FITTING SMOOTH SURFACES TO DENSE POLYGON MESHES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Venkat Krishnamurthy
June 2000

Abstract

Recent progress in acquiring shape from range data permits the acquisition of seamless million-polygon meshes from physical models. While dense polygon meshes are an adequate representation for some applications, many users prefer smooth surface representations for reasons of compactness, control, manufacturability, or appearance. In this thesis, we present algorithms and an end-to-end software system for converting dense irregular polygon meshes of arbitrary topology into tensor product B-spline surface patches with accompanying displacement maps. This choice of representation yields a coarse but efficient model suitable for interactive modification and animation and a fine but more expensive model suitable for rendering.

The first step in our process consists of interactively painting patch boundaries onto the polygonal surface. In many applications, the placement of patch boundaries is considered part of the creative process and is not amenable to automation. We present efficient techniques for representing, creating and editing surface curves on dense polygonal surfaces.

The second step in our process consists of finding a gridded resampling of each bounded section of the mesh. Our resampling algorithm lays a grid of springs across the polygon mesh, then iterates between relaxing this grid and subdividing it. This grid provides a parameterization for the mesh section, which is initially unparameterized. Our parameterization algorithm is automatic, efficient, and robust, even for very complex polygonal surfaces. Prior algorithms have lacked one or more of these properties, making them unusable for dense meshes. Our parameterization strategy also provides the user a flexible method to design parameterizations - an ability that previous literature in surface approximation does not address.

The third and final step of our process consists of fitting a hybrid of B-spline surfaces and displacement maps to our gridded re-sampling. This fitting method reproduces with high fidelity the geometry of the original polygon mesh. The displacement map is an image representation of the error between the fitted B-spline surfaces and our spring grid. Since displacement maps are just images our hybrid representation facilitates the use of image processing operators for manipulating the geometric detail of an object. They are also compatible with modern photo-realistic rendering systems.

The resampling and fitting steps of our process are fast enough to surface a million polygon mesh in under 10 minutes - important for an interactive system.

Acknowledgements

I would like to extend my thanks to the many people who have contributed to this thesis as well as to those who have provided me intellectual and moral support through my doctoral stint at Stanford.

First, I would like to extend my sincere gratitude and thanks to Marc Levoy, who as my advisor at Stanford expertly (and patiently) taught me the many elements that go into conducting stimulating academic research. Marc's infectious enthusiasm for Computer Graphics has been a constant source of inspiration to me through my stay at Stanford.

I would like to thank Pat Hanrahan and Leo Guibas for serving on my reading committee and for their many helpful comments and suggestions on my work. I would also like to thank Jean Claude Latombe and Stephen Boyd for serving on my orals committee and for providing several useful observations about this dissertation.

Next, I would like to thank my colleagues at the Stanford Graphics Lab for providing an intellectually stimulating and enjoyable research environment. In particular, I would like to express my thanks to Brian Curless, Greg Turk, Hans Pedersen, James Davis and Bill Lorensen for several thought-provoking and useful discussions. I would also like to extend a special thanks to Tamara Munzner for her invaluable help with my various video presentations.

Many thanks to David Addleman of Cyberware for the use of a scanner and to Lincoln Hu, Christian Rouet, Zoran Kacic-Alecic (of Industrial Light and Magic), Domi Piturro (of Synthesis Studio), Rob Letterman (of Dreamworks SKG), and George Dabrowski (of Cyberware), all of whom, at various times during my doctoral work, tested my software solutions on real-world problems and provided valuable feedback and suggestions for improvements.

My graduate work was sponsored by the National Science Foundation under contract CCR-9157767 and by Interval Research Corporation. My thanks to both organizations.

Special thanks to Amy for her unflagging support, patience and encouragement as I worked the long hours and weekends to complete this dissertation.

Special thanks also, to my parents Gita and Krishnamurthy and my sister Supriya, who have provided equal measures of love, advice and moral encouragement throughout my career. I dedicate this dissertation to them.

Contents

List of Figures

Chapter 1

Introduction

Advances in range image acquisition and integration allow us to compute geometrical models from complex physical models [Curless & Levoy 1996, Turk & Levoy 1994]. The output of these technologies is a dense, seamless (i.e. manifold) irregular polygon mesh of arbitrary topology. For example, the model in figure 1.1, generated from 75 scans of an action figure using a Cyberware laser range scanner, contains 350,000 polygons. Models like this offer new opportunities to modelers and animators in the CAD and entertainment industries.

Dense polygon meshes, such as the one shown in the figure, are an adequate representation for some applications such as stereolithographic manufacturing [Curless & Levoy 1996] or computer renderings. However, users in a number of application domains prefer smooth surface representations over irregular polygon meshes. By smooth surfaces we mean surfaces that have an underlying, higher order mathematical structure such as the existence of a global analytical derivative. In contrast to smooth surface representations, polygonal meshes are just a set of connected planar facets; they do not posses an analytical derivative.

Smooth surface representations offer useful advantages over an irregular polygonal mesh representation. Some of these advantages are:

- Smooth appearance: Several applications such as consumer product design require for aesthetic reasons that 3-D surface models possess a smooth appearance. Polygonal meshes cannot be used in these applications because they may appear faceted

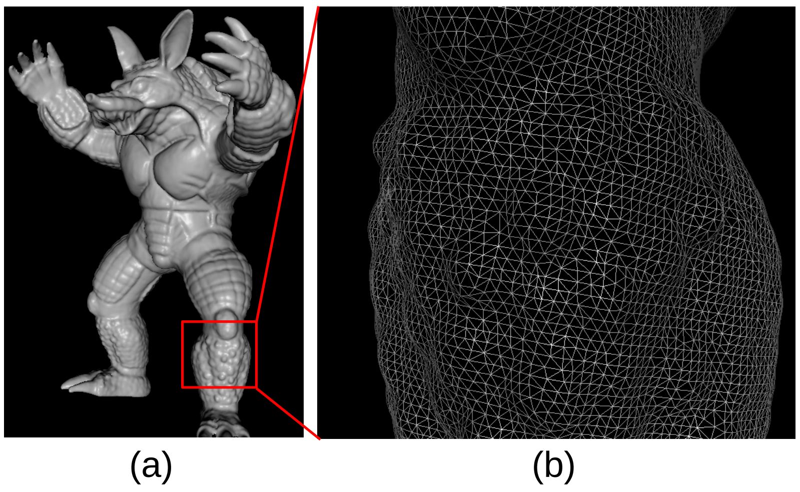

Figure 1.1. An example of a dense polygon mesh. (a) shows a polygonal mesh of an Armadillo model that is composed of over 350,000 polygons. The mesh was created by integrating about 75 range images (taken with a Cyberware laser range scanner) using the techniques of Curless et al. Note the fine surface detail present in the polygonal mesh. (b) shows a blow-up of a small region of the Armadillo's leg. Notice the density of the polygons and the irregular structure of the mesh. Note also that the mesh is a piecewise constant reconstruction i.e. it is composed of flat triangular facets.

(unless the polygons are made extremely small, which increases the expense of processing and storing the model).

- Compact representation: A smooth surface representation can usually represent complex surface shapes more efficiently than polygonal meshes.

- Flexible control: Smooth surface representations usually offer an easier interface to design, control and modify surface geometry and texture.

- Mathematical differentiability: Several applications use computational procedures that require the surface to be everywhere differentiable or curvature continuous (e.g. finite element analysis). For such applications, polygonal meshes cannot be used because they are merely piecewise linear surfaces.

- Manufacturability: Some manufacturing procedures such as CNC milling require a smooth surface representation to create high quality results.

- Hierarchical modeling: Creating manipulable hierarchies from smooth surfaces is a significantly simpler task than doing the same with dense, irregular, polygonal meshes.

Examples of smooth surfaces include parametric representations such as NURBS, B-spline and Bezier surfaces, implicit representations such as spheres and cylinders, algebraic representations based on explicit equations, and so on. To satisfy users that prefer smooth surface representations, techniques are needed for fitting smooth surfaces to dense polygonal meshes.

A notable property of our dense polygonal meshes is the fine surface detail present in the data (see for example the model in figure 1.1). This is due to both the accuracy of new digitizing techniques as well as the ability of newer integration techniques [Curless & Levoy 1996, Turk & Levoy 1994] to retain this fine detail in the polygon mesh reconstruction. Whatever fitting technique we employ should strive to retain this fine detail. Surprisingly, a unified surface representation may not be the best approach. First, the heavy machinery of most smooth surface representations (for example B-spline surfaces) makes them an inefficient way to represent fine geometric detail. Second and perhaps more important, although geometric detail is useful at the rendering stage of an application (e.g. for animation and game applications), it may not be of interest to the modeler. Moreover, its presence may degrade the time or memory performance of the modeling system. For these reasons, we believe it is advantageous to separate the representations of coarse geometry and fine surface detail.

Within this framework, we may choose from among many representations for these two components. For representing coarse geometry, modelers in the entertainment and CAD industry have long used NURBS [Farin 1990] and in particular uniform tensor product B-spline patches. In order to address their needs we have chosen uniform tensor product B-splines as our smooth surface representation. Following a standard practice in industry, our smooth surface models consist of a network of tensor product B-spline patches that are stitched together at the patch boundary curves. For most applications, the placement of these patch boundary curves is part of the creative process and not easily automatable. Therefore, in our solution the user specifies the patch boundaries.

For representing surface detail, we propose using displacement maps. The principal advantage of this representation is that displacement maps are essentially images. As such, they can be processed, retouched, compressed, and otherwise manipulated using simple image processing tools. Some of the effects shown in chapter 7 were achieved using Adobe Photoshop, a commercial photo retouching program.

In this thesis we present algorithms to fit a hybrid of smooth surfaces and displacement maps to dense polygonal meshes. Our surface fitting algorithms are efficient and have been used to create smooth surfaces at a higher complexity and quality than previous approaches. Furthermore, the hybrid surface representation of B-splines and displacement maps is a flexible one and as such allows for editing operations that would be hard to duplicate with existing unified smooth surface representations. We have also implemented an end-to-end surface fitting software system based on the algorithms developed in this thesis. In our discussions we will often draw on our software system to explain several practical aspects of our solution.

In the following section, we describe some of the uses and applications of solving the surface fitting problem described above (section 1.1). Next, in section 1.2 we analyze some important characteristics of our specific surface fitting problem. In section 1.3 we describe our surface fitting pipeline. In section 1.4 we list the principal contributions of this thesis. Finally, we end this chapter with an outline of the rest of this thesis.

1.1 Applications

Algorithms and practical tools for converting dense polygonal meshes to smooth surfaces have applications in a number of fields including entertainment, industrial design, reverse engineering, the web, medicine, and manufacturing.

1.1.1 Entertainment

Three dimensional models in movies, games and broadcast television programs use smooth surfaces or structured polygonal meshes (which are easily derived from smooth surfaces). Artists in this domain often sculpt physical maquettes of intended 3-D computer models to define the unique characteristics of the model and to get a better intuition for the model's shape. Current practice in this industry consists of employing touch probes and similar manual techniques to input these models into the computer. Not surprisingly, these processes are tedious and time-consuming. For these domains, the ability to create smooth surface models from laser scanned physical objects offers a quick and powerful alternative for model creation.

1.1.2 Industrial design

Designers in the industrial design community use parametric surfaces (almost exclusively) to create the exterior skin of consumer products such as audio speakers and perfume bottles. A large subset of this community consists of car styling applications. In this case, both the interior and exterior of cars are often built as physical mockups. Subsequently, time consuming processes such as touch probing are used to create computer models from the physical mockups. In these and similar domains, the ability to create complex smooth surface models from physical models has the potential of cutting design time and permitting the creation of models of a greater complexity than is possible using current methods.

1.1.3 Reverse engineering and inspection

Techniques that convert scanned data to smooth surfaces have obvious applications in reverse engineering. Specific examples include verifying the dimensions of parts against existing computer models and creating computer models of parts for which no such models yet exist. Interestingly, the laser scanning of objects is becoming increasingly common in this domain. Unfortunately, existing software tools that process the resulting data tend to have limited functionality and are usually manually intensive (see chapter 2 for details).

1.1.4 Medicine

In the medical industry, Computer Tomography (CT) and Magnetic Resonance Imaging (MRI) technologies are currently used to create both volumes and dense polygonal meshes from sequences of cross-sectional data. These computer models are employed mainly for visualization and diagnosis purposes. To enable the use of these models for a wider range of applications, such as the finite element analysis of bone structure or soft tissue deformation analysis, they must be converted to smooth surface representations.

1.1.5 Manufacturing

Certain manufacturing technologies such as CNC milling require smooth surface descriptions in order to produce high quality output. These technologies are widely used in industries such as manufacturing of car body exteriors, mechanical parts, toys, consumer goods etc. The ability to convert dense polygonal models to smooth surfaces permits the manufacture of these objects using these sophisticated manufacturing technologies.

1.2 Problem characteristics

Before we discuss our surface fitting solution in greater detail, it is instructive to examine some characteristics of our applications and of our input data. These characteristics have significant implications for any surface fitting solution. Surprisingly many existing solutions to surface fitting problems overlook one or more of these characteristics and as a result use inefficient algorithms or produce poor or simply un-usable smooth surfaces.

1.2.1 Application characteristics

A significant characteristic of our surface fitting applications is: **there is usually a non-automatable, creative component intrinsic to the process of smooth surface creation.** An example of a non-automatable component of the surface fitting process (when parametric surfaces are used) is the placement of patch boundary curves. This non-automatibility has three implications for any surface fitting solution:

- First, a completely automated surface fitting algorithm is not acceptable.
- Second, a good solution should provide efficient and intuitive tools to the user to provide the creative input.
- Third, a good solution should provide real-time feedback at every stage of the surface fitting process to provide the user as much control as is needed over the final solution.

1.2.2 Input data characteristics

Four important characteristics of our input data are:

1. **The input data is of known topology**. The fact that the data set is a polygonal mesh rather than just a cloud of points implies that a surface description already exists for the data set i.e. the data set is of known topology. Any algorithms and procedures we develop should strive to use this existing topological information in the smooth surface creation process. Several commercial and research systems (enumerated in section 2.3) work exclusively with point clouds. When presented with polygonal meshes such systems choose to discard the connectivity information provided by the polygonal mesh and convert the input once again to point clouds. This approach tends to be inefficient and compromises the quality of the resulting smooth surfaces.

2. **The input data has arbitrary topology**. This implies that it is not possible to solve our surface fitting problem using algorithms that work on height fields or objects of otherwise restricted topology. While there is an extensive literature on fitting surfaces to objects of restricted topology, these techniques are not general enough to work on arbitrary topology data and are therefore not applicable to our problem.

3. **The input data is dense**. This implies that our algorithms must be efficient if they are to provide a practical surface fitting solution. Several techniques in the literature are practical only on small polygonal meshes. Such techniques are not applicable to our problem.

4. **The input data possesses fine geometric detail**. This implies that our surface fitting algorithms must be capable of capturing this fine surface detail. Several fitting algorithms in the literature produce excessively smoothed out results. Such algorithms have traditionally been used for approximating noisy (or un-reliable) input data (see chapter 2 for a discussion of related work). While it is appropriate to use such algorithms for approximating noisy data, they are not acceptable for our surface fitting problem.

1.3 Surface fitting pipeline

We have incorporated the insights from the preceeding discussion into our surface fitting solution. In the following discussion, we present an overview of our solution in the form of a surface fitting pipeline which accepts as its input a dense polygon mesh and creates as its output a hybrid of B-spline surfaces patches and associated displacement maps. Figure 1.2 shows our surface fitting pipeline. The steps are as follows:

1. Interactive boundary curve painting, wherein a modeler defines the boundaries of a number of patches. This is accomplished with tools that allow the painting and editing of curves directly on the surface of the unparameterized polygonal model. Our tools employ both traditional space curve based editing paradigms (e.g. editing with a B-spline space curve) as well as novel surface curve based editing paradigms. For surface curve based editing we propose the use of *surface snakes*: a surface curve formulation for dense unparameterized polygonal meshes. The connectivity of the polygon mesh allows the use of local graph search algorithms to make curve painting and editing operations rapid. This property is useful when a modeler wishes to experiment with different boundary curve configurations for the same model. Each region of the mesh that a B-spline surface must be fit to is called a *polygonal patch*. Since patch boundaries have been placed for artistic reasons, polygonal patches are not constrained to be height fields. Our only assumptions about them are that each is a rectangularly parameterizable piece of the surface without holes. In this step, the user can also (optionally) specify one or more *feature curves* per polygonal patch (see step 2 below). These are to serve as constraints to the second (parameterization) stage of our pipeline.

2. Generate a gridded resampling for each polygonal patch. This is accomplished by an automatic coarse-to-fine resampling of the patch, producing a regular grid that is constrained to lie on the polygonal surface. We call this grid the *spring mesh*. Its purpose is to establish a parameterization for the unparameterized polygonal patch. Our resampling algorithm is a combination of relaxation and subdivision steps that iteratively refine the spring mesh at a given resolution to obtain a better sampling of

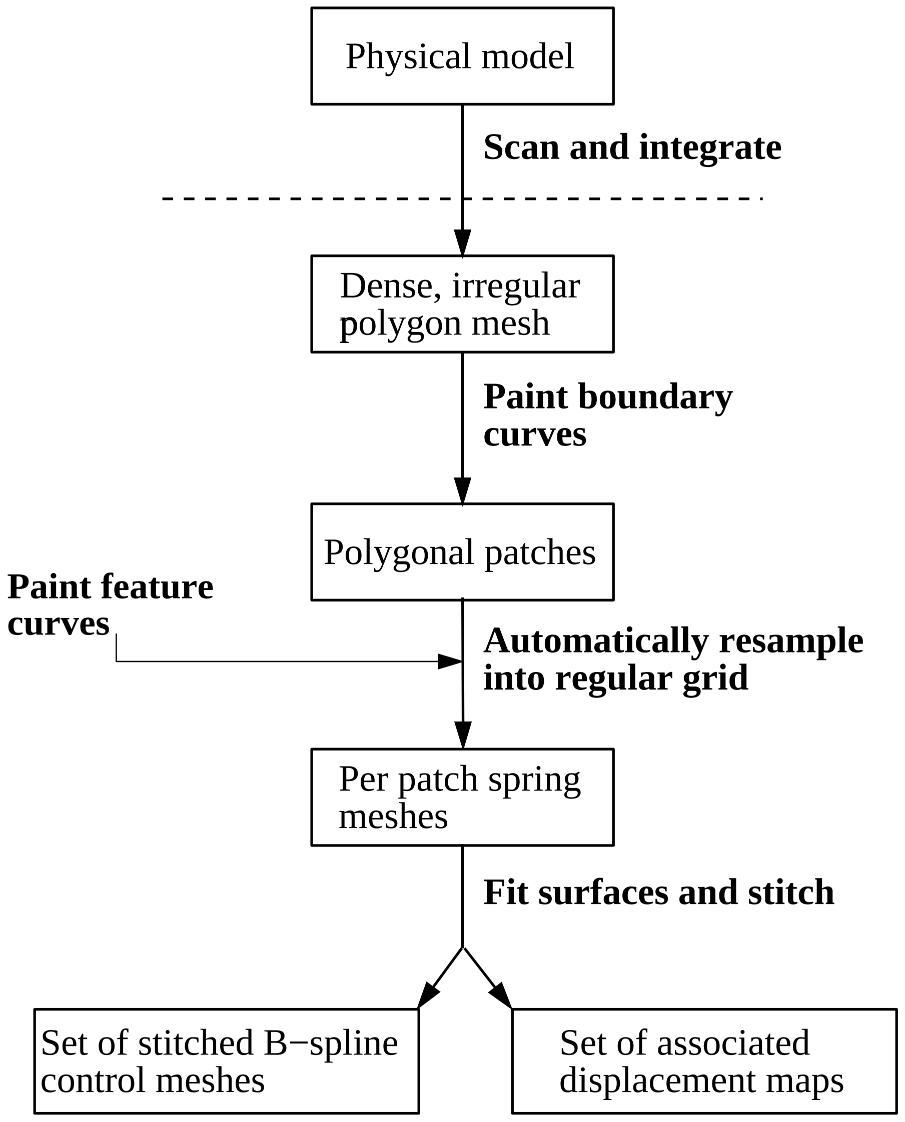
Physical model
Scan and integrate
Dense, irregular polygon mesh
Paint boundary curves
Polygonal patches
Paint feature curves
Automatically resample into regular grid
Per patch spring meshes
Fit surfaces and stitch
Set of stitched B−spline control meshes
Set of associated displacement maps

Figure 1.2. Our surface fitting pipeline. The input to our system is a dense irregular polygon mesh. In the first step, boundary curves for the desired spline patches are painted on the surface of the unparameterized polygonal model. These curves can be edited either with space curves based tools or with *surface snake* based tools. The output of this step is a set of bounded mesh regions that we call *polygonal patches*. Next, we perform an automated resampling of this polygonal patch to form a regular grid that lies on the surface of the polygonal patch. We call this regular grid a *spring mesh*. The user can supply a set of constraints called *feature curves* to further guide the structure of the spring mesh. Feature curves may be edited using the same set of tools used for editing boundary curves. Feature curves are interpolated during the resampling of the polygonal patch. In the final step, we fit surfaces to the spring mesh and output both a B-spline surface representation and a set of associated displacement maps to capture the fine detail.

the underlying polygonal patch. This refinement is explicitly directed by distortion metrics relevant to the spline fit. In cases where feature curves are specified, the parameterization is further constrained to interpolate them. The output of this step is a set of spring meshes to which surfaces will now be fit. The original polygon mesh is no longer needed and may be discarded.

3. We now use gridded data fitting techniques to fit a B-spline surface to the spring mesh corresponding to each polygonal patch. The output of this step is a set of B-spline patches representing the coarse geometry of the polygonal model. To represent fine detail, we also compute a displacement map for each patch as a resampling of the difference between the spring mesh and the B-spline surface. This regular sampling can conveniently be represented as a vector (rgb) image which stores a 3-valued displacement at each sample location. Each of these displacements represents a perturbation in the local coordinate frame of the spline surface. This image representation lends itself to a variety of interesting image processing operations such as compositing, painting, edge detection and compression. An issue in our technique, or in any technique for fitting multiple patches to data, is ensuring continuity between the patches. We use a combination of knot line matching and a stitching post-process which together give us G^1 continuity everywhere. This solution is widely used in the entertainment industry.

1.4 Contributions

This thesis has four principal contributions. First, we present a flexible formulation for representing surface curves on dense, unparameterized, polygonal surfaces. We call this the *surface snake* formulation. Prior work on surface curves has assumed the existence of an underlying smooth surface representation. Our formulation is computationally efficient and offers an intuitive paradigm for manipulating curves on dense polygon meshes. In the context of our surface fitting pipeline, we use this surface snake formulation for the specification and subsequent editing of boundary and feature curves. We demonstrate several surface curve editing operations using this formulation.

The second main contribution of this thesis is a coarse-to-fine parameterization strategy for dense polygonal surfaces. In the context of our surface fitting pipeline, this strategy is used to create a spring mesh resampling of each polygonal patch. The algorithm underlying our parameterization strategy is automatic, efficient, and robust, even for very complex polygonal surfaces. Prior algorithms have lacked one or more of these properties, making them unusable for meshes having hundreds of thousands of polygons. We also demonstrate that our parameterizations are of a higher quality than prior approaches. Since these parameterizations are eventually used for surface fitting purposes, our approximating B-spline surfaces are consequently of superior quality.

Our parameterization strategy also gives the user a flexible method to design parameterizations - an ability that previous literature in surface approximation does not address. In our system, the user can specify a number of constraints to the parameterization (i.e. *feature curves*) and the algorithm interpolates these constraints during the coarse-to-fine parameterization process. Feature curves enable the user to create the customized parameterizations required by many applications.

The third main contribution of this thesis is our hybrid surface fitting strategy. In particular, we fit a combination of B-spline surfaces (for coarse geometry) and displacement maps (for fine geometric detail) to our dense meshes. Our results reproduce with high fidelity the geometry of the original polygon mesh. Furthermore, our hybrid surface representation offers a flexible interface to manipulate the geometry of the model. We demonstrate this flexibility by implementing a number of surface editing operations that would have been difficult to achieve using a unified surface representation.

The fourth main contribution of this thesis is a practical end-to-end, interactive surface fitting system. The output produced by our system is in a usable form for most animation and design applications. Using this system users can create models of a very high complexity from scanned data.

1.5 Organization of this thesis

The remainder of this thesis is organized as follows. Chapter 2 reviews relevant previous work and places our work in the context of other surface interpolation and approximation schemes.

In chapter 3 we describe techniques for representing, painting and manipulating curves on polygonal meshes. We develop the surface snake formulation and demonstrate its uses in performing curve editing operations directly on the polygonal surface. These curve painting and editing tools are used to create patch boundary curves as well as feature curve constraints to parameterizations.

In chapter 4 we motivate and introduce our two-stage surface fitting strategy: first, the creation of a gridded resampling (the spring mesh) of each polygonal patch, and second, a least squares fitting to each spring mesh. We compare our fitting strategy to previous fitting schemes and explore the relative advantages and disadvantages of our approach. The particulars of the two-stage fitting strategy are explained in chapters 5 and 6. Chapter 5 first presents our coarse-to-fine, polygonal patch resampling algorithm. In this chapter we discuss both the basic unconstrained parameterization algorithm as well as methods to constrain and guide this parameterization using feature curves. This algorithm generates a spring mesh resampling per patch. Chapter 6 then explains our algorithms to create B-spline surfaces from the spring mesh.

In chapter 7 we describe our strategy for extracting displacement map from the scanned data and describe several methods of interacting with our hybrid representation. We explore some interesting applications of displacement maps and demonstrate surface editing operations that would be hard to duplicate with just a smooth surface representation.

In the presence of multiple patches, it is important to ensure continuity at patch boundaries. Chapter 8 discusses techniques for ensuring continuity for both the traditional B-spline surface representation as well as displacement maps. Finally, in chapter 9 we summarize the contributions of this thesis and discuss avenues for future work opened up by our research.

Throughout this thesis we will draw on examples from the entertainment industry. However, our techniques are generally applicable. Portions of this thesis are also described in [Krishnamurthy & Levoy 1996].

Chapter 2

Prior Work

There is a large literature on surface fitting techniques in the CAD, computer vision and approximation theory fields. We begin this chapter with an classification and overview of surface approximation problems (section 2.1) in a general setting. In the following sections we then discuss in depth a selection of these approximation solutions that are related to the work advanced by this thesis.

2.1 Surface approximation problem in a general setting

Surface fitting approaches can be broadly classified based on the following three criteria:

- Type of input data.
- Fitting method.
- Type of output surface.

Each of these three criteria can be further classified based on a number of sub-criteria. For example, the input data can be differentiated on the basis of topology (height fields or arbitrary topology), measurement accuracy (noisy or highly accurate) or the presence of explicit curve and surface constraints. Fitting methods can be further classified on the basis of fidelity of fit (e.g. approximation vs interpolation methods), mathematical measures of error (e.g. least squares error, moving least median of squares etc.), robustness to different kinds of input data (e.g. to non-uniform distributions of data, to outliers etc.), and so on. Output surfaces can be further classified as parametric (e.g. B-spline, Bezier), implicit and algebraic (e.g. quadrics, blobbies), subdivision (e.g. Catmull Clark, Doo Sabin), etc.

Each combination of these sub-categories corresponds to a distinct surface fitting problem. As can be expected, the above categorization spans a large body of work in a variety of application domains. A full description of these surface fitting problems and attempted solutions is beyond the scope of this thesis. We will instead focus here only on a subset of these fitting problems that provide useful insights into our own approximation problem.

Fitting method / Input data type	Geometric Interpolation	Geometric Approximation	Variational Interpolation
Height fields	Franke 1982 Sclaroff 1991	Schumaker 1979 Franke 1986 Terzopolous 1988 Forsey 1991	Celniker 1991
Arbitrary topology point clouds		Hoppe 1994 Bajaj 1995 Milroy 1995	Moreton 1992
Arbitrary topology polygon meshes	Reeves 1990 Lounsbery 1990 Loop 1994 Grimm 1995	Eck 1996 Krishnamurthy 1996 (This thesis)	Halstead 1993
Arbitrary topology mix of point clouds and curves			Welch 1994

Figure 2.1. A classification of surface fitting problems. This table organizes work in the field of surface fitting by type of input data and surface fitting method. See the text for a detailed explanation of this figure.

Figure 2.1 provides a more compact classification in the form of a table of surface fitting methods relevant to this thesis. We have organized this table as a cross product of four types of input data and three kinds of fitting techniques. The input data is classified on the basis of topology into the following categories:

- Height fields: the input data may be viewed as a set of z offsets from the xy plane. The data can either be a set of scattered points or there might be connectivity information included.
- Arbitrary topology point clouds: the input data is a set of unconnected points in space. The intended surface may be of arbitrary topology
- Arbitrary topology polygon meshes: the input data is a set of connected points forming an arbitrary topology manifold surface.
- Arbitrary topology mix of point clouds and curves: the input data is mix of unconnected curves and points. The intended surface may be arbitrary topology.

For each of the above kinds of data there are two fitting options:

1. Interpolation: the computed smooth surface passes through every data point.
2. Approximation: the computed surface does not necessarily pass through the data points. Instead it only approximates the data (i.e. passes near them).

Independent of these two options there are two kinds of surface fitting strategies:

1. Geometric: choose a specific geometric primitive (e.g. Bezier, Catmull-Rom, B-spline, etc.) as the smooth surface representation and approximate the data with this specific primitive.
2. Variational: formulate the approximating surface as the minimal energy solution to a corresponding variational problem [Weinstock 1974]. In this case the choice of a specific smooth surface element is incidental to the solution.

Variational methods are fundamentally different from geometric methods both in their solution strategy and in the relative advantages they offer. For example, it is usually easier to specify fairly complex curve constraints as part of a variational procedure [Welch & Witkin 1994].

Based on the above classification there are four fitting methods: geometrical interpolation and approximation and variational interpolation and approximation. For each combination of input data type and fitting method the appropriate box in the table lists a selection of relevant prior work. Combinations where there has been no previous work (to our knowledge) have been crossed out from the table. Note also that we have omitted entirely the column corresponding to variational approximation. We are not aware of prior work that successfully addresses this problem domain. The entries from table 2.1 that are of most relevance to our problem space are the approaches that create geometric approximations of arbitrary topology point clouds and polygon meshes.

In the remaining sections of this chapter, we provide an overview of each of the combinations discussed in the table. We will organize our discussion as follows: we begin by discussing geometric interpolation and approximation techniques for the various kinds of input data (sections 2.2, 2.3 and 2.4). Next in section 2.5 we discuss variational interpolation strategies. We conclude this chapter with a discussion of a selection of relevant work in the texture mapping domain 2.6.

2.2 Height fields

The problem of fitting surfaces to height field data arises fairly often in a number of problem domains ranging from visible surface determination [Terzopoulos 1988] to computer animation [Forsey & Bartels 1991] and free from surface design [Celniker & Gossard 1991].

2.2.1 Geometric Interpolation

Geometric interpolation to measured data is appropriate (practical) when the data has been measured with high accuracy and is manageable in size. Franke's survey on the subject of scattered data interpolation provides a good overview to techniques from a number of application domains [Franke 1982]. A specific example of interpolating to gridded data is the work of Sclaroff et al [Sclaroff & Pentland 1991]. The authors interpolate cylindrical face scans with generalized super-quadric surfaces. Since super-quadrics don't have enough degrees of freedom to interpolate an arbitrary height field, displacement maps are used to interpolate the gridded data set. While the techniques presented produced effective results for height fields, they cannot interpolate data sets of arbitrary topology.

2.2.2 Geometric Approximation

When the input height field is noisy or inaccurate, approximation techniques are preferable to interpolation. Schumaker's classic survey [Schumaker 1979] on the subject of approximating scattered data provides a good introduction to a large number of popular approximation methods. Another useful survey in this domain is Franke's comprehensive bibliography of both classical and recent approximation methods [Franke & Schumaker 1986] for scattered data.

Terzopoulos's work in the area of visible surface representation is a typical example of the problem of approximating to noisy height field data [Terzopoulos 1988]. In this particular problem, the input data is obtained from stereo image pairs using computer vision techniques. Terzopoulos proposes a controlled continuity spline surface representation for approximating this kind of data. The controlled continuity formulation introduces several smoothing terms during the approximation process to smooth out the noise in the input. Recent work in this field has focussed mainly on various enhancements to this basic strategy [Sinha & Schunck 1992].

Geometric approximation is also appropriate when the height field data is accurate but is too dense to be interpolated without recourse to surfaces of exceedingly high order. The work of Forsey et al. [Forsey & Bartels 1991] and Schmitt et al [Schmitt et al. 1986] are typical examples of solution strategies for this kind of input data. Forsey uses a hierarchical B-spline [Forsey & Bartels 1988] surface representation while Schmitt uses a set of Bezier surface patches as their smooth surface representation. These approximation techniques perform effectively for scattered height field data or regular grids of points but are not applicable for fitting data sets of arbitrary topology.

2.3 Arbitrary topology point clouds

In recent years there has been much interest in creating geometric approximations to point clouds of arbitrary topology. The input data set is usually obtained as the output of laser range scanners. Hoppe et al. [Hoppe et al. 1994] propose the use of subdivision surfaces [Halstead et al. 1993] for fitting point clouds of arbitrary topology. They assume that the point cloud has a uniform distribution of data points over the intended surface of the model. The solution produces smooth subdivision surface approximations through a three-stage optimization strategy that first produces a triangular mesh, then optimizes this mesh, and finally creates a subdivision surface from this optimized mesh. A similar approach is taken by Bajaj et al. [Bajaj et al. 1995]. Similar to Hoppe et al., the authors assume that the point cloud has a uniform distribution over the intended surface. Their solution works in three steps as well. First, a triangular mesh is constructed from the point cloud. Second a form of 3-D Delaunay triangulation is used to decompose space into uniform tetrahedra. The sections of the surface within each tetrahedra are approximated by Bezier patches. As a final step the approximated surface is smoothed.

Both these solutions produce high quality smooth surfaces. However, they have some shortcomings. First, they are sensitive to non-uniform distributions of data points. Second, these methods are not effective methods for capturing fine surface detail since they tend to smooth out the input data. Third, the smooth surfaces produced are either subdivision surfaces or a large number of stitched Bezier patches. Both these formats tend to be unusable in most practical applications because of the density of patches and because of the lack of flexible editing tools for these representations in popular commercial applications.

A second set of solutions that work with point clouds are manual B-spline surface fitting systems. Examples include commercial systems such as Imageware's Surfacer [Sinha & Seneviratne 1993], Delcam's CopyCAD, and the work of Milroy et al [Milroy et al. 1995]. These approaches begin by identifying a subset of points that are to be approximated. Parameterization of data points is usually accomplished by a user-guided process such as projection of the points to a manually constructed base plane or surface [Ma & Kruth 1995]. A constrained, non-linear least squares problem is then solved on this subset of the point cloud to obtain a B-spline surface for the specified region. While these solutions are widely applicable, then tend to be manually and computationally intensive. Secondly, as with the two automated solutions discussed earlier, these solutions do not satisfactorily capture surface detail. More comparisons of this category of techniques with our algorithms are supplied in chapter 4.

In general, the shortcomings of most point cloud techniques can be attributed to the fact that they fail to exploit topological information already present in the input (laser scanned) data. As demonstrated by Curless et al [Curless & Levoy 1996] and Turk et al [Turk & Levoy 1994], using this additional information can significantly improve quality of reconstruction.

2.4 Arbitrary topology polygon meshes

2.4.1 Geometric interpolation

Interpolating polygon meshes is a well researched subject in the geometric modeling literature. Typically, the meshes have a small number of vertices and the end goal is to create a piecewise smooth surface that interpolates the vertices of this network. In some cases additional information such as per-vertex normal information is provided to assist the interpolation process. See Lounsbery et al.'s survey [Lounsbery et al. 1992] for a good review of the various techniques that have been developed in this area.

An example of a use of this technique for interpolating surfaces to digitized data is the work of Pixar's animation group [Ostby 1986, Reeves 1990]. The data set in this case is obtained through the manual digitization of the polygon mesh network using a touch probe. Catmull-Rom splines are used to smoothly interpolate this data. Recent work in this field has focused on producing interpolants that generalize B-spline surfaces [Loop 1994] and interpolants that represent a single manifold surface [Grimm & Hughes 1995] instead of piecewise smooth surfaces.

The techniques in the geometrical interpolation domain generate as many (or more) patches as there are polygons. While this strategy is acceptable for small data sets, it is not a practical approach for dense polygon meshes.

