 15 had some negative or critical content, and the remainder were mainly explanatory (describing how to rotate the camera, for example).

The rest of the codes are relatively straightforward, and in some sense confirm my belief that meshmixer is a new, useful, and usable tool. I will consider some of these in more detail in the next section. One question was whether artists would "get" the motivation for the tool. However, the frequency of the *Purpose* code made it clear that artists could immediately see what meshmixer was meant to do. Most comments of this type were explanatory, for example the initial post in a forum thread that introduced meshmixer, which would then be followed by other discussion.

As negative or critical feedback was relatively infrequent, I have given these comments particular attention, both for for this evaluation and as a means to learn how to improve the tool. Of these few comments, the main focus was on topological issues, which will be discussed in depth in the following section. Software quality issues were also mentioned several times, focusing on crashes and the handling of very large meshes. The main other problem that some commenters had with meshmixer was that the "cut-and-paste" modeling capabilities of meshmixer may negatively impact the creativity of artists, who presumably could be pressured by managers to copy or re-use existing models instead of creating new ones.

Major Themes in the Forum Comments

In the following I will quote from these unsolicited public comments to give a glimpse into the public perception of meshmixer. Note that most internet forum posts do not

have exemplary spelling or grammar. Below I have reproduced the comments verbatim, except where noted.

In concert with the meshmixer software release, a demonstration video was posted on the video-sharing website YouTube[3]. In many cases, comments were based on this video alone. A frequent reaction was one of surprise and disbelief:

- "I don't believe what I just saw! is this thing for real?"
- "This looks to good to be true, I just have to try this out."
- "This is awesome! I can't believe I just saw it! Such a clever idea, wonder why no one had ever come up with it."

This reaction is encouraging because it confirmed our belief that this style of interactive, unconstrained PART drag-and-drop is not only a highly novel interaction, but also one that would appeal to artists. More encouraging was that those who appeared to have downloaded and tried the software remained largely positive:

- "Will have to play with this more... it seems to work quite well."
- "Very cool! It works a lot better than I've expected."
- "Meshmixer is pretty fun. (...) My kids have fun with it too. They can make all kinds of monsters and creatures."

I should note that this latter type of response to the actual usage of prototype modeling tools is the exact opposite of what one typically encounters.

Many authors commented on the usability of meshmixer:

- "This brings us a couple steps closer to a 3D app that our mothers (iow: anyone) can use"
- "What makes this amazing to me is how easy you can just 'drag' an object over another object, and it'll follow the shape wherever it can."
- "Its so freaking EASY. Just downloaded it, exported a missilelauncher from blender, a little getting used to the controls and voila, seamlessly integrated missilebunny"
- "Mr. Potato Head for the 3D printing generation has now officially become trivial."

These statements support the PART-based abstraction that meshmixer is based on. Artists clearly recognized the usability benefits of treating the PARTs as discrete objects and insulating the user from the myriad algorithmic details of the drag-and-drop interaction.

Perhaps the greatest compliment paid to meshmixer is that many 3D practitioners who tested the software expressed a strong desire for the drag-and-drop technique to be integrated into commercial tools:

[3] http://www.youtube.com

- "Would be awesome if someone could integrate such functionality in the upcoming blender 2.6"
- "This would open up a lot of new possibilities if it would be integrated into something like zbrush"
- "If you implement this stuff into 3ds Max... I'd kiss you. Like, seriously."

Many posts speculated about how meshmixer could be applied to modeling problems and integrated into existing workflows:

- "seem great idea's to use in a sculpt modeling workflow."
- "This looks like a very cool solution for those that like to save time by kit-bashing."[4]
- "This is brilliant for organic modeling"
- "That would be a great tool for police sketch artists"
- "I can still see this being INCREDIBLY useful for pre-viz and pre-production work, storyboarding, animatics etc."
- "Looks like something to release the inner surrealist in any 3D artist. Something the Salvador Dalis and H.R. Geigers of digital media would like to have in their toolbox."

What is interesting here is the relatively broad range of potential uses, given that meshmixer is essentially a demo of a single interaction technique. This agrees with my proposition that meshmixer's style of interactive Part-based composition fills a gap in current modeling tools. Several authors made explicit comments to this effect:

- "I can only imagine how much work that MeshMixer would have saved me over the years."
- "Saves a lot of welding."
- "not only toys, but with some premade 3D-data you can realize complex objects that are hard to create in conventional CAD-programs"
- "this is like what i always wanted booleans to do!"

The last comment is particularly revealing. Boolean operations are available in many SubD modeling and sculpting tools, such as Maya and ZBrush, and some authors seemed to think that meshmixer was simply another Boolean-based tool ("...a very good implementation of a boolean union operation.") However, artists such as the one above recognized that the composition approach in meshmixer is tailored to surfaces. The implication here is that while surface modelers clearly desired Boolean-like effects, actual

[4]The term kit-bashing refers to the practice of assembling a model by combining existing pieces.

solid modeling operators are unsatisfactory, presumably because they don't take the surface shape into account. Composition tools which respect the nature of surface PARTs will likely be more widely accepted.

It seems clear that one of the most immediate benefits of supporting PARTs in modeling tools is the ability to easily mix-and-match from existing models. A variety of posts anticipated these PART-based compositional workflows:

- "Make many little pieces in Sculptris, join in Meshmixer, then back to Sculptris for reshaping, smoothing, reduction and UVing. Then to Carrara for rigging, if necessary, and rendering."

- "So (in some time) all you need is a lot of model libraries to mix and match as you see fit. You know, grab an eye here, a nose there, a couple of fingers and a head, place it on a kangaroo body and you are set to go."

Although some lamented how this might change the practice of modeling:

- "...seems like modelling from scratch might become a lost art with software like that!"

I previously mentioned the video game *Spore*, in which players assemble their own characters using a simple PART-based modeling tool [Eletronic Arts Inc. 2010]. Several posts noted the similarity between meshmixer and Spore's creature creator:

- "Didn't know about Meshmixer. It's really fun, reminds me the assembling creature stuff you can do in Spore"

- "I've wondered why this wasn't a modeling tool ever since I saw something along the lines of this in Spore."

Note that to provide their PART-based interface, the developers of Spore manually designed each PART to ensure compatibility. As these comments make clear, meshmixer provides a similar modeling experience to the artist, but without requiring authored PARTs.

Excluding installation issues, crashes, and poorly-documented user interface controls, the most frequent complaint that was raised in discussions among 3D artists was about mesh topology. In current practice, professional artists rely almost exclusively on carefully designed quad meshes for production models. As noted below, these quad meshes contain carefully designed *edge loops* which support high-quality re-posing and animation. The regions where meshmixer introduces triangles to fill holes or stitch boundaries must be manually *re-topologized*. Some artists felt that this was a major limitation:

- "the resulting mesh needs a LOT of cleanup to work in a sub-d surface pipeline which requires close attention to topology to avoid nasty surface artifacts when rendered, not to mention the extra problems bad topology causes at UV mapping time and certainly animation time, as edge flow has a very strong affect on the way a mesh deforms when animated."

- "it's an interesting idea....but I'd be wary of thinking this is as useful as it seems...doing this kind of cut+paste is easy - it's the topology of the result that is important..."

This quad-topology question has come up repeatedly, and does appear to be the most pressing issue for artists and designers who may wish to use meshmixer. However, in the same threads where the above comments were made, rebuttals suggested that this topology issue may be side-stepped by larger developments in production modeling pipelines:

- "Topology is now a step in the pipeline and is sorted out later once sculpting and form is completed. It is also probably a mistake to think this is easy as I've never seen it implemented so simply."

- "Topology can be taken care of at a later date which it seems is how many artists work these days anyway. They mesh out an idea, form and basic sculpt then retopo is the next step. Adding this into the equation is no different."

- "The topology is only an issue if you plan on using the cut and paste live in an animation. If you just want to work out a concept, topology can be figured out later."

- "there's enough re-topo tools around that it's not a huge issue."

These comments are important because the need to preserve carefully-designed global mesh topology is largely at odds with the methods I have proposed in this thesis. Taking a Part-based approach implicitly means that the artist will work at higher levels of abstraction than mesh edges and vertices. The above statements indicate that even though global topology is clearly a concern in current practice, it will not be a major issue in the future. One user appears to have confirmed this supposition:

- "Put the idea in practice. Took an existing sculpt of mine, removed the head in Mesh Mixer, removed the toes in Sculptris, and finally generated the topology in 3Dcoat. Took a little trial and error but eventually I got the results I was after."

Although many of the above comments are highly promising and supportive of my Part-based approach, most are speculative in nature. As I noted, the majority of artist feedback is based on initial impressions, and in some cases the author of the post has clearly only viewed video demonstrations. Hence, a question which remains unanswered is the long-term utility of meshmixer. I have found several posts similar to the following:

- "I tried Mesh Mixer a long time ago and it's a nice enough tool but after the initial 'this is fun' it's not something I go back to at all really"

These comment suggest that some users have either not been able to integrate meshmixer into their workflows, or have had no desire to do so. One possible reason is the mesh topology issue described above.

To date I have little evidence of long-term use of meshmixer. The usage data I provided above does appear to indicate that users are working with the software, although there is no way to determine if this repeated use is maintained over time for individual

users. And I should mention the standard caveat that meshmixer is a research prototype, prone to crashes and with limited support for the million-poly meshes that many working artists deal with on a daily basis. Still, the statements I have quoted here strongly indicate that the style of Part-based interaction in meshmixer fills a void in the current modeling landscape. If not meshmixer itself, than some tool like it would clearly be of significant value to professional modelers.

I will close with an excerpt from detailed feedback provided by a modeling industry expert with extensive experience, who had this to say about meshmixer:

> "[This] is one of the most interesting research software applications we have come across in many years. Ultimately creating good 3D models is actually an artistic and design process not a scientific and mathematical process. This is the only application that we have ever seen where we can just load up parts and add them together interactively. It gives the artist a freedom that has not existed to date ... and returns a very enjoyable creative spontaneity."

Limitations of Artist Feedback Methodology

In the above evaluation I have taken the approach of collecting largely unsolicited and unstructured feedback from a wide range of potential users. I found this approach preferable to more structured feedback such as a user survey because it inherently avoids the possibility of unintentionally asking the artist to "tell me what I want to hear" (ie the *good subject* phenomenon). It also makes it possible to collect feedback on a broader range of topics, including those I had not anticipated.