2.4.2 Geometric approximation

Geometric approximation to dense polygonal meshes is the subject of this thesis. Part of our work can also be found in [Krishnamurthy & Levoy 1996]. An alternative fitting paradigm to ours is the one of Eck et al [Eck & Hoppe 1996]. The authors describe a method for fitting irregular meshes with a number of automatically placed bicubic Bezier patches. For the parameterization step, a piecewise linear approximation to harmonic maps [Eck et al. 1995] is used, and the number of patches is adjusted to achieve fitting tolerances. While this method produces high quality surfaces, it includes a number of expensive optimization steps, making it too slow for an interactive system. Furthermore, their technique does not control the specific placement of patch boundaries, rather the boundaries are automatically determined through a multi-step optimization strategy. Achieving adequate control over animations using their representation is presently an unresolved issue. Finally, their technique does not separate fine geometric detail from coarse geometry. Particularly for very dense meshes, we find this separation both useful and preferable, as already explained. We compare several specific aspects of the parameterization scheme of Eck et al. [Eck et al. 1995] with ours in greater detail in chapter 5 (section 5.7).

2.5 Variational interpolation

Variational interpolation methods have been used mostly in experimental free form design systems. The principal advantage offered by a variational interpolation method over a geometrical interpolation method is that the user can specify arbitrary point and curve constraints as inputs to the interpolation process. The variational fitting procedure automatically generates a smooth, energy minimizing surface that follows these user specified constraints that “fills in” unspecified information. In contrast the input to a geometrical interpolation method must be completely specified i.e. the fitting method cannot “fill in” the unspecified information.

Unfortunately, variational solutions to date have only been applied to problems consisting of a small number of manually specified point and surface constraints. For example, the work of Celniker et al [Celniker & Gossard 1991] interpolated a handful of points and curves that were restricted to represent height fields. This work was subsequently generalized to arbitrary topology configurations of points and curvature constraints at those points by Moreton et al [Moreton & Séquin 1992]. While the surfaces produced were of a high quality, the number of specified points was small and the optimization method used was expensive. Halstead et al [Halstead et al. 1993] generalized the last two schemes to interpolate arbitrary topology polygon meshes. However, as with other geometrical interpolation methods, the method is not suited for approximating large polygonal meshes.

Welch et al [Welch & Witkin 1994] presented a similar surface design system that was based on an arbitrary topology mix of curve and point constraints. The design techniques presented were the most flexible as well as efficient to date. However the maximum number of nodes that could be used without compromising interactivity were a few hundred. Once again, this is not a viable technique for large polygonal meshes.

While there have been attempts at variational interpolation of small data sets (as discussed above), no techniques exist yet for the variational approximation of dense data (either height fields or arbitrary topology) with or without curve constraints. The ideas advanced by this thesis take the first steps in this direction through the formulation of a variational parameterization algorithm (see section 5.12.1). However, our surface fitting method itself is based on traditional principles of geometrical approximation. True variational approximation to dense data remains an open problem for future research.

2.6 Relevant work in texture mapping

A key aspect of our method is an automatic parameterization scheme for irregular polygon meshes. As such, there are techniques in the texture mapping literature that address similar problems, notably the work of Bennis et al [Bennis et al. 1991] and that of Maillot et al [Maillot 1993]. Both of these papers present schemes to re-parameterize surfaces for texture mapping. These algorithms work well with regular data sets, such as discretized splines. However, they can exhibit objectionable parametric distortions in general [Eck et al. 1995]. Pedersen [Pedersen 1995] describes a method for texture mapping (and hence parameterizing) implicit surfaces. While the methods work well with implicit surfaces, they rely on smoothness properties of the surface and require the evaluation of global surface derivatives. Since irregular polygon meshes are neither smooth nor conducive to the evaluation of global surface derivatives, the above techniques cannot be applied to our input data sets.

Chapter 3

Surface curve formulation

Our surface fitting pipeline starts with a user interactively segmenting the polygonal model into a number of regions that are to be approximated by B-spline patches. In our system, this segmentation is accomplished by the placement of boundary curves at user specified locations. Since the user dictates the placement of boundary curves, they may have complex shapes relative to the surface geometry. Therefore, the ability to precisely position these curves relative to the surface geometry is crucial to our application. As such, curves that are constrained to the surface provide a better intuition for the shape and position of a boundary curve than unconstrained three dimensional curves. In the first place, unconstrained space curves could intersect and be occluded by the surface itself, making for poor visualizations. Secondly, in the absence of an explicit surface constraint, the user must manually ensure that the curve closely follows the surface geometry. If either the surface geometry or the curve shapes (or both) are complex this process is likely to be tedious. A more serious concern is that the process is likely to be error prone: for example using space curves it is possible to generate curves that do not obey the topology of the underlying surface. The fact that our input polygonal meshes are large only serves to further exacerbate these problems. To avoid the problems associated with the use of space curves we represent our patch boundaries as surface curves. By definition, this representation possesses a built in surface constraint and hence avoids the many problems associated with space curves.

In this chapter we describe efficient and intuitive techniques for specifying and editing surface curves. We begin (section 3.1) with a description of our surface curve representation. This representation is a discrete one since it exists on a piecewise flat surface. In section 3.2 we describe a method to create boundary curves.

Since the user might wish to subsequently edit the position of the curve, it is essential to provide flexible and robust curve editing tools. In our system, we provide two kinds of curve editing tools:

- Space curve based editing tools.
- Surface snake based editing tools.

Section 3.3 describes space curve based editing. This paradigm associates a face-point curve with a corresponding three dimensional (i.e. unconstrained) B-spline curve. An editing operation on the B-spline curve now translates to an analogous operation on the face-point curve. This paradigm allows us to use the flexibility and power of the traditional B-spline representation while still retaining the benefits of a surface curve representation.

While the space curve based editing paradigm is useful, it has several flaws. We explain these in section 3.4. To overcome these flaws we propose a *surface snake* formulation for our surface curves. This formulation facilitates the creation of a new set of flexible curve editing tools that overcome the shortcomings of space curve based editing. The theoretical framework for surface snakes is discussed in section 3.5. We discuss an implementation of the surface snake framework in section 3.6. A simplistic implementation of the surface snake theory can be inefficient. In section 3.7 we propose a method to speed up the basic implementation.

Tools based on surface snakes operate directly on the surface curve (rather than indirectly through a space curve) and hence are more flexible and intuitive to use than tools based on space curves. In addition, surface snakes also allow the use of surface properties such as vertex color to assist in the curve editing process. Such operations are not possible using just space curves. We conclude this chapter with a demonstration of these and other uses of surface snakes in section 3.8.

3.1 Curve Representation

A surface curve on a polygonal surface is accurately represented as a discrete polygonal geodesic [Mitchell et al. 1987]. This representation is essentially an "edge-point curve" i.e. a chain of edge-points such that two successive edge-points lie on some two edges (or a corner) of the same face.

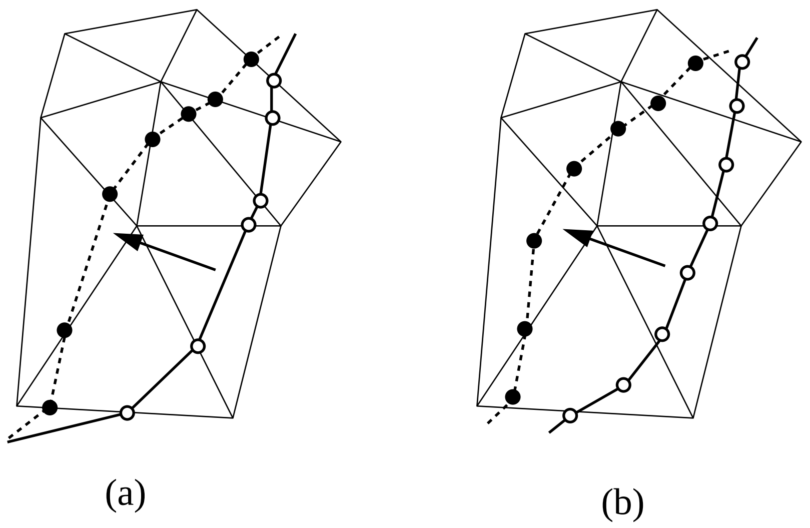

Figure 3.1. A comparison of the edge-point surface curve representation and our face-point approximation to it. (a) shows an editing operation on an edge-point curve on a polygonal surface. Notice that successive points on the curve before and after the editing operation must lie on some two edges (or a vertex) of each polygon that the curve crosses. This could make the representation inefficient for curve editing purposes: we must keep track of where the curve intersects the surface at each stage of the editing operation. (b) shows the same editing operation on our face-point representation. In this case, the editing operation consisted simply of moving individual face points to their new locations. The intersection of the face-point curve with edges of the polygon mesh does not need to be stored or computed at any point of an editing operation.

While this representation is a useful and elegant one for some operations, it is unsuitable for our purposes. For example figure 3.1(a) demonstrates a simple editing operation on the edge-point representation. Note that at each step of the editing operation the curve must be clipped against the edges of the underlying polygonal mesh in question. Unless carefully optimized this operation could be inefficient and therefore unwieldy in an interactive system. A simpler alternative to the edge-point representation is a face-point representation. A surface curve in this representation consists of a series of connected face-points, where a face-point is simply a point on some facet of the polygonal surface. We do not place an explicit constraint on the relative positions of successive face-points of the curve. Figure 3.1(b) shows the editing operation of figure 3.1(a) on our face-point representation. Since we do not explicitly maintain intersections of this curve with edges of the polygonal mesh, our curve editing operations can be made efficient. The curve is visualized (i.e. rendered) using a piecewise linear reconstruction through its constituent face-points. Since surface curves can be occluded (in a 2-D rendering) by the surface geometry itself we render them together with the surface geometry using a hidden surface algorithm.

Note that our linear re-construction of face-point curves might intersect the surface of the polygonal mesh. An example of this is shown in figure 3.2. A hidden surface rendering of our face-point curves could therefore result in portions of the surface curve being occluded by the surface geometry itself. Such a rendering would provide an unsatisfactory intuition for the placement of the face-point curve relative to the surface. However if the sampling density of the curve is reasonably high the rendering is satisfactory. In practice, we choose our sampling density so that on the average no two face-points are separated by more than the width of one polygon (i.e. successive face-points are either on the same face or on adjacent faces). We discuss one method to maintain a uniform sampling over the length of a face-point curve in section 3.6.3.

3.2 Curve painting

Given our choice to represent boundary curves as surface curves (i.e. as face-point curves), there are several possible methods for a user to place them on the surface. We choose to have the user paint the curves directly on the mesh surface. An alternate method might be to draw three dimensional space curves and then project these curves on the polygonal surface. Yet another method might be to use planes to cut sections of the object and have those sections define curves on the mesh surface. Such tools have been used with point cloud data by a few research systems as well as some commercial packages [Sinha & Seneviratne 1993, Milroy et al. 1995].

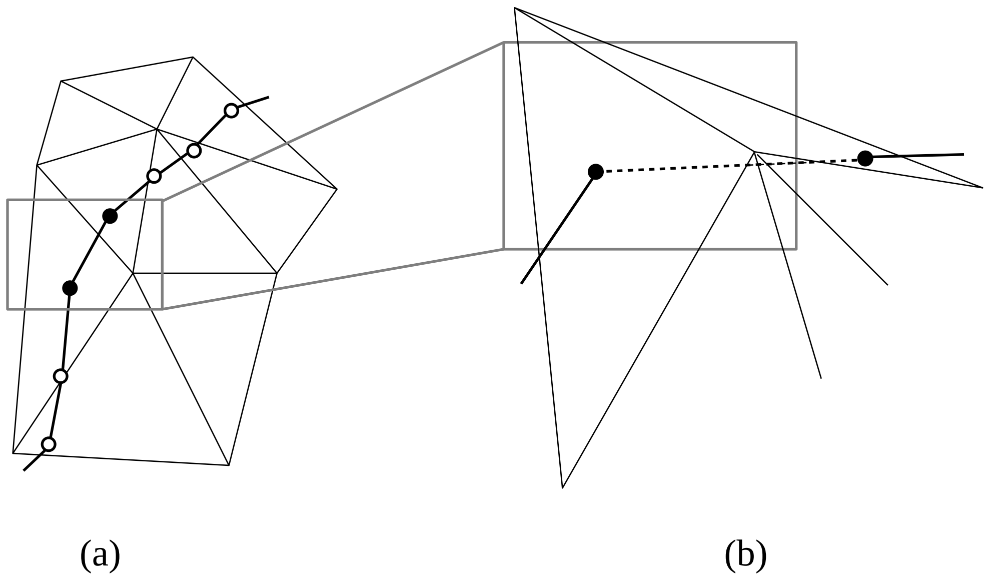

Figure 3.2. A linear reconstruction of the face-point curve representation might intersect the polygonal surface. For example (b) shows a section of the face-point curve in (a) that intersects the mesh. This is a potential drawback of a linear re-construction of the face-point curve. However in practice, if the sampling density of the curve is reasonably high the visualization is satisfactory. A method for ensuring uniform sampling density over the length of a face-point curve is explained in section 3.6.3.

While these tools are useful they do not provide the intuitive feedback that a surface painting tool can provide. For instance, the curve projection method is prone to failure in regions of the surface that have high curvature. In the case of a cutting plane tool the shape of the surface curve defined is severely limited: it can only be a planar contour on the surface. Furthermore, these tools do not use surface shape as an integral part of the curve specification process. Therefore, the user cannot rely on intuition about surface shape during the curve specification process.

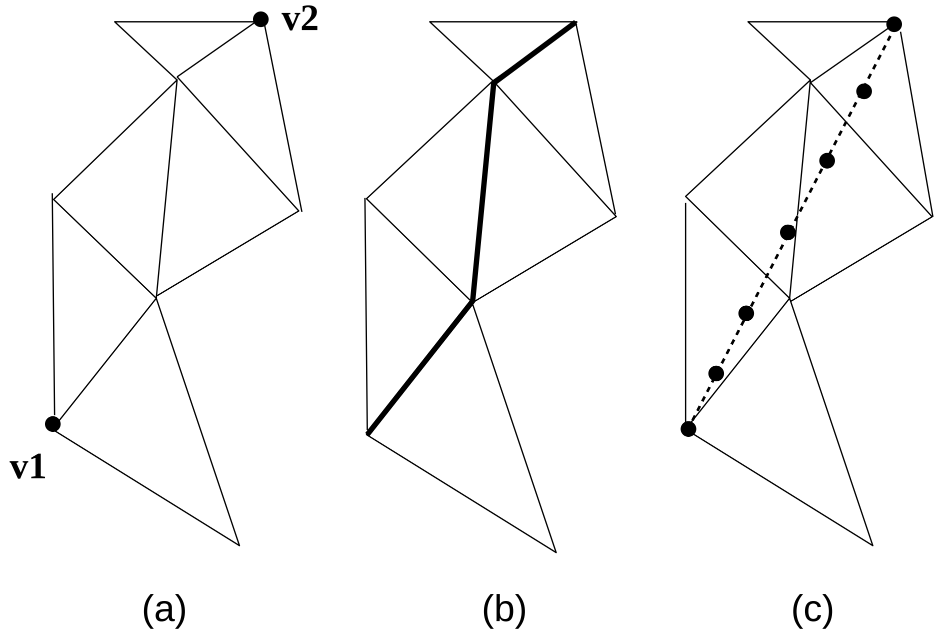

Figure 3.3. Boundary curve painting. Two successively picked vertices v1 and v2 are shown in (a). Between each pair of such vertices we compute the projection on the surface, of a straight line connecting the two. This is performed in two steps. First, we compute a greedy graph path between these two vertices. This path is a series of connected vertices of the mesh with v1 being the start vertex and v2 the end vertex. The intermediate vertices are chosen using a greedy algorithm: given a vertex on the path, the next vertex is chosen to be the closest neighbor of the destination vertex in Euclidean space. (b) shows the greedy graph path (as a thick polyline) corresponding to vertices v1 and v2. This path is then sampled into a face-point curve and smoothed into a straight line as illustrated in (c). The smoothing process is explained in detail in the text. Filled circles represent individual face points. The face-point curve now represents a sampling of the projection on the polygonal surface, of a line from v1 to v2.

Figure 3.3 explains the specifics of our curve painting process. Curve painting is accomplished in two steps. In the first step the user picks a sequence of vertices on the polygon mesh that represent a sequence of points that lie on the surface curve. The picking task is a straightforward one and can be accomplished in a number of ways. We use the following method: the user selects a series of points on a 2-D projection (or rendering) of the polygon mesh. This rendering stores an item buffer that associates for each screen pixel a set of mesh vertices that map to that pixel in the rendering. When a screen pixel is selected by the user the program searches the pixel's item buffer is used to compute the vertex of the mesh that is closest to the viewer. Another equally practical method for accomplishing this picking operation is to trace a ray through the screen position under the mouse and intersect it with a face-point on the mesh surface. We chose to implement the former approach for its simplicity and due to the fact that the operation is hardware accelerated on most high end workstations.

The second step in the curve painting process "chains together" surface points as the user is picking them. This chaining process creates a continuous surface curve out of the picked points. The problem of computing this continuous surface curve may be reduced to finding a surface curve section between each pair of successively picked points. Joining these curve sections together then gives us the overall curve through the points.

It is worth noting that there is no "correct" curve joining the two points: since the user has not supplied any information other than the picked points, there are many possible surface curves that we can create between each pair of surface points. The most obvious surface curve to use is the shortest surface curve between the two points i.e. a discrete geodesic [Mitchell et al. 1987]. Unfortunately true geodesic computation is prohibitively expensive ($O(n^2)$ in the size of the mesh [Chen & Han 1990]) and would severely limit the interactivity of our painting process. It is therefore not a viable option for our application. Of course, less computationally intensive approximations to geodesics could be attempted and this is an avenue for future exploration.

We choose instead to compute a curve that represents a sampling of the projection on the polygonal surface of a line joining each pair of successive points. We choose this over other possibilities because it is a plausible approximation to a geodesic between the two user points. Furthermore, the computation may be accomplished using purely local surface information and is therefore rapid. If two consecutively picked surface points lie on the same face, the projection operation is trivial. If this is not the case then we compute the face-point curve section in two steps. First, we compute a graph path between the start and end face-point of the curve section. This is defined as a sequence of connected vertices of the mesh, such that the start vertex is the closest to the starting face-point and the end vertex is closest to the end face-point. A greedy graph search algorithm computes this path. Given a vertex on the path, the greedy algorithm selects as the next vertex, a mesh neighbor that is closest in Euclidean space to the ending face-point. The graph path that we obtain from the previous step is sampled into a number of face-points. The resulting face-point curve is jagged since it conforms to the edges and vertices of the polygon mesh (figure 3.3b). As a second step we then smooth this face-point curve with a straight line joining the start and end points. The smoothing operation is based on procedure "CurveAttract" which is explained in section 3.3.1. In this procedure, the jagged graph path is first sampled into a set of face-points and then smoothed using a straight line joining the two points (shown in figure 3.3c). This two step smoothing operation is more robust than a direct one-step projection of a line on the surface. As explained earlier, a single step, direct projection to the surface can suffer catastrophic failures in regions of the polygon mesh that have high curvature. Our two step process is robust to abrupt changes in surface curvature and shape since it starts out with a good initial guess of the path on the surface (the graph path). Figure 3.3 summarizes the curve painting process.

It is worth noting that our procedure is not guaranteed to provide the shortest possible path through the picked points. Our graph paths were created using a local (greedy) heuristic and the subsequent smoothing operation moves face-points only locally on the surface. However, a shortest path through the points does not have any special significance for our application. A plausible approximation to it suffices for our purposes.

The painting process yields a face-point curve through a sequence of picked vertices. Between each pair of picked vertices the face-point curve section is the projection on the surface, of a straight line. Therefore, the painted surface curve may be viewed as a projection on the surface of a piecewise linear curve. In practice, the user will want to edit this curve to move it into a more desirable position. At the very least, the user will want to smooth the surface curve so it takes on more desirable appearance. In the next section, we discuss one paradigm for curve editing that uses space curves.

3.3 Curve editing using space curves

Space curve based editing tools are in turn based on a procedure that, given a face-point curve and an unconstrained three-dimensional space curve, uses the space curve to smooth the face-point curve. It accomplishes this by “attracting” the face-point curve (on the surface) to the space curve.

3.3.1 Attracting face-point curves to space curves

Recall that in our system a surface curve is a series of connected face-points. Let us assume for the moment that we are supplied a space curve that is to be used to smooth the face-point curve. As a first step in curve smoothing, we sample the space curve uniformly (by arc length) into a number of individual three dimensional points. We associate one such point in space to each face-point of the surface curve. The curve smoothing procedure simply slides each face-point of the curve to a new location on the surface such that it becomes the closest point on the surface to the corresponding point on the space curve. We call this the *CurveAttract* procedure. See Appendix A for one possible implementation of a procedure to slide points on polygonal surfaces. Figure 3.4 sums up this procedure.

If the space curve has a plausible projection on the surface, CurveAttract ensures that the face-point curve represents this projection. Furthermore, the algorithm is rapid since it uses purely local information when sliding points over the polygonal surface. However, if the space curve used for smoothing is not well chosen CurveAttract is prone to non-robust behavior. First, notice that the movement of a face-point does not affect its neighbors in the face-point curve: each face-point is treated on an individual basis when it is moved over the surface. Thus, if the space curve used for smoothing was itself non-uniformly sampled the smoothed face-point curve will also be non-uniformly sampled over its length. A second potential pitfall of the algorithm is that if either the space curve does not possess a plausible projection on the surface or if there are abrupt change in surface curvature, applying the algorithm could lead to an undesirable distribution of face-points within the

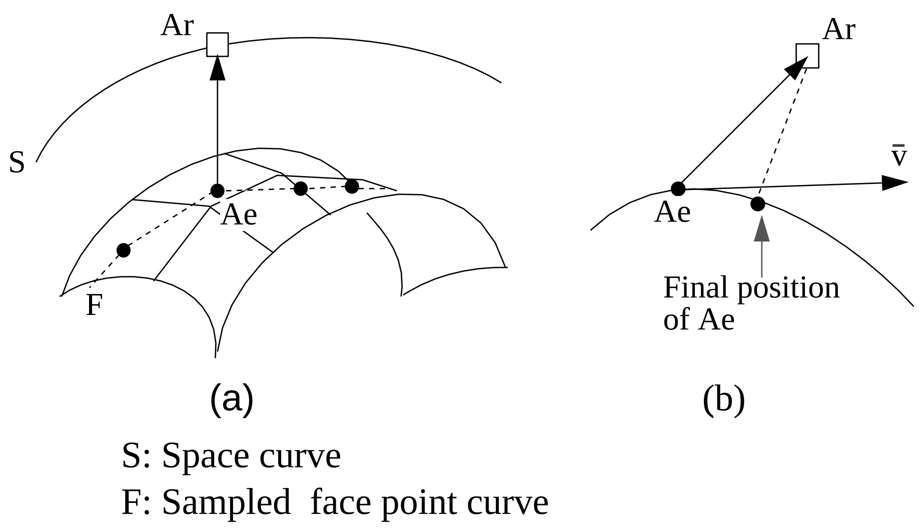

Figure 3.4. Procedure CurveAttract: smoothing a face-point curve using a space curve. (a) shows an attractor Ar on the space curve and an attractee Ae on the polygon mesh. (b) shows a 1-D version and the rest position of Ae.

smoothed face-point curve. This in turn could lead to poor visualizations. If the space curve used for CurveAttract was itself sampled uniformly over its length and had a plausible projection on the surface, the algorithm does well, i.e. the smoothed face-point curve is a faithful sampling of the projection on the surface of the space curve. Clearly, the choice of a suitable space curve is an important one. The next section first discusses our method for choosing a suitable space curve and then examines some curve editing operations based on this method.

3.3.2 Fitting a B-spline curve to the face-point curve

For editing operations that are based on space curve smoothing, we have chosen the space curve representation to be uniform B-splines. This representation is widely used by 3-D modelers and as such it enables the use of familiar editing operations (e.g. control vertex manipulation). Other parametric curve representations may be easily substituted into our approach.

We arrive at an initial location of the three dimensional B-spline (space) curve through a least squares fit of the face-point curve data. The equation for a uniform B-spline curve

$\mathbf{C}(u)$ can be written as:

$$\mathbf{C}(u) = \sum_{i=0}^{N} \mathbf{X}_i B_i(u) \tag{3.1}$$

In the equation, $B_i(u)$ is the uniform B-spline basis function [Bartels et al. 1987, Farin 1990] and $\mathbf{X}_i$ are the control vertices of the B-spline curve. Initially the control vertices are unknown to us; we must choose them to form a reasonable approximation to our surface curve. Therefore, given the number of control vertices N desired in the approximating B-spline curve and the face points $\mathbf{P}_j(x, y, z)$ of the surface curve, we need to determine the locations of a suitable set of control vertices (i.e. $\mathbf{X}_i$) such that the resulting B-spline curve $\mathbf{C}(u)$ is a good approximation to the face points.

This is a traditional curve approximation problem and is well studied in the literature. For our particular case the problem can be framed as a non-linear least squares problem [Dierckx 1993, Rogers & Fog 1989]. This is explained as follows. In equation 3.1 we need to find the $\mathbf{X}_i$ as well as u values for each data point $\mathbf{P}_j(x, y, z)$. Since, the B-spline basis functions we use are non linear (in general they are of order 4) in u the approximation problem is a non-linear one. One solution to this approximation problem proceeds as follows. Assume that we have assigned a parameter value (i.e. a u_j value) for each data point $\mathbf{P}_j(x, y, z)$. For the jth data point equation 3.1 can then be written as:

$$\mathbf{P}_j(x, y, z) = \mathbf{C}(u_j) = \sum_{i=0}^{N} \mathbf{X}_i B_i(u_j) \tag{3.2}$$

We get one such equation in the variables $\mathbf{X}_i$ for each face-point. In our application, this set of equations is usually over-constrained i.e. the number of face points is far greater than the number of control points approximating the face-point curve. If the number of face-points is M this over-constrained linear system can be written as:

$$\begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_M \end{bmatrix} = \begin{bmatrix} B_0(u_0) & B_1(u_0) & \dots & B_N(u_0) \\ B_0(u_1) & B_1(u_1) & \dots & B_N(u_1) \\ \vdots & \vdots & \vdots & \vdots \\ B_0(u_M) & B_1(u_M) & \dots & B_N(u_M) \end{bmatrix} \begin{bmatrix} \mathbf{X_0} \\ \mathbf{X_1} \\ \vdots \\ \mathbf{X_N} \end{bmatrix}$$

This is an over-constrained linear system and can be solved using traditional numerical techniques [Lawson & Hanson 1974]. Multiple linear constraints, such as fixing a set of control vertices at specific locations are added in a straightforward manner. Thus, once we have assigned parameter values to the data points the problem reduces to the numerically simple one of solving a set of linear equations. There are a number of techniques that have been proposed for assigning parameter values to the data points. See for example the methods explained in the works of Foley and Nielson [Foley & Nielson 1989, Nielson & Foley 1989] and the methods referenced therein.

For our application we have chosen to use the arc length metric for assigning parameter values to these face-points. This assigns a u value to each face-point that is equal to the ratio of the length of the curve up to that point to the total length of the curve. The length of our face-point curves itself (or sections thereof) is easily calculated based on the distances between successive face-points (i.e. the chord length). While this is an approximation to the true length of the curve, it is a satisfactory one if the curve is well sampled along its length. This parameterization method works well in practice. See section 3.6.3 for methods on maintaining a uniform sampling of face-point curves.

Based on the parameter values, we fit the face-point curve with a uniform B-spline curve of known resolution (i.e. N). This resolution is interactively chosen by the user. This choice allows the user to determine how closely a B-spline curve must approximate the surface curve for an editing operation. If the user wishes to smooth the surface curve by a substantial amount a coarse B-spline curve (i.e. fewer control vertices) could be chosen. In this case the B-spline approximation will be much smoother than the surface curve and hence the resulting smoothing operation will create a much smoother surface curve. Similarly, a larger number of control vertices for the B-spline approximation results in a much closer fit to the surface curve and hence the resulting smoothing is subtle. The next section briefly discusses further details of editing face-point curves with B-spline curves.

3.3.3 3-D B-spline curve based editing

The preceeding section outlined the process of obtaining a B-spline space curve from the face-point curve. In practice, the least squares fitting and smoothing method outlined in the last two sections can be performed at interactive speeds even for large polygonal meshes and dense face-point curves. This allows the user flexibility when using the smooth approximating space curve to perform curve editing. Coarser resolutions allow large scale smoothing (and editing) while higher resolutions allow the fine tuning of face-point curve placement. Editing is straightforward: the user manipulates either a control vertex or an edit point [Forsey & Bartels 1988] of the B-spline curve. This updates the location of the B-spline curve which in turn updates the surface curve based on the procedure *CurveAttract* discussed earlier. In practice all the standard methods for editing and controlling B-spline curves may be used in the above process. Examples include manipulating tangents and acceleration vectors at edit points.

The method has a several advantages. First it runs at interactive speeds even for large meshes. This is because the curve smoothing algorithm uses only local surface information to accomplish smoothing i.e. only those polygons that are actually touched by the face-point curve need be traversed during each smoothing step. A second feature of the algorithm is that it uses traditional B-spline curve editing techniques. This is a well studied area and commercial packages such as Alias, Softimage, Pro/Engineer allow a number of powerful tools to manipulate B-spline curves. Thus a modeler can use a powerful and familiar set of tools to manipulate the surface curve (albeit indirectly) while still benefitting from the additional intuition offered by a surface curve. Fine details may be edited by using more control vertices in the B-spline curve while coarse geometry can be editing using a smaller number of control vertices.

In the preceeding discussion one could also imagine using other curve representations (i.e. basis functions) such as a wavelet [Gortler & Cohen 1995, Finkelstein & Salesin 1994] or a hierarchical B-spline representation [Forsey & Bartels 1988]. The smoothing and editing techniques outlined above work with any underlying basis function for the space curve. In each of these cases the underlying mathematical structure of the space curve representation is inherited by the surface curve. We have chosen a uniform B-spline basis function because it is the representation of choice for most popular modeling packages.

3.4 Limitations of space curve based editing

While the curve editing paradigm outlined in the previous section has a number of advantages it has some shortcomings: first, editing based on space curves is usually not an intuitive operation for sophisticated editing scenarios. This is because the user does not directly manipulate the surface curve; rather, all editing operations must be accomplished indirectly by manipulating the B-spline space curve, which in turn controls the face-point curve. Furthermore, some editing operations on the space curve might produce no change in the surface curve (this can happen for example if the initial and final state of the space curve have the same projection on the surface). A second shortcoming of the space curve based editing procedure was outlined in section 3.3.1: if the space curve does not have a plausible projection on the surface (such as might happen in high curvature regions of the surface) the smoothing algorithm is prone to non-robust behavior.

Note that the shortcomings of a space curve based editing paradigm are independent of the specific representation of the space curve. Our choice of a B-spline basis for space curves does not worsen or alleviate these problems. Rather it is the editing paradigm itself that has these shortcomings.

3.5 Surface snakes: minimum energy surface curves

To overcome the shortcomings of space curve based editing, we propose a new framework for the direct manipulation of surface curves on dense meshes. We call this the *surface snake* framework. A surface snake is an energy-minimizing face-point curve. The energy of a surface snake is a combination of internal and external components. In the absence of external constraints a surface snake tends to minimize just its internal energy. As external constraints are added, the surface snake attains an energy state that minimizes both internal and external energy constraints. Using the surface snake framework we develop a curve editing tool that enables the user to edit surface curves by directly selecting and moving on the surface an arbitrary section of the curve (see section 3.8.1). The curve section deforms on the surface while staying continuous with the rest of the surface curve. Because the user directly manipulates the surface curve, the editing operation is intuitive. Furthermore, the surface snake formulation ensures that the editing operation is robust with respect to abrupt changes in surface curvature and geometry.

In this section we begin (in subsection 3.5.2) by reviewing a selection of related work in the area of surface curves as well as energy-minimizing curves in general. We follow this with a discussion of a theoretical formulation for energy minimizing curves on polygonal surfaces. Subsequent sections discuss an efficient implementation and applications of the formulation.

The surface snake framework is an extension to polygonal surfaces of snakes in two dimensions [Kass et al. 1988]. As such it draws several insights from two bodies of research: the variational modeling literature and prior work on surface curves (on other surface representations). In the following discussion, we discuss each of these in turn.

3.5.1 Snakes: energy minimizing curves in 2-D

Snakes (or active contours) are energy minimizing curves in two dimensions [Kass et al. 1988]. They were formulated for the purpose of assisting in the performance of high level image processing operations such as identifying subjective contours, motion tracking and performing stereo matching. Since these kinds of algorithms often require some form of higher level reasoning, low level automated algorithms usually perform poorly for these operations. Kass et al. argued that an interactive system that used minimum energy curves (or “active contours”) could be used to attain superior solutions. They called these curves snakes for their snake-like behavior during a series of energy minimizing steps. A snake under this definition is an energy minimizing 2-D curve associated with an underlying image. The energy of the snake is measured as a combination of internal constraints, image-based constraints and user supplied (external) constraints. For a curve $\mathbf{v}(s)$ that is parameterized by arc length this energy is written as:

$$E_{snake} = \int_0^1 E_{internal}(\mathbf{v}(s)) \ + \ E_{image}(\mathbf{v}(s)) \ + \ E_{constraint}(\mathbf{v}(s)) \, ds \tag{3.3}$$

In the equation above, the (arc length) parameter s varies from 0 to 1, the length of the curve. $E_{internal}$ represents the internal energy of the snake due to the internal stretching and bending of the curve. E_{image} represents the energy of the curve due to properties of the underlying image (e.g. intensity magnitudes or gradients) and $E_{constraint}$ represents external energy (e.g. based on user defined constraints). The snake attempts to reach a final state that minimizes E_{snake}.

To provide an intuition for what these energy terms represent we examine in more depth the internal energy term $E_{internal}$. This can be further expanded as:

$$E_{internal}(\mathbf{v}(s)) = \alpha(s)|\mathbf{v}_s|^2 + \beta(s)|\mathbf{v}_{ss}|^2 \tag{3.4}$$

where

$$\mathbf{v}_s = \frac{\delta \mathbf{v}}{\delta s}, \quad \mathbf{v}_{ss} = \frac{\delta^2 \mathbf{v}}{\delta s^2}$$

α and β are weighting functions. The presence of the first order term $|\mathbf{v}_s|^2$ minimizes the length of the snake. For an intuition of what this term accomplishes, imagine the behavior of a pliable, elastic band that is constrained to pass through a series of hooks. The elastic band shrinks its length to be as short as possible and still pass through those hooks. For this reason, the first order term is sometimes referred to as the *membrane energy* or *stretching energy* of the curve. The term locally characterizes a C^0 curve that need only be continuous but not differentiable i.e. the curve can have sharp corners.

The second order term $|\mathbf{v}_{ss}|^2$ minimizes the curvature over the length of the curve. For an intuition for what this accomplishes, imagine the behavior of a stiff wire that is constrained to pass through a series of hooks (like the membrane in our example above). Since the wire is resistant to bending, the resulting curve is smooth. For this reason, the second order term is sometimes referred to as the *thin plate energy* or *bending energy* of the curve. Quantitatively, the term locally characterizes a C^1 curve i.e. a curve that is continuous and has a continuous first derivative.

The functions $\alpha(s)$ and $\beta(s)$ are referred to as *continuity control* functions. By changing them one can control the relative strengths of the stretching and bending terms and therefore the shape of the curve. For example, when $\alpha(s)$ is zero the curve has a local discontinuity at s. When $\beta(s)$ is zero, the curve has a sharp corner at s. The curve produced by equation 3.4 is called a *controlled continuity spline* because one can change the continuity at any point along the curve by manipulating the continuity control functions $\alpha(s)$ and $\beta(s)$. Using the controlled continuity spline and judicious formulations for E_{image} and $E_{constraint}$ a variety of useful behaviors may be obtained for snakes. We refer the reader to the original paper by Kass et al. for further details [Kass et al. 1988].

3.5.2 Prior work on surface curves

While manipulating curves in Euclidean space ($\Re^2$ or $\Re^3$) is a well studied subject, there has been little work on the subject of manipulating curves on arbitrary curved manifolds. If the manifold is already a parameterized surface in 3-space, the curve definition problem is effectively transformed to a problem in the Euclidean plane (i.e. the curve coordinates lie on the parametric plane). Thus techniques for manipulating curves on parametric surfaces are well studied. However, when there is no obvious mapping of a surface to the Euclidean plane, such as in the case of implicit surfaces or polygon meshes, there are very few techniques that allow a user to manipulate curves directly on the surface. In the implicit surface domain, there has been some work on fitting curves onto general quadric surfaces [Dietz et al. 1993], but the work is not easily extensible to non-polynomial surfaces.

The work of Gabriel and Kajiya [Gabriel & Kajiya 1985], subsequently improved upon by Barr et al [Barr et al. 1992], comes closer to our needs. Both of these approaches model surface curves as a minimum energy solution to an optimization problem. The first approach works purely in parametric space and the second solves the problem of interpolating quaternions in 4-space. The interpolation methods proposed are not easily extensible to arbitrary surface definitions. Pedersen et al [Pedersen 1995] demonstrated a variant of the technique that could be used effectively to calculate smooth interpolating curves on implicit surfaces.