A valid critique of the unsolicited feedback methodology, however, is that large categories of potential users remain unrepresented. First, there is the set of users who would generally concur with the feedback I have presented above, but are not prone to commenting on internet forums or directly contacting software authors. We can only assume that the contributions of these artists would not significantly change the distribution of comments as provided above. More significantly, there are the artists who found either the video presentation or software tool to be so fundamentally unapealing as to not warrant the provision of any feedback at all. This "non-user" population can only be explored in a more structured environment, where sufficient random sampling will ensure that such users are represented in their statistical proportion.

One potential way to explore this user base outside of a controlled environment would be to further instrument the meshmixer software. For example, if the user opens meshmixer, tests it for a few minutes, and then closes it and does not return to it for several weeks, we could have polled the user with a questionnaire about their experiences. Presumably some of these users would be altruistic enough to complete such a survey, but if not, some sort of reward could be provided. Note that this tracking of session lengths and frequency would also be useful to further map the usage of the tool by those who do find it useful or compelling.

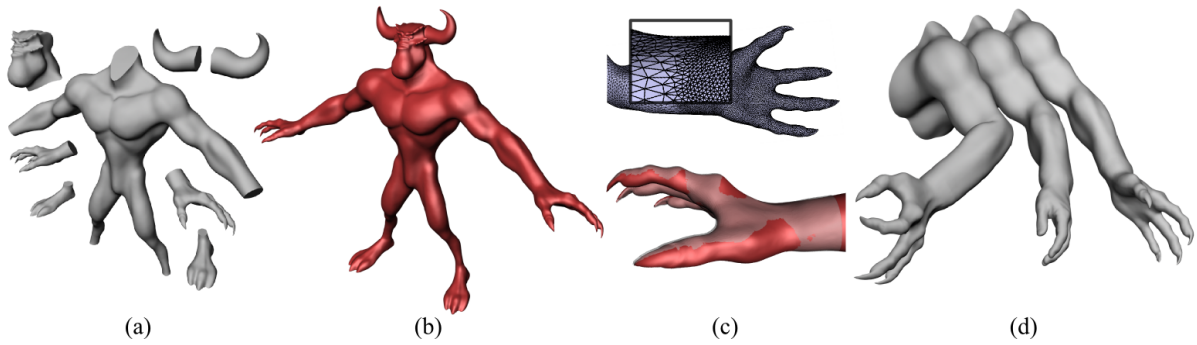


Figure 7.1: *We assembled 8 high-resolution components (a) into a manifold surface (b), spending 5-10 minutes per part. Stitching between varying levels of detail was handled automatically (c,top), and relatively accurate, rigid alignment with original* PARTs *(purple) was achievable without significant difficultly (c, bottom)). A* PART*-based approach allows components to be easily replaced, and if the model is decomposed into additional* PARTs*, even re-posed (d).*

7.2.3 Specific Applications

The critical response to meshmixer from artists and designers was overwhelmingly positive. We received feedback from a wide range of 3D practitioners, who wished to use meshmixer in many more ways than we had imagined. One aspect of this feedback which was particularly interesting was the variation in *workflows* and diversity of applications that formed sets of recurring tasks for different user groups. I will now describe how PART-based techniques could be integrated into some of these workflows.

Part Assembly I received a significant amount of feedback from special effects artists and product designers who must assemble overlapping 3D scans of physical objects into a single 3D model. Complex objects are often scanned in parts, with varying levels of resolution used for different regions. Assembly of these scans into a consistent surface is a tedious process, particularly if the object is a human who moves between scans.

A professional modeling studio provided a sample assembly task, shown in Figure 7.1, which I completed using meshmixer in under an hour. In terms of assembly quality, fair transitions between the different parts were achieved without difficulty, and although the PART attachment is based on deformation, minimal tweaking of the rigidity parameters was required to achieve relatively good alignment with the original parts.

Figure 7.1d demonstrates other applications of PART-based design in assembly workflows. Since the PARTs are merged in a way that takes surface shape into account, alternate PARTs can easily be swapped in, such as the human hand. Similarly, by manually decomposing the model into additional PARTs, I was able to re-pose it in a relatively convincing (and largely volume-preserving) way. In this example each PARTbased manipulation was baked in after completion. But nothing would prevent this PART decomposition from being captured in a Surface Tree, at which point it would have been possible to specify an animation of the arm simply by dragging the PARTs.

Related to the scan assembly problem is that of of assembling specific models from *part*

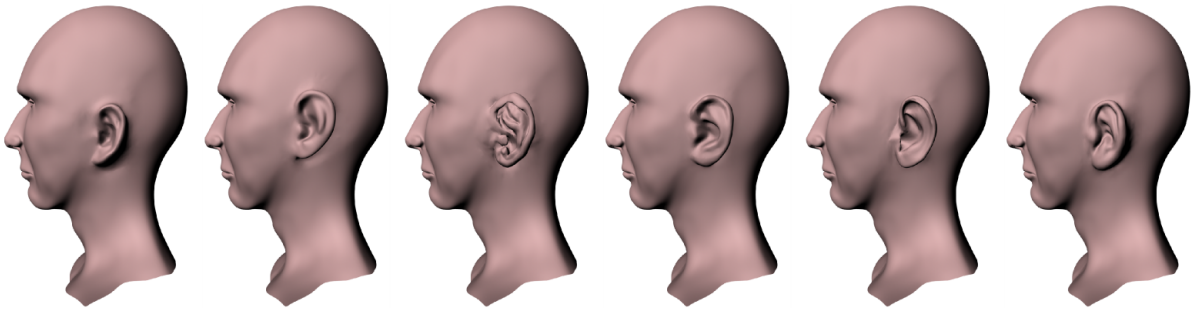


Figure 7.2: *Each ear on the left took only a few seconds to position, making it efficient to experiment with design variations.*

catalogs, in particular to provide customized human models. In this case the component meshes must be deformed to conform to the new surface, which is handled automatically by my surface Part. In addition, the complexity of current tools means that the customer must pick parts from images and then specify design changes to an artist. With a Part-based interface, the customer could efficiently experiment with different character designs, reducing the need for expert guidance. A mock-up of this sort of scenario is shown in Figure 7.2. Note again that picking a set of Parts implicitly defines a Surface Tree, at which point the customer could dynamically manipulate the properties of each shape on the combined surface.

Scan Repair The scan data in Part assembly tasks is generally of quite high quality, where great care is taken to capture all the important surface details. Perhaps the most enthusiastic testers of meshmixer were artists working with more ad-hoc meshed scan data. These scans rarely capture all the necessary surface detail, and often occlusion or motion results in holes and incorrect forms. Cleaning up this data is a tedious manual process. Users noted that being able to simply select and drag off a hole, leaving a fair surface behind, was itself a major improvement to their workflow. Similarly, filling in missing regions often requires the work of a skilled modeler. With a Part-based interface, one can drag in Parts from regions where the scan is more accurate, or from other models, allowing invalid regions to be efficiently repaired (Figure 7.3).

Rapid Base-Mesh Creation As I have discussed, in modern 3D modeling workflows, 3D sculpting tools are used to paint extreme levels of detail onto simplified base-meshes. Creating the necessary base-mesh for a sculpting task can be time consuming, and often 3D sculptors resort to stock models which must then be deformed or otherwise edited into a more suitable form. With a Part-based design tool, one could quickly assemble a suitable base-mesh from a library of stock parts. Furthermore, this assembly process can be captured in a Surface Tree, which would then allow the artist to later make procedural changes to Parts of the base mesh and have the brushed details update automatically. If the quad-mesh topology issues can be resolved to some degree, then Partdrag-and-drop would also make it practical to drop in stock parts as needed, or even remove and replace an existing part when a design variation proves unsuccessful.

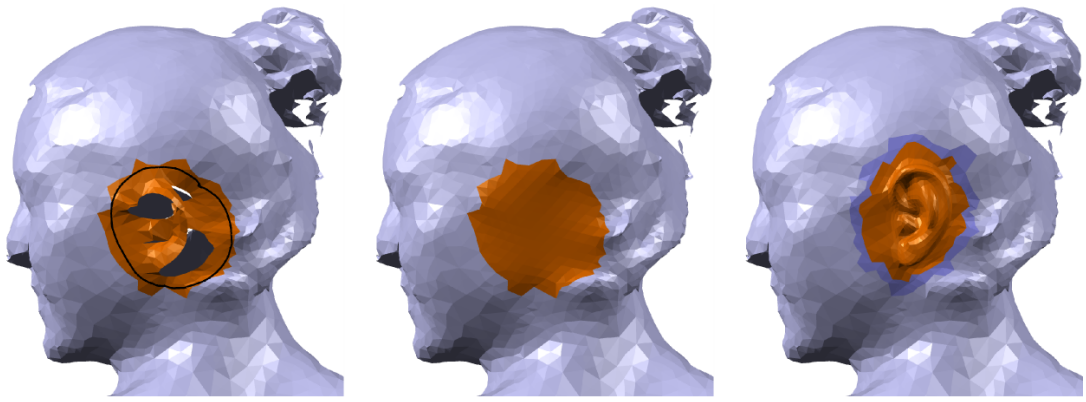


Figure 7.3: *Scanned data often includes regions with holes or missing detail (a). With meshmixer we can quickly drag off the invalid regions, leaving smooth fill surfaces behind (b). The missing detail can then be filled in using more accurate* PARTs*(c).*

Model Pre-Visualization A growing area of application for computer graphics in film and entertainment industries is *pre-visualization* (or *previz*). 3D mock-ups with varying levels of fidelity are now created for both digital and live-action shots, and used for virtually every production task, from basic staging and art direction to detailed camera, lighting, and rendering setup. Creating suitable models for higher-fidelity previz using digital sculpting tools is a time consuming and expensive process, as the models will ultimately be discarded. Several 3D artists noted that our composition tools would make modeling for previz much more efficient. I experimented with this use-case in Figure 7.4, creating a reasonably detailed lizard model. The final model was created in well under an hour, but would suffice for most digital mock-up purposes.

As in the case of PARTcatalogs, PARTdrag-and-drop makes composition simple enough that it may be possible for directors to explore the space of models themselves, rather than having to rely on an artist to translate ideas into 3D. In a similar vein, directors often want to see variations on a theme for a particular model. Instead of creating many independent models from scratch, an artist could author a parameterized Surface Tree model, which would then allow for much more efficient and finer-grained exploration of the design space.

Rapid Prototyping 3D printing has become relatively commonplace in many areas of industrial design, however realizing a 3D model as a physical object introduces its own set of particular challenges. For example, a model may need to be significantly modified so that parts can actually be assembled after printing. A related problem is that many 3D models which can be animated on the computer will either fall to pieces or be fixed in place when fabricated. Attaching the necessary fasteners and hinges via CSG operations involves tedious 3D manipulation and tends to result in sharp edges. Given a library of simple assembly connector surfaces, one could use PARTcomposition to quickly create assemblies suitable for fabrication. Some examples are shown in Figure 7.5.

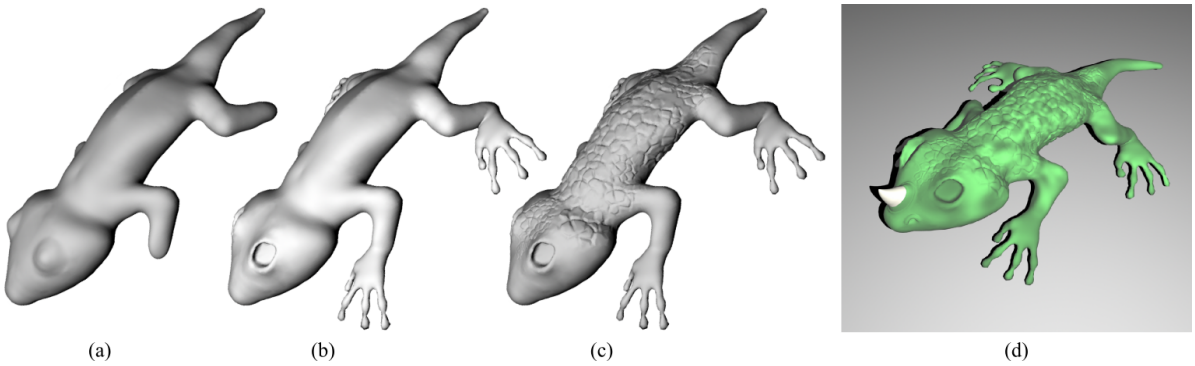


Figure 7.4: *To mock up a lizard model, I first sculpted a sphere into a coarse body shape (a) then transferred the more complex features from other models (b). Next I used a brushing tool to copy details from a rough, scale-like surface (c), then added a few more parts and rendered with a procedural texture (d). No more than 10 minutes was spent on each step.*

7.3 Representational Capability

The PART definition I have formulated, the DEM/COILS approach to implementing it, and the procedural Surface Tree structure, each have a variety of limitations. Some of these follow directly from limitations of the DEM and COILS techniques, while others are more fundamental.

7.3.1 High-Curvature Surface Regions

In general, I have assumed that it is possible to consistently generate relatively low-distortion parameterizations of the PART domain on the target surface. However, the fundamental properties of normal coordinates (Section 4.2) guarantee that the DEM will significantly deform across regions of highly varying curvature, and eventually fold-over at the cut loci, where the marching geodesic front intersects itself. These self-intersections

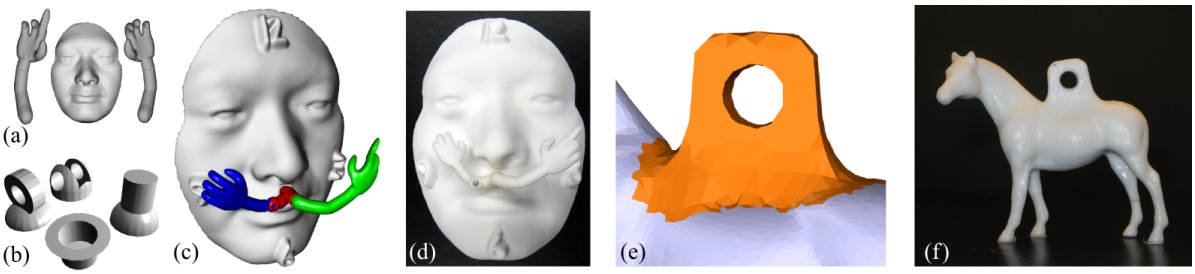


Figure 7.5: *Using a few parts (a) and a library of simple connectors (b), a static model was converted into a clock with a hinge in the hour hand, which allows time zone changes by bending the arm (c). A physical prototype of this clock was then fabricated with a 3D printer (d). In (e-f) a connector is dropped on to an existing model, creating an odd but functional Christmas tree ornament. (The Dali Clock was designed by Karan Singh.)*

can happen at relatively small geodesic radii if the surface has wide variations in curvature. A related problem occurs when the marching geodesic front intersects itself, which is relatively common on surfaces with many protruding features. In this case the Part domain is no longer a disc, but also no longer has a unique parameterization.

Assuming the Part domain is cut into a topological disc, more robust global optimization approaches can avoid these foldovers, at the cost of higher internal distortion, which will in turn deform the Part boundary in unexpected ways. Fundamentally, this is an unresolvable situation - if the Part domain is defined by a fixed mapping from a parameter space, distortion is guaranteed. Similarly, any use of a planar parameterization implicitly enforces the requirement that the part domain is a topological disc.

It seems difficult to devise Part-based interactions which support *generalized transport* that are not dependent on a parameterization. However, one interesting direction for future work would be to post-process the Part domain on the target surface, to improve preservation of the 2D boundary shape. The boundary rigidity approach of Section 5.9.1 could also be extended, to locally optimize regions of the boundary which are highly deformed in 3D. It is conceivable that this may lead to a way to represent the boundary as a sort of "magnetic wire", allowing it to be transferred from one surface to another without the need for a parameterization.

7.3.2 Part Domain Shape

My general strategy for representing an arbitrary surface region is to embed it in a parameterized geodesic disc. This approach assumes that the Part domain is efficiently represented by a geodesic disc - ie it is be bounded by a roughly convex polygon with an aspect ratio relatively close to 1. However, some Parts an artist may wish to use do not fit this restriction. For example, most designers would consider a feature line running the length of a car to be a single Part but my approach would not be particularly efficient to represent it. Even more problematic is that the parameterization of the large geodesic disc that must be grown to contain a long, skinny Part is much more likely to have foldovers or self-intersections.

A clear alternative would be to directly represent the Part boundary, rather than a point-with-geodesic-radius encoding. However, as I discussed in Section 3.6.1, this will significantly complicate the process of moving a Part from one location to another. While reasonably rigid *surface transport* could be computed via vector fields [Crane et al. 2010], *generalized transport* from one surface to another seems very difficult. As I mentioned above, representing the boundary as a 3D curve undergoing rigid deformation may be a viable approach.

It should be noted that the Surface Tree framework I have constructed does assume that the Part domain can be represented relative to a single anchor point that can be embedded in a composite global parameterization. This use of a single point greatly simplifies tree manipulation operations. These operations could perhaps still be implemented with more complex representations of the boundary, but this will certainly impact the performance of the Surface Tree.

An alternative to direct representation of the boundary is to devise more complex encodings than the geodesic disc which are still derived from a single point. Biermann's

multiresolution copy-and-paste system [Biermann et al. 2002] proposed a strategy for representing long-and-skinny regions, and even branching regions. The artist would define a geodesic "spine" for the PART in the source region. Branches of the spine were recursively encoded as geodesics in the tangent space, which could then be recomputed on the target surface. As with Biermann's other techniques, the parameterization was only defined once the PART boundary was fixed, which is not as flexible as my intermediate-parameterization approach, and the artist had to manually define an appropriate spine. However, it may be possible to adapt this strategy to automatically represent more complex PARTs.

Both my geodesic disc encoding, and Biermann's spine technique, are based on encoding relative to a single point. This has the advantage that it is efficient for the artist to interact with, as a 2D cursor can be used to unambiguously specify the point. But as the PART boundary deviates significantly from a roughly circular convex region, it becomes questionable whether a single point will be flexible enough for the artist. Of course it is always possible to locally manipulate the PART domain on the target surface after the initial transfer is applied, as was done in Section 4.6.3. However, with the growing ubiquity of multitouch interfaces, we now have the potential for the artist to accurately specify multiple points. The development of a multi-point encoding of a complex PART boundary would potentially provide the artist with more expressive capabilities. The same can be said for gestural and sketch-based input.

7.3.3 PARTs with Non-Disc Topology

In Section 3.4 I described three forms of the ON operator that attaches a PART to another surface. The *Deformation* form is a deformation of a disc-shaped region of the mesh, while the *Edit* form adds refinement capabilities and more general topologies, and the *Insertion* form replaces the PART domain with a new mesh. In Figure 3.8 I showed that the Insertion form is highly general, supporting a wide variety of topological changes.

It would seem that since the Insertion form is the most capable, the Deformation and Edit forms are redundant. However, because it is restricted to topological discs, only the Deformation form is guaranteed to define an invertible mapping between the input and output surfaces. I took advantage of this in the Surface Tree, where it was necessary to be able to map anchor points between node input and output surfaces during tree modifications. Hence, PARTs with non-disc topology can restrict the Surface Tree in certain ways. In particular, when nodes lack a one-to-one mapping between the input and output surface it may not be possible to consistently update the anchor points of dependencies after tree modifications. Although I described some stopgap measures to potentially deal with these issues in an interactive tool in Section 6.3.5, at the conceptual level my hierarchical representation does not support many topological changes.