None of the above-mentioned techniques have addressed the issue of intuitiveness of control, computational complexity of the optimization steps or the interactivity of their algorithms. Further, the methods all make assumptions about the smoothness properties of their underlying surface representation. While the assumptions were valid for the problem domain being tackled (i.e. quaternion spheres, implicit surfaces or quadric surfaces), they do not hold for polygonal meshes.

3.5.3 Formulating snakes for polygonal surfaces

Our surface snake formulation is a extension of the 2-D snake formulation of Kass et al. [Kass et al. 1988] for dense polyhedral surfaces. As such, the key difference of our formulation as distinct from theirs are twofold:

1. We add the additional constraint that ensures that our snakes lie on the surface of the polygonal mesh while still satisfying the minimum energy criteria.
2. We re-formulate each of the energy terms in equation 3.5 for the face-point curve representation.

We explain both these modifications in the following discussion.

We use a controlled continuity spline to model our surface curves. The internal and external energy terms of a surface snake can be written as:

$$E_{surf-snake} = \int_0^1 E_{internal}(\mathbf{v}(s)) \ + \ E_{surf}(\mathbf{v}(s)) \ + \ E_{constraint}(\mathbf{v}(s)) \, ds \qquad (3.5)$$

where the only modification we have made to the 2-D formulation is the replacement of E_{image} with E_{surf} to indicate surface based constraints (rather than image based constraints). The surface curve acts autonomously to correct its own shape by finding a minimum to the energy functional. Before we can use this formulation we must address the two issues listed above. In the following discussion we use the formulation for $E_{internal}$ to clarify both these issues. The other energy terms generalize in a similar fashion. The previous section examined the two components of the internal energy for a curve in $\Re^2$. Our curves are face-point curves: these are essentially curves in $\Re^3$ with the additional constraint that they lie on the polygonal mesh surface. Consider equation 3.4 in the context of a face-point curve. The first order term minimizes the integral of the tangent at each point along the curve. By definition, the tangent vector of a surface curve is tangent to the surface. Therefore for purposes of evaluating the first order term we may ignore the surface constraint.

The second order term does not generalize to surface curves in as straightforward a manner. One intuition for this term is explained in figure 3.5. The curvature of a surface curve can have components that are both normal and tangential to the surface that the curve

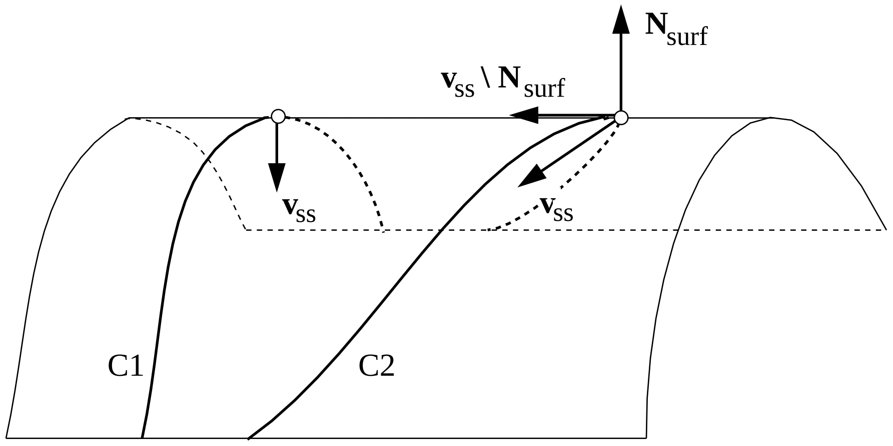
$\mathbf{N}_{\text{surf}}$
$\mathbf{v}_{\text{ss}} \setminus \mathbf{N}_{\text{surf}}$
$\mathbf{v}_{\text{ss}}$
$\mathbf{v}_{\text{ss}}$
C1
C2

Figure 3.5. Normal and tangential curvatures for a surface curve. This figure shows our motivation for penalizing just that component of the thin plate term that is tangential to the surface. The thin plate term attempts to minimize the integral of the magnitude of the curvature $\mathbf{v}_{ss}$ of the curve $\mathbf{v}$. $C1$ is a surface curve with non-zero curvature in space at the point shown. However, note that the tangential component on the surface of $\mathbf{v}_{ss}$ is zero. This implies that at the point shown, $C1$ already minimizes curvature on the surface despite the fact that it has a non zero curvature in space. $C2$ shows yet another curve on the same surface. In this case, the curvature in space of a specific point on $C2$ is shown as $\mathbf{v}_{ss}$. The normal to the surface at that point is $\mathbf{N}_{surf}$. Note that at the point shown, $C2$ has a non-zero curvature component tangent to the surface. This is shown in the figure as $\mathbf{v}_{ss}\backslash\mathbf{N}_{surf}$. It is this component of the curvature that we shall minimize.

lies on. It is the component of the curvature that is tangential to the surface that we are interested in minimizing. This is easily explained. Whichever path the curve takes on a surface, at each point on the curve its curvature vector will in general have a component that is normal to the surface at that point. This vector is actually the curvature of the surface itself along the tangent to the curve at that point. Therefore a minimum energy surface curve may possess a non-zero normal component to its curvature at each point along its length. As before, this component corresponds to the curvature of the surface along the direction of the tangent to the minimum energy curve at that point. Thus the normal component of a surface curve's curvature is a function of the underlying surface. As such it is unavoidably non-zero if the surface has non-zero curvature and should not be penalized. Therefore our second order term only penalizes the component of the surface curve's curvature that is tangent to the surface. With the foregoing discussion in mind the internal energy can now be written as:

$$E_{internal}(\mathbf{v}(s)) = \alpha(s)|\mathbf{v}_s|^2 + \beta(s)|\mathbf{v}_{ss}\backslash\mathbf{N_{surf}}(\mathbf{v}(\mathbf{s}))|^2 \, ds \tag{3.6}$$

where $\mathbf{N_{surf}}(\mathbf{v}(\mathbf{s}))$ refers to the normal to the surface at the surface point corresponding to the point on the curve given by $\mathbf{v}(s)$. Borrowing Barr et al's ([Barr et al. 1992]) notation, we use $\mathbf{v1}\backslash\mathbf{v2}$ to denote a vector defined as follows:

$$\mathbf{v1}\backslash\mathbf{v2} = \mathbf{v1} - \frac{\mathbf{v1}.\mathbf{v2}}{\mathbf{v2}.\mathbf{v2}}\mathbf{v2}$$

This ensures that $(\mathbf{v1}\backslash\mathbf{v2}).\mathbf{v2} = 0$ i.e. that $\mathbf{v1}\backslash\mathbf{v2}$ is the vector obtained by subtracting from $\mathbf{v1}$ its component in the $\mathbf{v2}$ direction. For convenience, in the following discussion we will continue to use $\mathbf{v}_{ss}$ to explain our surface curve implementation. In light of the foregoing discussion, this term should be interpreted as the tangential component of the curvature vector i.e. as $\mathbf{v}_{ss}\backslash\mathbf{N_{surf}}$.

Note that equation 3.6 does not explicitly ensure that the curve will remain on the surface. Rather it ensures that the energy of the surface curve is computed in a meaningful manner. One method for imposing the additional surface constraint could be to add a mathematical condition to equation 3.6 that accomplishes the desired effect. This is a reasonable option if the surface has a global description for surface position (e.g. $F(\mathbf{P}) =$

0). In this case one could substitute the curve equation into the surface equation (e.g. $F(\mathbf{v}(s)) = 0$) and optimize the curve with this as a constraint. However, our polygonal meshes are unparameterized and do not have a closed form equation that describes them. Given our curve representation (a series of face-points), an effective method of imposing the surface constraint is to ensure that each of the individual face-points of a curve always stay on some facet of the polygonal mesh even as the curve moves towards its minimum energy configuration.

3.6 Surface snakes: an implementation

The problem posed by equation 3.6 is a type of variational problem [Weinstock 1974]. Practical algorithms that solve variational problems come in two flavors: the finite difference approach and the finite element approach. The finite difference approach starts by considering an equivalent statement of equation 3.4 as a set of differential equations. These come from a practical result of the calculus of variations [Weinstock 1974] that shows that a curve that minimizes equation 3.4 also minimizes the set of differential equations (the Euler equations) given by:

$$E_{Euler}(\mathbf{v}) = \frac{d^2\beta\mathbf{v}_{ss}}{ds^2} - \frac{d\alpha\mathbf{v}_s}{ds} = 0 \tag{3.7}$$

For the sake of simplicity we are considering here just the internal energy terms. We extend the formulation to include other kinds of energy terms in later sections.

Finite difference approaches approximate the continuous solution to the Euler equations using a set of discrete difference equations. This strategy reduces the curve representation to a set of points in space. Therefore, we lose the original continuity of the solution. However it is a computationally simple and an easily extensible approach.

A second approach to solving our minimum energy equation is the finite element method. In this case the desired solution is represented as a weighted sum of a set of carefully chosen basis functions. The optimization process seeks to find the optimal set of weights (for these basis functions) that would result in a minimum energy solution. We refer the reader to Celniker and Gossard's work [Celniker & Gossard 1991] on deformable curve and surface design for an example of using the finite element technique to create minimum energy curves and surfaces. For our application, a finite element solution would require that we construct an appropriate set of smooth basis functions over an irregular, unparameterized polygonal manifold. This is a challenging task in itself and remains an open problem [Welch & Witkin 1994, Eck et al. 1995]. A variation of this strategy could be to use basis functions in $\Re^3$ (i.e. no surface constraints). However this strategy would ultimately have to resort to creating projections of space curves on the polygonal surface. This solution is not acceptable to us since it runs counter to our original reasons for creating a surface curve formulation (i.e. that projection is in general non-robust and could lead to un-intuitive interactions). We have therefore chosen to use a finite difference solution on our sampled face-point surface curve representation.

3.6.1 A finite difference solution

Our goal is to generate a minimum energy face-point curve given a set of user-imposed constraints and perhaps a set of surface-based constraints. Our solution strategy is to find the discretized version of the Euler equations and solve the resulting difference equations for our solution. For simplicity, and without loss of generality, we explain the minimization process using only the stretching and bending energy terms (a curve's internal energy). Once our basic solution paradigm is explained we may extend it in a straightforward manner by adding other energy terms.

Given just the stretching and bending terms of equation 3.6 the corresponding Euler equations are as given in equation 3.7. The discrete form for the internal energy terms is written as follows:

$$E_{internal}(\mathbf{v}) = \sum_i E(\mathbf{v}(i)) \tag{3.8}$$

where $\mathbf{v}(i)$ is the i-th face-point and the summation is assumed to be over the face-points of the surface curve. Approximating the derivatives with their corresponding finite difference approximations, the "energy" contribution of the i-th face-point is given by:

$$E(\mathbf{v}(i)) = \alpha_i \|\mathbf{v}_i - \mathbf{v}_{i-1}\|^2/2h^2 + \beta_i \|\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}\|^2/2h^4 \tag{3.9}$$

where h is the sample spacing (assumed to be uniform here).

The discrete version of the Euler equations (for an individual face-point) is therefore given by the discrete version of equation 3.7:

$$\begin{aligned} E_{Euler}(\mathbf{v}(i)) &= \beta_{i+1}(\mathbf{v}_i - 2\mathbf{v}_{i+1} + \mathbf{v}_{i+2}) + \beta_{i-1}(\mathbf{v}_{i-2} - 2\mathbf{v}_{i-1} + \mathbf{v}_i) \\ &\quad -2\beta_i(\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}) \\ &\quad -(\alpha_{i+1}(\mathbf{v}_{i+1} - \mathbf{v}_i) - \alpha_i(\mathbf{v}_i - \mathbf{v}_{i-1})) \\ &= 0 \end{aligned} \tag{3.10}$$

Note that we have dropped the denominators from the expressions for the discrete derivative and curvature in this equation. In reality, these weights affect the relative importance of the first and second terms; however, in our discussions we will subsume these weights into the β's. In two dimensions the above equation can be conveniently expressed in the form of a matrix equation that may be solved using an implicit Euler method. We refer the reader to the work of Kass et al. [Kass et al. 1988] for a discussion of the issues involved in such a solution. What is worth noting here is for our implementation, a matrix based solution cannot factor in the point-on-surface constraint. Our iterations must therefore be explicit Euler iterations [Strang 1986], i.e. we must explicitly enforce the surface constraint at each iteration step.

We propose an explicit iterative solution that at every iteration moves each face point of the surface curve by some distance on the surface. When we start the minimum energy iteration the internal energy of the curve is non-zero. The exact numerical value is given by the value of E_{Euler}. The direction and distance moved by the surface-snake (on the polygonal mesh) are computed so that the value of E_{Euler} is reduced at each iteration step. The iteration proceeds in this manner until the surface curve reaches its minimum energy configuration. There are two steps in our iterative energy minimization process (which is essentially a relaxation process):

1. First, based on the expression for E_{Euler} we infer a distance and direction to be moved by each individual face-point.

2. Second, we slide the face-points to new positions on the surface, according to the

computed distance and direction.

The distance and direction to be moved by each face-point at each iterative step is computed in the form of a "force" acting on the individual face-points. We call the set of forces computed from the expression for E_{Euler} as the *variational Euler forces*.

To our knowledge, no prior work on surface curves has attempted a solution to our kind of variational problem based on a decomposition of the energy terms as forces. It is worth noting that, where possible, an implicit Euler solution such as the one exemplified by Kass et al. is preferable for reasons of efficiency over an explicit solution such as ours. However as noted earlier, accomodating the additional constraint of keeping our curves on the polygonal mesh surface while using this solution is an open problem.

The first step in our solution then is to decompose the discretized Euler equations into a set of forces on individual face-points. We rewrite equation 3.10 for $\mathbf{v}_i$ (i.e. the i-th face-point) as follows:

$$E_{Euler}(\mathbf{v}(i)) = -(\alpha_i F_{i,i-1} - \alpha_{i+1} F_{i,i+1} - 2(2\beta_i C_i - \beta_{i-1} C_{i-1} - \beta_{i+1} C_{i+1})) \tag{3.11}$$

where

$$F_{i,j} = \mathbf{v}_j - \mathbf{v}_i,\ C_i = \frac{\mathbf{v}_{i+1} + \mathbf{v}_{i-1} - 2\mathbf{v}_i}{2}$$

$F_{i,i-1}$ is proportional to the discrete backward tangent at $\mathbf{v}_i$, $F_{i,i+1}$ is proportional to the discrete forward tangent at $\mathbf{v}_i$ and C_i is proportional to the discrete curvature at $\mathbf{v}_i$ (assuming the sample spacings are uniform). At some intermediate stage of our iterative energy minimization process the above expression is non zero. In order to reach equilibrium (i.e. zero out the expression) we propose to exert a force $F_{resultant}(i)$ on the i-th face-point that would allow the face-point to reach its minimum energy state. $F_{resultant}$ is given by:

$$\begin{aligned} F_{resultant}(i) &= -E_{internal}(\mathbf{v}(i)) \\ &= \alpha_i F_{i,i-1} + \alpha_{i+1} F_{i,i+1} \\ &\quad +2(2\beta_i C_i - \beta_{i-1} C_{i-1} - \beta_{i+1} C_{i+1}) \end{aligned} \tag{3.12}$$

We can now interpret each of the five discrete terms in the above equation as individual component forces acting on the face-point $\mathbf{v}_i$. We call these five forces the *variational*

Euler forces to indicate that they're derived from the variational equations corresponding to our minimum (internal) energy curve equation.

Figure 3.6 supplies more intuition for these forces. The forward tangent $F_{i,i+1}$ moves

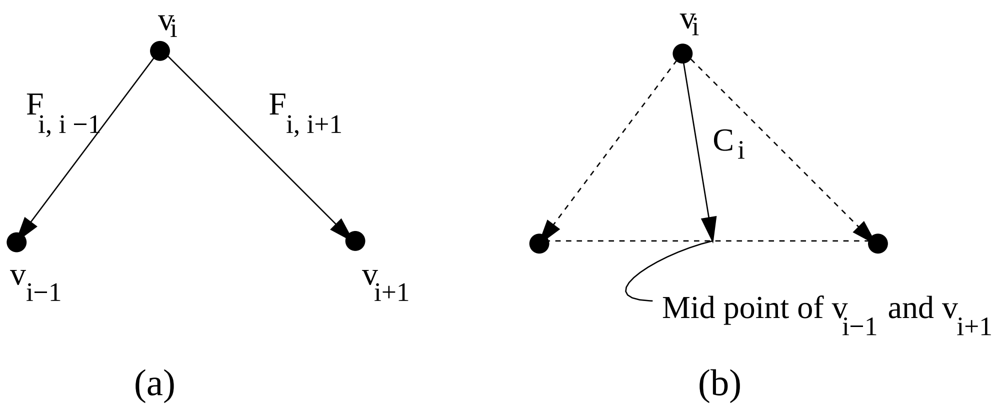

Figure 3.6. Decomposing the discrete Euler equations into forces on individual face-points. (a) shows the forces corresponding to the discrete forward ($F_{i,i+1}$) and backward tangents ($F_{i,i-1}$) at $\mathbf{v}_i$. (b) shows the force corresponding to the curvature term C_i.

$\mathbf{v}_i$ towards $\mathbf{v}_{i+1}$ and the backward tangent $F_{i,i-1}$ moves it towards $\mathbf{v}_{i-1}$. The curvature force C_i moves $\mathbf{v}_i$ towards the mid-point of its two neighboring points.

Consider the resultant of the first two variational Euler forces. This term represents the force due to the membrane energy term. Figure 3.7a supplies an intuition for this term. If the weights are chosen appropriately, this term moves the point to a new location that lies in-between its two neighbors. Similarly, the resultant of the last three variational Euler forces represents the force due to the thin plate term. Figure 3.7b supplies an intuition for this term. This force makes the curve locally curvature continuous to the neighboring curve segments. Note all these forces are defined in Euclidean space i.e. they do not encode our knowledge of surface geometry. Following the arguments of section 3.5.3 we use only the component of these forces that lies on the local tangent plane of the surface. This projected force is used to move the surface point to a new location on the surface.

To summarize our basic surface snake formulation: each energy minimization term is reduced to a set of appropriate forces acting on individual face-points of our surface curve. The component of these forces that is tangent to the surface is used to slide these face-points over the surface to their new positions on the surface. At equilibrium, the resultant

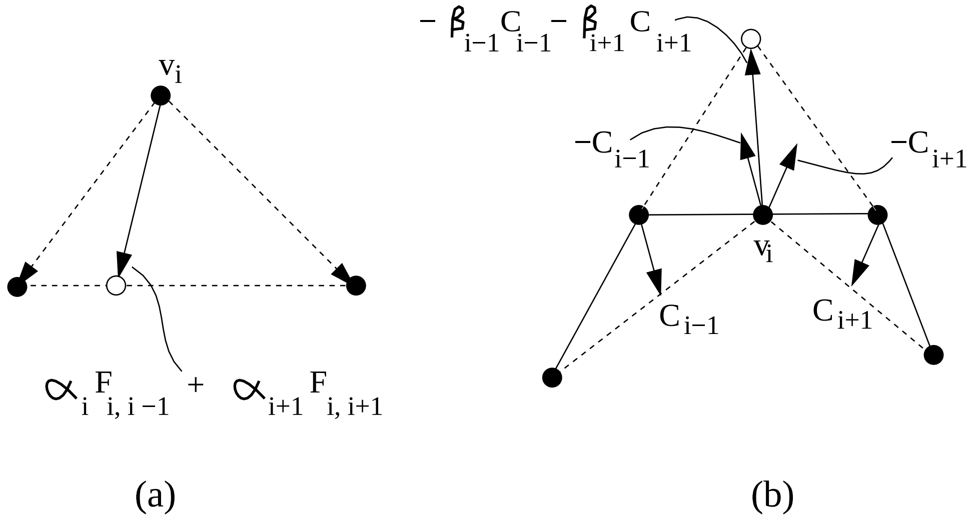
v_i
$\alpha_i F_{i,i-1} + \alpha_{i+1} F_{i,i+1}$
(a)
$-\beta_{i-1} C_{i-1} - \beta_{i+1} C_{i+1}$
$-C_{i-1}$
$-C_{i+1}$
v_i
C_{i-1}
C_{i+1}
(b)

Figure 3.7. Variational Euler forces: a 2-D visualization of resultants corresponding to each of the membrane and thin plate terms. (a) shows the resultant of the first two terms (i.e. the membrane term) from equation 3.11. The force vector corresponding to this resultant is a linear combination of the forward and backward tangent vectors. If $\alpha_i + \alpha_{i+1} = 1.0$ then at least in two dimensions the resultant will move $\mathbf{v}_i$ to a point on the line joining its two neighbors (shown as a hollow circle). (b) shows the resultant of the second term (i.e. the thin plate term) in the equation when $C_i = 0$. This force attempts to make the curve at $\mathbf{v}_i$ locally curvature continuous to the neighboring curve segments. Once again the final position of $\mathbf{v}_i$ is shown as a hollow circle.

tangential force on each face-point is zero and the surface snake reaches its desired minimal energy state.

3.6.2 Setting the continuity control functions

An important issue to address in our energy minimization strategy is what the continuity control functions (i.e. the weights α_i and β_i) should be. Varying the relative strengths of these functions controls the "stiffness" of the curve (see section 3.5.1). In an interactive environment, it is desirable to provide user control over this parameter. Unfortunately, this issue has not been adequately addressed in the literature on 2-D snakes. We offer here one possible strategy for user control over these functions for surface snakes.

One method to provide the user control over curve stiffness might be to offer local control over the value of α_i and β_i for each face-point. While this would be straightforward to implement, it would provide a cumbersome interface to the user. Instead, let us make the assumption that both α_i and β_i are constant functions over the length of some section of the curve i.e.

$$\alpha_i = \alpha, \ \beta_i = \beta$$

Varying α and β now controls the stiffness of the entire curve section. For this curve section equation 3.12 can now be re-written as:

$$\begin{aligned} F_{result}(i) &= \alpha F_{i,i-1} + \alpha F_{i,i+1} + 2(2\beta C_i - \beta C_{i-1} - \beta C_{i+1}) \\ &= 2\alpha C_i + 2\beta(2C_i - C_{i-1} - C_{i+1}) \end{aligned} \tag{3.13}$$

since

$$F_{i,i-1} + F_{i,i+1} = 2C_i$$

We can further re-write this equation as:

$$F_{result}(i) = K_{fair}F_{fair}(i) + K_{curvature}F_{curvature}(i) \tag{3.14}$$

where

$$K_{fair} = 2\alpha, \ \ K_{curvature} = 2\beta$$

are the new (constant) continuity control functions and

$$F_{fair} = C_i, \quad F_{curvature} = 2C_i - C_{i-1} - C_{i+1}$$

We call these last two expressions the *fairness* and *curvature* forces respectively. From our earlier discussion the fairness force simply represents the membrane energy term. It moves each point to the mid-point of its two neighbors. Similarly the curvature force represents the thin plate energy term. It makes the curve locally curvature continuous with respect to the neighboring curve segments (figures 3.6b and 3.7b). Our use of the term *fairness* is motivated by its use in the geometric interpolation literature [Halstead et al. 1993]. In that domain, the fairness of an interpolating curve or surface is a measure of its smoothness.

Note that the fairness and curvature forces have potentially conflicting goals. The fairness term attempts to minimize the length of the curve. Intuitively, this means that it straightens out (on the surface) curved parts of the face-point curves. The curvature term on the other hand attempts to make a surface curve's (tangential) curvature, continuous over its entire length i.e. it smooths the curve only to the extent necessary to attain curvature continuity. Equation 3.14 provides a simple framework for controlling the global curvature properties of a surface curve. Assigning K_{fair} a higher weight produces tight, length minimizing surface curves and assigning $K_{curvature}$ a higher weight produces curves that tend to be smooth and yet retain their existing curvature properties (i.e. a highly "curvy" surface-curve is likely to retain its high curvature properties in an equilibrium state). Figure 3.8 shows an example where the relative weights of the curvature and fairness term are varied for a surface curve. As the figure illustrates, a flexible set of editing operations may be performed with constant values for $K_{curvature}$ (and K_{fair}) over a section of the curve. The details of the interactions for these editing operations is explained in section 3.8.1.

3.6.3 Creating uniformly sampled surface snakes

When dealing with discretized curves it is important to ensure that the curves are adequately sampled along their length. A curve that has an irregular distribution of points over its length is liable to miss detail in sparsely sampled regions. More importantly since our

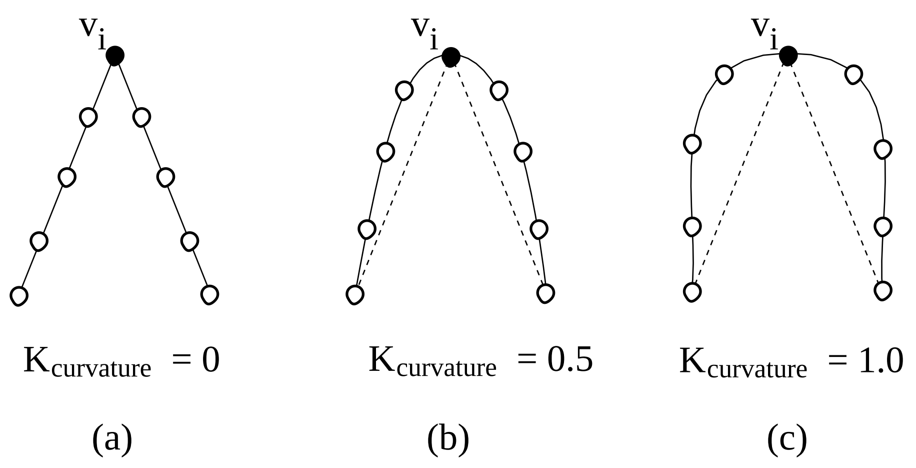

Figure 3.8. Snake continuity control functions. This particular editing scenario illustrates the flexibility offered by the surface snake formulation. The point $\mathbf{v}_i$ and the two end points of the surface curve are constrained to be fixed during the energy minimizing iteration of the surface snake and the weight of the curvature term $K_{curvature}$ is varied. (a) (b) and (c) show the final state of a relaxation iteration as the value of $K_{curvature}$ is gradually increased When $K_{curvature}$ is zero only the membrane (fairness) term is active. The resulting curve behaves like a tight elastic band passing through $\mathbf{v}_i$ and the two end points. The curve has a discontinuous first derivative at $\mathbf{v}_i$. As $K_{curvature}$ increases the curve behaves more like a stiff wire (thin plate) and distributes its curvature over the entire curve.

snake based energy computations are made only at the discrete set of face-points, an irregular sampling leads to an inaccurate estimate of snake energy. This in turn can adversely affect surface snake based editing operations. Figure 3.9 illustrates these problems with an example. Previous literature on minimum energy 2-D curves has not addressed this

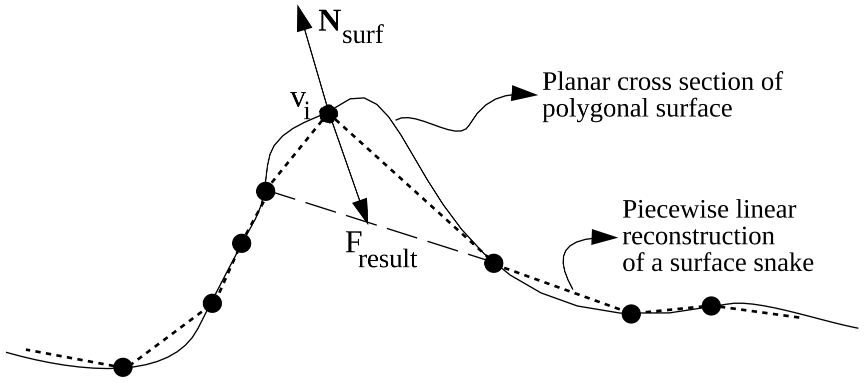
$\mathbf{N}_{surf}$
v_i
Planar cross section of polygonal surface
F_{result}
Piecewise linear reconstruction of a surface snake

Figure 3.9. Maintaining a uniformly sampled surface snake. The figure shows an irregularly sampled surface curve on a planar cross section of a surface. This curve minimizes the resultant due to the variational Euler forces. Note however, that because it is not uniformly sampled along its length it is an aliased representation of the underlying surface cross section. Secondly, since the sampled curve is an aliased representation of the underlying surface section, the snake energy terms for the curve are incorrectly computed. This in turn can result in incorrect and un-intuitive editing interactions.

problem because the problem simply does not arise in 2-D: the fairness force would move each point towards the midpoint of its neighbors in two dimensions ensuring that the points were evenly distributed over the length of the curve. It is with the addition of the curve-on-surface constraint that the non-uniform sampling problem is introduced. Previous literature on surface curves has not adequately addressed this issue either.

We solve the problems associated with non-uniform surface snake sampling by adding an additional force term to our minimum energy snake formulation. This force moves each face-point so that it stays equidistant from its neighbors in the surface-curve. We call this the snake arc length force term. A desirable characteristic for the arc length term is that it should not change the actual character or shape of the curve itself. Rather it should simply re-distribute the face-points of the surface curve so that they are evenly spaced.

Figure 3.10 provides the intuition in two dimensions, for the arc length term. The goal is to move $\mathbf{v}_i$ to a new location, such that it becomes equidistant from its neighbors $\mathbf{v}_{i-1}$ and

$\mathbf{v}_{i+1}$. Clearly, any destination point on the perpendicular bisector of the edge $(\mathbf{v}_{i-1}, \mathbf{v}_{i+1})$ will achieve this goal. However, any of the positions on this bisector would change the shape of the curve. Our goal is to choose a destination on this bisector that minimally changes the global shape of the curve. One method for selecting this optimum position could be to fit a non-local higher order (polynomial) curve and redistribute points on the surface according to this curve. However, this strategy has a problem: the fitted curve would have to once again be projected to the surface. As pointed out earlier curve projection is a non robust operation in general. Figure 3.10 shows three other candidate positions that we could move $\mathbf{v}_i$ to. We have found that moving a face-point directly towards its more distant neighbor (i.e. $\mathbf{v}_{i+1}$) produces satisfactory results. This strategy introduces a certain minimal local smoothing of the curve but tends to preserve the global shape and character of the curve.

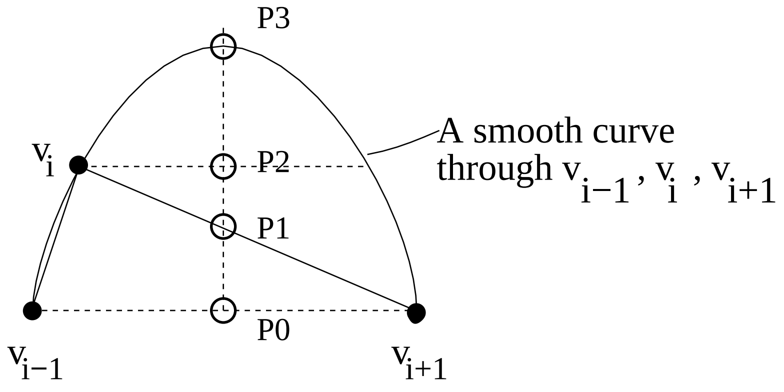

Figure 3.10. The arc length criterion. The figure shows a section of a face-point curve in two dimensions. $\mathbf{v}_i$ is to be moved to a new location, such that it satisfies two requirements. First, it should become equidistant from its neighbors $\mathbf{v}_{i-1}$ and $\mathbf{v}_{i+1}$. Second, it should not excessively alter the shape (or character) of the face-point curve. P0, P1, P2 and P3 show four candidate destination locations. They all satisfy the first criterion since they lie on the perpendicular bisector of the chord joining $\mathbf{v}_{i-1}$ and $\mathbf{v}_{i+1}$. However, P0 and P3 do not satisfy criterion 2. P0 is the mid point of $\mathbf{v}_i$'s neighbors. Moving $\mathbf{v}_i$ to P0 would smooth out the curve section excessively. P3 is computed based on a smooth interpolating curve through the three points. Moving $\mathbf{v}_i$ to P3 also change the shape of the curve section excessively. P1 and P2 are two viable alternatives for a new location of $\mathbf{v}_i$. Our implementation of the arc length term chooses P1 to be the new destination for $\mathbf{v}_i$. See the text for further details.

We compute the arc length force as follows:

$$F_{arc}(\mathbf{v}_i) = |(\| F_{i,i-1} \| - \| F_{i,i+1} \|)| \frac{\mathbf{t}_{max}}{\| \mathbf{t}_{max} \|} \tag{3.15}$$

where $\mathbf{t}_{max}$ is the larger of $F_{i,i-1}$ and $F_{i,i+1}$ (in magnitude) i.e.

$$\mathbf{t}_{max} = \begin{cases} F_{i,i-1} & \text{if } \|F_{i,i-1}\| > \|F_{i,i+1}\| \\ F_{i,i+1} & \text{if } \|F_{i,i+1}\| > \|F_{i,i-1}\| \end{cases}$$

If the magnitudes of the forward and backward tangents happen to be equal the arc length force is zero.

Using the arc length constraint within our surface snake minimum energy iteration is trivial: we simply add the arc length force to the variational Euler forces. The minimum energy iteration then automatically ensures that the surface curve has a uniform distribution of face points along its length.

Note that a more efficient representation for surface curves might be one that has a lower number of sample points in regions of lower surface curvature (and a higher concentration of face-points in regions of higher surface curvature). As explained, our solution uses uniformly sampled curves that are sampled at dense enough resolution that two adjacent face-points are either on the same face of the mesh, or on adjacent faces. Therefore, in flat areas of the surface we might be using more face-points than are actually necessary to accurately represent the surface curve. An adaptive representation might be more economical with regard to memory usage. However in practice this is not a significant enough savings to warrant a non-uniform representation for the situations we have encountered in our areas of application. Furthermore, a non-uniformly sampled surface curve would have to be dynamically updated with regard to face-point distribution as it moved over the surface geometry. In contrast a uniform surface curve representation is trivial to maintain and update. For these reasons we have chosen to use a uniform surface curve representation over a non-uniform (i.e. adaptive) one.

3.7 Speeding up the surface snake implementation

The previous section suggested a straightforward implementation of the surface snake formulation: at each iteration step compute the resultant force on every face-point and move that face-point to a new position on the surface. Continue this iteration (which is essentially a relaxation process) until a minimum energy configuration is reached.

While this is a reasonable relaxation strategy, it is not a particularly efficient one for the density of our underlying data sets. Recall that we chose our face-point curve's sampling rate to be such that two face-points were separated on average by no more than the width of one polygon. Therefore, a dense underlying polygon mesh will in turn require that the face-point curves be densely sampled. The straightforward surface snake implementation on dense face-point curves gives rise to flaccid or unresponsive surface snakes i.e. shape changes of the snake propagate slowly. Note that this problem is distinct from the one of maintaining a surface snake that adapts to surface curvature. The cause for this inefficiency is easily understood with the help of an example. Consider an editing operation (see section 3.8.1) where a user pulls on one end of the snake to move the end point to a new location on the surface i.e. an *impulse* at one end of the snake. Assuming that the end point that the user pulls stays fixed to its new location on the surface, our surface snake must now attain a new minimum energy state through the snake relaxation process. For the snake to reach this state the effect of the editing operation must be allowed to propagate throughout the length of the snake. We call this the *impulse propagation* cost for a surface snake since it measures the time an impulse at one end of the snake takes to propagate throughout its snake.

Consider the impulse propagation cost for a simplistic relaxation process. Since each minimum energy iteration propagates the impulse by at most 2 face-points (the curvature term affects upto two neighboring face-points) it takes $O(\frac{N}{2})$ iterations for the effect of the editing operation to reach the other end of the snake. Since the cost per relaxation iteration is N, the cumulative computational cost for the impulse to reach the opposite end of the snake is $O(N^2)$. Note that this cost still doesn't measure the number of iterations beyond these $O(N^2)$ steps that are needed to reach a minimum energy state. It merely measures the time for an impulse at one end of the snake to reach the other end. In practice, several of these $O(N^2)$ impulse propagation steps will be needed to achieve the final equilibrium state of the surface snake. However, the impulse propagation cost is a reliable measure of the computational complexity of the snake relaxation process since the impulse propagation process is central to every minimum energy iteration.

It is worth noting that in the absence of a surface constraint the impulse propagation cost could be reduced to $O(N \log N)$ simply by using an implicit matrix based solution similar to the one employed by Kass et al. [Kass et al. 1988]. However, we cannot use this solution strategy because it would mean compromising our curve-on-surface constraint.