Figure 7.6 graphically depicts the possible combinations of PART domain and shape in two cases: addition of a single PART, as in meshmixer, or hierarchical PART assembly, as in a surface tree. Note that I had previously stated in Section 3.6.1 that PARTs domains containing topological handles could not be supported by a domain representation based on parameterization. However, in the table I have indicated partial support. This is due to the ability of the DEM to "flatten" some handles into the plane when using

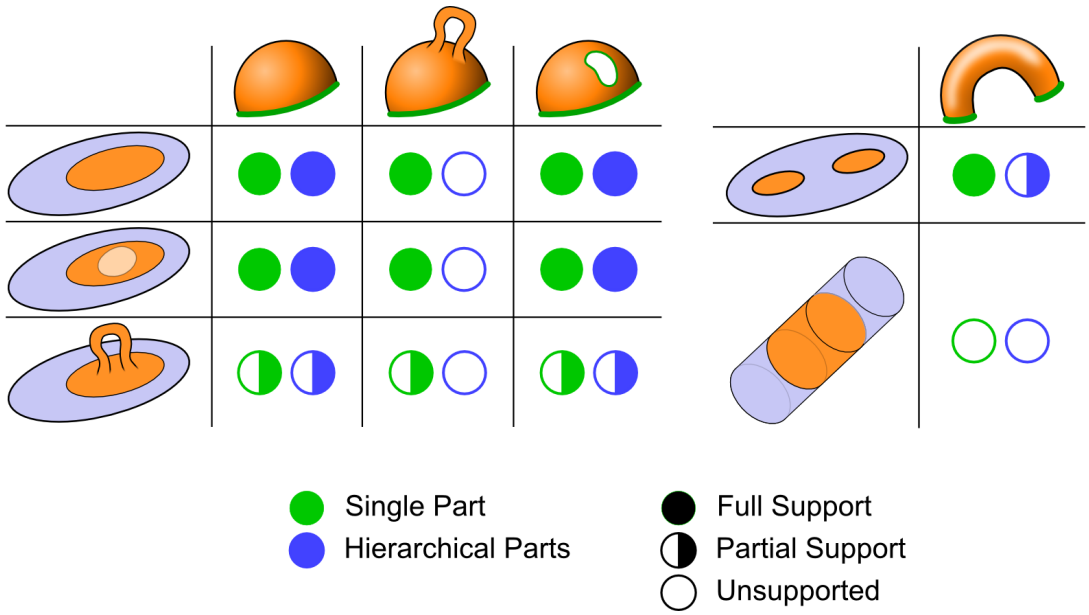


Figure 7.6: *Supported combinations of* Part *domain* $\mathcal{U}$ *(rows) and shape* $\mathcal{V}$ *(columns) for the addition of a single* Part*, as in the meshmixer tool, or hierarchical* Part *assembly in a surface tree.*

smoothed normal fields, as shown in Figure 4.6. All cases with topological handles are marked as partially-supported for hierarchical Parts because, without a one-to-one parameterization, we cannot properly anchor Parts on another containing a topological handle. Parts that are topological handles with multiple boundaries are marked as partially supported because although they do admit a cylindrical parameterization, there is no clear mapping from the input domains if the Part is to be removed.

As I have mentioned, the COILS deformer also has some limitations with respect to non-disc topology. The geodesic front propagation underlying the COILS deformer will split-and-merge when encountering both "holes" in the manifold, and topological holes and handles. In both cases the resulting discontinuities will introduce non-smooth "tears" in the deformation. This issue, and potential solutions, are discussed at length in Section 5.6.

7.3.4 What makes a good Part?

In this work I have focused on how to *represent* arbitrary surface Parts, and avoided making any statements about the *quality* of individual Parts. However, during the development of my Part-based modeling tools I have spent an extensive amount of time experimenting with surface Parts. From this experience, I have drawn some conclusions that may inform future attempts to at quantifying Part quality, for example in Part inference techniques such as mesh segmentation.

At the semantic level it is relatively clear that some surface regions make sense as Parts and others do not. For example, in Figure 7.7a I have clearly selected the *face*, while attempting to textually describe the selections in b and c will make it clear that

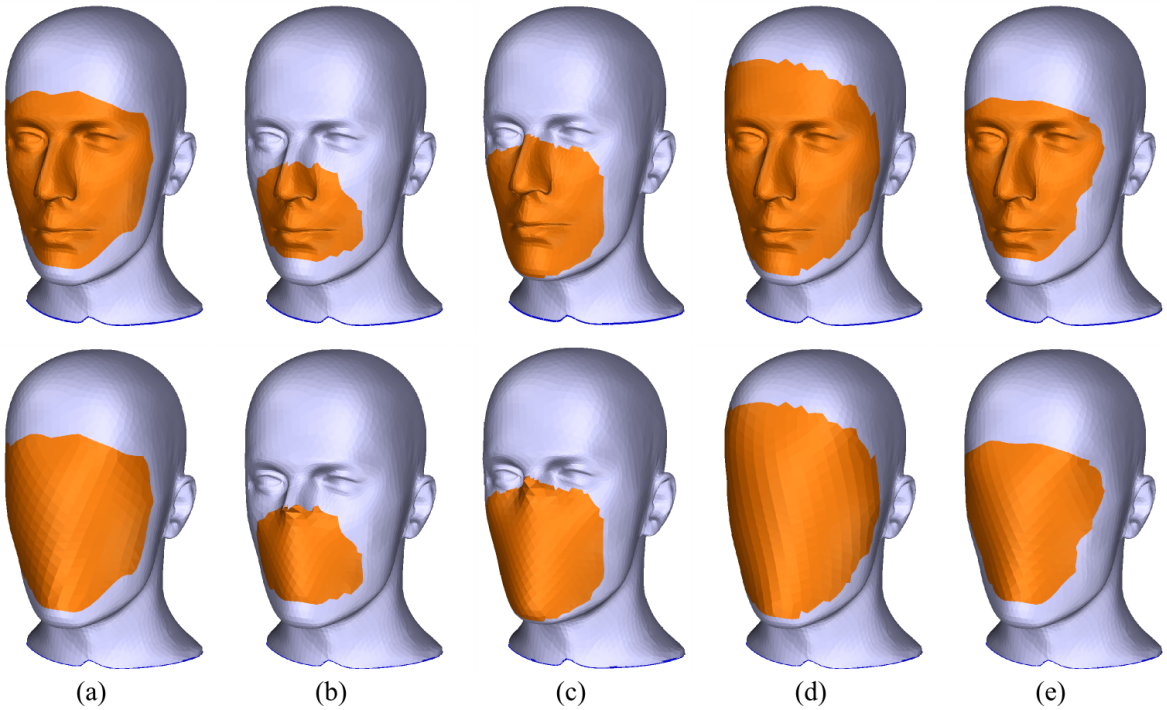


Figure 7.7: *It seems sensible to consider (a) a* Part *but not (b) or (c). Note that (d) and (e) appear to define the same* Part *as (a). If we infer the underlying smooth surfaces, we see that (a),(d), and (e) are all very similar, while (b) and (c) have significant differences around the nose.*

they are not particularly atomic Parts. But as these semantics are not available to an algorithm. Is there anything intrinsically different about these selections that we can detect?

To see that there is, consider Figure 7.7d and e. I would argue that these are also valid selections of the *face*' Part. But then we must accept that the *face* does not actually have a specific boundary. Instead the face blends into the head surface across some transition region - in other words, the face Part*overlaps* the underlying surface.

This observation still does not tell us what is different between the face Part and the selection in Figure 7.7b. Consider, though, that if the Part blends with the underlying surface, then to talk about the Part shape we must also consider the underlying surface. Since this surface is not known, we will have to infer it, and in the absence of any additional information the simplest assumption to make about this missing surface is that it is smooth. Then for a given Part boundary, we can infer the missing surface shape simply by smoothly interpolating the boundary conditions, as I have done in the bottom row of Figure 7.7.

Comparing these underlying surfaces, we can observe an interesting phenomenon. For the three different *face* selections, the inferred *head* is highly similar in each case. In contrast, the inferred underlying surfaces for the two non-Partsare quite different, mainly in the nose region.

So then for this *face* Part, there is clearly a *region* within which the Part boundary

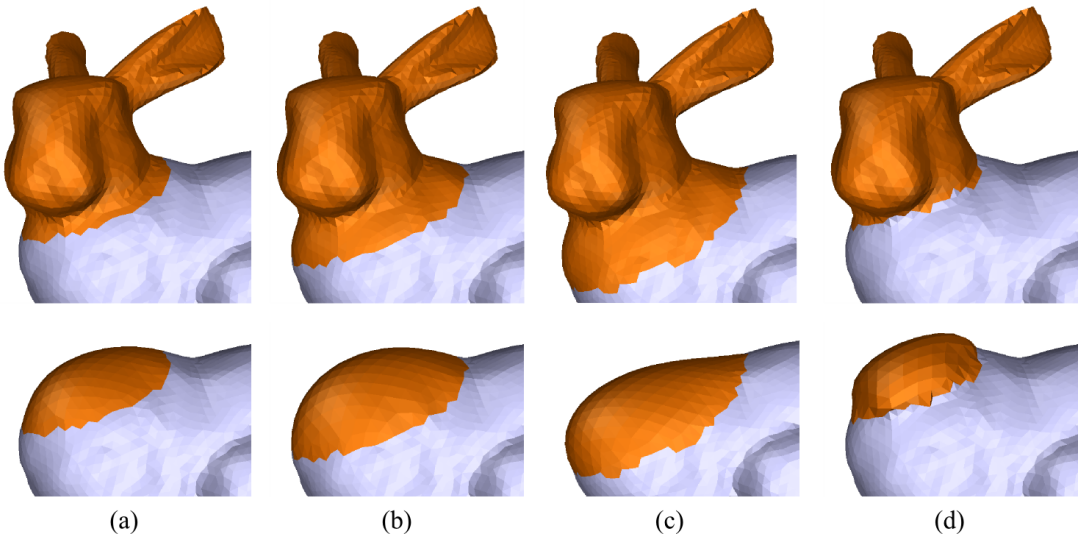


Figure 7.8: *Expanding the* PART *boundary in (a) to that in (b) has a small effect on the inferred underlying surface. However, expanding a bit too far (c) produces a very different decomposition, as does (d) even a small change in the other direction.*

curve can vary without significantly affecting the inferred underlying shape. This region is finite, and we can determine its extents by perturbing the boundary and observing how the underlying surface changes. An example is shown in Figure 7.8, where I created an initial PART selection and then manually offset the selection via geodesic contours. There were clear critical values of this offset where the fill surface diverged, which occurred when the selection boundary reached ridges of significant curvature change.

Assuming that there is a band of finite extent bounding a PART in which the inferred base surface is consistent, then this region is precisely the *overlap* between the PART and base surface. Furthermore, in this overlap region the spatial distance between the inferred PART and base surfaces is very small. Note that this is exactly what one would expect if the PART was inserted into the surface with a smoothly-blended transition region. Hence, as long as the fixed boundary curve resides within this overlap region, the decomposition into PART and base surfaces will be stable.

Assuming these observations hold, then a given boundary loop on a surface will be a potential PART boundary if it can be perturbed without causing a significant change in the inferred underlying surface. It seems relatively straightforward, although perhaps expensive, to interpret this metric algorithmically. Then to find a surface PARTs we simply need to search through the space of boundary loops and find families of loops which score highly on this metric. Of course, most local regions of a surface have this property, so one must also require that the PART interior have some geometrically salient features. One potential metric in this respect would be to measure the difference between the PART interior and the inferred fill surface.

An obvious direction for future work would be to try to implement this PART-detection approach (and then to figure out how to make it efficient). This may result in surface decompositions that more accurately represent semantic surface PARTs. As I have previ-

ously stated, the disjoint *partitions* generated by current segmentation algorithms often do not correspond with intuitive notions of surface Parts. I believe that this is because surface Parts frequently *overlap*, and based on the above observations, the Part boundary should reside in this overlap region. But if these overlap regions are the blending transitions between Part and base surface, then they are precisely the regions where any notable geometric features will be suppressed. In fact, boundaries that are close to geometrically salient features such as curvature ridges may not result in particularly good Part-based modeling components, as I showed in Figure 7.8.

Chapter 8

Conclusion and Future Work

In Chapter 1, I noted that the three main goals of this thesis were to determine how to define and represent a PART of a 3D surface, construct a hierarchical, procedural PART-based surface representation, and explore how PART-based techniques can be applied in 3D design tools.

In Chapters 3 to 6 I have developed a framework for PART-based surface modeling. After deciding on an domain/shape PART decomposition in Chapter 3, I presented the Discrete Exponential Map (DEM) to encode PART domains on surfaces (Chapter 4), and the COILS deformer to encode PART shapes (Chapter 5). Then in Chapter 6 I described the Surface Tree, a hierarchical, procedural, PART-based representation of complex 3D surface models. Each of these is a novel contribution to the field of shape modeling.

In each of the technical chapters I have also described a novel PART-based 3D design interface which significantly improves on the capabilities of previous works which addressed similar problems. My decal texturing tool allows one to quickly assemble complex surface texture by layering texture PARTs. Meshmixer enables artists to mix-and-match arbitrary surface PARTs from existing 3D models with a level of interactivity and automation that has not been previously demonstrated. And the Surface Tree editor provides a sketch-based interface for adding layered procedural PARTs to an initial base surface, including topological handle PARTs, each of which can then be procedurally manipulated. I also demonstrated that meshmixer could be re-purposed as a tool for decomposing an existing model into a Surface Tree.

Perhaps the most pressing question that arises after considering these techniques and tools is whether they would be of any practical use to artists and designers. Both meshmixer and my interactive Surface Tree editor involve "drag-and-drop" composition techniques which combine surface-constrained positioning, global deformation, and geometry merging into an atomic action controlled by a single parameter: the position of mouse cursor over the 3D surface.

On the one hand, even casual computer users with no 3D modeling experience have been able to immediately use this tool. Based on that experience, I claim some level of success in creating a simple and usable interface for PART-based 3D surface composition. However, in simplifying the interface to this extent I also significantly constrained part positioning, deformation, and merging. It was unclear whether such a restricted tool would be useful to professional practitioners of 3D modeling. Hence, in Chapter 7 I

explored the utility of this Part-based interaction by performing a a large-scale informal evaluation to solicit a wide range of feedback.

To carry out this evaluation, I publicly released each of my tools on the internet, to collect feedback from graphics hobbyists, educators, and industry professionals. The simple Part composition tool, *meshmixer*, was the most successful of these systems. Meshmixer was widely publicized on internet forums, blogs, and news websites, and remains in active use today. Based on public internet discussions about meshmixer, and direct feedback I received from artists and designers, I was able to provide evidence that Part-based interaction provides capabilities which are unavailable in current tools.

I will now discuss a few potential extensions and applications of this work.

Increased Generality While our interactive system demonstrates the potential of Surface Tree modeling, there is extensive room for improvement. One fundamental limitation is the use of parameterization to encode Part domains. I discussed how we can replace the DEM with more robust global parameterizations to avoid foldovers, but in the process will inherit increased parametric "wobbliness". One possibility would be to define *smoother* parameterizations, which sacrifice some local distortions to increase more global smoothness metrics (perhaps a simple approach would simply be to parameterize a smoothed version of the surface). Another possibility would be to directly optimize the Part domain on the surface, to preserve salient geometric properties like boundary curvature or domain. Hofer [Hofer 2007] surveys a variety of techniques for performing constrained energy minimization of curves on surfaces.

I also noted one of the major limitations of my Surface Tree, which is that it does ultimately require some kind of global parameter space in which to *anchor* each node. I conjectured that this is an unavoidable side-effect of defining surface Parts, but perhaps some workaround can be found. Assuming a global parameterization is necessary, the ideal choice would be a Riemannian manifold structure with a low-distortion atlas. However, such a general manifold suitable for interactive shape modeling has yet to be developed. The design of such a global, or even local, manifold parameterization is a key topic for future study. In particular, characterization of its behavior under deformation will be critical for predictable interactive surface design.

Parametric Surface Modeling In Chapter 6 I demonstrated some very simple functional modeling operations, like linking parameters between copies of a Part. This capability can simplify many repetitive modeling tasks, and also can be extended much further. It is entirely possible to functionally link parameters between arbitrary nodes - one can imagine a whole set of edits whose parameters are driven by a few simple controls. A sensible interface for constructing linked parameter networks is one direction for future research.

Another interesting possibility would be to incorporate CAD-style parametric modeling techniques into the surface editing domain. In solid modeling the types of parametric constraints one wishes to preserve are often based on straightforward Euclidean information - lengths, angles, volumes, and so on. To adapt parametric modeling to surface Parts, first we must devise a vocabulary of constraints between Parts that can be used

to express properties we might like to preserve. Geodesic distances, surface domains, and Part volumes would likely be included in this vocabulary. The next task would be to determine how to solve these constraint systems. Solid modeling constraint solvers may be adaptable, but it seems likely that a more heuristic approach may be required, along the lines of the iWires system [Gal et al. 2009].

Construction History One potential application of the Surface Tree is as a *construction history* for existing interactive modeling operations. If each operation in a modeling session can be automatically captured as a Surface Tree node, then the full power of a procedural, hierarchical representation would be available if the artist wished to use it. It is straightforward to see how some kinds of modeling operations, such as the displacement painting found in sculpting tools, or relayering operations [Igarashi and Mitani 2010], can be integrated into this framework. However, more complex deformations are usually defined with respect to 3D *handles* which are arbitrarily oriented in 3D space (control points, curves, and so on). For example, recent linear variational mesh editing techniques such as Laplacian deformation [Botsch and Sorkine 2008] are specified by interactively re-positioning a rigid portion of the surface. To capture such a deformation as a Surface Tree Node, this handle and its global 3D position must somehow be represented relative to the base surface, in such a way that when the base surface changes, the handle can be re-computed in a way that the designer finds reasonably predictable.

Other Surface Types In this thesis I have focused on mesh surfaces. However, other representations like NURBS, multiresolution SubDs, and and even point set surfaces are in broad use. My procedural Parts can be adapted to each of these representations. Point set surfaces are perhaps the easiest, as the key components of my approach - the Discrete Exponential Map, the COILS deformer, and the procedural mesh data structure - can be applied directly to point sets. A Surface Tree created using the mesh-based interface could even be "played back" on a point set surface.

If we wished to integrate a NURBS or SubD Part into a NURBS or SuBD base surface, then the main challenge is to determine how to implement the graph insertion operations. Note, however, that in most modeling tools NURBS and SubD are simply high-level controls for designing meshes. Hence, from the point-of-view of most 3D designers, NURBS and SubD are not *surface representations* but rather *modeling interfaces*. It is debatable whether these are *good* modeling interfaces, however because of their ubiquity and even dominance in interactive modeling tools, every 3D artist knows how to use them, to the point of being able to in most cases accurately predict how the underlying surface will respond to manipulation of a particular control point.

Because of this familiarity, providing a NURBS or SubD-style interface to local shape manipulation has significant practical benefits. This suggests an interesting possibility - local NURBS and SuBD patches could be encapsulated as PartsṪhese Parts would provide the familiar control-point interfaces, but integration into the base surface mesh would be handled by the Surface Tree framework. One immediate advantage of this approach over traditional NURBS and SubD models is that the artist would be freed from having to explicitly manage global topology.

Reverse Engineering A practical challenge which any new surface representation faces is that of compatibility with previous representations. It seems clear that designers will want to utilize Part-based modeling techniques with their existing databases of 3D models. The main challenge here is that these existing models must be decomposed into a meaningful semantic hierarchy of Parts. As I have shown, the meshmixer interface can be used to manually carry out such a segmentation, although it is somewhat tedious. For large databases of models some automated scheme would be necessary. However, existing segmentation algorithms focus on splitting the model up into disjoint polygon groups, rather than a layered decomposition. As I discussed in Section 7.3.4, there is reason to believe that segmentation algorithms could be extended to infer Part overlaps. Even a hybrid system based on user guidance would be useful in many application domains.

Physical Modeling Workflows A common workflow used in many shape modeling tasks is to first work in physical media like clay, and then transfer the scanned model to the computer. The resulting point cloud lacks any structure whatsoever, usually necessitating time-consuming "re-topologizing" with NURBS or Subdivision surfaces.

An alternative would be to capture scans as the sculpting progresses. The differences between these scans would define a Part decomposition of the final surface, and could be represented by a Surface Tree. In addition to being a potentially useful interface for constructing Surface Trees, the digital representation provides benefits for the physical modeler. For example, it is much easier to enforce symmetry in the digital model. With the rise of rapid prototyping, it is then possible to transfer virtual manipulations back into the physical world, enabling an iterative physical/digital workflow.

Animation As noted above, parametric constraint techniques have seen little application in surface modeling. This is an interesting area which may have many benefits, particularly for animation. One general advantage of procedural representations is that they can easily be animated, simply by manipulating parameters over time. Surface Trees provide some interesting possibilities because the layered features can be independently deformed, providing visual effects that would be difficult to generate otherwise.

One can even imagine applying simulation to animate Parts. For example, applying gravity to constrained "cloth" Partscould be used to add sagging jowls or a furrowing brow to a face. Dynamics-based procedural Parts could be a useful tool to quickly create complex surface features that would be time-consuming or difficult to model. Another interesting possibility would be to maintain dynamic Parts as active simulations. This would allow animators to combine direct control of some facial features with dynamic simulation of others. From an artistic standpoint, such a dynamics-with-control approach may ultimately be more efficient and expressive than both manual and fully-automatic techniques [Sifakis et al. 2006].