To make our surface snakes more rigid and responsive, a faster implementation of our minimum energy iteration is necessary. We propose a coarse-to-fine relaxation procedure that works as follows: for every face-point on the surface snake we compute the forces on that face-point based on a hierarchy of resolutions of the surface snake. To better understand this computation, let us consider the forward and backward tangent forces on the i-th face-point $\mathbf{v}_i$ based on this hierarchy of resolutions of a surface snake. Figure 3.11 provides an intuition for these forces. In the figure, resolution 0 is the default resolution of the surface snake (i.e. the highest resolution). It has N_0 face-points. For purposes of force computation, resolution R of the snake has

$$N_r = \frac{N_0}{2^R}$$

face-points (i.e. we drop every other face-point as we increase R). At resolution R, the force on $\mathbf{v}_i$ due to the face-point on the immediate "left" is labeled F_l^R and the force due the neighbor on the immediate "right" of $\mathbf{v}_i$ is labeled F_r^R. These forces are indicated in the figure for 3 different resolutions of the snake. The resultant force F_l on face-point $\mathbf{v}_i$ is now given by a weighted combination of the forces at all the resolutions of the snake i.e.

$$F_l^{result} = w_0 F_l^0 + w_1 F_l^1 + w_2 F_l^2 + w_3 F_l^3 + \ldots + w_M F_l^M \qquad (3.16)$$

where

$$M = \log_2(N_0)$$

A similar expression can be written for the resultant force from the right of $\mathbf{v}_i$. The weights are computed so that the coarser the resolution of the snake, the smaller the weight of the forces at that resolution. We have used the following weighting mechanism.

$$w_R = \frac{w_0}{2^R}$$

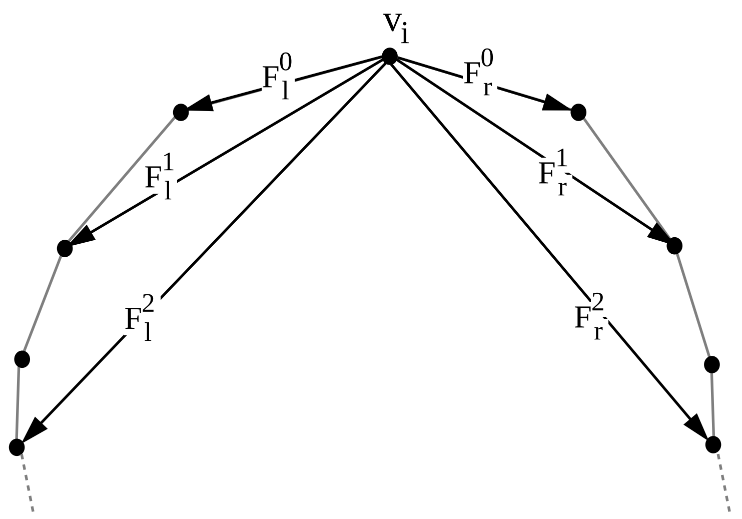

Figure 3.11. Coarse-to-fine relaxation for snakes. The figure shows the forces on a face-point $\mathbf{v}_i$ of the snake due to its immediate neighbors at three different resolutions of the snake. The resultant force due to the "neighbors" of $\mathbf{v}_i$ is a weighted combination of the forces at the different resolutions of the snake. These forces are then used in the snake minimum energy relaxation. The resulting snakes are more "rigid" and responsive than snakes produced by a static relaxation process. See the text for a detailed discussion.

The variational Euler forces on $\mathbf{v}_i$ are now computed based on these resultant forces. For example, the *fairness* force on $\mathbf{v}_i$ is now given by:

$$F_{fair}(i) = F_l^{result}(i) + F_r^{result}(i)$$

The other variational Euler forces are computed in a similar fashion to the one above. In practice, this coarse-to-fine computation is computed by simply selecting out the appropriate face-points from the high resolution snake. Thus it is not necessary to explicitly compute a number of resolutions of the snake for every single face-point $\mathbf{v}_i$. The coarse-to-fine snake computations can therefore be implemented efficiently.

Let us now consider the computational gains obtained due to our coarse-to-fine relaxation process. Computing Eulerian forces in the manner shown in equation 3.16 implies that the effect of an impulse at any point of the surface snake is immediately propagated to every other point on the curve in the very first iteration. However, the cost of each force computation is now $O(\log N_0)$ (there are $\log N_0$ terms in equation 3.16). Therefore the total impulse propagation cost our coarse-to-fine iteration is $O(N \log N)$.

Thus, instead of the $O(N)$ iterations of cost $O(N)$ each, that were required to propagate an effect in the static implementation (i.e. computations based on just the highest resolution), we now need only a single iteration of cost $O(N \log N)$. Because of this, snakes that use our coarse-to-fine iteration strategy achieve their minimal energy configuration an order of magnitude faster than fixed resolution snakes. In practice this speedup makes our surface snakes rigid and responsive which in turn makes them usable in an interactive setting.

3.8 Applications of surface snakes

The last two sections described an efficient implementation of our surface snake formulation. In this section we now explain how our our base surface snake formulation can be extended to include external energy constraints (i.e. $E_{constraint}$ from equation 3.5) as well as surface based energy constraints (i.e. E_{surf} from equation 3.5). We have implemented two specific extensions of our base surface snake formulation to illustrate each of these kinds of constraints. The first extension adds user-defined constraints to the formulation that enable a user of our system to perform editing operations directly on the surface curve. Our second extension enables the user to manipulate surface curves using arbitrary scalar fields defined on the surface. We demonstrate this using the example of surface color (we use only a single channel of the color) as scalar values at every mesh vertex.

Recall that the surface snake is essentially a controlled continuity spline whose internal energy terms are based on a combination of a membrane and a thin plate term. Our solution strategy converts the discretized variational form of the surface snake's minimum energy equation (equation 3.5) into a set of forces acting on individual face-points. These forces are in turn used in an iterative procedure that pushes the surface snake into its minimum energy configuration.

It is straightforward to see that any new energy measures that are different from the ones our surface snakes already possess could be added to our solution using the strategy outlined above i.e.

- First, add the energy measure to the existing thin plate and membrane energy terms.
- Second, derive the discrete variational form of the new minimum energy equation.
- Third, compute a new set of forces on individual face-points based on the discrete Euler equations.

However, since we eventually use a force based relaxation strategy a simpler alternative is to bypass the first two steps and directly derive a new set of forces based on our intuitions for the energy measure being included. In the next two sections we have employed this last strategy for the two extensions mentioned above.

3.8.1 Curve editing with surface snakes

Surface snakes can be edited using mechanisms similar to those used for conventional 2-D snake editing. For example Kass et al. [Kass et al. 1988] discuss the use of “volcanos” (a radial vector field originating at a point) to influence edit snake behavior. Figure 3.12 explains our surface snake editing paradigm. First, the user identifies a section of the surface curve that is to be edited. In our system this is accomplished by picking a face-point of the surface curve and by specifying a symmetrical length of curve around this face-point. We call the picked face-point an *edit point* of the surface snake and the symmetrical section around the edit point an *edit section*. The edit point can be an arbitrary face point of the surface snake and the length of the edit section can take on an arbitrary value limited only by the length of the surface snake itself.

Once the edit point and edit section have been identified, the user pulls on the edit point in an arbitrary direction in screen space. Our system converts this user interaction into a force on the edit point that is tangential to the surface at that point. This force now replaces the variational Euler forces at the edit point. It is worth noting that the variational Euler forces are no longer used to move the edit point itself. Rather its movement is dictated solely by the user specified force. Therefore if the user ceases the application of the force at the edit point, it does not move from its new location on the surface.

Let us assume that the user has moved an edit point to a new location on the surface using the editing interaction described above. As a next step we must update the surface curve based on the users input. To accomplish this we use our relaxation strategy selectively on the edit section. We modify our relaxation process such that only the face-points of the edit section, with the exception of the edit point itself, are moved. Therefore, during the minimum energy iteration, the two end points of the section are fixed to their original positions on the surface while the edit point is fixed to its user specified location on the surface. This strategy produces a smooth interpolation on the surface of the edit point and the end points of the section. Except for the edit section, the rest of the surface curve is not modified by this editing operation. Therefore, in a global sense our surface curve no longer satisfies our minimum energy criteria. However, the edit section itself attains a smooth minimum energy configuration under the constraints that its two end points and the edit point are at fixed locations. As before, the user can vary the relative weights of the curvature and fairness terms to modify the shape of the edited curve section (see section 3.6.2).

The curve editing operation described above two main advantages over the space curve based editing paradigm that we described in section 3.3:

- It is more intuitive to use since the user directly manipulates the surface curves rather than through an indirect mechanism such as the manipulation of control vertices of an associated B-spline space curve.
- It is more robust to abrupt changes in surface curvature. As explained earlier, if the space curve that is being used for editing purposes does not have a plausible projection on the surface (e.g. at high curvature regions of the surface) the space curve editing paradigm is prone to non-robust behavior.

Surface snake editing offers another notable advantage over space curve editing: the ability to use surface properties for curve editing purposes. This would be difficult to accomplish with a space curve based editing approach. We discuss this in the next section. Despite

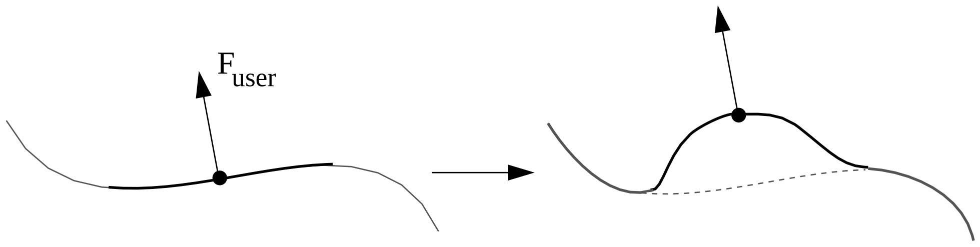

Figure 3.12. Surface snake based curve editing. The figure shows the interaction that we have provided for surface snake based curve editing. A user picks an *edit point* on the surface curve (shown as a solid circle) and an *edit section* around that point (shown darker than the rest of the curve). The edit point can be any arbitrary face-point of the surface curve. The user performs an editing operation by pulling on the edit point in an arbitrary direction (in screen space). This interaction is modeled as a force on the edit point that is tangential to the surface at that point. This force is then used in conjunction with our variational Euler forces in the surface snake relaxation process. The result of this relaxation process produces our edited result. One example of an edited surface snake is shown in the figure on the right. Note that the edit section stays smoothly attached to the rest of the curve i.e. it maintains C^1 continuity at the boundaries of the edit section. It is worth noting that extreme deformations of the edit section can generate a sharp looking corner at the boundaries of the section. This is because our surface-curve representation is discrete. As such if the sampling density is insufficient, high curvature regions of the curve can appear sharp i.e. as C^1 discontinuities.

the pitfalls associated with space curve editing, the approach does have some advantages. For example it is a familiar editing paradigm (see section 3.3). Furthermore, the problems with space curve based editing approach may be partially alleviated when it is used in conjuction with the surface snake formulation. Therefore, in our system we have provided the user with both surface snake based editing tools as well as space curve based editing tools. The user can choose to use either one of these tools (or a combination thereof) to manipulate a patch boundary curve. Some examples of the editing techniques explained in this section are demonstrated in figure 3.13.

3.8.2 Using surface color for curve editing

In this section we propose a tool for surface snake *color attraction*. Using this tool a surface snake can be made to conform to a shape defined by the color information on the underlying polygonal mesh vertices. Kass et al. [Kass et al. 1988] demonstrate similar tools for 2-D snakes on images. In our discussions we will treat color as a scalar entity. Converting an (r, g, b) color value to a scalar can be accomplished in a straightforward way by either using a single color channel or by computing the magnitude of the color vector. As such, our tool may be used with trivial modifications for arbitrary scalar fields defined on the polygonal surface (e.g. Gaussian curvature).

This tool has several useful applications. Consider an input polygonal mesh created by scanning in a physical object on which the intended patch boundaries are physically painted on it. Our input in this case would be a dense polygonal mesh with an additional per vertex color. In this situation our color attraction tool allows a user to precisely position the patch boundary curves based on surface color. Another application of this tool is for creating surface curves that follow in creases or folds (i.e. highly curved regions) of the polygonal mesh. In this case, Gaussian curvature (being a scalar value) may be substituted for color in the color attraction tool. The tool could then be used to automatically attract boundary curves into the required creases and folds on the surface.

As before, our method for implementing the color attraction tool works in two steps: first, we derive a new set of *color attraction forces* (on the individual face-points of a surface curve) that capture our intuitions for a desirable minimum energy solution. Second, we use

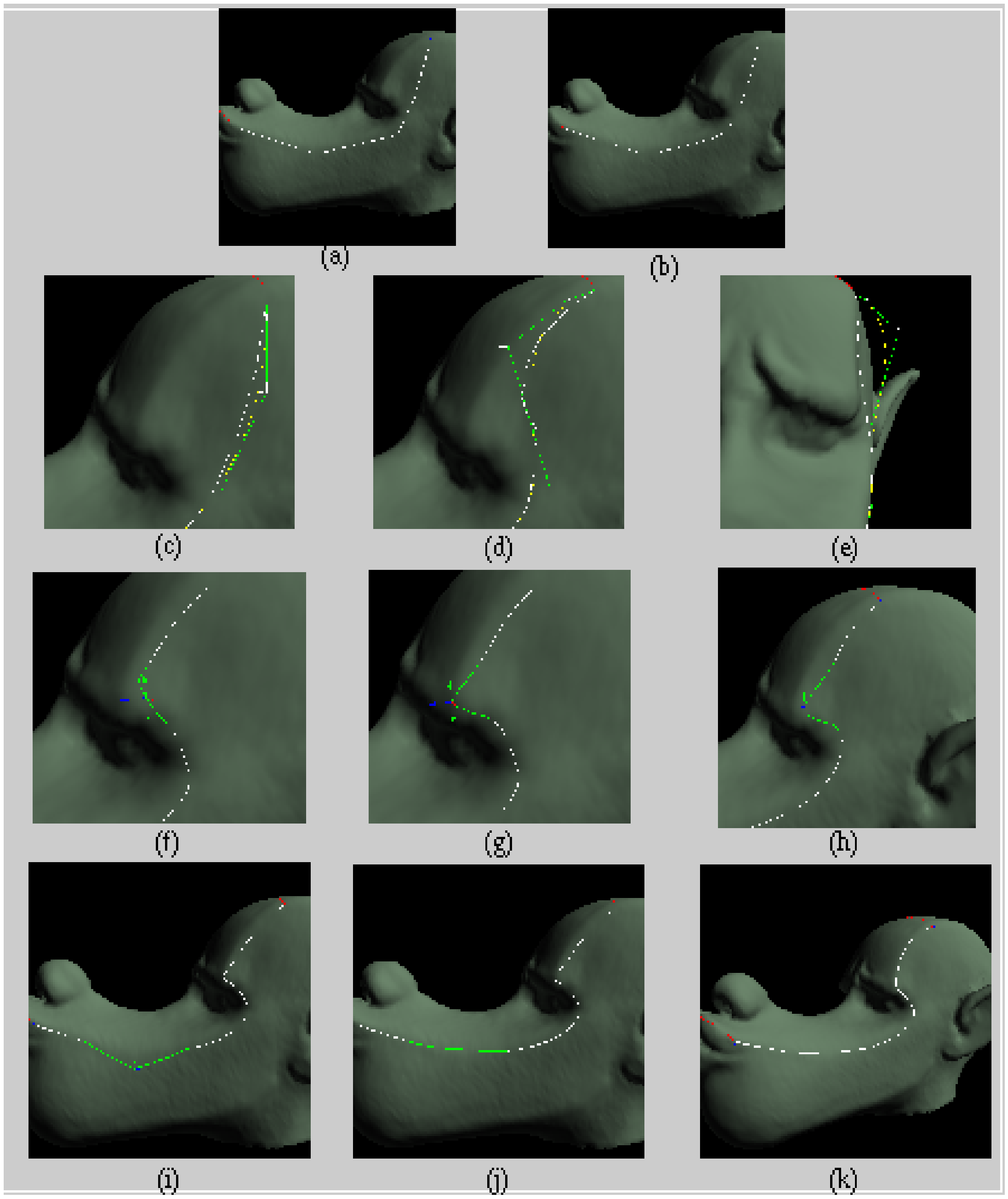

Figure 3.13. This figures shows a sequence of steps in painting and editing a surface curve on the polygonal mesh of a wolf's head. In each of the images the white curve shows the surface curve being edited. (a) shows the crude initial curve painted by the user. (b) shows the results of a minor smoothing operation that smooths the curve without changing its character. This result is obtained by using just the internal energy surface snake terms. (c) through (e) show space curve based editing and some problems associated with the paradigm. A B-spline curve that was fit to the surface curve is used to move a section of the surface curve from the position in (c) through (d). The control mesh for this curve is shown as a green polyline with control points shown as white crosses. The yellow curve shows the position of the space curve. As the spline curve is manipulated in space, it drags the surface curve with it. Note that the edited surface curve shown in (d) appears to "wiggle". The reason for this becomes apparent in image (e): the user has moved the space curve (i.e. its control points) far away from the surface. As a result space curve projection is no longer robust. (f) through (j) show a sequence of surface snake edits that achieve the final result in (k). Note that snake based editing does not suffer from the limitations of space curve based editing. The edit sections are shown in green and the edit point is shown as a set of colored arrows. Notice that the user can choose arbitrary edit points on the curve, and arbitrary lengths for the edit section. The painting and editing operations shown here were all performed at interactive speeds. This model was supplied by Industrial Light and Magic.

these set of forces in conjuction with existing minimum energy forces in our surface snake relaxation algorithm.

In accordance with this two-step approach, let us first focus on the computation of the color attraction forces. Since we would like our surface snakes to conform to underlying surface color, a reasonable strategy for computing these forces might be to attract individual face-points to any nearby mesh vertex with a non-zero color value on the polygonal surface. Unfortunately, this straightforward approach produces unstable minimum energy configurations. The reasons for this non-robust behavior are two-fold. First, the magnitude of surface color tends to vary irregularly over the surface. A colored section of the polygonal mesh that we are attracting our snake to is typically a ribbon (of colored vertices) of irregular width rather than a well defined, thin, smooth ribbon of color. Second, since the color data is defined only at mesh vertices the boundary of a colored region tends to posses a "jagged" outline.

A better solution is to derive a force based on the color gradient. The magnitude of the color gradient establishes the edge (or boundary) of a painted section of the mesh while the direction of the color gradient is used to propel the surface curve into the smooth zero gradient interior of the colored region. We define the *gradient force* as the force at an arbitrary face-point due to the surface color gradient. Figure 3.14 explains the gradient force computation for irregular polygonal surfaces for a vertex $\mathbf{p}_0$ with color magnitude S_0. Let the M vertices connected to $\mathbf{p}_0$ be labeled as $\mathbf{p}_1$ through $\mathbf{p}_M$ with color magnitudes S_1 through S_M respectively. We compute the color gradient at $\mathbf{p}_0$ as:

$$\mathbf{e}_0 = \sum_{i=1}^{M} (S_i - S_0)(\mathbf{p}_i - \mathbf{p}_0) \tag{3.17}$$

We compute and store this gradient at each of the mesh vertices. At an arbitrary face-point the gradient force is given by an interpolation of the gradients at the vertices of that face, based on the barycentric coordinates of the face-point. This gradient force now becomes the color attraction force on that particular face-point.

Figure 3.15 provides an intuition for how our computed color attraction forces are used in the surface snake relaxation strategy. At each face-point of the surface curve, we add to the variational Eulerian forces an additional force due to color attraction. This force is

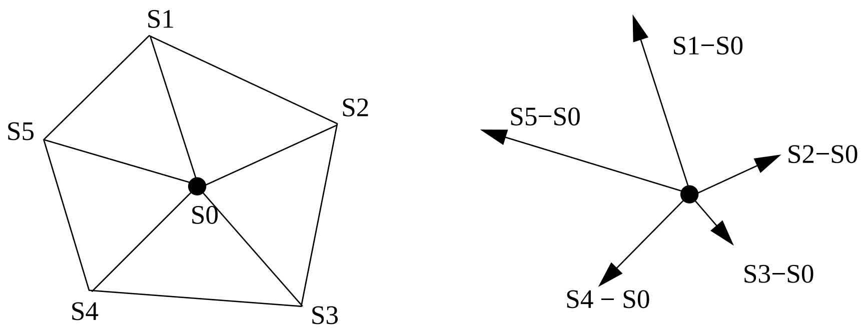

Figure 3.14. Computation of the color gradient at a vertex. $S0$ through $S5$ represent the color magnitudes at the vertices shown. We compute the color gradient as a weighted combination of vectors corresponding to outward pointing edges from the vertex to all its neighbors. Weights are assigned based on the differences in the magnitudes of scalar values at the vertices. See equation 3.17 and the text for further details.

computed as the component of the gradient force at that face-point that is in the direction of the normal (on the surface) to the curve i.e. we do not use the component of the gradient force that is tangential to the curve at that face-point. This is because the tangential component of the gradient force can only cause a redistribution of face-points along the length of the curve. Recall that the distribution of our face-points along the surface curve is already addressed by the arc length criterion. Therefore, including an additional tangential force at each face-point of the surface curve will interfere with the arc length criterion. This is undesirable since it can cause a bunching up of face-points along sections of the surface snake. For this reason we exclude the tangential component to the curve (on the surface) of the gradient force from our color attraction force computation.

The resultant force on a particular face-point $\mathbf{v}_i$ of the surface snake is given by a weighted combination of the color attraction force and the variational Euler forces.

$$F_{result}(i) = K_{internal}F_{internal}(i) + K_{color}F_{color}(i) \tag{3.18}$$

By varying the relative weights of these terms a user can determine how closely the surface snake follows surface color information. A higher relative value of K_{color} makes the surface snake conform more closely to surface color (at the expense of internal energy). This setting is appropriate for cases where the supplied color information is precise and well defined. On the other hand if the supplied color information is noisy or irregular, a lower K_{color} value

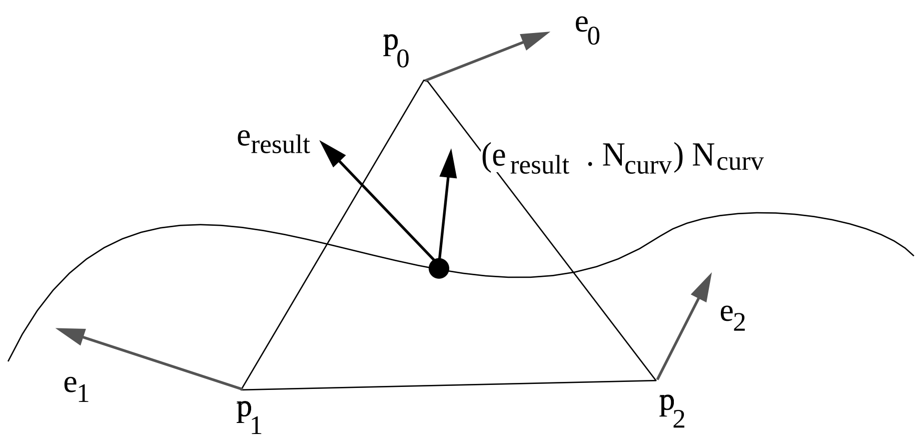

Figure 3.15. Using the color attraction force during surface snake relaxation. The figure shows the computation of the color attraction force at an arbitrary face-point (shown as a solid circle) of a surface curve. First we compute the gradient force e_{result} at the face-point. Second we extract the component of this gradient force that is normal to the surface curve (and tangential to the surface itself) at the face-point. Therefore, if the normal (on the surface) to the curve is given by N_{curv} the color attraction force is given by $(e_{result} \cdot N_{curv})N_{curv}$. This force is now used in conjunction with the variational Euler forces to move the surface curve to its minimum energy location.

is preferable. This weighting ensures that the snake is only roughly guided by the noisy surface color information. Some results of the techniques of this section are demonstrated in figure 3.16.

3.9 Summary

Curve editing operations such as the ones explained in the last section demonstrate the flexibility of our surface snake formulation. First, since tools based on surface snakes operate directly on the surface curve (rather than indirectly through a space curve) they are more intuitive to use than tools based on space curves. Second, they are more robust to abrupt changes in surface curvature. Finally, our surface snake formulation allows the effective use of surface properties such as vertex color to assist in the curve painting and editing process. Such operations would not have been possible to duplicate using space curve based editing.

Our curve painting and editing tools (i.e. both space curve and surface snake based

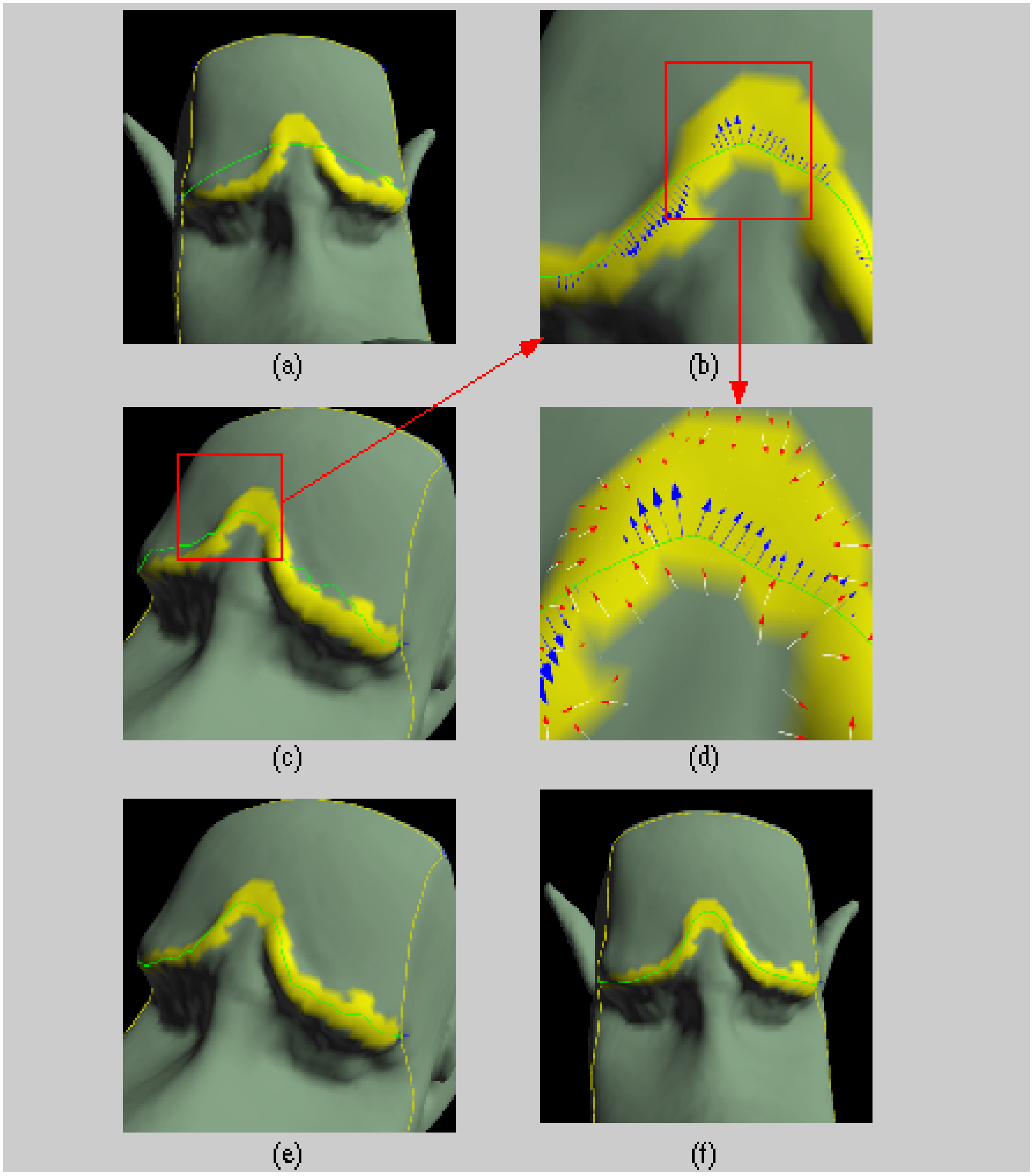

Figure 3.16. (a) through (f) demonstrate color attraction of a surface curve (a feature curve) drawn on the wolf's head. For this particular example, the yellow band of color was painted in by a user. In general, this color could have come from an arbitrary source e.g. as input from a laser scanner that captures color as well as depth data. Note that in this particular case, the curve could have been painted in directly by the user. However, we are using this example merely to demonstrate the concept of color attraction. The user specifies a crude initial surface curve shown in green. The surface color then attracts this crude surface curve so that it passes smoothly through the irregularly painted surface. The final location of the curve is shown in (f). Two intermediate locations of the surface curve during the color attraction process are shown in (c) and (e). (b) shows a detail of a region of the surface curve. The magnitude and direction of the color attraction force field are shown as blue arrows. (d) shows a further magnification of part of the image in (b). This image shows the gradient field on the surface due to the surface color in the form of red tipped arrows. The vectors are selectively displayed at mesh vertices. The color attraction force field is computed based on this gradient field. Note that the final surface curve in (f) passes smoothly through the interior of the colored region.

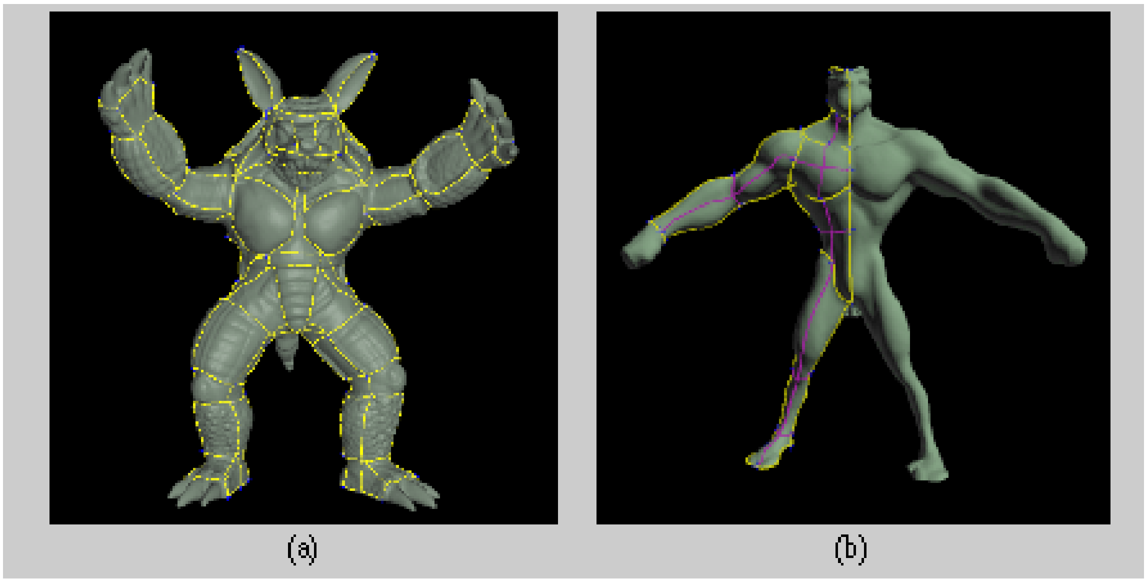

Figure 3.17. The Armadillo (shown in (a)) has about 200 surface curves and 104 patches. As noted earlier it has about 350,000 polygons. The curve placement and editing of this curve set took about 2 hours. Most of this time was spent in interactive operations rather than computation. (b) shows a polygonal mesh of an action figure called *Bofar* (Bofar appears here courtesy of Domi Piturro of Synthesis Studio, San Fransisco). The Bofar mesh has about 450,000 polygons. It has 12 patches and 33 boundary curves. In addition each patch has one or more feature curves (shown in purple) that were added to it. These patches and boundary curves were carefully placed to allow for subsequent animation. Once again, the curve specification process took about 2 hours with the bulk of that time being spent on precisely tuning the final location of the surface curves. As an interesting aside note that even though Bofar is less detailed than the Armadillo it has more polygons because it was scanned in from an 18 inch clay mold at 1 mm resolution. The Armadillo was only about 6 inches high and was scanned in at about 0.2 mm resolution.

tools) allow a user to efficiently specify complex curves on dense polygonal meshes. Figure 3.17 shows an example of two complex models with a complete set of patch boundary and feature curves specified on the model using the techniques of this chapter.

Chapter 4

Parametric surface approximation: an introduction

We approximate our dense polygonal meshes with a network of tensor product B-spline surface patches and associated displacement maps. Each region of the polygonal mesh that must be approximated by our hybrid surface representation is bounded by a set of four boundary curves. These curves lie on the polygonal surface, but they need not align with the vertices and edges of the mesh. We call each such four-sided section of the polygonal mesh a *polygonal patch*. As explained earlier, the placement of the boundaries of these B-spline surface patches is part of the creative process and is not easily automatable. Therefore, in the first step of our approximation process the user provides a set of boundary curves as an input. These curves are specified using the curve painting tools explained in the previous chapter. Our surface approximation problem now reduces to the following simpler problem:

Given a polygonal patch, find a combination of uniform tensor product B-spline surface and displacement map that best approximates this polygonal patch.

Recall from chapter 1 that our proposed surface fitting solution is a two-step process that first creates a spring mesh for each polygonal patch and then fits the B-spline surface and displacement map to the spring mesh. In this chapter, we examine the characteristics that govern our approximation problem and explain our reasons for using this two-step fitting strategy.

We begin by examining the problem parameters in a general surface approximation problem (section 4.1). Then, in section 4.2 we examine the characteristics that govern our specific approximation problem. Based on these characteristics we then identify several features of a desirable solution to our approximation problem. In section 4.3 we discuss fitting strategies for two closely related approximation problems and explain why each of these strategies is inadequate for our purposes. This discussion motivates our two step approximation strategy of first generating a spring mesh and second fitting to this spring mesh. Finally in section 4.4 we outline our new fitting strategy.

4.1 Surface approximation in a general setting

It is instructive to place our problem in the context of a more general surface approximation problem. The general problem of surface approximation in $\Re^3$ is simply stated [Schumaker 1979]:

Suppose F is a real valued function defined on $\Re^3$ and we are given the values $F_i = F(P_i)$ of F at some set of (data) points P_i in $\Re^3$, $i = 1, 2, \ldots, n$. Find a fitting function F_{fit} defined on $\Re^3$ which reasonably approximates F.

As can be inferred from the broad scope of this problem statement, there is no single definitive strategy for the general approximation problem. Rather, the statement can be viewed as a general template for defining and classifying specific surface approximation problems (and solutions). One can identify three characteristics in the problem statement that must first be well defined to create an instance of the general problem. These are as follows:

1. Characteristics of the input data (e.g. input data topology, statistical properties of noise in the data etc.)

2. The choice of a smooth surface primitive for the fitting function F_{fit}.

3. Characteristics of the fit (e.g. should the fit interpolate or approximate the data, how should we measure the quality of the fit etc.)

The field of approximation theory is populated with a variety of instantiations of the surface approximation problem (in a diverse set of domains). In general, for any surface approximation problem it is important to carefully examine the specific problem parameters in each of the three categories above before arriving at an appropriate solution strategy. For a fairly exhaustive survey of this field see Nielson's [Nielson 1993], Schumacher's [Schumaker 1979] and Franke and Schumacher's [Franke & Schumaker 1986] surveys on the subject. See also our classification of other relevant problem domains in chapter 2. In the following discussion we develop only those insights from this field that apply to our specific approximation problem.

4.2 Characteristics of our surface fitting problem

Let us examine our surface approximation problem in the light of the each of the characteristics listed above. We begin by considering the input data characteristics.

4.2.1 Characteristics of the input data

Note that while our polygonal meshes have arbitrary topology, each individual polygonal patch is itself u-v parameterizable i.e. it is some arbitrary manifold deformation of a four sided planar surface. We do not make any other assumptions about the topology or the geometry of a polygonal patch. In particular, polygonal patches are not restricted to be height fields.

From a statistical perspective, the polygonal meshes that are input to our surface fitting system represent an optimal reconstruction from a set of laser scanned range images under a certain set of well-founded assumptions such as Gaussian statistics for the noise in the laser scanned data (see [Curless & Levoy 1996] for further details). We are therefore approximating not just the raw (potentially noisy) data but an aggregated surface representation of it. This has a two significant implications for our surface fitting system. First, there is nothing sacrosanct about our mesh vertices since they do not necessarily themselves represent the original scanned data points i.e. we need not interpolate them.

Second, although we do not feel compelled to interpolate the vertices in our input mesh, neither do we feel justified in excessively smoothing or fairing the surface we fit to it. Several surface approximation schemes [Eck & Hoppe 1996, Sinha & Seneviratne 1993] add smoothing (or regularizing [T. Poggio & Koch 1985]) terms such as the thin plate spline [Terzopoulos 1988] during the surface fitting process. Such smoothing terms are relevant only in the context of fitting to sparse data where the approximation problem is under-specified (or ill posed [T. Poggio & Koch 1985]) or in the context of fitting to noisy input data where smoothing is appropriate. Our input meshes are neither sparse nor noisy and therefore any desirable solution to our fitting problem should avoid smoothing out the approximating B-spline surfaces using regularizing terms.

An obvious but often overlooked characteristic of our input data is that polygonal patches are composed of a set of connected polygonal facets rather than just a set of scattered data points. As will be demonstrated in the following chapters, we can use this connectivity information (and associated topological structure) to our advantage allowing us to produce superior surface approximations. By contrast, several commercial systems (Imageware's Surfacer [Sinha & Seneviratne 1993], CDI's Surface Design, DelCAM's CopyCAD) and research systems [Milroy et al. 1995] work exclusively with point clouds. When presented with polygonal meshes such systems choose to discard the connectivity information offered by the polygonal mesh and convert the input once again to point clouds. This strategy can produce incorrect results. Figure 4.1 demonstrates this with a simple example. There are other serious objections to converting our polygonal mesh fitting problem to a point cloud fitting problem. These include a loss in the efficiency and robustness of the fitting procedure as well as a loss in the quality of the fitted surface. We discuss these drawbacks in section 4.3.1 where we discuss point cloud fitting in greater detail.

4.2.2 The choice of surface primitive for the fitting function

The second of the three characteristics enumerated in section 4.1 is the choice of a smooth surface representation for the fitting function F_{fit}. Our choice of smooth surface representation is already specified; it is a combination of a tensor product B-spline surface and

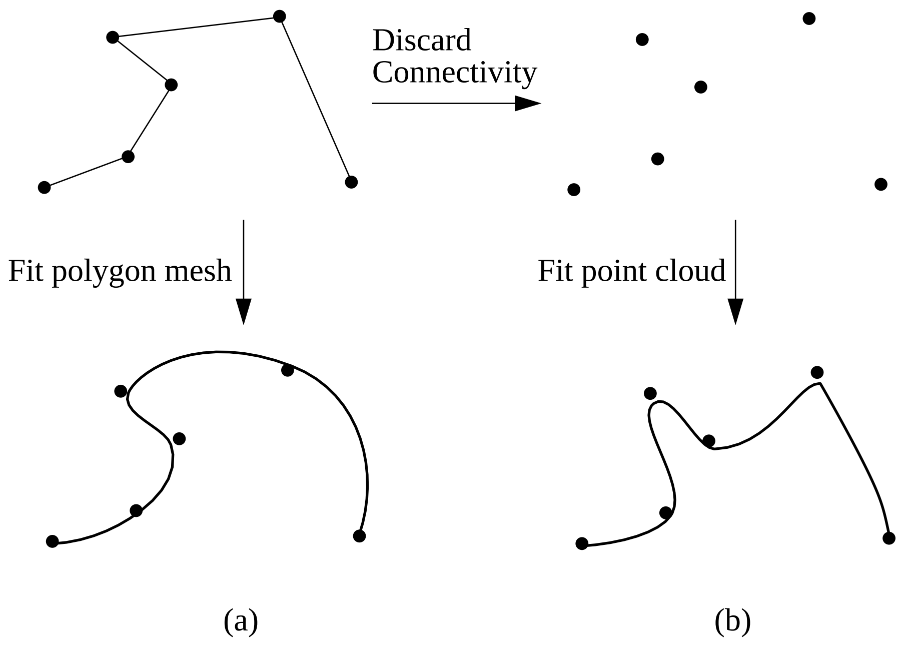

Figure 4.1. Discarding connectivity information before fitting to the polygonal mesh can result in incorrect fits. (a) shows a polyline that is fit with a smooth curve. (b) shows a similar fit to the input data after the connectivity information has been discarded. Note that the fitted curve is different in the two cases. The result produced by point cloud fitting is clearly incorrect in this case since it violates the connectivity of the original data set. For surfaces in three space point cloud fitting can produce surface fits that violate the topology of the input data set.

displacement map. Our reasons for choosing this hybrid representation were discussed in chapter 1. Briefly, the uniform tensor product B-spline surface was chosen to capture the coarse geometry of our polygonal patch, while the displacement map was chosen to capture fine surface detail. It is worth noting that our B-spline surface representation is one specific choice of a parametric surface; our general approach is extensible to other parametric surfaces (such as Bezier, NURBS, Catmull-Rom, H-splines etc.).

Given our hybrid representation, one obvious question is: how do we decide the separation between the extent of geometry represented by the spline surface and that represented by the displacement map? Since our goal was to build an interactive surface fitting system, we leave this decision to the user. In our approach, the user has full control over the resolution of the B-spline surface to be fit to the geometry. The displacement map is calculated to automatically capture the residual geometry that was not captured by fitting the smooth B-spline. This provides the user with a lot of flexibility and for the application areas we have explored (mainly in entertainment), it is perceived by users as one of the principal advantages of our approach. In order to successfully implement this strategy our fitting must satisfy several criteria.

First, if the user is to make a informed decision about the resolution of B-spline surface, the fitting process should be fast enough to be used in an interactive setting. Only this would enable the user to conveniently experiment with a number of resolutions of B-spline surface before settling on a particular resolution.

Second, we would like to compute the best possible B-spline surface approximation to the underlying polygonal surface given a fixed number of control vertices. This is because, in our system the user may choose to create an extremely high resolution spline surface that completely captures all the surface detail. Therefore our surface fitting algorithms should be capable of capturing this surface detail entirely within the spline surface if required. It is worth noting that our smooth surface representation i.e. a B-spline surface with a uniform knot spacing is intrinsically incapable of capturing certain kinds of surface detail like sharp corners and edges. These must inevitably be captured by the displacement map.

4.2.3 Characteristics of the fit

There are two important characteristics of any fit that must be specified before we may choose (or devise) a fitting algorithm. The first of these is the issue of how closely the fitting function approximates the input data. If the fitted function is constrained to pass through the data points, it is an interpolating function; otherwise it is an approximating function. As explained in section 4.2.1, we are interested in a surface approximation to the polygonal mesh surface rather than just the mesh vertices. Therefore, an interpolating function in our context would be one that interpolated the polygonal mesh surface rather than just the mesh vertices. The density of our input data makes this an impractical objective since it could potentially require a very high number of control vertices to exactly interpolate the mesh surface. However, we are still interested in a high quality approximation to the mesh surface itself.

The second characteristics of our fit that must be specified is what it means for a smooth surface (or B-spline and displacement map) to be a “high quality approximation” to the polygonal mesh surface (i.e. in a mathematical sense). One method of measuring the quality of an approximating smooth surface is to examine the error of fit. Using this method, a lower error of fit would imply a higher quality surface approximation. However this quantification of surface quality does not necessarily ensure desirable results. As has been noted in the geometric interpolation literature [Lounsbery et al. 1992, Halstead et al. 1993] it is possible to generate zero error of fit surfaces (i.e. interpolating surfaces) that display “unfair” characteristics (i.e. the fitted surfaces tend to shown unsightly bumps and wiggles). Thus for two fits of equal error, one might be preferable to the other because it is fairer.

Furthermore, a unique property of parametric surface fits is that even given two fits of equal error and fairness, one might be preferable to the other because it has a better parameterization i.e. a fair surface fit could still possess an unsatisfactory parameterization. Thus a high quality approximation in our context is one that possesses a low error of fit (in the sense described above), is geometrically fair and that has a high quality parameterization. We quantify the notion of what it means for a parameterization to be of a “high quality” in chapter 5.

4.2.4 The defining features of our solution strategy

Based on the preceeding discussion, let us summarize a set of defining (or desirable) features of our solution strategy.

1. **Our input polygonal patches have no topological or shape constraints other than that they are u-v parameterizable** i.e. each polygonal patch is an arbitrary manifold deformation of a four sided planar surface. No further assumptions are made about patch topology or geometry. In particular, patches are not required to be height fields.

2. **The control vertex resolution of the B-spline surface is chosen by the user**. Our fitting strategy should create the "best possible" B-spline surface approximation at this user-specified resolution. The remaining geometry is captured in the displacement map. We would like this fitting procedure to work at interactive speeds to allow the user to easily experiment with multiple resolutions of B-spline surface (and hence displacement map).

3. **It should be possible to capture all the geometry in the B-spline surface without smoothing**. As explained earlier the user might choose to capture the entire geometry of a polygonal patch with just the B-spline approximation (rather than use a displacement map). If smoothing terms (such as membrane or thin plate terms) are introduced these will likely produce overly smoothed approximations that cannot capture all the geometry in the B-spline surface.

4. **The combination of B-spline surface and displacement map need only approximate the mesh surface, rather than interpolate the mesh vertices**. As explained in the preceeding sections, our polygonal mesh represents an aggregated surface representation of the raw data and our mesh vertices do not necessarily represent the original scanned data points. We are thus interested in a high quality approximation to the polygonal mesh surface rather than just the mesh vertices.

5. **The mesh connectivity should be used to assist the surface fitting process**. This is important to note because traditional fitting solutions often discard connectivity

thereby converting the problem to one of point cloud fitting. As explained in section 4.2.1 this approach can compromise correctness, robustness, efficiency and quality of fit.

4.3 Parametric surface fitting: two traditional formulations

Our surface fitting strategy is motivated in part by the solutions to two traditional approximation problems that are closely related to ours: B-spline approximations to irregular point clouds and B-spline approximations to organized or gridded data. These problems and their solutions provide useful insights into possible strategies for solving our approximation problem. In the following discussion we will assume uniform cubic (order 4) tensor product B-spline surfaces but the formulation will hold for other kinds of parametric surfaces as well.

4.3.1 Fitting to point clouds in $\Re^3$

Although we do not propose to employ fitting to unorganized point clouds, a brief consideration of this problem provides useful insights into our own fitting problem. In general, parametric curve and surface fitting to point cloud data can be formulated as a non-linear least squares problem [Dierckx 1993, Rogers & Fog 1989]. The equation for a B-spline surface $\mathbf{F}(u, v)$ can be written as:

$$\mathbf{F}(u,v) = \sum_{j=0}^{n}\sum_{k=0}^{m} \mathbf{X}_{j,k} B_j(u) B_k(v) \tag{4.1}$$

where u and v are parameter values in the two parametric directions of the surface, $\mathbf{X}_{j,k}$'s are control points (in $\Re^3$) of the B-spline surface and B_j and B_k are fourth order, end-point interpolating B-spline basis functions. This equation associates a point $\mathbf{P}(x, y, z)$ in $\Re^3$ for each point (u, v) in parameter space i.e.

$$\mathbf{F}(u,v) = \mathbf{P}(x_{u,v}, y_{u,v}, z_{u,v})$$

The B-spline basis functions B_j have been well studied in the literature. We refer the reader to the books by Bartels et al. [Bartels et al. 1987] and Farin [Farin 1990] for further details on the mathematical and computational details associated with using these and other similar parametric basis functions. In our formulation above, the control vertices of the B-spline surface completely determine the shape of the surface. Thus finding a B-spline surface approximation of a fixed resolution corresponds to finding the appropriate set of control vertices.

Now let $\{\mathbf{P}_i(x, y, z)|i = 1 \ldots L\}$ be a point cloud (i.e. a set of points with no connectivity information) in $\Re^3$ to which a B-spline surface of fixed resolution must be fit. Given the desired final resolution (m by n) of the B-spline surface, the fitting problem is: find the control vertices $X_{i,j}$ such that the resulting B-spline surface represents a reasonable approximation to the data points.

To accomplish this fit we must associate a point (u_i, v_i) in parameter space to each data point $\mathbf{P}_i$. This association, called a parameterization of the data, is not given to us initially. Let us assume for the moment that we are given a parameterization of the data. From equation 4.1 we can then write, for each data point $\mathbf{P}_i$:

$$\mathbf{P}_i = \mathbf{F}(u_i, v_i) = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} \mathbf{X}_{j,k} B_j(u_i) B_k(v_i) \quad i = 1 \ldots L \tag{4.2}$$

The right hand side of this equation is linear in the unknown $\mathbf{X}_{j,k}$'s. Thus we get L linear equations in the control points $\mathbf{X}_{j,k}$ (one for each data point). Writing these linear equations in matrix form:

$$\begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_L \end{bmatrix} = \begin{bmatrix} B_0(u_0)\mathbf{B}_n(v_0) & B_1(u_0)\mathbf{B}_n(v_0) & \ldots & B_m(u_0)\mathbf{B}_n(v_0) \\ B_0(u_1)\mathbf{B}_n(v_1) & B_1(u_1)\mathbf{B}_n(v_1) & \ldots & B_m(u_1)\mathbf{B}_n(v_1) \\ \vdots & \vdots & \vdots & \vdots \\ B_0(u_L)\mathbf{B}_n(v_L) & B_1(u_L)\mathbf{B}_n(v_L) & \ldots & B_m(u_L)\mathbf{B}_n(v_L) \end{bmatrix} \begin{bmatrix} \mathbf{X}_n(0) \\ \mathbf{X}_n(1) \\ \vdots \\ \mathbf{X}_n(m) \end{bmatrix}$$

where

$$\mathbf{B}_n(v_i) = [B_0(v_i) B_1(v_i) \ldots B_n(v_i)]$$

and

$$\mathbf{X}_n(i) = [X_{i,0}X_{i,1} \ldots X_{i,n}]$$

Re-writing the above equation in a more compact form:

$$\mathbf{P}_{vec} = \mathbf{B}_{mn,L}(u,v)\mathbf{X}_{vec} \tag{4.3}$$

In this equation the dimensions of the matrices are as follows: $\mathbf{P}_{vec}$ has dimensions $L \times 1$, $\mathbf{B}_{mn,L}(u,v)$ has dimensions $L \times mn$ and $\mathbf{X}_{vec}$ has dimensions $mn \times 1$. In a surface approximation problem, the number of data points L is usually far in excess of the number of control vertices mn being used to specify the spline surface. These equations therefore represent an over-constrained (since $L > mn$) linear system where the $X_{i,j}$ are unknowns. Assuming the matrix $\mathbf{B}_{mn,L}(u,v)$ is well conditioned, this set of equations is easily solved using a conventional least squares method [Lawson & Hanson 1974]. If $L \sim mn$ (i.e. the solution is under-constrained) or the input data is not evenly distributed among parameter values then the matrix $\mathbf{B}_{mn,L}(u,v)$ will usually not be invertible or will produce unstable results. Approximation problems that are characterized by either of these conditions are ill-posed problems. In such cases it is usually necessary to add an additional regularization or smoothing function to the fitting equation to make the problem well posed [Dierckx 1993].

The above discussion shows that given an accurate parameterization of the data set, solving the B-spline approximation is relatively straightforward. However, we are not given this parameterization. Therefore in equation 4.3 both the u, v values for each data point as well as the control vertices are unknowns. Furthermore, since our B-spline basis functions are non-linear (actually: cubic) the equations that comprise the over-constrained "linear" system above are in fact not linear. Note that any non-constant parametric basis function would make our problem a non-linear one.

Therefore the general problem of parametric surface fitting is a non-linear least squares problem. To solve this problem we must find both a parameterization of the data set as well as the best possible set of control vertices that approximate the data set given this parameterization.

A typical method of solving this problem uses an iterative optimization strategy [Milroy et al. 1995, Rogers & Fog 1989]. An outline of this fitting strategy is shown in figure 4.2. The optimization procedure begins by guessing an initial set of parameter values for the data points. Given this parameterization, equation 4.3 is solved for a first (usually crude) B-spline approximation to the data. This B-spline approximation is then used to re-parameterize the data and equation 4.3 is then solved again with the re-parameterized data. Re-parameterization of each data point is accomplished by a search in parameter space for a point on the intermediate B-spline surface that is closest to the data point. The motivation for re-parameterization and re-fitting is to improve the initial crude parameterization. The optimization strategy iterates in this manner between the parameter refinement step and the fitting step until some fitting tolerance is achieved.

There are several important aspects to note about this iterative fitting procedure. First, the convergence of the iteration (and hence the quality of the fitted B-spline surface) is strongly dependant on the initial parameter values. A crude initial parameterization invariably results in the iteration falling into local minima or, in the worst case, an iteration that does not converge. This in turn results in poor quality B-spline approximations. This loss in quality of fit manifests itself as unsightly bumps and wiggles in the approximating spline surface [Hoschek & Lasser 1993, Ma & Kruth 1995]. Surfaces that exhibit these artifacts are said to be "un-fair" surfaces. To prevent these unsightly artifacts, researchers often add regularization functionals to the fitting procedure [Terzopoulos 1986, Eck & Hoppe 1996]. These regularization terms are typically variants of thin plate or membrane splines and alleviate the fitting problems by smoothing out the approximating surface. While the use of these regularizing (smoothing) terms might be appropriate in the context of approximating sparse or noisy data, including them as part of our fitting problem would sacrifice surface detail and therefore the quality of fit. Our input data is dense and of a high quality (in the sense discussed in section 4.2.1). Therefore, using regularizing terms in the manner described above would only serve to cover up (i.e. smooth out) the fundamental problems created by a crude parameterization of the input data. We would prefer to not introduce these smoothing terms to our surface fit.

A second noteworthy aspect of this fitting procedure is the time complexity of each iterative step. At each step, we must first solve the linear least squares problem for an intermediate approximation and second we must refine the parameter values based on this

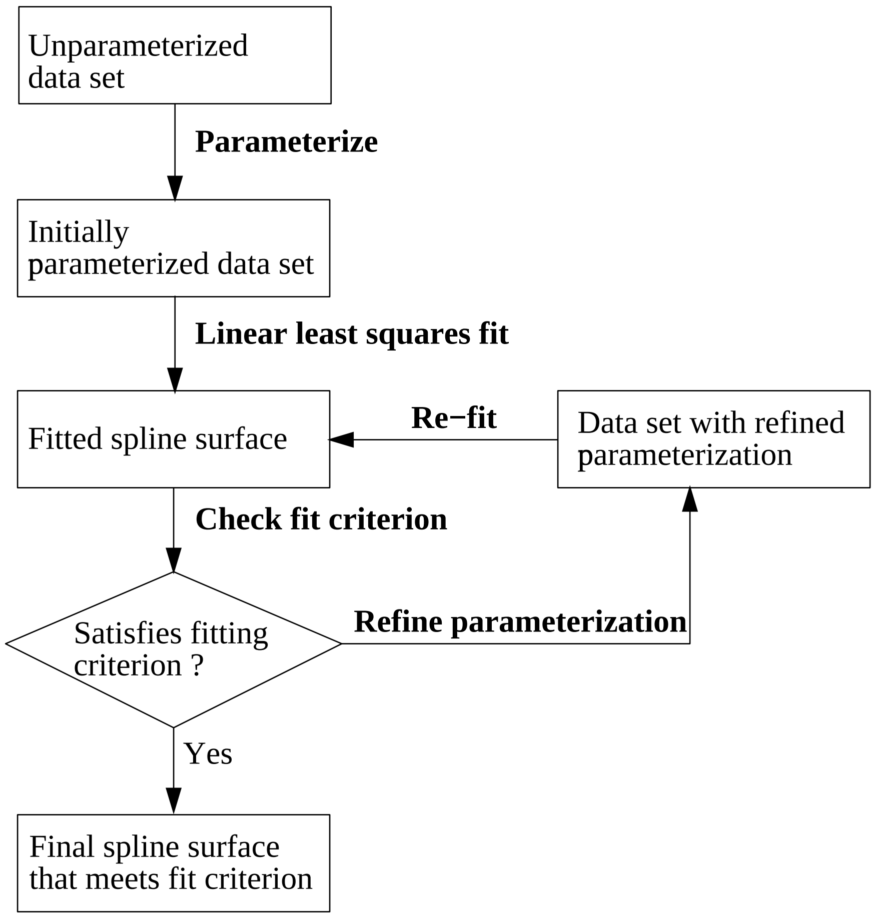
Unparameterized data set
Parameterize
Initially parameterized data set
Linear least squares fit
Fitted spline surface
Re−fit
Data set with refined parameterization
Check fit criterion
Satisfies fitting criterion ?
Refine parameterization
Yes
Final spline surface that meets fit criterion

Figure 4.2. A traditional solution to the problem of surface fitting to point cloud data. The first step is to create an initial parameterization of the data points. This is usually accomplished through manual techniques (see the text for details) and produces a crude initial parameterization of the data set. This parameterization is used in the surface fitting step (i.e. the linear least squares problem of equation 4.3) to obtain a surface approximation. If this approximation meets the fitting criterion (which is usually based on the residual error of fit) the fitting process terminates. If the approximation does not meet the fitting criterion the data set is re-parameterized based on the most current surface approximation. The iteration continues in this manner until the fitting criteria are met. In this iterative optimization process the initial parameterization of the data set is crucial to the quality of the final surface approximation. See the text for a detailed discussion of this process.

intermediate approximation. The dimensions of the matrix B in equation 4.3 are L by mn. The most expensive operation in solving the linear system involves "inverting" B [Lawson & Hanson 1974]. For a typical numerical solution (such as QR factorization [Golub & Loan 1993] (section 5.3.4)) the time complexity of this operation is:

$$2m^2n^2(L - \frac{mn}{3})$$

In practice the number of data points L is significantly more than the number of control vertices mn and we can rewrite the time complexity of the least squares problem as $O(m^2n^2L)$. Refining the parameter values is typically $O(mnL)$ (parameter values for each data point are refined using a search through the intermediate spline surface's parameter space). From a practical implementation point of view parameter refinement is an expensive calculation since it involves computing spline surface partial derivatives at each of the original data points [Hoschek 1988, Sarkar & Menq 1991].

A third noteworthy aspect of this iterative optimization procedure is that if the user wishes to fit a different resolution of spline surface (i.e. a different number of control vertices) to the data, at least some of the parameter refinement iterations must be performed again. This is because a B-spline surface approximation with a different resolution will induce a different parameterization on the data set.

The preceeding discussion clearly points to the fact that obtaining a good initial parameterization of the data points is central to the surface fitting problem. Techniques for parameterization in the CAD literature use manually intensive schemes such as projection of data points to manually chosen planes or a manually constructed base surface (e.g. a lofted surface) [Ma & Kruth 1995, Milroy et al. 1995, Sinha & Seneviratne 1993]. These techniques, even with extensive manual intervention are not guaranteed to produce good parameterizations. Furthermore, in our case the data set is dense, making manual intervention for the parameterization step impractical (i.e. laborious).

Note that by discarding the connectivity information in our polygon mesh we could trivially convert our approximation problem to that of a conventional point-cloud fitting problem and use the fitting approach described above. In such a case the vertices from the source polygonal mesh would form the point cloud. However, we have chosen not to follow this strategy since conventional point cloud fitting strategies can produce topologically incorrect results (e.g. see figure 4.1), are prone to non-robustness (stemming from convergence issues) and often produce poor quality surfaces due to iterations falling into local minima. They are also very inefficient (i.e. $O(Lm^2n^2)$ per iterative step) and therefore unusable in an interactive setting.

4.3.2 Fitting to gridded data

A second approximation problem related to ours is that of fitting to a gridded data set. Although our data set is not in this format, gridded data fitting techniques provide useful intuitions into our problem. Fitting tensor product surfaces to gridded data can be conveniently decomposed into a sequence of simpler curve fitting processes [Forsey & Bartels 1991]. As a result, gridded data fitting algorithms are simpler, more efficient and more robust than point cloud fitting algorithms. If the data set is given by the grid of points $\{\mathbf{P}(i,j)|i = 1 \dots M, j = 1 \dots N\}$ such that $\mathbf{P}(i,j)$ has parameter values (u_i, v_j), then the least squares equation for the gridding data fitting problem is written as follows:

$$\mathbf{P}_{grid} = \mathbf{B}_{m,M}(u)\mathbf{X}_{grid}\mathbf{B}^T_{n,N}(v) \tag{4.4}$$

where $\mathbf{P}_{grid}$ is the regular grid of data points:

$$\mathbf{P}_{grid} = \begin{bmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \dots & \mathbf{P}_{0,N} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & \dots & \mathbf{P}_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{P}_{M,0} & \mathbf{P}_{M,1} & \dots & \mathbf{P}_{M,N} \end{bmatrix}$$

$\mathbf{X}_{grid}$ are the control vertices that are to be solved for:

$$\mathbf{X}_{grid} = \begin{bmatrix} \mathbf{X}_{0,0} & \mathbf{X}_{0,1} & \dots & \mathbf{X}_{0,n} \\ \mathbf{X}_{1,0} & \mathbf{X}_{1,1} & \dots & \mathbf{X}_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{X}_{m,0} & \mathbf{X}_{m,1} & \dots & \mathbf{X}_{m,n} \end{bmatrix}$$

and

$$\mathbf{B}_{m,M}(u) = \begin{bmatrix} B_0(u_0) & B_1(u_0) & \dots & B_m(u_0) \\ B_0(u_1) & B_1(u_1) & \dots & B_m(u_1) \\ \vdots & \vdots & \vdots & \vdots \\ B_0(u_M) & B_1(u_M) & \dots & B_m(u_M) \end{bmatrix}$$

$\mathbf{B}_{n,N}$ is obtained by substituting n for m and N for M in the above matrix.

In equation 4.4 the dimensions of the matrices are as follows: $\mathbf{P}_{grid}$ is $M \times N$, $\mathbf{B}_{m,M}(u)$ is $m \times M$, $\mathbf{B}_{n,N}$ is $n \times N$ and $\mathbf{X}_{grid}$ has dimensions $m \times n$. In a surface approximation problem, the number of data points in each direction of the grid are usually far in excess of the number of control vertices in the same direction of the grid i.e. $M > m$ and $N > n$. For purposes of comparison we will assume that the number of data points in the point cloud formulation of the previous section is about the same as the number of data points in the regular grid formulation above i.e.

$$L \simeq MN$$

The parameterization of data points on a regular grid is trivial because a regular grid is intrinsically parameterized [Forsey & Bartels 1991]. Since the data set in equation 4.4 is already parameterized, solving the surface fitting problem for regular gridded data reduces to solving equation 4.4 once for the control vertices. Solving this equation involves “inverting” both $\mathbf{B}_{m,M}(u)$ and $\mathbf{B}^T_{n,N}(v)$. The computational costs of these operations are $O(Mm^2)$ and $O(Nn^2)$ respectively (assuming once again, that we use some standard numerical technique such as QR factorization [Golub & Loan 1993]). The cost of solving this system is therefore $O(Mm^2 + Nn^2)$. This is a significantly simpler and less expensive computation than the non linear iteration associated with equation 4.3. In that case the cost *per iteration step* was $O(MNm^2n^2)$.

It is worth pointing out that since regular gridded data is by definition uniformly distributed in parameter space, the matrices $\mathbf{B}_{m,M}(u)$ and $\mathbf{B}_{n,N}{}^T(v)$ are individually more well conditioned than $\mathbf{B}_{mn,L}(u,v)$ (from equation 4.3). From a purely numerical perspective, this means that gridded data fitting is usually numerically more robust than point cloud fitting. Further details of gridded data fitting are discussed in chapter 6.

To summarize, the solution to the surface approximation problem for regular gridded data is simpler, computationally less expensive and numerically more robust than the solution for surface fitting to point cloud data. The fundamental reason for these advantages is that since the data is ordered, the surface fitting problem is decomposable into a sequence of simpler curve fitting problems. A second major advantage of fitting to gridded data is that the results produced are of a higher quality than those produced by point cloud fitting. This is because the quality of an approximating surface is strongly influenced by the quality of the underlying parameterization of the data, and regular grids are trivial to parameterize compared to irregular point clouds

4.4 A new surface fitting formulation

Clearly, gridded data fitting techniques are to be preferred over point cloud fitting techniques. Since our data is not in the form of a regular grid, these methods are not immediately applicable. However, if we were to convert our data into the above format we could exploit the advantages of gridded data fitting over point cloud fitting. This motivates a new two-step solution strategy for our surface fitting problem:

1. First, resample each irregular polygonal patch into a regular surface grid, which we call the *spring mesh*.

2. Second, use gridded data fitting techniques on the spring mesh to obtain a surface fit.

Our two step fitting strategy allows us to avoid the complexity, cost and non-interactivity of the non-linear optimization process traditionally used for irregular data. See chapter 5 (section 5.7) for a detailed discussion of the relative benefits of our surface fitting approach over traditional strategies.

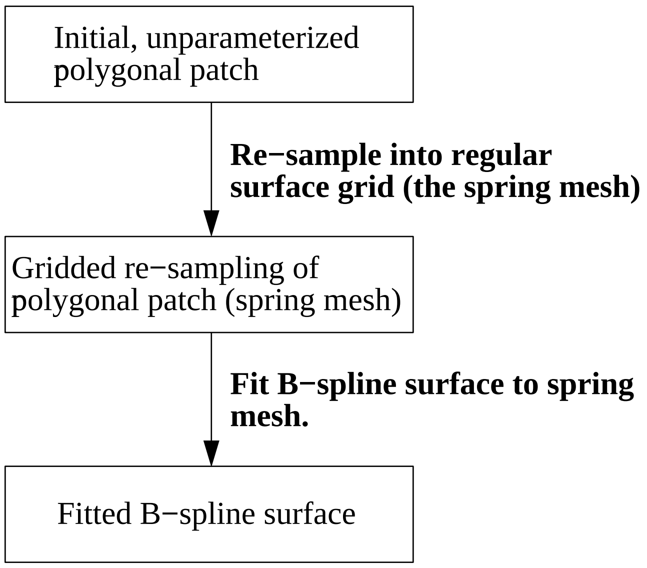
Initial, unparameterized polygonal patch
Re−sample into regular surface grid (the spring mesh)
Gridded re−sampling of polygonal patch (spring mesh)
Fit B−spline surface to spring mesh.
Fitted B−spline surface

Figure 4.3. Our two step surface approximation strategy. In the first step, we re-sample the polygonal patch into a regular surface grid called the spring mesh. This process is explained in chapter 5. In the second step we apply gridded data fitting techniques to the spring mesh to obtain our final B-spline surface approximation. This two step strategy allows us to avoid the complexity, cost and non-robustness of point cloud fitting algorithms. We have not shown the the computation of a displacement map here. Recall that our displacement maps are extracted simply as the residual error of fit after the B-spline approximation is obtained. See chapter 7 for a discussion of displacement map computation.

The spring mesh may be viewed as a high quality parameterization of the underlying data. In our application we use the spring mesh itself as the (gridded) data for the surface fitting step. As discussed in section 4.2.3 we are more interested in a high quality approximation to the mesh surface itself rather than the mesh vertices. As long as the spring mesh we produce is a reasonably careful sampling of the polygonal mesh, surface quality and detail are not compromised.

If in other applications it is required to fit the original mesh vertices (e.g. to obtain tolerances to the original data), this can be accomplished by first parameterizing the mesh vertices using our regular spring grid and then running the standard non-linear optimization process described in section 4.3.1. For these approaches, our gridding process offers a robust, automatic, high-quality mesh parameterization scheme. However, we do not pursue this alternative strategy in this thesis.

The next chapter explores in detail, our gridding strategy for each polygonal patch and its advantages in an interactive setting. We also discuss strategies for adding flexible constraints to the parameterization process in the context of an interactive surface fitting system.

Chapter 5

Gridded resampling

As outlined in the previous chapter, our fitting strategy works in two steps. First, each polygonal patch is re-sampled into a regular surface grid called the spring mesh. Second, a B-spline surface is fit to the spring mesh. In this chapter, we discuss the details of the first of these two steps i.e. the creation of the spring mesh.

The input to our gridding process is a polygonal patch. A polygonal patch for the purposes of our discussion is an arbitrary four-sided polygon mesh section. The only constraints to this mesh section are that it must be rectangularly parameterizable and must not have holes. Both these assumptions are reasonable: first, in order that a tensor product B-spline surface be fit to the surface it must be rectangularly parameterizable. Second, the models input to our system are seamless or can easily be made so either by acquiring and integrating more scans or by using recent hole-filling techniques [Curless & Levoy 1996]. There are no further constraints (with respect to either topology or geometry) on polygonal patches. In particular, they are not restricted to be height fields.

For the first part of this chapter, we will assume that the only user input is the polygonal patch itself (i.e. the four boundary curves of the patch). A procedure is described to automatically generate a spring mesh re-sampling for this polygonal patch. Given just the four boundary curves of a polygonal patch as input, there are several plausible distributions of spring points. We begin this chapter (section 5.1) by examining the properties that define a good spring mesh re-sampling. This leads to a qualitative measure for evaluating the desirability of a spring mesh re-sampling. In section 5.2 we quantify these characteristics in the form of a set of parameterization rules. In the following sections we outline an efficient implementation that generates spring meshes that obey our parameterization rules. This implementation is based on a coarse-to-fine relaxation process that incrementally builds up a better parameterization for the polygonal patch. In section 5.7 we discuss several desirable properties of our coarse-to-fine parameterization strategy. We follow this discussion with an analysis of the computational cost of our parameterization algorithms (section 5.8).

In the second half of this chapter we introduce the notion of *feature curves*. These are user-specified curves that may be used to arbitrarily (and precisely) customize the distribution of points within the spring mesh. We call the resulting parameterizations *feature-driven parameterizations*. We introduce feature curves by examining our parameterization algorithm from a design perspective (section 5.9). From this perspective feature curves are a natural extension to our basic parameterization strategy. In section 5.10 we discuss how our parameterization rules may be modified in the presence of feature curves. Then in section 5.11 we discuss an implementation of feature curves. Our feature driven parameterization technique bears several similarities to the variational surface design literature. In section 5.12 we discuss an intuition for our feature driven parameterization algorithm that is based on a variational surface design perspective [Welch & Witkin 1994]. We conclude this chapter (section 5.13) with a summary of its principal contributions and some practical examples of the uses of our parameterization algorithms.

5.1 Desirable characteristics for a spring mesh

The spring mesh is essentially a regular grid of points that lie on the polygonal surface. We will refer to these points as the “spring points”. The spring mesh can be viewed in two ways: first, it can be viewed as a parameterization of the polygonal patch. Second, since spring points lie on the polygon mesh surface, the spring mesh can be viewed as a *re-sampling* of the patch into a regular grid. We will draw on both these views of the spring mesh for our intuitions about the parameterization process. In the following discussion we use the terms parameterization and spring mesh re-sampling interchangeably.

Given a polygonal patch there are infinitely many distributions of spring points and therefore parameterizations of the patch. From the space of all these parameterizations some are preferable over others. Therefore, before we create a procedure to generate a parameterization, we must first answer the question: what does it mean for a parameterization (i.e. spring mesh) to be “preferable” over others? Clearly, since our spring meshes are used for smooth surface fitting, we should restrict our space of acceptable parameterizations to those that enable a high quality surface approximation. In section 5.1.1 we examine this issue in more detail.

Note that the above restriction does not yet narrow down our space of acceptable parameterizations: it is still possible to generate several different spring parameterizations each of which works just as well for approximating the geometry of the polygonal patch. However, not all of these parameterizations might be satisfactory to the user. Since users in our target application areas (i.e. Entertainment and Industrial Design) play an important role in determining the desirability of a surface parameterization, whatever parameterization procedure we propose should take care to satisfy the users requirements in addition to producing good approximations to surface geometry. Accordingly, in section 5.1.2 we examine our parameterization procedure from a user perspective.

5.1.1 A sampling perspective

Since the spring mesh is essentially a re-sampling of the polygonal patch we begin our effort to characterize the desirability of an arbitrary spring mesh from a sampling perspective. The spring mesh is generated with the intent of fitting a combination of B-spline surface and displacement map to it. In order that our fitted surface not lose any detail that was in the original polygonal patch, it is desirable that the spring mesh be as faithful a resampling of the polygonal patch as possible. Only if the spring mesh satisfies this requirement may we dispense with the irregular polygonal representation and use just the regular spring mesh for fitting purposes. Furthermore, we would like our spring mesh to be minimal in the sense that there should be no “wasted” spring points (samples). We therefore ask the question: What is the distribution and density of spring points that gives us a faithful sampling of the polygonal patch?

Consider a simpler version of the same question: for a given resolution of the spring mesh what is the distribution of spring points that represents a reasonable sampling of the surface? Since our triangulations come from real models that have been sampled uniformly over their surfaces, our triangle meshes tend to be uniformly dense (in terms of the number of polygons per unit area of the surface) across different parts of the polygonal model. If we assume that geometrical information is distributed according to polygon mesh density then for a spring mesh of a fixed resolution to sample the surface optimally each sample should represent an equal area on the polygonal surface. This gives us one qualitative measure of goodness for our spring mesh from a sampling perspective. Furthermore, in order to dispense with the polygonal patch, the density of the spring mesh should be high enough that the finest details of the polygonal patch are captured by the re-sampling. This gives us a second qualitative measure. Our parameterization metric can therefore be encapsulated in the form of two qualitative rules:

1. Each spring point represents a constant size area of the polygon mesh.
2. The density of the spring mesh should be high enough to capture the finest details of the polygonal patch.