One issue with animating a Part-based surface model is that many animation techniques depend on the ability to easily *deform* the model surface. Such deformations may be challenging to integrate into the Part-based model, as I will now discuss.

Deformation and In-Context Part Manipulation One challenge in integrating Part-based modeling into existing modeling tools will be in determining how current surface modeling techniques should be applied to decomposed models. Ideally one would represent all model changes procedurally, so for example a global deformation applied to a model would simply be another node in a Surface Tree. However, it may be more practical to "bake" such operations into the existing set of Parts, particularly in certain types of interfaces like sculpting tools.

This raises a tricky problem: the Part is always presented to the user in a deformed state. The question then is how to apply standard modeling operations like deformation to the Part geometry. There are two alternatives. One is to map the modeling operations back into the original Part space. So, for example, in a Laplacian deformation this would imply that the control handle manipulations must be transferred from the deformed to un-deformed configurations. However the surface deformations I have considered (my COILS deformer, the RIC deformation) do not support this inverse mapping. Even if a spatial deformation were inferred, for example as in Sumner et al. [Sumner et al. 2007], the result of this indirect manipulation may not be intuitive to the user. The other option is to apply the deformation to the deformed Part geometry. This is likely to be preferable to the artist, as the behavior will then be the same as traditional tools. However, we must now somehow transfer the deformed configuration of the deformed Part back into the un-deformed Part space. Again, without an invertible spatial deformation this problem seems quite challenging to address.

This is a significant issue, although one must also consider whether it is necessary for a Part-based tool to support in-context manipulation. In solid modeling tools, Parts are largely manipulated in isolation, and then assembled. Taking the same approach to surface Part editing does seem less natural, given that the surface Parts are meant to smoothly integrate into a composite surface. However, one can observe that in many cases where I have used complex Parts that may need to be edited, they are not extensively deformed except near the boundary, and near the boundary there is generally a region of low detail to ensure a smooth transition. In these cases it seems completely plausible that one would edit the Part in isolation, and rely on indirect manipulations (blending parameters, controls in the Part domain, etc) to refine the transition shape.

Cases where significant deformation is evident (and desirable) generally involve displacement-like Parts, but changes in displacement vectors are much easier to transfer back to the original Part configuration. This line of reasoning does lead us towards scenarios where we have different classes of Parts depending on the type of feature being represented, but perhaps this is unavoidable if we wish to support a wide range of different Part manipulations.

Interaction Finally, working with the Surface Tree itself involves a number of interaction and visualization problems which cannot easily be solved with existing techniques. I have mentioned Visual Scaffolding [Schmidt et al. 2007], which exposes some of the internal structure of a hierarchical 3D model, although the semantics of the visualization are still limited to a flat list of parts. Integrating a representation of the model hierarchy into visual scaffolding would be highly useful, particularly if it also allows the tree to be

edited. Though it does not expose the global structure, one idea is to manipulate the visual representation of selected items to convey the local structure of the model tree. This may sidestep the visual complexity associated with displaying the full model tree, although at the cost of global comprehension. One highly desirable property of any tree visualization technique will be to support drag-and-drop of tree nodes between parents. Again, to be applicable to layered surfaces, this must be done directly on the 3D surface. Note that visual scaffolding has yet to be applied to layered surface representations - the simple volumetric techniques currently proposed [Schmidt et al. 2007] may need to be adapted to represent more complex surface Parts.

Part-based representation and manipulation of surface models has many advantages over the current state-of-the-art. However, at a more basic level, perhaps the largest benefit of Part-based interaction is its capability to make 3D surface modeling more accessible. Even the most intuitive sketch-based interfaces can provide little guidance in mastering the skills necessary to express an idea as a detailed and realistic 3D model. 3D sculpting tools still have relatively difficult interfaces, and there is a steep learning curve which must be traversed if one is to become capable of creating even moderately complex surface models. I believe that this is one of the most significant hindrances to wider artist adoption of 3D design. This barrier may perhaps be lowered if the novice is able to rely on libraries of parts to add the details they are not yet able to create themselves. I personally have experienced this effect myself, while creating many of the figures in this thesis. It is this "making hard things easy" aspect of Part-based surface modeling that I find the most compelling. My hope is that this thesis provides others with the the initial tools necessary to perform further exploration of the broad range of possibilities for Part-based surface modeling.

Bibliography

Adobe Systems Inc., 2007. Adobe Illustrator. www.adobe.com/illustrator.

Adobe Systems Inc., 2007. Adobe Photoshop. www.adobe.com/photoshop.

Alexa, M., and Adamson, A. 2009. Interpolatory point set surfaces—convexity and hermite data. *ACM Trans. Graph. 28*, 2, 1–10.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. 2001. Point set surfaces. In *Proc Visualization '01*, 21–28.

Alexa, M. 2002. Linear combination of transformations. *ACM Trans. Graph. 21*, 3, 380–387.

Alexa, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 105–114.

Allègre, R., Barbier, A., Galin, E., and Akkouche, S. 2004. A hybrid shape representation for free-form modelling. In *Proc. Shape Modeling International*.

Angelidis, A., Cani, M.-P., Wyvill, G., and King, S. 2004. Swirling-sweepers: Constant-volume modeling. In *Proc. Pacific Graphics*, 10–15.

Angelidis, A., Wyvill, G., and Cani, M.-P. 2004. Sweepers: Swept user-defined tools for modeling by deformation. In *Proc. Shape Modeling International*.

Attene, M., Mortara, M., Spagnuolo, M., and Falcidieno, B. 2008. Hierarchical convex approximation of 3d shapes for fast region selection. In *Proc. Symposium on Geometry Processing 2008*, 1323–1332.

Autodesk Inc., 2010. 3ds Max 2011, July. http://www.autodesk.com/3dsMax.

Autodesk Inc., 2010. Alias design 2011, July. http://www.autodesk.com/aliasstudio.

Autodesk Inc., 2010. AutoCAD 2011, July. http://www.autodesk.com/autocad.

Autodesk Inc., 2010. Inventor 2011, July. http://www.autodesk.com/inventor.

Autodesk Inc., 2010. Maya 2011, July. http://www.autodesk.com/maya.

Autodesk Inc., 2010. Mudbox 2011, July. http://www.autodesk.com/mudbox.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3, 1–9.

BARGHIEL, C., BARTELS, R., AND FORSEY, D. 1994. Pasting spline surfaces. In *Mathematical Methods for Curves and Surfaces*, 31–40.

BARR, A. H. 1984. Global and local deformations of solid primitives. *SIGGRAPH Comput. Graph. 18*, 3, 21–30.

BARTELS, R. H., AND FORSEY, D. R. 1993. Spline overlay surfaces. Tech. rep., University of British Columbia.

BARTELS, R., BEATTY, J., AND BARSKY, B. 1995. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan-Kaufmann.

BELKIN, M., AND NIYOGI, P. 2008. Towards a theoretical foundation for laplacian-based manifold methods. *Journal of Computer and System Sciences 74*, 8, 1289–1308.

BENDELS, R., SCHNABEL, R., AND KLEIN, R. 2006. Detecting holes in point set surfaces. *Journal of WSCG*.

BIEDERMAN, I. 1987. Recognition-by-components: a theory of human image understanding. *Psychological Review 94*, 2, 115–147.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph. 21*, 3, 312–321.

BISCHOFF, S., PAVIC, D., AND KOBBELT, L. 2005. Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4, 1332–1352.

BLINN, J., AND NEWELL, M. 1976. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10, 542–547.

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *SIGGRAPH Comput. Graph. 16*, 3.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics*. (To Appear).

BOTSCH, M., AND KOBBELT, L. 2003. Multiresolution surface representation based on displacement volumes. *Comput. Graph. Forum 22*, 3, 483–492.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Trans. Vis. Comp. Graph. 14*, 1, 213–230.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. *Comp. Graph. Forum 26*, 3, 339–347.

BREWIN, L. 2009. Riemann normal coordinate expansions using cadabra. *Class. Quantum. Grav. 26*, 175017.

BRODERSEN, A., MUSETH, K., PORUMBESCU, S., AND BUDGE, B. 2007. Geometric texturing using level sets. *IEEE Trans. Vis. Comp. Graph.*. to appear.

BROWN, S., MORSE, B., AND BARRET, W. 2009. Interactive part selection for mesh and point models using hierarchical graph-cut partitioning. In *Proc. Graphics Interface 2009*, 23–30.

BRUN, A. 2008. *Manifolds in Image Science and Visualization*. PhD thesis, Institute of Technology, Linkoping University.

BRUNETON, E., AND NEYRET, F. 2008. Real-time rendering and editing of vector-based terrains. *Comput. Graph. Forum 27*, 2, 311–320. Proc. Eurographics '08.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design 10*, 6, 350–355.

CHAN, K., MANN, S., AND BARTELS, R. 1997. World space surface pasting. In *Proc. Graph. Interface '97*, 146–154.

CHEEGER, J., AND EBIN, D. G. 1975. *Comparison Theorems in Riemannian Geometry*. North-Holland Mathematical Library.

CHEN, G., ESCH, G., WONKA, P., MÜLLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. *ACM Trans. Graph. 27*, 3, 1–10.

CHEN, X., GOLOVINSKIY, A., , AND FUNKHOUSER, T. 2009. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3.

CIPRIANO, G., AND GLEICHER, M. 2007. Molecular surface abstraction. *IEEE Transactions on Visualization and Computer Graphics 13*, 6, 1608–1615.

CONRAD, B., AND MANN, S. 2000. Better pasting via quasi-intepolation. In *Curve and Surface Design: Saint-Malo 1999*, Vanderbilt University Press.

COOK, R. L. 1984. Shade trees. In *Proc. SIGGRAPH 84*, 223–231.

COYNE, B., AND SPROAT, R. 2001. Wordseye: an automatic text-to-scene conversion system. In *Proc. SIGGRAPH 2001*, 487–496.

CRANE, K., DESBRUN, M., AND SCHRÖDER, P. 2010. Trivial connections on discrete surfaces. *Computer Graphics Forum (Proc. SGP 2010)*.