Although these qualitative rules summarize our requirements concerning obtaining a good approximation to geometry, they do not necessarily capture a users notion of a desirable parameterization. This is easily illustrated with the help of an example. Figure 5.1 shows three possible "parameterizations" of a planar distorted patch. The parameterization in figure 5.1(a) might seems desirable at first since each spring mesh opening covers about the same area of the patch (barring the boundary regions). However, the spring mesh shown is not a regular grid i.e. every horizontal (vertical) curve of the spring mesh does not have the same number of sample points as every other horizontal (vertical) curve. This parameterization therfore does not suit our purposes despite the fact the sampling of the patch is somewhat even. Our space of acceptable parameterizations is restricted to *regular surface grids* because that is the parameterization intrinsic to a tensor product parametric basis. Furthermore, the entire spring mesh must be present within the polygonal patch i.e. the parameterization must be a regular, tensor product parameterization with its four boundaries being the same as the patch boundaries.

Consider now the parameterizations shown in figure 5.1(b) and (c). Both are regular surface grids. Note that both spring meshes have a singularity at the lower, collapsed

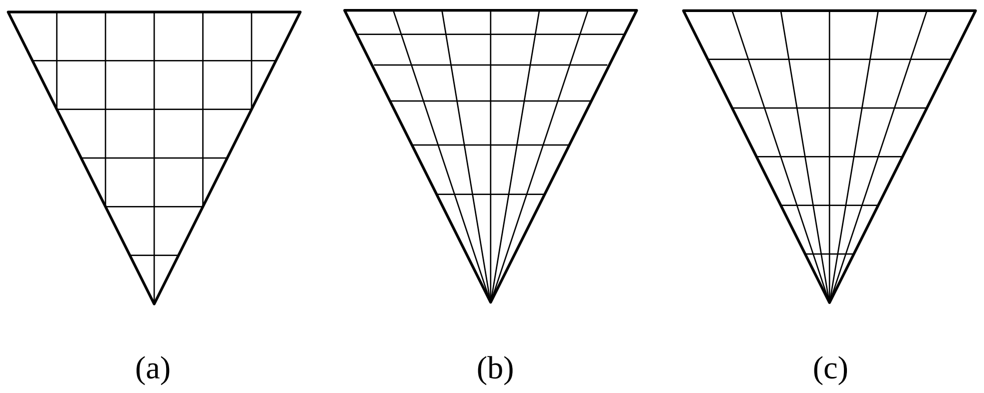

Figure 5.1. Three examples of possible spring meshes for a planar triangular patch. The spring mesh of (a) seems desirable at first since each interior spring point covers about the area of of the patch. However, the parameterization is not acceptable to us because it is not a tensor product parameterization i.e. each u (v) iso-curve does not contain the same number of points. (b) and (c) are both tensor product parameterizations with the same number of u and v iso-curves. Note that no tensor product parameterization of this patch can avoid a pole at the lower, collapsed boundary curve. The spring mesh of (b) minimizes the area distortion of the patch. However, note that the parameterization is distorted with respect to the arc length along a vertical iso-curve: the distance between horizontal iso-curves get compressed as one moves away from the pole. Also note the aspect ratio distortion: area elements closer to the pole are thin and long while area elements farther from the pole are broad and short. Finally (c) shows a parameterization that minimizes the arc length distortion and has a lower aspect ratio distortion than the spring mesh in (b). However note that it is distorted with respect to area. Our parameterization scheme chooses to create the more “uniform” spring mesh of (c). See the text for a discussion of our motivations for making this choice.

boundary curve (i.e. the pole of the spring meshes). The singularity at the pole is a limitation of the tensor product nature of the parameterization rather than a limitation of the specific distribution of spring mesh points i.e. no tensor product parameterization of this patch could avoid such a pole or the distortion associated with it. For example, the spring mesh of (b) minimizes area distortion. However, note the extreme distortion induced with respect to the arc length and aspect ratio. On the other hand figure 5.1(c) shows a parameterization that minimizes the arc length distortion in both parametric directions of the spring mesh. While it has some aspect ratio distortion, it is significantly lower than the distortion in spring mesh of (b). However note that it is more distorted with respect to area than (b). In particular the area elements near the pole are smaller than the ones farther from it.

Note that the spring meshes of both (b) and (c) allow us to create acceptable surface approximations as long as the density of the spring mesh in each case is high enough. So which of (b) or (c) is preferable for our application ? To resolve this question let us examine our parameterization problem from a design (i.e. users) perspective.

5.1.2 A parameterization design perspective

The application domains that are the principal focus of this thesis are entertainment and industrial design. For applications in these domains, the design of 3-D parametric surfaces has two key components: first, the design of the geometry of the surface, and second, the design of its parameterization. It is important to note that in general these are two separate activities even though commercial modeling packages such as Alias or Pro/Engineer use modeling paradigms (such as lofting or skinning [Farin 1990]) that treat them as a unified activity.

In our application the user is not designing so much the geometry of the fitted surface (which is ideally the same as that of the the input polygonal patch) as its parameterization. The design of a surface's parameterization is of critical importance to most modelers because it influences several important properties of the surface. For example, surface parameterizations determine the flow of a texture map over the surface. A surface that is not parameterized according to the users specifications is unlikely to appear satisfactory when texture mapped. Surface parameterizations also determine the precise changes in surface geometry when a control vertex of the surface is moved in space to a new location. An unsatisfactorily parameterized surface will usually deform in an unexpected (i.e. undesirable) fashion when a control vertex is moved. For these and other reasons having control over surface parameterization is crucial for most applications. To take a specific example from entertainment, the animation of a parametric surface representing a human face is better performed when the parameter lines of the surface flow according to anatomical features such as the eyes and the mouth of the face [Parke & Waters 1996]. Furthermore, animators often concentrate a greater number of parameter lines (hence control vertices) on and around regions of the surface that require fine geometric control i.e. areas of the face that are the most expressive such e.g. the eyes. Similar examples can be found in the field of Industrial Design.

The foregoing discussions illustrate that it is important to evaluate any parameterization algorithm from a design perspective. Let us therefore ask the question: What parameterization did the user implicitly *design* by specifying a polygonal patch through its four patch boundaries?

We tackle this question in two parts. First, let us assume that our only user input is a set of four patch boundary curves and that an underlying polygonal surface does not exist i.e. our input is just a set of four space curves. Based on just this input, a reasonable spring mesh output is a *smooth* blend in space of the four boundary curves. Furthermore, in the absence of other information it is reasonable to assume that the user intended the spring mesh to be *uniformly* spaced in both parametric directions. An example of the geometry and parameterization created by this kind of operation is a Coons patch [Farin 1990].

Second, let us now add the additional polygonal surface constraint (i.e. the intermediate iso-curves must now be constrained to the polygonal surface). In this case, as an obvious extension to the first argument, the user could reasonably expect to have designed *a spring mesh that represents a uniform and smooth blend on the polygonal surface of the four boundary curves*.

Thus, given that the input from a user of our system are the four boundary curves of a polygonal patch, a reasonable parameterization for the polygonal patch is a spring mesh that is *uniformly* spaced in both the u and v directions and that represents a *smooth* blend on the surface of the boundary curves. Therfore, from a design perspective the parameterizations shown in figure 5.1(c) is preferable to the one shown in (b) because it represents a more uniform blend on the surface of the boundary curves. The parameterization in (c) distributes iso-parametric curves uniformly on the surface between each pair of opposing boundary curves.

The preceeding discussion demonstrates that, while minimal area distortion is a reasonable goal to strive for in a parameterization, it is not always desirable to achieve this goal. We must therefore create practical rules that generate parameterizations which encode our intuitions based on both the sampling perspective and on the design principles explained above. In the next section we quantify our qualitative characterization of the preceeding discussion in the form of simple and implementable parameterization rules.

It is worth noting that the foregoing discussion indicated that the choice of how to trade off different kinds of distortions in parameterizations is often subjective (i.e. application specific). Therefore a natural question to ask is: what if the user did indeed want to create the parameterization shown in figure 5.1(b) ? Ideally our parameterization rules should be extensible enough to accommodate such scenarios. We explore such issues at greater depth in the second half of this chapter (sections 5.9 onwards).

5.2 Quantifying the parameterization metric

We would like practical rules that capture the qualitative parameterization metrics explained in the last section. Ideally, these rules should create parameterizations that have minimal area distortion (where permitted by patch topology) and as well that represent a uniform blend on the polygonal surface of the four patch boundary curves. In the discussion below, “iso-curves” refer to iso-parametric curves in one of the principal directions of the tensor product parameterization. An iso-curve is just a series of spring points that constitute a grid line of the spring mesh. We refer to the two principal directions of the spring mesh as the u and v directions. A u (v) iso-curve is an iso-parametric grid line with a fixed u (v) value and progressively increasing (or decreasing) v (u) parameter values.

We begin our quest for practical parameterization rules by observing that regardless of the shape of the polygonal patch, it is always possible to create a spring mesh that minimizes distortion in grid spacing along a particular u or v iso-curve. For example, if we follow a fixed iso-curve in the examples shown in figures 5.1, the spacing (on the surface) of the spring mesh points along that iso-curve is uniform. We also note that to create an even sampling in both the u and v directions, the spring mesh points in each direction should be about the same distance apart. These observations are encapsulated in the following two parameterization rules:

1. *Arc length uniformity*: the spring mesh spacing along a particular iso-curve should be uniform.

2. *Aspect ratio uniformity*: the spring mesh spacing along a u iso-curve should be the same as the spacing along a v iso-curve.

When evaluated from a sampling perspective, the arc-length criterion accounts for the fact that geometrically interesting detail is equally likely along any given section of an iso-curve along the surface. Similarly, the aspect-ratio criterion captures the fact that detail is equally likely in either direction of a gridding of the polygonal patch.

Unfortunately, neither of these rules encapsulate the notion of a "minimal" parameterization. Consider for example a small section of a spring mesh shown in figure 5.2. Note that the spring mesh section in figure 5.2a satisfies both the arc length and aspect ratio criteria specified above. However, the parameterization of the spring mesh section is still unsatisfactory: while it does encode our notion of a *uniform* blend on the surface of the four boundary curves, it is not a *smooth* blend. From a practical perspective, the parameterization shown would exhibit undesirable characteristics such as a wavy appearance under texture mapping. Similarly, pulling on a control vertex of the surface would induce an undesirable wavy deformation on the surface. Clearly the aspect ratio and arc length uniformity rules do not restrict the space of acceptable parameterizations strongly enough. Noting this fact, we add a regularization rule (a minimal energy rule) to the above two criteria.

3. *Parametric fairness*: Every u and v iso-curve should be of the minimum possible length given the first two criteria.

The term "fairness" has its roots in the geometric interpolation literature where it is used to convey the concept of minimum energy interpolating surfaces [Halstead et al. 1993]. Our

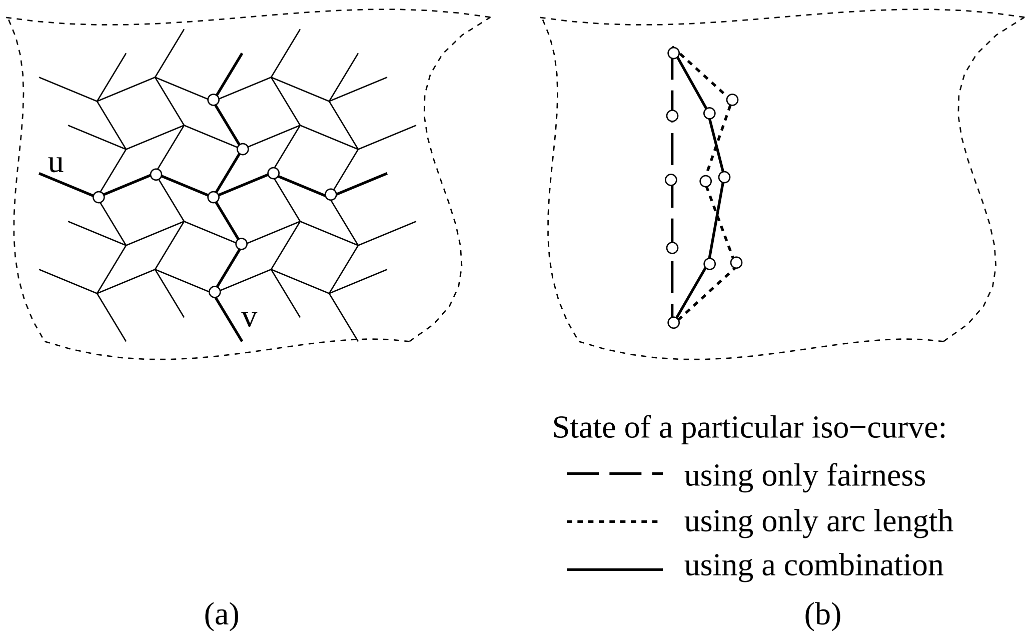
u
v
State of a particular iso−curve:
— — - using only fairness
········ using only arc length
——— using a combination
(a)
(b)

Figure 5.2. Parametric fairness as a regularization term. (a) shows a section of a larger spring mesh that satisfies just the arc length and aspect ratio uniformity criterion. Spring points are shown as hollow circles. Note that spacings along a particular u or a v iso-curve are uniform (arc length uniformity). Also note that the spacings along every u iso-curve are the same as the spacings along every v iso-curve. However, despite these facts, the parameterization is clearly unacceptable indicating that these two rules do not restrict the space of parameterizations sufficiently. (b) shows the interaction of the parametric fairness rule with the two other rules on a section of a single v iso-curve. The parametric fairness rule by itself creates minimal length curves. When used in conjunction with the other parameterization rules, this rule produces reasonable parameterizations.

use of the term fairness is in the context of achieving a minimal energy parameterization hence: parametric fairness. Previous applications have addressed the issue of fairness only to the extent that it affects surface shape, not as it affects surface parameterization.

The three qualitative parameterization rules listed above now give us minimum energy (i.e. parametrically fair) spring meshes that minimize arc length and aspect ratio distortions. It is worth noting that an implicit assumption we have made during the formulation of these rules is that surface detail in a region of the mesh is dictated solely by the number of polygon mesh vertices (or polygons) in that region. Since for our meshes the vertex density is fairly uniform (with respect to surface area) over the entire polygonal mesh it is desirable that our parameterization rules evenly distribute spring points on the mesh surface.

It is easy to see that at least from a sampling perspective our parameterization rules are conservative: they will in general require more samples than might be necessary for an optimal sampling of a polygonal patch. For example, a flat region of the surface will use up as many spring points as a highly curved region of the surface as long as they possess the same surface area (and hence the same number of mesh vertices). If our parameterization rules took into account properties such as surface curvature, the number of spring points could potentially be optimized so as to be concentrated on more complex regions of the surface. Despite this potential shortcoming, our strategy has two desirable characteristics. First, because it is conservative we are at least guaranteed to never miss any surface detail. Second, on a practical note, it is conceptually and numerically straightforward to perform computations on a uniform spring mesh representation compared to a more involved but optimal representation (such as a hierarchical spring mesh with a quad-tree like structure).

It is worth comparing our problem to those that arise in the finite element literature, specifically in the area of numerical grid generation [Thompson 1991]. The objective in this domain is to generate (either structured or unstructured) grids in $\Re^n$ as well as on surfaces in $\Re^n$. Specifically, extensive research has been done on various strategies for generating surface grids in $\Re^3$. However, these techniques are not directly applicable because they assume the existence of a higher order surface description (such as a B-spline surface) to start with. A higher order surface description allows the use of several smoothness properties such as global differentiability that are not available to us. As pointed out earlier, while it is possible to make local approximations to surface curvature for irregular polygonal surfaces [Welch & Witkin 1994], there is no scheme to evaluate global derivatives at arbitrary surface positions.

5.3 A fast gridding algorithm

In the last section we specified a set of parameterization rules to generate fair spring meshes with low aspect ratio and arc length distortion. In this section, we describe an algorithm that efficiently implements these parameterization rules. Our algorithm is an iterative, coarse-to-fine procedure that for each polygonal patch, incrementally builds a successively refined spring mesh. Specifically, the procedure iterates between two steps: a relaxation step and a subdivision step. The relaxation step works at a fixed resolution of the spring mesh. It ensures that the spring points at this fixed resolution are moved on the surface so that the resulting spring mesh satisfies our three parameterization rules. The subdivision step increases the resolution of the spring mesh by adding a new set of spring points to the existing parameterization. We continue this coarse-to-fine iteration until the spring mesh density is close to the density of vertices of the original polygonal patch.

We begin our coarse-to-fine iteration with an initial guess for the spring mesh iso-curves that is based on Dijkstra's shortest-path algorithm with an appropriate aspect ratio for this patch. We restrict this computation to just the vertices and faces of the polygonal patch rather than the entire mesh. Since the polygon patch is just a connected graph, identifying a set of vertices and faces of the polygonal patch that belong to a polygonal patch is easily performed using a breadth first seed fill operation. In the next few sections we describe the details of each step of our gridding algorithm.

5.4 Spring mesh initialization

To obtain a coarse initial guess for a set of u and v iso-curves, we use Dijkstra's single-source, single-destination, shortest-path algorithm [Aho & Ullman 1979] to compute a path between corresponding pairs of points along opposing boundary curves. The initial numbers of iso-curves in each direction are chosen to be proportional to the aspect ratio of the

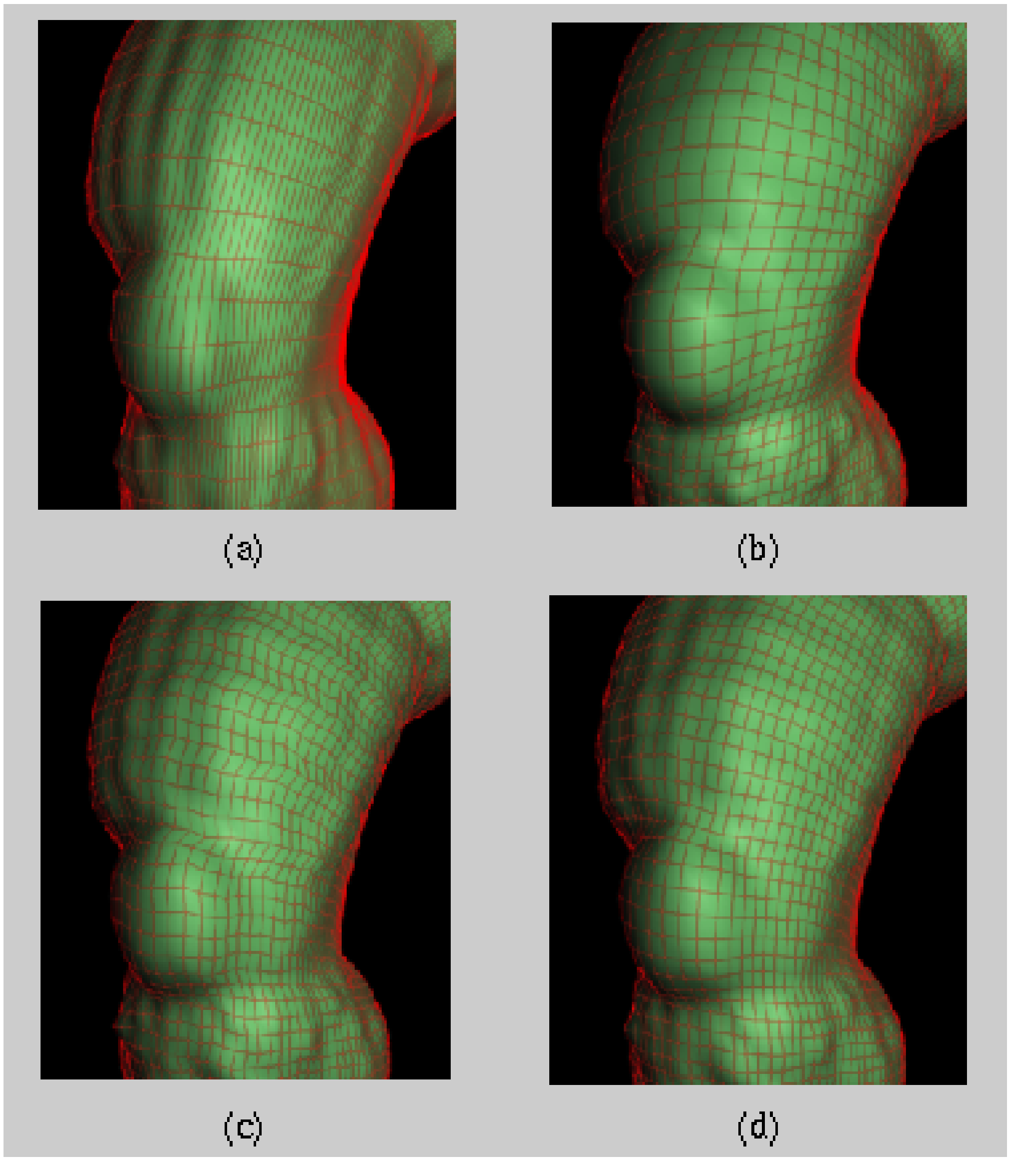

Figure 5.3. This figure explores the three sampling criteria on part of the right leg of the model in figure 1.1. Each of the above images represents a triangulated and smooth shaded spring mesh at a very low resolution. In each case, the number of spring points sampling the polygon mesh was kept the same. The differences arise from their redistribution over the surface. The spring edges are shown in red. (a) shows what happens when the aspect ratio criterion is left out. Notice how a lot of detail is captured in the vertical direction, but not in the horizontal. (b) shows the effect of leaving out the arc length criterion. Notice how the kneecap looks slightly bloated and that detail above and around the knee region is missed. This is because few samples were distributed over the knee resulting in a bad sampling of this region. (c) shows a missing fairness criterion. The iso-curves exhibit many “wiggles”. Finally (d) shows the result when all three criteria are met. See figure 5.6 for a summary of resampling process working on the leg patch.

patch. The aspect ratio of a polygonal patch is computed as the ratio of the longer of the two boundary curves in either direction. The starting spring mesh points are computed as intersections of these initial iso-curves; the curves must intersect if the patch is rectangularly parameterizable. Dijkstra's algorithm is $O(N \log(N))$ in the number of vertices of the polygonal patch. Since polygonal meshes are actually planar graphs they possess a simpler structure than arbitrary graphs. As such a linear-time algorithm exists to perform the shortest patch computation for planar graphs [Klein et al. 1994]. However, there have been no reports of practical implementations of these algorithms. We have found Dijkstra's algorithm to perform adequately for our purposes. Since we search for only a small set of initial iso-curves, this procedure is rapid.

5.5 Spring mesh relaxation

The initial guess for the spring mesh as obtained above is generally of poor quality. Furthermore, we have chosen to compute this initial guess at a low resolution. Despite these potential drawbacks, our initial spring mesh serves as a good starting point for our relaxation process.

In the relaxation step, we move the spring points (at a given resolution of the spring mesh) to new locations on the polygonal patch such that the resulting spring mesh satisfies our parameterization rules. Note that we have already implemented the rule of aspect ratio uniformity with our initialization step. As the spring mesh is uniformly subdivided this aspect ratio is maintained. Therefore, we do not explicitly enforce this criterion as part of our relaxation procedure. We enforce just the two remaining parameterization rules of arc length and parametric fairness.

The relaxation algorithm works in two steps. First a "resultant force" is computed on each spring point. This force is based on our parameterization rules of arc length and parametric fairness. In the second step this force is then used to slide the spring point to a new position on the polygonal surface (within the polygonal patch). All spring points are moved in this manner until the spring mesh attains its equilibrium state, i.e. when the resultant forces on all the spring mesh points are zero.

In the next few subsections, the "force" between two spring points is computed as the

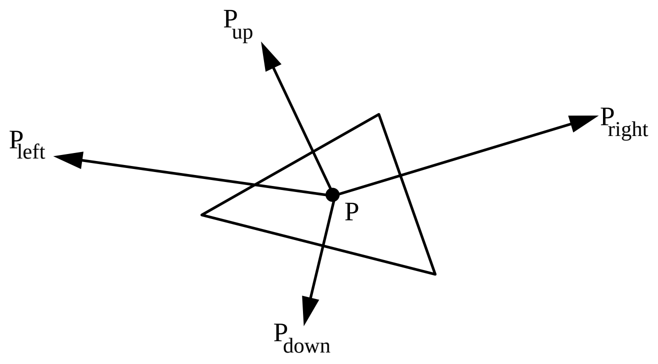

Figure 5.4. This figure shows the neighbors of a face point P of the spring mesh. The resultant force on the spring point during the relaxation step is a linear combination of these forces. See the text for details.

Euclidean vector joining the two points i.e.

$$\mathbf{Force}(P_a, P_b) = P_a - P_b$$

To understand the details of our relaxation procedure let us focus on the forces on just an individual spring point P. Let P_{up}, P_{down}, P_{left} and P_{right} represent the positions of the P's four neighbors in the spring mesh in the u and v directions. We first examine the arc length and fairness forces in turn and then discuss how these forces may be combined in a relaxation strategy.

5.5.1 The arc length criterion

Minimizing arc length distortion along one of P's iso-curves is achieved by simply moving P towards whichever neighbor (along the same iso-curve) is farther away from it. This computation is very similar to the computation for arc length based forces for snakes (see section 3.6.3). The two iso-curves of the spring mesh that pass through P may be thought of as individual snakes for force computations based on arc length.

As explained in section 3.6.3 the force due to the arc length corresponding to the up-down iso-curve (i.e. P_{up} and P_{down}) is:

$$F_{arc-ud}(P) = (|\mathbf{Force}(P_{up}, P)| - |\mathbf{Force}(P_{down}, P)|)\frac{\mathbf{t}_{max}}{\|\mathbf{t}_{max}\|} \tag{5.1}$$

where

$$\mathbf{t}_{max} = Max(\mathbf{Force}(P_{up}, P), \mathbf{Force}(P_{down}, P))$$

where $\mathbf{t}_{max}$ is the larger of $\mathbf{Force}(P_{up}, P)$ and $\mathbf{Force}(P_{down}, P)$ (in magnitude) i.e.

$$\mathbf{t}_{max} = \begin{cases} \mathbf{Force}(P_{up}, P) & \text{if } \mathbf{Force}(P_{up}, P) > \mathbf{Force}(P_{down}, P) \\ \mathbf{Force}(P_{down}, P) & \text{if } \mathbf{Force}(P_{down}, P) > \mathbf{Force}(P_{up}, P) \end{cases}$$

If the magnitudes of the forces happen to be equal the arc length force is zero. The two expressions **Force(**P_{up}, P) and **Force(**P_{down}, P) represent forces on the spring point P due to each of the spring points P_{up} and P_{down}. We perform a similar computation in the other direction (left-right) as well to get a force F_{arc-lr}. The resultant of F_{arc-lr} and F_{arc-ud} is the net arc length based force acting on the spring point P i.e.

$$F_{arc}(P) = F_{arc-lr}(P) + F_{arc-ud}(P)$$

5.5.2 The parametric fairness criterion

Let us now examine the force on P due to the parametric fairness criterion. Recall that this criterion attempts to make every iso-curve of the minimum possible length i.e. it tries to pull each iso-curve as "tight" as possible given the other constraints on the spring mesh. As with the arc length criterion, we can draw on our surface snake formulation for intuitions about this force term. Recall that the first order term (i.e. the membrane term) in the surface snake formulation minimizes the length of the snake. Writing out just the first order term we have:

$$E_{internal}(\mathbf{v}(s)) = \int_0^1 |\mathbf{v}_s|^2 \, ds \tag{5.2}$$

The various terms are used to mean the same quantities they did in section 3.6.1. In our current context, we interpret $\mathbf{v}(s)$ to be an iso-curve of the spring mesh and $E_{internal}$ to be the internal energy of the iso-curve. As explained in section 3.6.1, the discretized variational formulation of this equation gives rise to a single resultant force that moves a point to the mid-point of its two neighbors along the curve. Computing this for an "up-down" iso-curve we obtain the following:

$$F_{fair-ud}(P) = \mathbf{Force}(P_{up}, P) + \mathbf{Force}(P_{down}, P) \tag{5.3}$$

We obtain a similar expression for the other iso-curve passing through P. The resultant force due to parametric fairness is therefore:

$$F_{fair}(P) = F_{fair-lr}(P) + F_{fair-ud}(P)$$

We note that the parametric distortion is therefore minimized by moving the point P to a position that minimizes the energy corresponding to a set of springs consisting of P and P's immediate neighbors along both iso-curves that pass through it i.e.

$$F_{fair} = \mathbf{Force}(P_{up}, P) + \mathbf{Force}(P_{down}, P) + \mathbf{Force}(P_{left}, P) + \mathbf{Force}(P_{right}, P)$$

In this sense our regular surface grid behaves like network of springs that are constrained to the polygonal surface. Our use of the phrase "spring mesh" to describe our surface grid was motivated in part by the behavior of our surface grid under the effects of the parametric fairness criterion.

5.5.3 Relaxation using the arc length and fairness criterion

The last two sections explained how to compute the force on an individual spring point P based on our two parameterization rules of arc length and parametric fairness. The point P is now moved according to a force given by a weighted sum of the arc length force $F_{arc}(P)$ and parametric fairness force $F_{fair}(P)$.

$$F_{result}(P) = \alpha * F_{fair}(P) + \beta * F_{arc}(P) \tag{5.4}$$

The relative weights of these terms are assigned as follows. We start each iteration with a higher weight for the arc length force (i.e. $\alpha = 0$ and $\beta = 1$). As the iteration progresses we gradually reduce the weight of the arc length term and increase the effect of the fairness term (i.e. $\alpha = 0.5$, $\beta = 0.5$). The parametric fairness force is meant mainly as a regularizing force. Its purpose is to create a minimal parameterization that obeys the other rules. As such the spring mesh relaxation should be guided mainly by the arc length parameterization rule. Our weighting of the relative effects of the arc length and fairness forces is based on this intuition. In practice, this strategy has proved to produce satisfactory results.

Note that we have used only Euclidean forces in our relaxation step, i.e. forces that represent the vector joining the two spring points under consideration. A relaxation step based purely on Euclidean forces is efficient but not guaranteed to always generate robust results. Figure 5.5a shows an example where Euclidean forces alone fail to produce the desired effect.

An alternative to using Euclidean forces is to compute *geodesic forces*. These are simply forces that are computed along the surface of the mesh. Geodesic forces would produce the robust and correct motion for the spring points in the above case. Therefore, one approach to solving the problem exemplified by figure 5.5a, would be to use geodesic forces, or approximations thereof, in the relaxation step (instead of Euclidean forces). However as discussed in chapter 3 computing discrete polygonal geodesics is an expensive proposition since the fastest algorithm for point to point geodesics is $O(n^2)$ in the size of a polygonal patch [Chen & Han 1990]. Even approximations to geodesics such as those that are based on local graph searches are $O(n)$ and would be too expensive to perform at every relaxation step (for every force computation).

Spring mesh subdivision alleviates the problem motivated by figure 5.5b: we create a new spring point $P_{mid-point}$ that lies on the surface halfway between $P1$ and $P2$. This new point generates new Euclidean forces acting on the original points, moving them towards each other on the surface.

5.6 Subdividing the spring mesh

Spring mesh subdivision is based on a graph search and refinement algorithm. Our subdivision algorithm is based on a procedure that, given any two spring points $P1$ and $P2$ computes a new face point P_{mid} at the mid-point on the surface of the two spring points. We call this procedure ***FindMidPoint***($P1$, $P2$, P_{mid}). The details of the implementation of

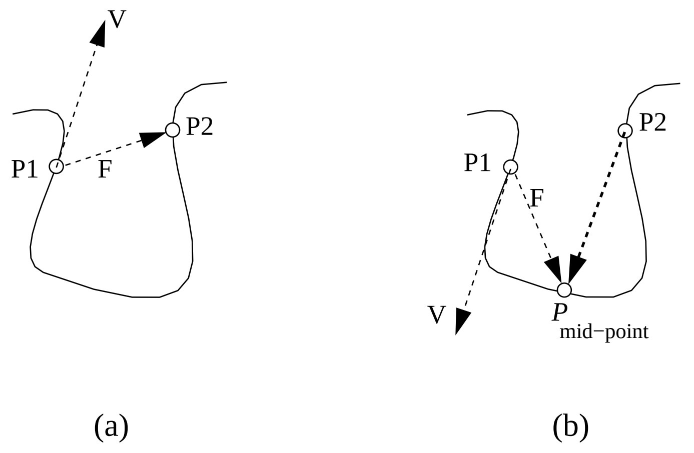

Figure 5.5. This figure shows a case where relaxation alone fails to move a spring mesh point in the desired direction. In each case F represents the force on $P1$ from its right neighbor and V represents the resulting direction of motion. The desired motion of the point $P1$ is into the cavity. In (a) just the opposite occurs; the points move apart. (b) shows how this case is handled by subdividing the spring mesh along the surface. See the text for details.

this procedure are as follows:

1. Find the two closest vertices v1 and v2 on $P1$ and $P2$'s faces.

2. Compute a greedy graph path from v1 to v2. The mid-point of this path serves as a first approximation to P_{mid}'s location.

3. Refine this location by letting the forces given by *Force*($P1$, P_{mid}) and *Force*($P2$, P_{mid}) act on P_{mid}, moving it to a new position on the surface.

Note that the last step in the subdivision process is simply the application of a fairness force locally at the point P_{mid} to make the curve section formed by $P1$, $P2$ and P_{mid} as tight as possible. Subdivision along boundary curves is based on a static resampling of our face point curve representation; these points are never moved during the relaxation and subdivision steps.

Based on the above algorithm, the subdivision algorithm uniformly subdivides the spring mesh in both parametric directions. Note that since the aspect ratio criterion was enforced at the time the spring mesh was created, it is maintained through successive subdivisions of the spring mesh.

Our coarse-to-fine strategy thus works by iterating between a relaxation step and a subdivision step. An issue still to be addressed is the termination criterion for this iterative process. As explained earlier, we would like our spring mesh at the highest resolution to represent a faithful re-sampling of the polygonal patch. For this to occur our spring mesh must capture all the fine surface detail in the original polygonal patch. Recall that, a fundamental assumption of our re-sampling strategy was that this surface detail is uniformly distributed (or at least equally likely) over the surface of the polygonal patch. Since our spring mesh uniformly samples the polygonal surface, a conservative strategy for capturing all the surface detail present in the original patch is to use at least as many spring points as mesh vertices. Thus we terminate spring mesh subdivision when the number of spring points is comparable to the number of vertices in the original polygonal patch. As noted in section 5.2 this termination criterion is a conservative one. A more aggressive strategy for terminating spring mesh subdivision might be to quantitatively measure the cumulative error between a reconstruction of the spring points and the underlying polygonal surface. In practice we have found our conservative termination criterion to work well in an interactive setting.

5.7 Discussion

The spring meshes generated by the algorithm described in the previous section are minimally distorted with respect to aspect ratio and arc lengths. In addition the parametric fairness criterion ensures that the spring meshes represent minimal energy parameterizations (in the sense defined earlier) given the other constraints. In practice we have found that our resulting spring meshes have low area distortion as well. As explained earlier, our subdivision and relaxation criteria enforce this at least partially. An example of this property is evidenced by a parameterized section of the Armadillo's leg shown in figure 5.3. Note that we do not *guarantee* minimal area distortion for our parameterizations. Indeed, as discussed in section 5.1 it is not always desirable to produce a minimal area distortion parameterization for arbitrarily shaped polygonal patches.

Our conservative termination criterion for spring mesh subdivision ensures that our spring meshes are dense enough that they capture the finest surface details of the polygonal

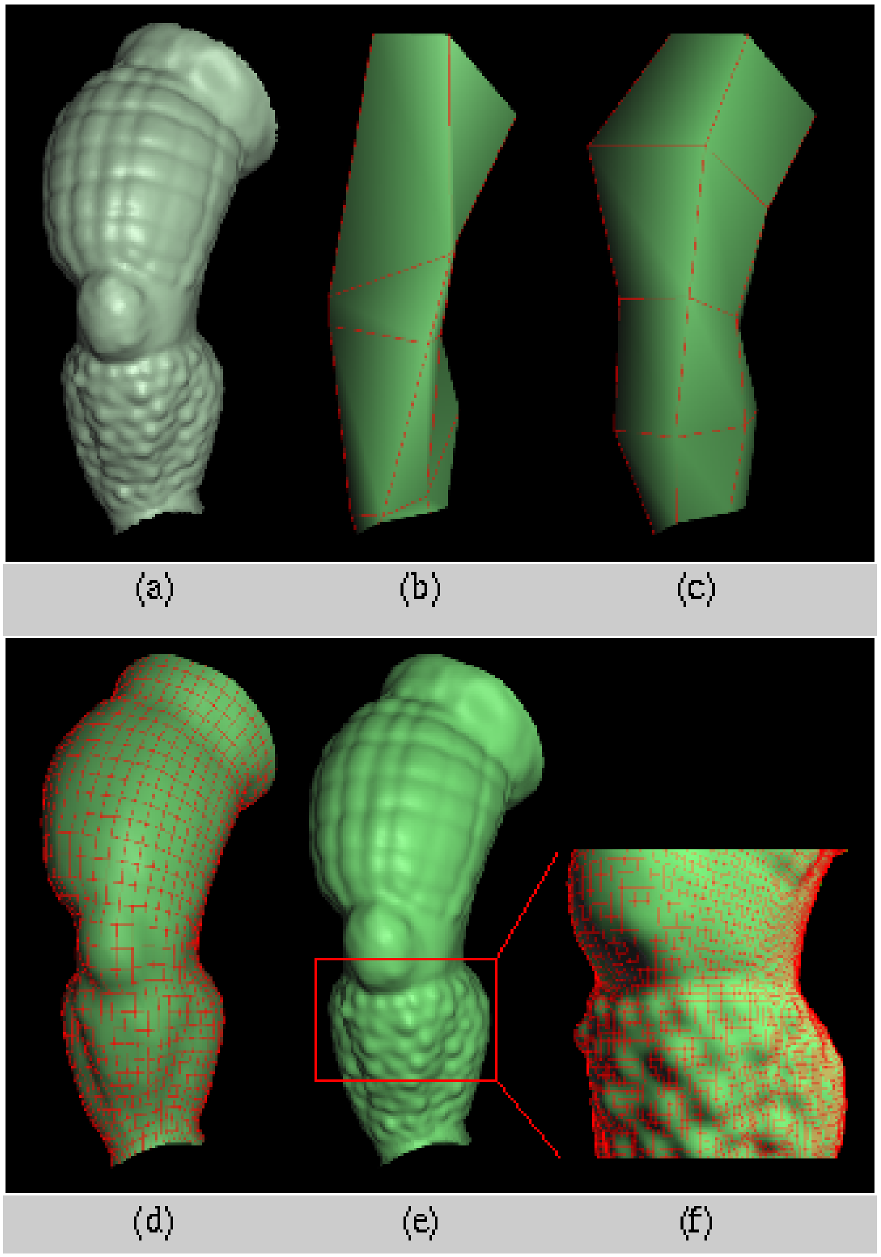

Figure 5.6. The figure summarizes with an example our strategy for resampling a polygonal patch into a regular grid. (a) shows the original polygonal patch (the right leg from the Armadillo model in figure 1.1). This particular patch is cylindrical and has about 25000 vertices. (b), (c), (d) and (e)show a triangulated and smooth shaded reconstruction of the spring mesh at various stages of our re-sampling algorithm. We omit the lines from (e) to prevent clutter. A detail of the spring mesh in (e) is shown in (f). (b) shows the initial guess for u and v iso-curves (under 4 seconds). Notice that the guess is of a poor quality. (c) shows the mesh after the first relaxation step (under 1 second). (d) shows the spring mesh at an intermediate stage, after a few relaxation and subdivision steps (under 3 seconds). (e) shows the final spring mesh without the spring iso-curves. Notice how the fine detail on the leg was accurately captured by the re-sampled grid. This entire resampling process took about 20 seconds. All times are on a 250 Mhz Mips R4400 processor.

patches. This, combined with the minimal distortion characteristics of our parameterization effectively satisfy the two qualitative criteria we proposed in section 5.1.1 i.e. our parameterizations are both minimally distorted as well as represent a faithful re-sampling of the polygonal patch. It is in this sense that our spring meshes also represent a high quality parameterization of the underlying polygonal patches.

Our coarse-to-fine gridding algorithm has several desirable properties that make it practical and useful in the context of an interactive surface fitting system. First, the coarse-to-fine nature of our algorithm makes it efficient. The average cost of the relaxation and subdivision steps of our algorithm are $O(N)$ in the size of the polygonal patch. Earlier polygon mesh parameterization algorithms (e.g. the algorithm of Eck et al [Eck & Hoppe 1996]) run an order of magnitude slower ($O(N^2)$) than the one we have described above. The details of the complexity analysis for our relaxation and subdivision algorithms are supplied in section 5.8.

A second advantage of our parameterization strategy is that although our algorithm is completely automated, it can be effectively controlled and directed by the user. For example, the user can pause the algorithm at a coarser resolution of the spring mesh to view the intermediate results. If desired, these coarser spring meshes can be used immediately for surface fitting purposes. This is a useful property of our system, especially when dealing with large polygonal meshes. In particular, subdivision to higher levels can be postponed until the model designer is satisfied with a patch configuration and parameterization.

A third advantage of our coarse-to-fine parameterization algorithm is that it offers a flexible framework within which to add further constraints to the parameterization. For example, in section 5.10 we discuss a powerful set of constraints called *feature curves* that the user can employ to create arbitrarily complex parameterizations.

Finally, our parameterization strategy has useful implications for the later surface fitting process (i.e. the gridded data fitting process). Specifically, once a spring mesh has been generated, the gridded data fitting process is fast (see chapter 4 for details). Furthermore, since our spring meshes are generated only once per patch (i.e. no further patch re-parameterization is required), the user can change the resolution of the fitted spline surface and the system responds at interactive rates with the fitted results. Systems that iterate between fitting and re-parameterization typically cannot respond at these rates and are therefore unsuitable for use in an interactive setting. Overall, we have found our pipeline to be a useful interactive tool for a modeler, especially when he or she wishes to explore the design space of tradeoffs between explicitly represented geometry and displacement mapped detail. We discuss further details of this aspect of our surface fitting pipeline in the next two chapters.

As discussed in chapter 2 there are other schemes that may be used to parameterize irregular polygon meshes. In particular, the harmonic maps of Eck et al[Eck et al. 1995] produce parameterizations for mesh vertices with low edge and aspect ratio distortions. However, the scheme has two main drawbacks for our purposes. First, it can cause excessive area distortions in parameter space, especially in the interior of the patch. Second, the algorithm employs an $O(N^2)$ iteration (in the size of a polygonal patch) to generate final parameter values for vertices of the mesh and no usable intermediate parameterizations are produced. As pointed out in the discussion above, we have found intermediate parameterizations useful in an interactive system. Finally, our algorithm allows for a powerful set of user defined constraints in the form of feature curves which can be used to create arbitrarily complex and customizable parameterizations. The parameterization technique of Eck et al does not offer the ability to specify such constraints.

5.8 Algorithm complexity

In the following discussion, we assume that the size of the polygonal patch is N (i.e. the number of vertices in the patch). The initialization cost for our coarse-to-fine strategy as explained earlier is $O(N \log N)$. We initialize the spring mesh at a low (constant) resolution. Our subsequent coarse-to-fine, iterative parameterization strategy has two distinct steps: spring mesh relaxation and subdivision. We consider the cumulative computational cost of each of these steps in turn.

5.8.1 Relaxation cost

A reasonable measure of computational cost for the relaxation step is the number of iterative steps required to reach the final state of the spring mesh. Unfortunately, this is not an easily quantifiable measure since the path followed by a spring point in reaching its minimum energy position is highly dependant on the geometry of the polygonal patch as well as the initial position of the spring point. To overcome this difficulty we use as a measure of cost the cumulative distance (on the surface) moved by all the spring points. At each level of subdivision, each spring mesh point must traverse some fraction of the polygons in a polygonal patch as it relaxes. Since our initial guess tends to be of a poor quality, in the worst case a spring point might traverse a significant fraction of the polygonal patch. Therefore in the worst case, the cost of this relaxation depends linearly on the size of the polygonal patch N. It obviously also depends on the size of the spring mesh. If these two were equal, as would occur if we immediately subdivided the spring mesh to the finest level, then the cost of running the relaxation would be $O(N^2)$.

If, however, we employ the coarse-to-fine strategy described in the foregoing sections, then at each subdivision level, four times as many spring mesh points move as on the previous (coarser) level, but they move on average only a fraction as far. Let us assume that this distance is a fraction $1/F$ of the distance moved at the coarser level. Let the distance moved by the spring points at the coarsest resolution of the spring mesh be D. If this resolution of the spring mesh has K spring points then the cumulative cost of the relaxation step over all subdivision levels is given by the expression:

$$C_{relax} = KD + 4K\frac{D}{F} + 4^2K\frac{D}{F^2} + \ldots + 4^MK\frac{D}{F^M} \tag{5.5}$$

where the number M is chosen according to our termination criterion. The criterion states that the final number of spring points 4^MK should be comparable to the number of mesh vertices in the polygonal patch i.e. N. Therefore,

$$M = \log_4\frac{N}{K}$$

For convenience let us rewrite equation 5.5 in the form:

$$C_{relax} = KD\mathbf{S}(F) \tag{5.6}$$

where

$$\mathbf{S}(F) = 1 + \frac{4}{F} + (\frac{4}{F})^2 + \ldots + (\frac{4}{F})^M \tag{5.7}$$

Let us now evaluate the cumulative cost of the relaxation step based on the values taken by F. Since $1/F$ represents the fraction of the distance moved by spring points at successively finer resolutions of the spring mesh it measures the relative gains of using a coarse to fine scheme.

In the worst case spring points at finer resolutions move as far as the points at coarser resolutions. In this case $F = 1$ and $\mathbf{S}(F)$ therefore evaluates to:

$$\begin{aligned} \mathbf{S}(1) &= 1 + 4 + 4^2 + \ldots + 4^M \\ &= \frac{4^{M+1} - 1}{4 - 1} \\ &= \frac{\frac{4N}{K} - 1}{3} \\ &= O(N) \end{aligned} \tag{5.8}$$

Therefore in the worst case:

$$\begin{aligned} C_{relax}(F = 1) &= O(DN) \\ &= O(N^2) \end{aligned}$$

since the average distance D moved at the coarsest resolution is $O(N)$. This means that in the worst case the coarse-to-fine relaxation scheme is computationally no better (at least from the relaxation perspective) than starting with the highest resolution of spring mesh and relaxing the spring mesh at this resolution. Of course, in this case the initialization cost for creating the highest resolution spring mesh would be much higher ($O(N \log N)$ per spring mesh point, which is $O(N^2 \log N)$).

However, F is usually much higher i.e. the finer resolution spring mesh points tend to move far less than the coarser resolution spring points. For example if the finer resolution spring points move about half as far as the coarser resolution spring points then $F = 2$ and:

$$\mathbf{S}(2) = 1 + 2 + 2^2 + \ldots + 2^M$$

$$\begin{aligned} &= \frac{2^{M+1}-1}{2-1} \\ &= \sqrt{\frac{4N}{K}} - 1 \\ &= O(\sqrt{N}) \end{aligned} \tag{5.9}$$

and therefore

$$C_{relax}(F=2) = O(N\sqrt{N})$$

Similarly if the coarser resolution spring points move about a fourth as far as finer resolution spring points then $F = 4$ and:

$$\begin{aligned} \mathbf{S}(4) &= 1 + 1 + 1^2 + \ldots + 1^M \\ &= M + 1 \\ &= O(\log N) \end{aligned} \tag{5.10}$$

and

$$C_{relax}(F=4) = O(N \log N)$$

In fact typically, spring points at finer resolutions tend to move by far less than even a fourth as far as the coarser spring points. This is because the finer spring points are created as the mid points of the coarser spring points. Therefore, unless positions of the coarser spring points in the finest spring mesh deviate drastically from their positions at coarser resolutions, the newer spring points tend to be created at or near their final positions. To take an extreme case consider a completely flat and square polygonal patch. In this case, the finer resolution spring points will not have to move at all since the subdivision step ensures that they will be created at their final destinations (and the relaxation at lowest resolution moves the coarse spring points to their final positions). In such cases $F > 4$ and therefore:

$$\begin{aligned} \mathbf{S}(F > 4) &= 1 + d + d^2 + \ldots + d^M; \;\; d < 1 \\ &= O(1) \end{aligned} \tag{5.11}$$

and

$$C_{relax}(F > 4) = O(N)$$

Thus the typical cumulative cost of the spring mesh relaxation step is usually $O(N)$ in the size N of the polygonal patch. Note that in the above computation we have assumed that at the coarsest resolution spring mesh points move a distance $O(N)$. This was a worst case assumption because our initial guesses tend to be of a poor quality to begin with. A better quality initial spring mesh would further reduce our average time complexity.

5.8.2 Subdivision cost

The cumulative cost of subdividing a given resolution of the spring mesh is equal to the cost of subdividing each of the iso-curves of the coarse spring mesh and creating the additional spring points at the middle of each pair of adjacent iso-curves in the two parametric directions. Let the total number of spring points at a given resolution R of the spring mesh be K_R i.e.

$$K_R = K_0 4^R = K4^R$$

Let us first assume that our polygonal patch does not have an extremely distorted aspect ratio. Let us consider the implications of this assumption. If the number of iso-curves in the u and v directions (at a particular spring mesh resolution) are n_u and n_v respectively then:

$$P = n_u n_v$$

Under our assumption:

$$n_u \simeq n_v \simeq \sqrt{K_R}$$

i.e. the number of iso-curves in the u and v directions are each $O(\sqrt{K_R})$.

Recall that our subdivision strategy is based on a greedy graph search algorithm between each pair of adjacent spring points. Therefore, the cost of subdividing an iso-curve is proportional to the cumulative number of polygons in a greedy graph path through each of the spring points in the iso-curve. Since our patch does not have a distorted aspect ratio this graph path is on the average $O(\sqrt{N})$. It is worth noting here that the relaxation process assists in simplifying the subdivision: relaxation prior to subdivision ensures that the iso-curves we subdivide are already near their final locations.

The total cost of subdivision at the resolution R of the spring mesh is therefore:

$$O(\sqrt{N}\sqrt{K_R}) = O(\sqrt{N}\sqrt{K_0 4^R})$$

The cumulative cost of subdividing the spring mesh (temporarily ignoring the O notation) to its highest resolution is therefore given by the expression:

$$\begin{aligned}
C_{subdivide} &= \sqrt{N}\sqrt{K_0 4^0} + \sqrt{N}\sqrt{K_0 4^1} + \sqrt{N}\sqrt{K_0 4^2} + \ldots + \sqrt{N}\sqrt{K_0 4^M} \\
&= \sqrt{NK_0}(1 + 2^1 + 2^2 + \ldots + 2^M) \\
&= \sqrt{NK_0}(2^{M+1} - 1) \\
&= \sqrt{NK_0}(\sqrt{\frac{4N}{K_0}} - 1) \\
&= O(\sqrt{N}\sqrt{N}) \\
&= O(N) \qquad (5.12)
\end{aligned}$$

Therefore we see that if the polygonal patch does not have an extremely distorted aspect ratio the cumulative cost of spring mesh subdivision is $O(N)$.

Let us now consider those cases when our aspect ratio assumption does not hold. An example of a case where this can occur is when the polygonal patch is excessively long and thin. We must make the following two changes to the above computation. First, the total number of iso-curves could be proportional to the number of spring points K_R i.e. either $n_u \simeq K_R$ or $n_v \simeq K_R$. Second, the cost of subdividing an iso-curve could be linear in the size of the patch i.e. if the patch is long and thin, subdividing iso-curves in the "longer direction" could be $O(N)$.

Therefore, the subdivision cost at a resolution R of the spring mesh is:

$$O(NK_R) = O(NK_0 4^R)$$

The computation of the cumulative cost of subdividing the spring mesh (ignoring the O

notation) now changes to:

$$\begin{aligned} C_{subdivide} &= NK_0 4^0 + NK_0 4^1 + NK_0 4^2 + \ldots + NK_0 4^M \\ &= NK_0(1 + 4^1 + 4^2 + \ldots + 4^M) \\ &= O(N^2) \end{aligned} \quad (5.13)$$

So if the polygonal patch has a pathologically distorted aspect ratio the cumulative cost of spring mesh subdivision is $O(N^2)$.

5.8.3 Total cost

To summarize the discussion of the last two sections, the typical cost of our coarse to fine parameterization strategy in terms of the size N of the polygonal patch is as follows:

1. Initialization: $O(N \log N)$.
2. Relaxation: $O(N)$ cumulative cost.
3. Subdivision: $O(N)$ cumulative cost.

Note that the initialization step of our process is more expensive than both the relaxation and subdivision together. This is directly related to our use of Dijkstra's algorithm to compute the coarse, initial spring mesh. As noted earlier this problem could be alleviated in theory, by using a recently proposed linear time algorithms to perform shortest path computations on planar graphs [Klein et al. 1994]. Using this algorithm would bring our average parameterization cost to $O(N)$. However, we have found no reports of practical implementation of this algorithm. In practice Dijkstra's algorithm performs adequately for our purposes.

In the worst case the last two steps are both $O(N^2)$. The conditions under which this occurs are:

1. Relaxation: spring points at finer levels of the spring mesh move just as far as the spring points at coarser levels i.e. we derive little or no benefit from relaxation at coarser levels.

2. Subdivision: the polygonal patch has an excessively distorted aspect ratio.

In practice, both these conditions do not occur except in pathological cases. A simplistic alternative to our coarse-to-fine strategy could be to immediately subdivide the spring mesh to the finest level and relax the spring mesh at this level. In this case, since the individual spring points must now all move through a distance $O(N)$ as well as the spring mesh resolution at the finest level is $O(N)$, the cost of running the relaxation is typically $O(N^2)$. Thus from a computational point of view the coarse-to-fine strategy is preferable to this strategy. As noted in section 5.5.3 the coarse-to-fine strategy is more robust than a single step strategy.

5.9 A parameterization design perspective

Given that the only input supplied by the user consists of a set of patch boundaries (i.e. a polygonal patch) our automated coarse-to-fine relaxation algorithm generates an acceptable (i.e. a minimally distorted as well as fair) parameterization. A reasonable question to ask of any such automated parameterization scheme is: what if the automatically generated parameterization is not acceptable to the user?

To explore the possibilities raised by this question, let us re-examine our parameterization algorithm from a design perspective. In section 5.1.2 we posed the question: what parameterization did the user implicitly *design* by specifying a polygonal patch through its four patch boundaries. We argued there that a user can reasonably expect to have designed *a spring mesh that represents a smooth and uniform blend on the polygonal surface of the four boundary curves*. Note that this is the result that our automatic parameterization procedure generates: the iso-curves are uniformly spaced in both the u and v directions (based on the arc length constraint) and represent a smooth blend on the surface of the boundary curves (based on the parametric fairness constraint). In addition, each surface element (spring mesh opening) is as close to square as possible (based on the aspect ratio constraint).

With this design perspective let us re-phrase the question we asked at the beginning of this section: what if our (automatically generated) blend on the polygonal surface of the four boundary curves does not create an acceptable parameterization of the polygonal patch? As explained in section 5.1.2 the design of the parameterization of a surface cannot, in general, be automated. Clearly this situation calls for the user to specify additional input in the form of constraints to the parameterization procedure. There are several possible kinds of constraints a user could specify. We discuss one kind in the next section: feature curves. Our ability to add constraints of the sort is indicative of the generality and extensibility of our parameterization strategy.

5.10 Feature curves

We define a *feature curve* as an iso-parametric curve painted on the polygonal patch, i.e. it is a face-point curve that starts at any one boundary curve and ends at the opposite boundary curve. Exactly one iso-curve of the spring mesh is constrained to follow this feature curve. This means that the spring points that correspond to the selected iso-curve are constrained to lie on the feature curve. During the parameterization process, the entire spring mesh is coerced into smoothly interpolating the feature curve constraint. In general, a user can specify arbitrarily many feature curves all of which are used as constraints to the parameterization procedure. Feature curves are a form of scalable constraint: in the limit, the user could in theory manually specify every single iso-curve of the spring mesh to arbitrarily customize the patch parameterization. Naturally, this would be an extremely laborious and therefore undesirable procedure and is not an expected use of the feature curve paradigm.

Our strategy of using feature curves for parameterization design has not been attempted previously in the surface fitting domain. Previous work in this domain has focussed exclusively on capturing just the geometry of the input mesh, rather than on the design of a suitable parameterization. In contrast we separate the two tasks of first designing a parameterization and second approximating surface geometry. It is worth noting that there has been some work on parameterization design in the texture mapping literature [Maillot 1993, Bennis et al. 1991]. While these algorithms work well with regular data sets, such as discretized splines, they can exhibit objectionable parametric distortions in general [Eck et al. 1995].

To a user of our system, feature curves are a natural extension of the parameterization strategy. As discussed in the previous section, a useful interpretation of our automatically generated spring meshes is as a blend on the polygonal surface of the four boundary curves.

The interpretation extends in a straightforward manner to feature curves. First, assume we are given just the four boundary curves and feature curves in space (i.e. there is no underlying polygonal surface). A reasonable spring mesh based on this input is a blend in space of all the curves involved. This operation is referred to as *skinning* in the field of parametric surface design. As such, it is a straightforward extension of the more common Coon's patch blend [Farin 1990]. In the presence of an underlying polygonal mesh, a reasonable result for the spring mesh is *a smooth blend on the surface of both the boundary and the feature curves*. We use this intuition to modify our parameterization algorithm in the presence of feature curves.

5.10.1 Specifying a feature curve constraint

Feature curves (or simply *features*) may be specified using exactly the same painting procedure that was used to create patch boundaries (see chapter 3). Our feature curves are meant to be iso-curves of the final spring mesh. Therefore three rather obvious constraints are imposed during the painting of feature curves:

- Features must lie on the polygonal mesh surface.
- Features must lie within the polygonal patch being parameterized.
- Features must start and end at points on opposing boundary curves.

We impose no further constraints on the placement of feature curves. To specify a feature a user paints (on the surface) a face-point curve that starts at a point on one boundary curve and terminates at a point on the opposing boundary curve. The user must paint reasonable feature curves to obtain sensible parameterizations. An example of an unreasonable set of feature curves are two u (or v) feature curves that intersect with each other at one or more points. In this case the spring mesh generated by our algorithm will fold on itself since the feature curves folded onto themselves to begin with. We leave these kinds of design choices to the user rather than alleviate them by adding constraints to our parameterization procedure.

Imposing the first two constraints (i.e. that feature curves lie on the polygonal mesh and within a polygonal patch) is trivial: we simply restrict the painting procedure to vertices and polygons of the specific polygonal patch. The third constraint is imposed as follows. When a user picks a start point for a feature curve, we identify the closest point on the boundary curve that is nearest to the face-point (or mesh vertex) picked by the user. This selects out both a closest boundary curve and a point on that boundary curve that is closest to the start point. We implement this operation using a brute-force approach: we compute the Euclidean distance between the user defined point to each face-point of each boundary curve and pick the closest face-point. Despite being a brute force algorithm, this operation is rapid in practice. A graph search based algorithm might be more efficient but in practice it is not needed for this particular operation.

Once we have determined the start point on a specific boundary curve, we join the point to the first user picked point using a procedure similar to the one explained in section 3.2. The user now paints the rest of the feature curve using the usual curve painting process. The boundary curve on which the end point lies is the one opposite to the start point's boundary curve. To terminate the feature curve, we must add a curve section that reaches from the last user picked point to the closest point on this opposing boundary curve. This is accomplished using a procedure similar to the one used to compute the start point of the feature curve. First, we find the face-point on the opposing boundary curve that is closest in Euclidean space to the last user-picked face-point. Second, we create a curve section between the two points and append this to our feature curve.

Once painted, feature curves may be edited using exactly the same kind of tools that were used to edit boundary curves. The result of our feature curve painting step is a polygonal patch annotated with one or more u and v iso-curves. These iso-curves are now used as constraints to the spring mesh relaxation process.

5.10.2 Using feature curves to guide spring relaxation

While feature curves are a fairly intuitive extension of our parameterization strategy from a users viewpoint, the resulting spring meshes are likely to be even more globally distorted than an unconstrained spring mesh. An example of the additional distortion introduced due to feature curves is shown in figure 5.7. The additionally distorted spring mesh is the desired result since it is the the user who has specified the feature curves in the

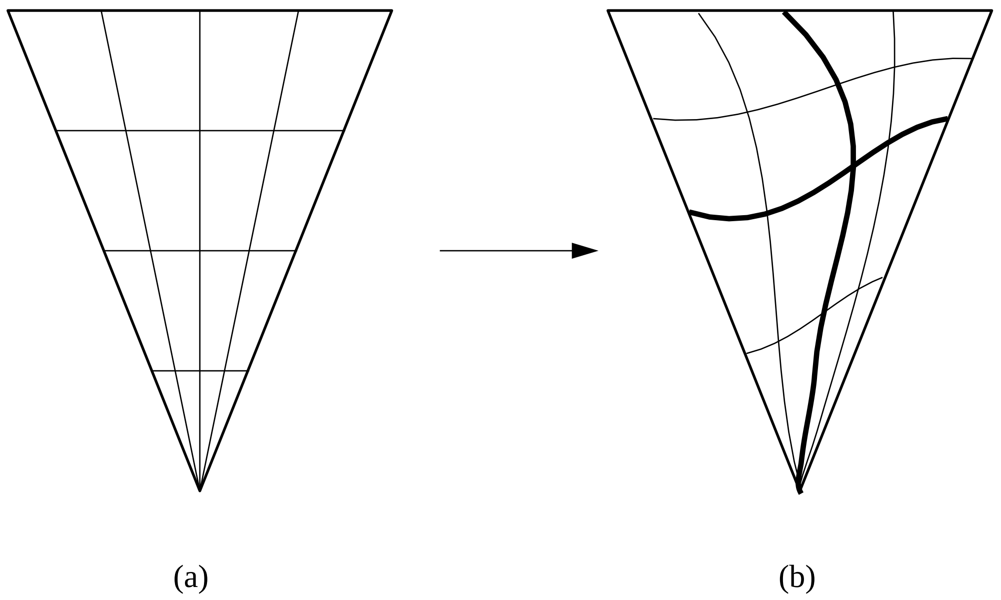

Figure 5.7. Parametric distortion introduced by feature curves. (a) shows an already distorted patch shape and the associated spring mesh (shown as a grid). There are no feature curves in this patch. Note that the spring mesh satisfies our three parameterization rules but is still distorted with respect to area. (b) shows the same patch shape but with two feature curves as well (shown as thicker curves). Note that the resulting spring mesh is even more distorted than the one shown in (a).

first place. Let us therefore re-examine our three sampling criteria to accommodate the distortion introduced by feature curves. In our discussions we will find it useful to use the notion of a *spring section* and a *spring sub-patch*. These are defined as follows:

- *Spring mesh section*: a set of spring points that lie in-between two successive u or v feature curves.
- *Spring mesh sub-patch*: a set of spring points that are bounded by adjacent feature curves (or boundary curves) in both the u and v directions.

These two terms are further illustrated in figure figure 5.8.

Now consider what it means for a feature curve $\mathbf{F_u}$ to be the m-th iso-curve of the spring mesh in the u direction. For a given resolution of spring mesh what this means is that the particular set of spring mesh points that correspond to the m-th iso-curve of the spring mesh (in the u direction) must all lie on the feature curve $\mathbf{F_u}$. As the spring mesh resolution becomes finer the spring mesh iso-curve corresponding to $\mathbf{F_u}$ simply has more spring points. All of these spring points must continue to lie on $\mathbf{F_u}$. With these definitions in mind, let us now consider each of our three parameterization rules in the presence of feature curves.

5.10.3 Modifications to the arc-length criterion

Figure 5.9 motivates the need for a modification to the original arc length criterion when feature curves are present. The patch in the figure has two feature curves in the v direction i.e. F_{v1} and F_{v2}. The corresponding spring sections are labeled S_{v1}, S_{v2} and S_{v3} respectively. Let us assume for a moment that we've already created a satisfactory spring mesh re-sampling of this patch. Now consider two iso-curves I_1 and I_2 of the final spring mesh re-sampling as shown in the figure.

First observe that the part of I_1 that lies within section S_{v1} has the same number of spring points as the part of I_2 that lies within section S_{v1}. Similarly, both iso-curves have the same number of spring points in the parts that lie within each of section's S_{v2} and S_{v3}. These numbers are noted in the figure as n_{sv1}, n_{sv2} and n_{sv3} respectively.

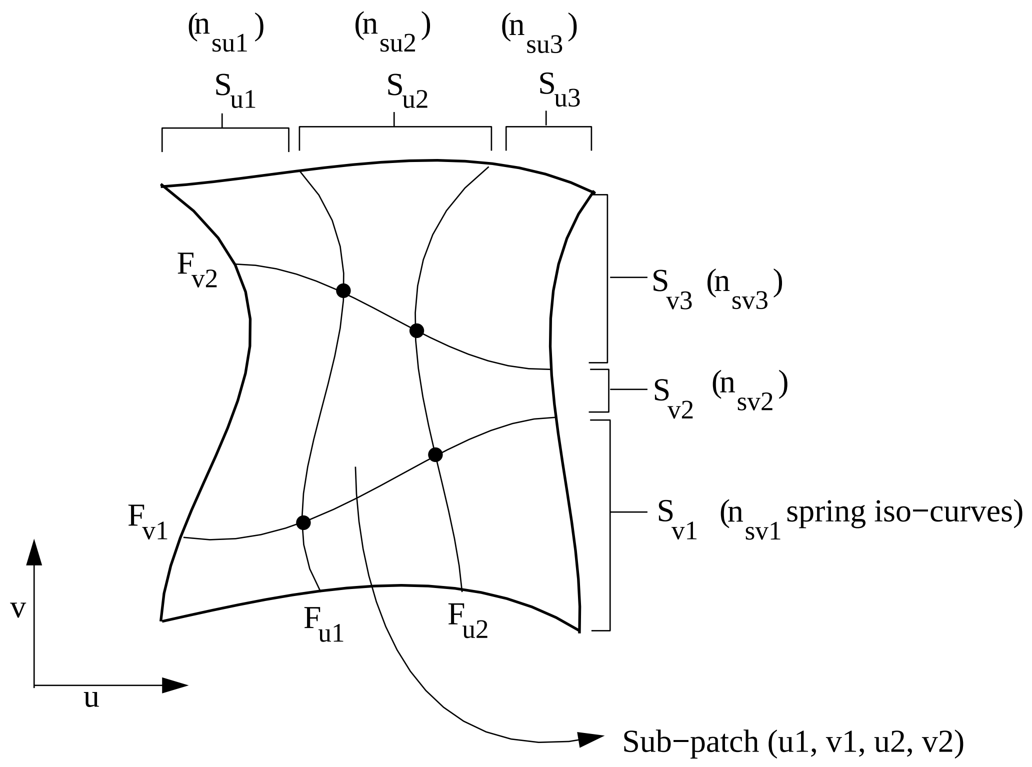

Figure 5.8. Spring mesh sections and sub-patches. The polygonal patch shown here has four feature curves - two in each parametric direction. The feature curves in the u direction are labeled F_{u1} and F_{u2} and the feature curves in the v direction are labeled F_{v1} and F_{v2}. The intersections of these feature curves are shown as solid circles. A *spring mesh section* is a set of spring points that lie in-between two adjacent feature curves or boundary curves. Thus the two u feature curves F_{u1} and F_{u2} create three u sections. These are labeled as S_{u1}, S_{u2} and S_{u3}. Thus, for example S_{u1} consists of all the spring points that lie between the boundary curve on the left and F_{u1} and so on. The three v sections created by F_{v1} and F_{v2} are labeled in similar fashion. The number of *iso-curves* in each section S_k are indicated alongside as n_{sk}. There are 9 *sub-patches* created by the 4 feature curves shown. Each sub-patch is bounded by four feature or boundary curves. Sub-patches are indexed according to the u and v values of the four surrounding feature (or boundary) curves. The central sub-patch is therefore labeled in the figure as *Sub-patch*$(u1, v1, u2, v2)$. For convenience we will assign the boundary curves u and v values e.g. the boundary curve on the left is assigned the value $u0$, while the one on the right is assigned the value $u3$.

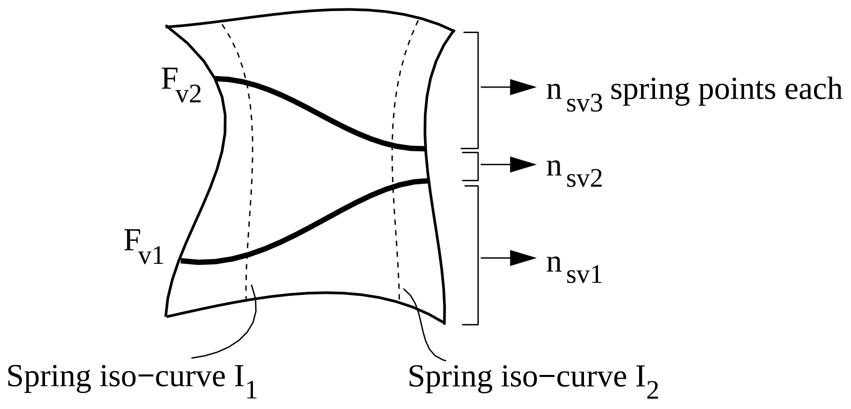

Figure 5.9. Changes to the arc length criterion. The patch shown has two feature curves F_{v1} and F_{v2}. The two iso-curves I_1 and I_2 that are shown here each have the same number of spring points within a particular section of the patch. See the text for details on why this fact implies that we cannot satisfy our original arc length criterion.

Also observe that the number of spring points of I_1 within section S_{v1} is independent of the number of spring points of I_1 within each of sections S_{v2} and S_{v3} i.e. the parts of a given iso-curve that lie within different sections are independent of each other. This is because each feature curve is an actual iso-curve of the spring mesh. Therefore, spring points in one section cannot cross over to neighboring sections. This observation implies that unless we choose the relative number of sample points for an iso-curve within each section, we cannot hope to maintain arc length uniformity over the entire length of the iso-curve.

Let us assume that we compute the number of sample points for each section of I_1 such that we have a uniform arc length distribution of spring points along I_1 i.e. assume we have fixed the numbers n_{sv1}, n_{sv2} and n_{sv3} such that the arc length criterion holds for I_1. Now, from the figure we note that the piece of I_1 within section S_{v1} is much shorter than the piece in section S_{v2}. Therefore in order for arc length uniformity along I_1 at least the following must be true:

$$n_{sv2} > n_{sv1}$$

i.e. the number of spring point of I_1 within S_{v2} must be greater than the number of spring points of I_1 within S_{v1}.

Now using exactly the same arguments for I_2 as we did for I_1:

$$n_{sv2} < n_{sv1}$$

i.e. for I_2 to satisfy the arc length criterion, the number of spring points of I_2 within section S_{v1} must be greater than the number of spring points in section S_{v2}.

Clearly the two requirements inferred from the figure are in contradiction. This indicates that in the presence of feature curves we cannot in general satisfy our original criterion for arc length. Thus we cannot ensure that an arbitrary iso-curve will uniformly sampled over its entire length. However note that *within a particular section* we can in fact achieve a uniform arc length for each iso-curve, This observation forms the basis for our modified arc length criterion:

- *Arc length uniformity*: the spring point spacing along a particular iso-curve should be uniform within each piece of the iso-curve that is within a spring mesh section (or sub-patch).

Note that an alternate formulation could have been to allow for a certain minimal arc length non-uniformity within each section. An example of such a scheme is one that blends arc length non-uniformity across feature curves. This could be a useful modification for some applications, but we have found our method to work well in practice.

5.10.4 Modifications to the aspect-ratio criterion

The parametric distortion induced by feature curves affects our aspect ratio criterion as well. In this case our original criterion is compromised by the fact that each spring mesh sub-patch dictates its own preferred aspect ratio. Since we have a tensor product parameterization these preferred aspect ratios conflict with each other. Take for example the patch and feature curve set shown in figure 5.10. We have labeled four spring mesh sub-patches in the figure as S_1, S_2, S_3 and S_4. S_1 and S_3 are tall and thin (i.e. more resolution in the $\mathbf{v}$ direction) while S_2 and S_4 are short and broad (i.e. more resolution in the $\mathbf{u}$ direction). Since we have a tensor product parameterization S_1 and S_4 have the same number of $\mathbf{u}$ subdivisions, as do S_2 and S_3. These numbers are labeled as n_{su1} and n_{su2} respectively.

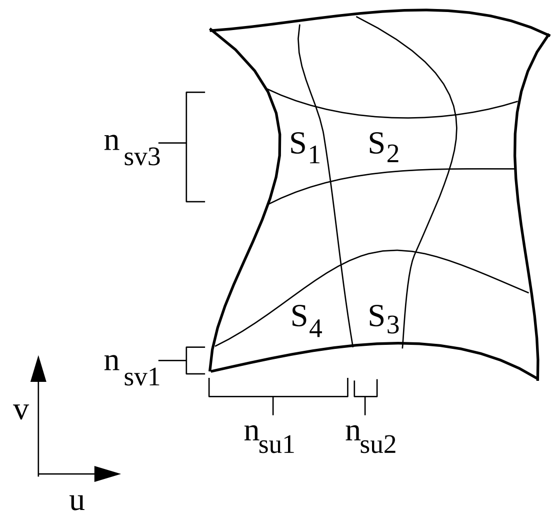

Figure 5.10. Modifications to the aspect ratio criterion. The text explains why, because of this particular set of feature curves, the aspect ratio criterion as defined earlier can no longer be satisfied.