DASSAULT SYSTEMES / SOLIDWORKS CORP., 2010. SolidWorks 2010, July. http://www.solidworks.com/.

DAVIS, J., MARSCHNER, S., GAR, M., AND LEVOY, M. 2002. Filling holes in complex surfaces using volumetric diffusion. In *Proc. Intl. Symp. 3D Data Proc., Vis., Trans.*

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Comp. Graph. Forum 21*, 3, 383–392.

DETURCK, D. M., AND KAZDAN, J. L. 1981. Some regularity theorems in riemannian geometry. *Annales scientifiques de l'Ecole Normale superieure 14*, 3, 249–260.

DEY, T. K., AND GOSWAMI, S. 2004. Provable surface reconstruction from noisy samples. In *Proceedings of the 20th annual symposium on Computational geometry*, 330–339.

DIJKSTRA, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Prentice Hall.

DO CARMO, M. P. 1994. *Riemannian Geometry*. Birkhauser.

DOLGOV, A. D., AND KHRIPLOVICH, I. B. 1983. Normal coordinates along a geodesic. *General Relativity and Gravitation 15*, 11, 1033–1041.

EBERT, D. S., Ed. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann. ISBN 1558608486.

ELBER, G. 2005. Geometric texture modeling. *IEEE Comput. Graph. Appl. 25*, 4, 66–76.

ELETRONIC ARTS INC., 2010. Spore Creature Creator, August. http://www.spore.com.

FARBMAN, Z., HOFFER, G., LIPMAN, Y., COHEN-OR, D., AND LISCHINSKI, D. 2009. Coordinates for instant image cloning. *ACM Trans. Graph. 28*, 3, 1–9.

FARIN, G., HOSCHEK, J., AND KIM, M.-S. 2002. *Handbook of computer aided geometric design*. North Holland.

FARIN, G. 2002. *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. Morgan-Kaufmann.

FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. T. 2005. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3, 544–552.

FLOATER, M., AND HORMANN, K. 2005. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*. Springer, 157–186.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., HUGHES, J. F., AND PHILLIPS, R. 1993. *Introduction to Computer Graphics*. Addison-Wesley.

FORSEY, D., AND BARTELS, R. 1988. Hierarchical b-spline refinement. In *Proc. SIGGRAPH 88*, 205–212.

FU, H., TAI, C.-L., AND ZHANG, H. 2004. Topology-free cut-and-paste editing over meshes. In *Proc. Geometric Modeling and Processing*, 173–182.

Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., and Dobkin, D. 2004. Modeling by example. *ACM Trans. Graph. 23*, 3, 652–663.

Gal, R., Sorkine, O., Mitra, N., and Cohen-Or, D. 2009. iwires: an analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28*, 3, 1–10.

Geometry Systems Inc., 2010. GSI Studio, July. http://www.geometrysystems.net.

Glaser, B. G., and Strauss, A. L. 1967. *The discovery of grounded theory.* Aldine.

Google Inc., 2007. SketchUp 6, December. www.sketchup.com.

Google Inc., 2010. Google Analytics, August. http://www.google.com/analytics.

Gramkow, C. 2001. On averaging rotations. *Int. J. Comput. Vision 42*, 1-2, 7–16.

Greenberg, S., and Buxton, B. 2008. Usability evaluation considered harmful (some of the time). In *Proc. CHI 2008*, 111–120.

Grimm, C., and Hughes, J. 1995. Modeling surfaces of arbitrary topology. In *Proc. Siggraph 1995*, 359–369.

Grimm, C., and Zorin, D. 2006. Surface modeling and parameterization with manifolds: Siggraph 2006 course notes. In *ACM SIGGRAPH 2006 Courses*, 1–81.

Grimm, C. 2004. Parameterization using manifolds. *International Journal of Shape Modeling 10*, 1, 51–80.

Grimm, C. 2005. Spherical manifolds for adaptive resolution surface modeling. In *Proc. Graphite 2005*, 161–168.

Gu, X., He, Y., and Qin, H. 2006. Manifold splines. *Graph. Models 68*, 3, 237–254.

Guennebaud, G., and Gross, M. 2007. Algebraic point set surfaces. In *Proc. SIGGRAPH '07*, 23.

Guskov, I., Khodakovsky, A., Schroder, P., and Sweldens, W. 2002. Hybrid meshes: multiresolution using regular and irregular refinement. In *Proc. Symp. Comp. Geom.*, 264–272.

Haeberli, P. E. 1988. Conman: a visual programming language for interactive graphics. *SIGGRAPH Comput. Graph. 22*, 4, 103–111.

Hanrahan, P., and Haeberli, P. E. 1990. Direct wysiwyg painting and texturing on 3d shapes. In *Proceedings of SIGGRAPH 90*, vol. 24, 215–223.

Hassner, T., Zelnik-Manor, L., Leifman, G., and Basri, R. 2005. Minimal-cut model composition. In *Proc. SMI '05*, 72–81.

Havemann, S. 2005. *Generative Mesh Modeling.* PhD thesis, TU Braunschweig.

HOFER, M. 2007. Constrained optimization with energy-minimizing curves and curve networks - a survey. In *Proc. 23rd Spring Conference on Computer Graphics*, 31–39.

HOFFMAN, D. D., AND RICHARDS, W. A. 1984. Parts of recognition. *Cognition 18*, 65–96.

HOFFMANN, C., AND ROSSIGNAC, J. 1996. A road map to solid modeling. *IEEE Transactions on Visualization and Computer Graphics 2*, 1, 3–10.

HOFFMANN, C. 1989. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann.

HORN, B. K. 1987. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Amer. 4*, 629–642.

HSU, W. M., HUGHES, J. F., AND KAUFMAN, H. 1992. Direct manipulation of free-form deformations. In *Proceedings of SIGGRAPH '92*, 177–184.

HUANG, X., FU, H., AU, O. K.-C., AND TAI, C.-L. 2007. Optimal boundaries for Poisson mesh merging. In *Proc. SPM '07*, 35–40.

IGARASHI, T., AND HUGHES, J. 2001. A suggestive interface for 3d drawing. In *Proc. UIST '01*, 173–181.

IGARASHI, T., AND MITANI, J. 2010. Apparent layer operations for the manipulation of deformable objects. *ACM Trans. Graph. 29*, 4, 1–7.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proc. SIGGRAPH '99*, 409–416.

JIN, J., GARLAND, M., AND RAMOS, E. A. 2009. Mls-based scalar fields over triangle meshes and their application in mesh processing. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 145–153.

KALOGERAKIS, E., HERTZMANN, A., AND SINGH, K. 2010. Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics 29*, 3.

KANAI, T., SUZUKI, H., MITANI, J., AND KIMURA, F. 1999. Interactive mesh fusion based on local 3D metamorphosis. In *Proc. Graphics Interface '99*, 148–156.

KENDALL, W. S. 1990. Probability, convexity, and harmonic maps with small images i: uniqueness and fine existence. *Proc. Londond Math. Soc. 61*, 2, 371–406.

KIMMEL, R., AND SETHIAN, J. 1998. Computing geodesic paths on manifolds. *Proc. of National Academy of Sci. 95*, 15 (July), 8431–8435.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH '98*, 105–114.

KOBILAROV, M., CRANE, K., AND DESBRUN, M. 2009. Lie group integrators for animation and control of vehicles. *ACM Trans. Graph. 28*, 2, 1–14.

KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004) 23*, 23, 861–869.

KRAEVOY, V., JULIUS, D., AND SHEFFER, A. 2007. Shuffler: Modeling with interchangeable parts. *Proc. Pacific Graph. '07*, 129–138.

KRAEVOY, V., SHEFFER, A., COHEN-OR, D., AND SHAMIR, A. 2008. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2008)*.

KRISHNAMURTHY, V., AND LEVOY, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *Proc. SIGGRAPH '96*, 313–324.

LANCASTER, P., AND SALKAUSKAS, K. 1981. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 87, 141–158.

LANDRENEAU, E., AND SCHAEFER, S. 2010. Scales and scale-like structures. In *Proc. Symposium on Geometry Processing 2010*.

LAWRENCE, J., AND FUNKHOUSER, T. 2003. A painting interface for interactive surface deformations. In *Proc. Pacific Graphics 2003*, 141–150.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2005. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5, 444–465.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)*.

LEUNG, R., AND MANN, S. 2003. Distortion minimization and continuity preservation in surface pasting. In *Proc. Graphics Interfaces 2003*.

LEVIN, D. 1998. The approximation power of moving least squares. *Math. Comp. 67*, 224, 1517–1531.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of ACM SIGGRAPH 2001*, 417–424.

LEWIS, T., AND JONES, M. W. 2004. A system for the non-linear modelling of deformable procedural shapes. *Journal of WSCG 12*, 2, 253–260.

LI, M., LANGBEIN, F. C., AND MARTIN, R. R. 2010. Detecting design intent in approximate cad models using symmetry. *Compupter-Aided Design 42*, 183–201.

LIEPA, P. 2003. Filling holes in meshes. In *Proc. SGP '03*, 200–205.

Lipman, Y., O.Sorkine, Levin, D., and Cohen-Or, D. 2005. Linear rotation-invariant coordinates for meshes. In *Proc. SIGGRAPH '05*, 479–487.

Lipman, Y., Cohen-Or, D., Gal, R., and Levin, D. 2007. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph. 26*, 1.

Lipp, M., Wonka, P., and Wimmer, M. 2008. Interactive visual editing of grammars for procedural architecture. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 1–10.

Lounsbery, M., DeRose, T. D., and Warren, J. 1997. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Trans. Graph. 16*, 1, 34–73.

Luxology Inc., 2010. Modo 401, July. http://www.luxology.com/modo/.

Ma, M. 2000. *The Direct Manipulation of Pasted Surfaces.* Master's thesis, University of Waterloo.

MacCracken, R., and Joy, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *Proceedings of SIGGRAPH '96*, 181–188.

Mantyla, M. 1988. *Introduction to Solid Modeling.* W.H. Freeman & Company.

Milliron, T., Jensen, R., Barzel, R., and Finkelstein, A. 2002. A framework for geometric warps and deformations. *ACM Trans. Graph. 21*, 1, 20–51.

Mitchell, J. 2000. *Geometric Shortest paths and network optimization.* Elsevier Science, ch. Handbook of Computational Geometry, 633–702.

Mitra, N. J., Guibas, L., and Pauly, M. 2006. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics 25*, 3, 560–568.

Mortari, D. 2001. On the rigid rotation concept in n-dimensional spaces. *J. Astronautical Sciences 49*, 3, 401–420.

Muller, M. J., and Kogan, S. 2010. Grounded theory method in hci and cscw. Tech. Rep. TR2010.09, IBM Watson Research Center.

Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. 2006. Laplacian mesh optimization. In *Proc. ACM GRAPHITE*, 381–389.

Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3.

Osher, S., and Fedkiw, R. 2003. *Level Set Methods and Dynamic Implicit Surfaces.* Springer.

Pakdel, H.-R., and Samavati, F. 2005. Incremental catmull-clark subdivision. In *Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, 95–102.

PASKO, A., ADZHIEV, V., SOURIN, A., AND SAVCHENKO, V. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8, 429–446.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3, #43, 1–11.

PEDERSEN, H. K. 1995. Decorating implicit surfaces. In *Proceedings of SIGGRAPH 95*, 291–300.

PEDERSEN, H. K. 1996. A framework for interactive texturing operations on curved surfaces. In *Proceedings of SIGGRAPH 96*, 295–302.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Trans. Graph. 23*, 3, 635–643.

PENNEC, X. 2006. Intrinsic statistics on riemannian manifolds: Basic tools for geometric measurements. *J. Math. Imaging Vis. 25*, 1, 127–154.

PETTERSSON, T., 2010. Sculptris v1.0, July. http://www.sculptris.com/.

PEYRÉ, G., AND COHEN, L. 2005. Geodesic computations for fast and accurate surface remeshing and parameterization. *Progress in Nonlinear Differential Equations and Applications 63*.

PILGWAY INC., 2010. 3D-Coat 3.3, July. http://www.3d-coat.com.

PIXOLOGIC, INC., 2011. ZBrush 3.5R3, July. http://www.pixologic.com/zbrush/.

POPA, T., JULIUS, D., AND SHEFFER, A. 2006. Material-aware mesh deformations. In *Proc. SMI '06*.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Proceedings of SIGGRAPH 84*, vol. 18, 253–259.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 24*, 3, 626–633.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, 465–470.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1991. *The Algorithmic Beauty of Plants*. Springer.

RAITT, B., AND MINTER, G. 1998. Digital sculpture techniques: How to apply the principles of traditional sculpture to make stunning 3d characters. *Interactivity magazine* (August).

REQUICHA, A. A. G., AND VOELCKER, H. B. 1983. Solid modeling: Current status and research directions. *IEEE Comp. Graph. & Appl. 3*, 25–37.

RICCI, A. 1973. A constructive geometry for computer graphics. *Computer Graphics Journal 15*, 2, 157–160.

ROBERT MCNEEL & ASSOCIATES, 2010. Rhino 4.0, July. http://www.rhino3d.com/.

RUSTAMOV, R. 2010. Barycentric coordinates on surfaces. *Computer Graphics Forum 29*, 5. Proc. SGP 2010.

RVACHEV, V. L. 1963. On the analytic description of some geometric objects. *Reports of the Ukrainian Acadamey of Sciences*, 153, 765–767.

SATHERLEY, R., AND JONES, M. 2001. Vector-city vector distance transform. *Computer Vision and Image Understanding 82*, 3, 238–254.

SCHAEFER, S., JU, T., AND WARREN, J. 2007. A unified, integral construction for coordinates over closed curves. *Comput. Aided Geom. Des. 24*, 8-9, 481–493.

SCHMIDT, R., AND SINGH, K. 2008. Sketch-based procedural surface modeling and compositing using Surface Trees. *Computer Graphics Forum 27*, 2, 321–330. Proceedings of Eurographics 2008.

SCHMIDT, R., AND SINGH, K. 2010. Conformal parameterization of point-sampled surfaces. Tech. Rep. CSRG-605, Dept. Computer Science, University of Toronto.

SCHMIDT, R., AND SINGH, K. 2010. Drag-and-drop surface composition. Tech. Rep. CSRG-604, Dept. Computer Science, University of Toronto.

SCHMIDT, R., AND SINGH, K. 2010. Drag, drop, and clone: An interactive interface for surface composition. Tech. Rep. CSRG-605, Dept. Computer Science, University of Toronto.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2005. Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 53–62.

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics 25*, 3, 605–613.

SCHMIDT, R., ISENBERG, T., JEPP, P., SINGH, K., AND WYVILL, B. 2007. Sketching, scaffolding, and inking: A visual history for interactive 3d modeling. In *Proc. NPAR '07*, 23–32.

SCHMIDT, R. 2006. *Interactive Modeling with Implicit Surfaces.* Master's thesis, University of Calgary.

SCHNEIDER, R., AND KOBBELT, L. 2001. Mesh fairing based on an intrinsic PDE approach. *Computer-Aided Design 33*, 11, 767–777.

SCHNEIDER, J., GEORGII, J., AND WESTERMANN, R. 2009. Interactive geometry decals. In *Proc. Intl. Conf. on Vision, Modelling, and Visualization (VMV)*.

SCHRÖDER, F. 1995. ape – the original dataflow visualization environment. *SIGGRAPH Comput. Graph. 29*, 2, 5–9.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4, 151–160.

SHAH, J., AND MANTYLA, M. 1995. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. Wiley-Interscience.

SHAMIR, A. 2008. A survey on mesh segmentation techniques. *Computer Graphics Forum 28*, 6, 1539–1556.

SHAPIRA, L., AND SHAMIR, A. 2009. Local geodesic parametrization: An ant's perspective. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration, (Series: Mathematics and Visualization)*, Springer, T. Moller, B. Hamann, and R. Russel, Eds.

SHAPIRO, V., AND VOSSLER, D. 1995. What is a parametric family of solids? In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, 43–54.

SHARF, A., ALEXA, M., AND COHEN-OR, D. 2004. Context-based surface completion. In *Proc. SIGGRAPH '04*, 878–887.

SHARF, A., BLUMENKRANTS, M., SHAMIR, A., AND COHEN-OR, D. 2006. Snappaste: an interactive technique for easy mesh composition. *Vis. Comput. 22*, 9, 835–844.

SHEFFER, A., PRAUN, E., AND ROSE, K. 2006. Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis. 2*, 2, 105–171.

SHEWCHUK, J. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Lecture Notes in Computer Science*, vol. 1148. Springer-Verlag, 203–222.

SIDE EFFECTS SOFTWARE INC., 2007. Houdini 9, September. http://www.sidefx.com.

SIFAKIS, E., SELLE, A., ROBINSON-MOSHER, A., AND FEDKIW, R. 2006. Simulating speech with a physics-based facial muscle model. In *Proc. SCA '06*, 261–270.

SIMARI, P., AND SINGH, K. 2005. Extraction and remeshing of ellipsoidal representations from mesh data. In *Proc. Grpahics Interface*.

SIMARI, P., KALOGERAKIS, E., AND SINGH, K. 2006. Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In *Proc. Eurographics Symposium on Geometry Processing*.

SINGH, K., AND FIUME, E. 1998. Wires: a geometric deformation technique. In *Proceedings of SIGGRAPH '98*, 405–414.

SINGH, K., AND KOKKEVIS, E. 2000. Skinning characters usnig surface oriented free-form deformations. In *Proc. Graphics Interface 2000*.

SINGH, K., PEDERSEN, H., AND KRISHNAMURTHY, V. 2004. Feature based retargeting of parameterised geometry. In *Proc. IEEE Geometric Modeling & Processing 2004*, 163–172.

SIQUEIRA, M., XU, D., GALLIER, J., NONATOAND, L. G., MORERA, D. M., AND VELHO, L. 2009. A new construction of smooth surfaces from triangle meshes using parametric pseudo-manifolds. *Comput. Graph. 33*, 3, 331–340.

SIU, S., AND MANN, S. 2003. Computer aided ferret design. In *Proc. Geometric Modeling and Graphics*, 195–200.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proc. Symp. Geom. Proc.*, 175–184.

SUGIHARA, M., WYVILL, B., AND SCHMIDT, R. 2010. WarpCurves: A tool for explicit manipulation of implicit surfaces. *Computers & Graphics 34*, 3, 282–291. Proc. Shape Modeling International (SMI) 2010.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3.

SURAZHSKY, V., SURAZHSKY, T., KIRSANOV, D., GORTLER, S. J., AND HOPPE, H. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3, 553–560.

SUZUKI, H., SAKURAI, Y., KANAI, T., AND KIMURA, F. 2000. Interactive mesh dragging with an adaptive remeshing technique. *The Visual Computer 16*, 3-4, 159–176.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. SIGGRAPH '95*, 351–358.

THE BLENDER FOUNDATION, 2010. Blender 2.49b, July. http://www.blender.org.

TRIPLE SQUID SOFTWARE DESIGN, 2010. Moment of Inspiration 2.0, July. http://www.moi3d.com/.

TSANG, C., AND MANN, S. 1998. Animated surface pasting. Tech. Rep. CS-98-19, University of Waterloo.

TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 99*, 335–342.

VARADY, T., MARTNI, R., AND COX, J. 1997. Reverse engineering fo geometric models - an introduction. *Computer Aided Design 29*, 4, 255–268.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.

WELCH, W., AND WITKIN, A. 1992. Variational surface modeling. In *Proc. SIGGRAPH '92*, 157–166.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proceedings of SIGGRAPH 94*, 247–256.

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data Structure for Soft Objects. *The Visual Computer 2*, 4, 227–234.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Comp. Graph. Forum 18*, 2, 149–158.

XU, K., ZHANG, H., COHEN-OR, D., AND XIONG, Y. 2009. Dynamic harmonic fields for surface processing. *Computers and Graphics 33*, 391–398.

XU, W., WANG, J., K.YIN, ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. *ACM Trans. Graph. 28*, 3, 1–9.

YING, L., AND ZORIN, D. 2004. A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. Graph. 23*, 3, 271–275.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZELINKA, S., AND GARLAND, M. 2004. Similarity-based surface modelling using geodesic fans. In *Proceedings of the Eurographics Symposium on Geometry Processing*, 209–218.

ZIMMERMANN, J., NEALEN, A., AND ALEXA, M. 2008. Sketching contours. *Comp. & Graph. 32*, 3, 486–499.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proc. SIGGRAPH 97*, 259–268.