Similarly, S_1 and S_2 have the same number of **v** subdivisions as do S_4 and S_3. These numbers are labeled n_{sv3} and n_{sv1} respectively. It is straightforward to show that it is not possible to satisfy the aspect ratio criteria of all four of these sub-patches. Let us assume that we were able to satisfy the aspect ratio criterion for each of the four sub-patches. Since S_1 is tall and thin we have the equation:

$$\frac{n_{sv3}}{n_{su1}} = 1 + \alpha_1 \;\; \alpha_1 > 0$$

A similar argument for each of S_2, S_3 and S_4 gives us the following equations:

$$\frac{n_{su2}}{n_{sv3}} = 1 + \alpha_2 \;\; \alpha_2 > 0$$

$$\frac{n_{sv1}}{n_{su2}} = 1 + \alpha_3 \;\; \alpha_3 > 0$$

$$\frac{n_{su1}}{n_{sv1}} = 1 + \alpha_4 \;\; \alpha_4 > 0$$

Taking a product of the left and right hand sides of these four equations gives us:

$$\frac{n_{sv3}}{n_{su1}}\frac{n_{su2}}{n_{sv3}}\frac{n_{sv1}}{n_{su2}}\frac{n_{su1}}{n_{sv1}} = \Pi_{j=1}^{4}(1+\alpha_j) \ \ \alpha_j > 0$$

i.e.

$$1 = \Pi_{j=1}^{4}(1+\alpha_j) \ \ \alpha_j > 0$$

which is clearly a contradiction since the right hand side of the equation is strictly greater than 1.

We conclude that, in general it is not possible to simultaneously satisfy all the aspect ratio constraints created by a set of feature curves. Unfortunately unlike the the arc length criterion, there is no reasonable subset of the aspect ratio criterion that we can hope to satisfy. Therefore, we choose to preserve our goal of aspect ratio uniformity with the additional caveat that in general it is not possible to attain optimal aspect ratios for every single sub-patch; in other words:

- *Aspect ratio uniformity*: the spacing along a u iso-curve should be the same as the spacing along a v iso-curve for every spring mesh sub-patch to the extent feasible within the constraints imposed by the feature curves.

This is a weak criterion at best and this is reflected in our implementation strategy for this criterion: we keep the decision regarding the best initial aspect ratios to the user.

5.10.5 Modifications to the fairness criterion

Let us turn now to our parametric fairness criterion. Recall that this term was introduced as a regularization term to create a parametrically fair spring mesh. The term makes each iso-curve as tight (or minimal length) as possible given the other two constraints. We would like our parameterizations to always possess this property regardless of the number of constraints added. Figure 5.11 shows the differences in the fairness criterion in the presence of feature curves. If we consider any single iso-curve (I_1 or I_2 in the figure) as it passes through several sections we note that there are two kinds of fairness that we would like to ensure for the iso-curve:

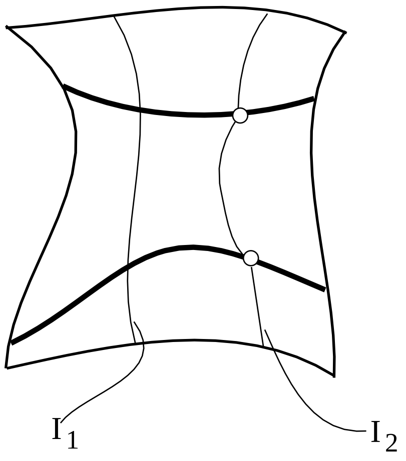

Figure 5.11. The parametric fairness criterion in the presence of feature curves.

1. Parametric fairness within a spring section.
2. Parametric fairness at the interface of two adjoining spring sections.

Thus in the figure shown I_1 is an iso-curve that satisfies both properties while I_2 does not satisfy property 2 i.e. I_2 is un-fair at the juncture of two sections. Our statement of the fairness criterion however need not change since for an iso-curve to be of minimal length in the presence of feature curves it must automatically satisfy both of the properties above. Therefore:

- *Parametric fairness*: Every u and v iso-curve should be of the minimum possible length given the first two criteria.

5.11 Feature curves: an implementation

To summarize, we have proposed changes to each of our three parameterization rules in the presence of feature curves. The changes are:

- The arc length criterion holds within a spring section but not globally.
- The aspect ratio criterion has been weakened to state that the criterion holds only where it is feasible (given the constraints imposed by the feature curves).

- The parametric fairness criterion stays the same as before. The main difference is that our implementation must consider fairness both within a spring section as well as at the interface of two adjoining spring sections.

From an implementation perspective each spring mesh sub-patch essentially behaves like an independent patch except for the *parametric fairness* constraint which is propagated across feature curves to neighboring sub-patches. Intuitively then, feature curves act as semi-permeable patch boundaries that only let the parametric fairness criterion through to neighboring sections.

In this section we describe modifications to our earlier coarse-to-fine parameterization strategy in the presence of feature curves. Our discussion is in four parts. First, we discuss changes to fundamental spring point data structures to accommodate feature curves. Second, we discuss how to initialize a spring mesh based on a set of feature curves. Third, we examine the modifications to the spring mesh relaxation process. Finally, we explain the changes to our spring mesh subdivision step.

5.11.1 Data structure changes

Thus far, the only constraint on our spring points was that they should lie on the surface of the polygonal mesh i.e. a spring point was just a face-point. With the addition of feature curves, additional constraints must be imposed on spring points. These changes merely account for those spring points that are either constrained to feature curves or lie at the intersection of two feature curves. Figure 5.12 motivates these additional changes.

First consider the face-points that lie at the intersection of two feature curves. Since each of the feature curves eventually corresponds to an iso-curve of the spring mesh, the intersection of two curves must correspond to a spring point. Since the feature curves do not shift position during the parameterization procedure, neither does this intersection spring point. We call such immovable spring points *anchors* or *anchor spring points*. In figure 5.12 the spring point P_1 corresponds to such a point. Note that all spring points that lie on boundary curves are effectively anchors since they never move during relaxation.

Now consider those spring points that lie on a feature curve but not at the intersection

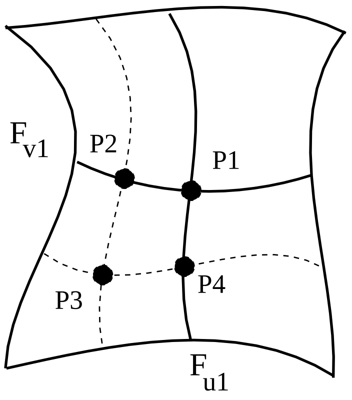

Figure 5.12. Three different kinds of spring points. The figure shows the 2 new kinds of spring points that must be introduced to accommodate feature curves. First, the anchor points: points that lie at the intersection of two feature curves. They are never moved during our relaxation process. $P1$ is an example of an anchor point. The second kind of point introduced are the curve-points: these are points that are constrained to lie on a feature curve. $P2$ and $P4$ are curve points. $P3$ represents an example of a regular face-point.

of two features. These points are constrained to stay on the feature curve at all times during the relaxation procedure. We call these points *curve spring points* or *curve points*. In the figure P_2 and P_3 are curve points. We will find it useful to define a function *MovePointOnCurve* for the curve-points. We will use this operation on curve-points much like we used *MovePointOnSurface* on face-points earlier. Appendix A describes the details of this implementation.

Finally, there are spring points that are not directly affected by the feature curve constraint. These are exactly the points of the spring mesh that aren't anchors or curve points. P_3 in the figure is one such point. These points behave exactly as they did in our original unconstrained re-sampling strategy. With these simple change to our basic data structures let us now examine how to initialize, relax and subdivide feature curve enhanced spring meshes.

5.11.2 Initializing the spring mesh

Our initialization procedure is split into two steps. In the first step we create a spring mesh that is based on just the anchor points of the spring mesh. This is found by computing the intersections of all pairs of u and v feature curves. This initial spring mesh might not meet the requirements of the aspect ratio criterion. Therefore, in the second step we subdivide the initial spring mesh from the first step to achieve a satisfactory aspect ratio.

Our first step is to compute the intersections of each feature (and boundary) curve in the u direction with each feature (and boundary) curve in the v direction. These spring points are all anchors and constitute our initial spring mesh. Let us consider one of these computations i.e. the intersection of two feature curves; one each in the u and v directions. Given a pair of any two surface curves, they may intersect in an arbitrary number of points. The intersections might include complex cases such as tangential intersections. However, we are concerned with a much simpler problem. In our case, the two surface curves are iso-parametric curves in the u and v direction respectively and usually have one well defined intersection. In a tensor product parameterization every single u iso-curve must intersect every v iso-curve in at least one point.

Given two feature curves F_u and F_v, procedure *IntersectSurfaceCurves(F_u, F_v)* finds a face-point P_{uv} that lies on both feature curves. Recall that all our surface curves are represented as face-point curves. Therefore, procedure *IntersectSurfaceCurves* must compute the intersection of two face-point curves. Note that these curves need not have any face-points in common even if the surface curves they represent (i.e. sample) do in fact intersect. Even a linear (or higher order) reconstruction of the face-point curves is not guaranteed to intersect in an exact, mathematical sense. Therefore, a reasonable implementation of the procedure might be to find a point on a linear re-construction of the first face-point curve that is closest in $\Re^3$ to a linear re-construction of the second face-point curve. However, note that this implementation could yield incorrect results in pathological cases e.g. if the face-point curves are close together at certain points in space even though at those points they are widely separated on the surface. To alleviate this problem we restrict our closest point computation between the face-point curves to an *intersection neighborhood* on each face-point curve. The intersection neighborhood is computed as follows. First, we compute graph paths corresponding to each surface curve by chaining together a sequence of vertices that are each closest to consecutive face-points on the curve. Since the feature curves follow the two iso-parametric directions, they cross each other on the surface. Therefore, the computed graph paths for the feature curves intersect at a mesh vertex. On each curve, we use as our intersection neighborhood a small (constant size) set of face-points in the neighborhood of the face-point on that curve that corresponds to the mesh vertex.

An initial spring mesh that is based on just the intersection of the feature curves may not satisfy our aspect ratio criterion. Since our subdivision step uniformly subdivides our spring mesh, the initial aspect ratio will be preserved in the final spring mesh. The second step of our initialization process is to subdivide the spring mesh sections to achieve a better initial aspect ratio. As we observed earlier, it is not in general possible to satisfy our aspect ratio criterion globally. In fact the user might desire a distorted aspect ratio in the first place. For these reasons we allow the user to supervise this second stage of the initialization process. In our system, we allow the user to selectively subdivide spring mesh sections to attain a desirable initial aspect ratio.

5.11.3 Relaxing the spring mesh

The goal of our relaxation step is the same as before: to re-distribute spring points on the surface such that our criteria of arc length and parametric fairness are met. As before, we achieve this goal by first computing forces on individual spring points (based on the parameterization rules) and second by using these forces to move the points over the mesh surface. The equilibrium position of the spring points minimizes the resultant forces due to our rules and therefore represents the desired minimal energy spring mesh.

Anchor points by definition do not move during the relaxation procedure. Let us consider the relaxation based forces for the other two kinds of face-points: regular spring points and curve-points.

Relaxation for regular spring points remains exactly the same as before (see section 5.5.3). These are the points that are completely within a spring sub-patch. Since all forces are computed locally, these points do not interact directly with the spring points of other spring sub-patches. They interact with one or more of the other spring points in the same spring sub-patch or with the anchors or curve-points on bounding feature (or boundary) curves. In both cases, the forces on a regular spring points may be computed in exactly the same fashion as they were earlier.

Let us now consider the curve-points. By definition these points are constrained to lie on the feature curves. Therefore, any forces on them will move them only on the feature curve itself. As mentioned earlier feature curves act as semi-permeable boundaries: they propagate fairness across neighboring sub-patches but not arc length. There are two kinds of forces acting on the curve-point: a force based on parametric fairness and a force based on arc length. To illustrate how these forces are computed let us consider the curve-point P shown in figure 5.13. As shown in the figure, P is constrained to lie (or move) on a v iso-curve F_v. The two neighbors of P along the feature curve are labeled P_{vl} and P_{vr}. Since P, P_{vl} and P_{vr} are all constrained to a fixed feature curve F_v it doesn't make sense to impose a fairness constraint on P based on P_{vl} and P_{vr}. This is because we cannot hope to make the the iso-curve of the spring mesh that corresponds to F_v any "tighter" than F_v itself. Therefore these two neighbors of P are used only in arc length computations.

Now consider the other two neighbors of P, labeled P_{ua} and P_{ub} in the figure. Since P lies on a feature curve, these points belong to different spring mesh sections (and therefore sub-patches). Recall that feature curves are impermeable to the arc length constraint i.e. arc-length uniformity is not maintained across a feature curve. Therefore these two neighbors of P are not used for arc length computations. We use them only for the fairness computation.

To summarize, P_{ua} and P_{ub} are used to compute the parametric fairness based force and P_{vl} and P_{vr} are used to compute the arc length based force on P. The resultant of these two forces moves P to a new position along the feature curve F_v. Let us turn now to the computations of these forces.

The arc length force should move the curve-point P along the feature curve F_u to a new point that lies at the mid-point, on the curve, of P_{ul} and P_{ur}. This is indicated in figure 5.13a. The arc length force on P is therefore computed as follows. Let $\mathbf{L}_F(P_1, P_2)$ be a function that returns the arc length between two points P_1 and P_2 on a feature curve F. If $\mathbf{v}_t$ represents the tangent vector at P, then the arc length force on P is given by:

$$\mathbf{F}_{arc} = (\mathbf{L}_{F_u}(P_{ur}, P) - \mathbf{L}_{F_u}(P, P_{ul}))\mathbf{v}_t \tag{5.14}$$

The goal of the parametric fairness based force now is to ensure that the iso-curve on which P, P_{ua} and P_{ub} lie is as fair (or "tight") as possible. This will occur when the three points are as close to being in a straight line on the surface as possible. An example of

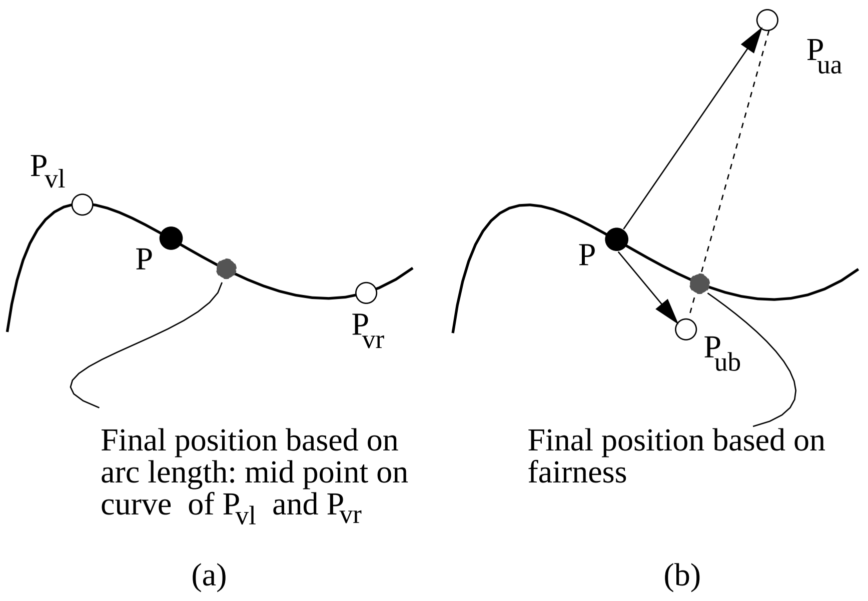
P_vl
P
P_vr
Final position based on arc length: mid point on curve of P_vl and P_vr
(a)
P_ua
P
P_ub
Final position based on fairness
(b)

Figure 5.13. Curve point motion. During the relaxation process curve points can move only along the feature curve on which they were created. The figure shows the neighbors of a curve point P both along the feature curve (P_{vl} and P_{vr}) and along the iso-curve in the other iso-parametric direction (P_{ua} and P_{ub}). (a) shows the final position of the curve point due to arc length forces. (b) shows the final position of the curve point due to fairness forces.

a final state of P based on the fairness force is indicated in figure 5.13b. To compute the fairness force, we must first define the forces on P due to each of P_{ua} and P_{ub}. These are defined as:

$$\mathbf{F}_{up} = P_{ua} - P$$

and

$$\mathbf{F}_{down} = P_{ub} - P$$

The vectors defined above do not in general lie on the surface tangent plane at P. Let the corresponding vectors that are projected to the tangent plane at P be $\mathbf{F}^t_{up}$ and $\mathbf{F}^t_{down}$. Since the arc length constraint is not propagated across feature curves, we are not concerned with the actual lengths of these vectors. Examining the desired rest position of P on the curve F_u we find that if the unit vectors in the directions corresponding to the two forces defined earlier face in opposite directions, our fairness criterion will be satisfied. Accordingly if we let the resultant of these vectors on the tangent plane at P be:

$$\mathbf{F}^t_{fair} = \frac{\mathbf{F}^t_{up}}{\|\mathbf{F}^t_{up}\|} + \frac{\mathbf{F}^t_{down}}{\|\mathbf{F}^t_{down}\|} \tag{5.15}$$

and the fairness criterion is given by

$$\mathbf{F}_{fair} = (\mathbf{F}^t_{fair} \cdot \mathbf{v}_t)\mathbf{v}_t \tag{5.16}$$

where $\mathbf{v}_t$ once again is the tangent to the curve at P.

Thus the fairness and arc length based forces pull the curve-point P in potentially different directions along the iso-curve F_v. The resultant force on P is given by a weighted combination of $\mathbf{F}_{arc}$ and $\mathbf{F}_{fair}$.

$$\mathbf{F}_{result} = \alpha_{curv}\mathbf{F}_{arc} + \beta_{curv}\mathbf{F}_{fair} \tag{5.17}$$

In practice when our relaxation iteration starts the points are re-distributed predominantly according to the arc length criterion (i.e. $\alpha_{curv} = 1$ and $\beta_{curv} = 0$). As the relaxation progresses progresses the fairness criterion is given increasing weight (i.e. $\alpha_{curv} = 0$ and $\beta_{curv} = 1$). This simple strategy yields satisfactory results. We have found it useful to supply the user with the option to interactively vary the relative weights of α_{curv} and β_{curv} during the relaxation process.

5.11.4 Spring mesh subdivision

Spring mesh subdivision in the presence of feature curves is a straightforward extension of our earlier subdivision process. Feature curves provide additional flexibility in that a user can selectively subdivide individual sections of the patch and not affect other sections in the process. Note that it is not possible to subdivide an arbitrary spring sub-patch because our spring mesh is a regular grid. To uniformly subdivide a sub-patch in both directions we must subdivide both sections that the sub-patch belongs to. This is an inherent limitation of the non-hierarchical representation of our spring mesh and of a tensor product B-spline surface rather than a limitation of the techniques explained herein.

The subdivision process proceeds in much the same way as already explained in section 5.6. The one new case we encounter with feature curves is the subdivision of an iso-curve that corresponds to a feature curve. This requires us to find the mid-point of each pair of adjacent curve-points that lie on the same feature curve. This is accomplished simply by finding a new spring point that lies at the mid-point on the feature-curve of the two original spring points.

The ability to selectively subdivide each spring mesh section allows the user to selectively refine regions of the mesh where more spring points might be needed. Since, the fitted spline surfaces closely follow the underlying spring mesh this effectively allows the user to place more spline iso-curves in any section. It is worth noting that the region a user chooses to selectively subdivide does not necessarily have to be geometrically complex or dense. Take for example the spline control meshes of human (or humanoid) faces that are generated for animation. Even geometrically simple regions such as the eye socket or the region around the mouth are often represented using a dense control mesh to provide for convincing facial expressions [Parke & Waters 1996]. Thus the choice of which region to selectively subdivide is often domain dependant and is better determined by the user than by an automated algorithm.

5.12 Feature curves: discussion

From a surface design perspective, feature curves are a natural extension to our basic parameterization strategy (section 5.10). From an implementation perspective, feature curves are straightforward to incorporate into our parameterization strategy. Although it is possible to create poor parameterizations using feature curves, the principal benefits of our parameterization strategy that were outlined in section 5.7 continue to hold true in the presence of feature curves. In addition, the user can now create arbitrary parameterizations by specifying just a few feature constraints. Figure 5.14 shows an example of feature curves in action.

5.12.1 A variational perspective of our parameterization algorithm

A useful property of feature curves is that a user specifies only those constraints that are important in their application. The rest of the parameterization is “filled in” automatically, interpolating those constraints with a minimum energy (i.e. distortion) spring mesh. In this sense, our parameterization strategy is similar in philosophy to variational (or free-form) surface design [Welch & Witkin 1994] where a minimum energy surface geometry is automatically computed given a small set of user specified constraints. In our pipeline, the geometry of the surface is always constrained to approximate the original polygon mesh; it is instead the parameterization of each patch which is computed to satisfy user-specified constraints. From this perspective, our parameterization problem may be viewed as a *variational parameterization design* problem and our specific spring mesh based solution may be viewed as a finite difference solution to this design problem.

5.13 Results

In this chapter we presented a robust and efficient parameterization algorithm for irregular polygonal patches. Several desirable properties of our parameterization algorithm were analyzed in section 5.7. In this section we demonstrate these properties using two practical examples: the Armadillo and the Bofar model. Figure 5.15 shows some results of our

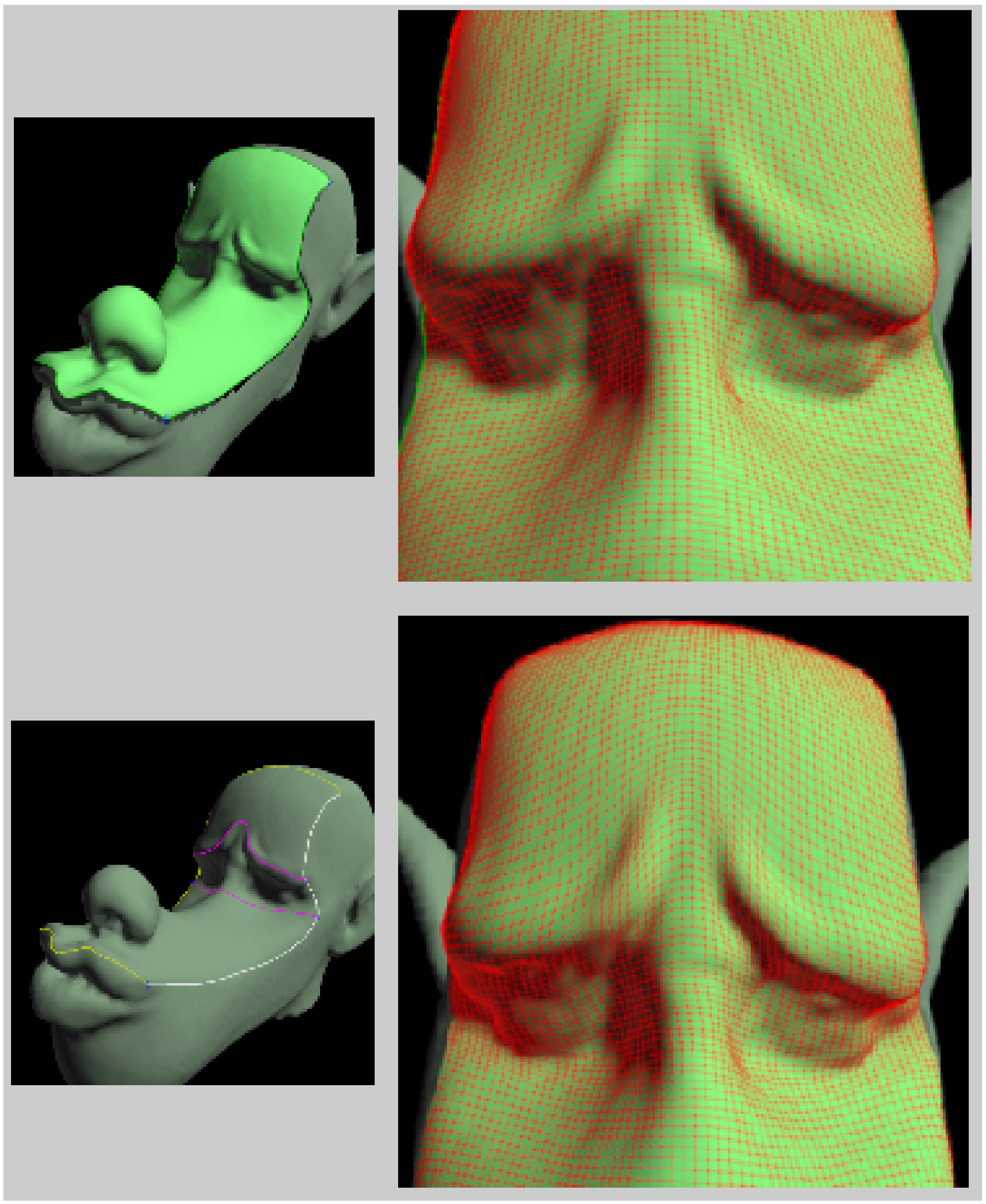

Figure 5.14. (a) shows the final spring mesh re-sampling (reconstructed as triangles) of a polygonal patch defined on the wolf's head. This re-sampling did not use any feature curves. (b) shows a detail of this automatically generated spring mesh around the wolf's eyebrows. Note that the parameter lines do not follow the curve of the eyebrows. In particular, note that in certain regions the geometry of the eyebrow runs diagonal to the two principal directions of the parameterization. Modifying the geometry of the eyebrow using this parameterization would be very un-wieldy. Also note that the spring mesh samples are uniformly distributed over the entire polygonal mesh surface. (c) shows (in purple) a set of two feature curves that were specified on the polygonal patch to guide the parameterization. These feature curves were introduced to accomplish two goals. First, to have the spring mesh follow the curve of the eye-brows. This change now allows the user to control the geometry of the eyebrow using a single (or a few adjacent) iso-parametric curve. Second, we wanted to generate a denser set of samples in the region around the wolf's eyes (i.e. between the two feature curves). This change gives the user greater control (more iso-parametric lines) over the geometry of the eyes of the wolf. (d) shows a detail of the final spring mesh re-sampling of the wolf's head in the presence of these two feature curves. Note that the spring mesh generated by our feature-driven parameterization process satisfies both goals.

parameterization algorithms on these models. Note the use of feature-driven parameterizations for the Bofar model.

The first notable aspect of these parameterizations is the fidelity of our final spring mesh to the original polygonal model. In figures 5.15(a) and (c) we have shown split views of the two models. In both cases the left half of the model is a triangular reconstruction of the spring mesh. The right half of the model is the original polygonal data. Note that the triangular reconstruction of our spring mesh is indistinguishable from the original polygonal data. A more detailed example of the fidelity of our spring mesh re-sampling algorithm was shown earlier in figure 5.6.

The second notable aspect about the parameterizations of the Armadillo and Bofar is the rapidity with which the parameterizations were created. The Armadillo has about 350,000 polygons and 104 patches. The final gridded re-sampling of the entire Armadillo took under 8 minutes (the figure shows just half of the final spring meshes) The Bofar model had about 450,000 polygons. It had 12 patches and 33 boundary curves. In addition each patch had one or more feature curves that were added to it. These patches and boundary curves were carefully placed to allow for subsequent animation. The entire spring mesh re-sampling took under 2 minutes for this set of patches. All computations were timed on a MIPS 250 MHz R4400 processor. It is worth noting that in both these cases, the precise positioning of boundary and feature curves took significantly more time than the actual parameterization process itself (see chapter 3).

Note that despite the fact that Bofar has a larger number of polygons than the Armadillo it has less geometric detail. Therefore our termination criterion generates spring meshes that contain more spring points than are strictly necessary. However note that neither the speed of our algorithm nor the quality of our results are compromised as a result of this conservative termination criterion.

A third notable aspect about the parameterizations of the models shown is that they were constructed in an interactive environment. Our algorithm's coarse-to-fine relaxation strategy was invaluable in this environment. The modeler could pause the relaxation at a coarser resolution and make modifications to the input (such as modifying boundary curves or adding feature curves) and preview results at interactive speeds. This dynamic user interaction would not have been possible with a static (i.e. slow) parameterization scheme.

The ability to interpolate a few (important) feature curves and automatically fill in the rest of the spring mesh, is a unique and useful ability of our parameterization algorithm. For example, the Bofar model shown in figure 3.17b was constructed to meet the needs of an animation. Note the use of feature curves to guide the parameterization of the model. These feature curves were specified to match certain anatomical characteristics of the model. Note that only a few key feature curves were needed to specify the required parameterization (see figure 3.17b). Our feature driven parameterization algorithm automatically created a parameterization that interpolated these feature constraints. The figure shows an example of the changes induced by the feature curve constraint.

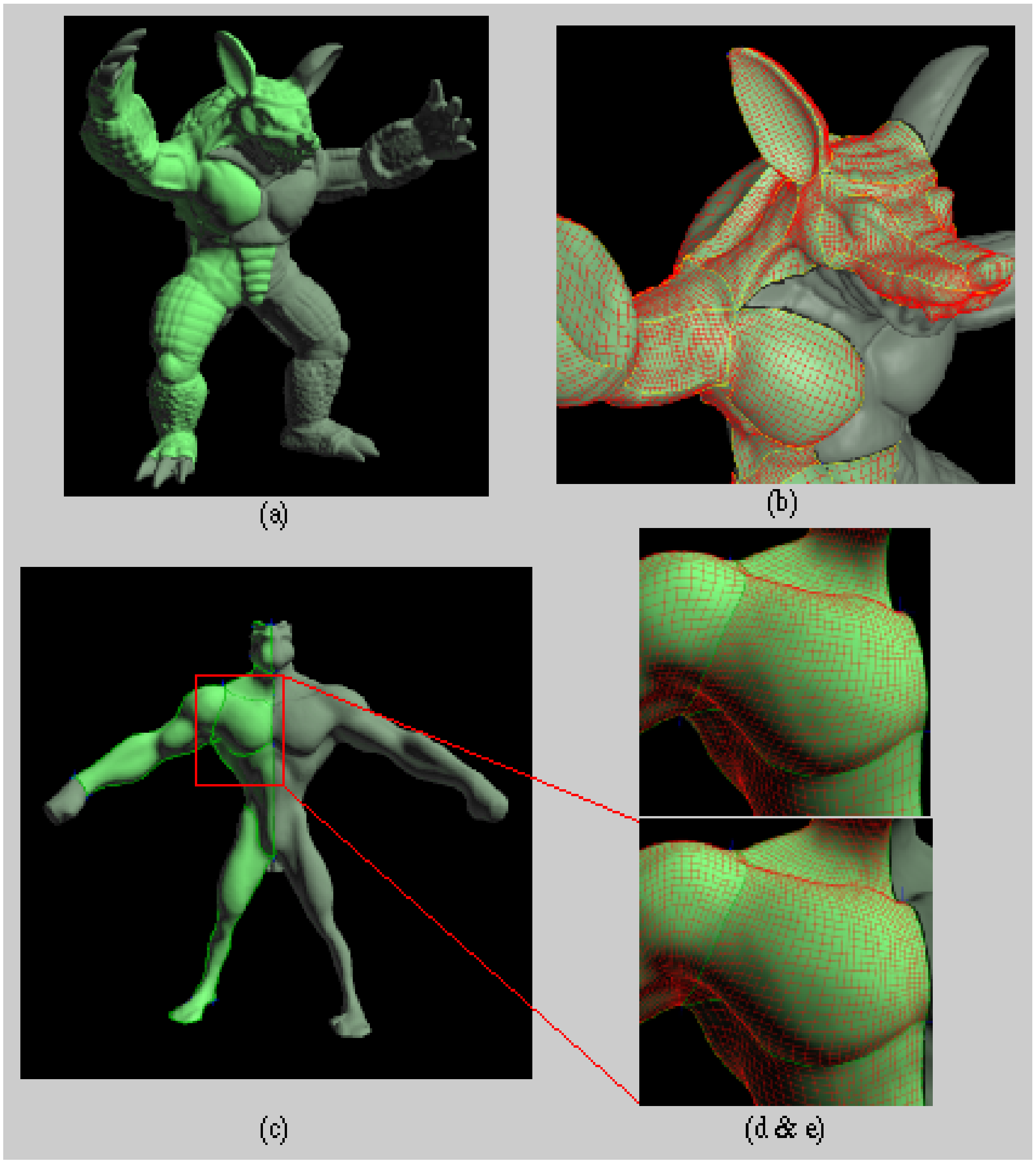

Figure 5.15. (a) shows a split view of the Armadillo model. The left (bright green) half of the model is a smooth triangular reconstruction of our final, per-patch spring meshes (about one-half of 104 patches are shown here). The right half (gray) of the model is the original polygonal mesh. We have omitted the patch boundary lines from this image to avoid clutter. A detail of some of the Armadillo spring meshes at a lower resolution is shown in (b). The spring mesh relaxation process took under 8 minutes for the entire 104 patches (of which only half are shown here). (c) shows a similar split view of Bofar. Only the left half of Bofar was “patched”. The spring mesh relaxation process for the 12 patches took under 2 minutes. All timings were taken on a MIPS 250 MHz R4400 processor. (e) shows a detail of the spring meshes around Bofar’s chest and shoulder regions at an intermediate stage of the parameterization process. Note how the feature curves (see figure 3.17 for the placement of Bofar’s feature curves) on Bofar affect the flow of spring mesh iso-curves. For comparison we have shown in (d) the same set of spring meshes (at similar resolutions) when feature curves were not used. The parameterizations generated in (d) are not incorrect. In fact they are perfectly acceptable for surface fitting purposes. Bofar’s feature curve driven parameterization was created, for purposes of animation, to match certain anatomical features. Notice that for both models, the spring mesh re-sampling accurately captures the detail present in the original polygon mesh. In the next step of our surface fitting pipeline we fit B-spline surfaces to these spring meshes.

Chapter 6

Spline surface fitting

Once our input polygon mesh has been parameterized, the next step in our pipeline is to fit a combination of spline surfaces and displacement maps to the parameterized surface. As explained in chapter 1, the spline surface captures the coarse geometry of the surface while the displacement map captures fine detail. The resolution of the spline surface is chosen by the user and both the spline surface and the associated displacement map are automatically determined based on this resolution. In this chapter we devote ourselves to the discussion of how the B-spline surface fit is obtained once the polygonal patch has been parameterized. We discuss the computation of displacement maps in the next chapter.

Once we have generated the final spring meshes we could use them in one of two ways: first, we could use the spring mesh to assign parameter values to the mesh vertices and proceed with a traditional non-linear optimization strategy to fit to the mesh vertices (since the mesh vertices are distributed irregularly on the polygonal surface). A second approach is to fit our surfaces directly to the spring mesh. As explained in chapter 4 we have chosen to use the second approach i.e. we fit our B-spline surfaces directly to the spring mesh. We are more interested in an approximation to the mesh surface itself rather than just the mesh vertices. As such if the spring mesh is a faithful re-sampling of the polygonal surface then fitting to the spring points does not compromise the quality of our fitted B-spline surface. As the results from the previous chapter demonstrate, the spring meshes generated by our coarse-to-fine gridding algorithm do in fact satisfy the requirements for surface fitting. Furthermore, as explained in chapter 4, fitting to the spring mesh is computationally more efficient.

This chapter describes how we obtain a B-spline fit of a given (user specified) resolution from our spring mesh. We begin (section 6.1) by first characterizing our approximation measure. In particular we address the issue of what it means for a B-spline surface to reasonably approximate a set of data points. Section 6.2 then discusses our specific surface fitting implementation. We then conclude this chapter (section 6.3) with some results of our surface fitting algorithm and discuss its useful properties in an interactive setting.

6.1 Characterizing an approximation measure

Our fitting function is a uniform tensor product B-spline surface of a fixed, user specified resolution. The data set we are fitting to is the spring mesh which is a regular grid of face-points. The goal of the fitting process is to assign positions in $\Re^3$ to the B-spline surface control vertices such that the resulting spline surface is a "reasonable approximation" to the spring mesh. As such there are several approximation strategies that could meet this criterion.

A simple strategy is to sub-sample the spring mesh points and use these as the edit points [Forsey & Bartels 1988] of a B-spline surface. This is clearly some kind of approximation to the spring points. However it is a poor one. Signal processing theory suggests that sub-sampling and reconstruction (in our case tensor product B-spline reconstruction) of a data set can produce an aliased version of the surface that the data set is meant to represent. Of course, there are reasonable strategies in the signal processing literature that overcome such aliasing problems and these could be applied to alleviate the problems with this simplistic solution. However, such modifications involve smoothing out the spring mesh, potentially losing surface detail that would otherwise have been captured by the spline surface. Therefore such a strategy is not acceptable to us.

Instead we seek a B-spline surface output that represents the *best possible* approximation to the spring mesh for a given input control vertex resolution. In other words we would like our fitting method to *minimize the error of fit between the approximating B-spline surface and spring mesh.*

Recall that the gridded data surface fitting problem can be framed in the form of an over-determined linear system of equations (chapter 4, section 4.3.2). It is a well known fact that for such a system the least squares approximation minimizes the error of the solution [Strang 1986]. Therefore, we have chosen a least squares approximation strategy to solve our surface fitting problem. As such, this is a well studied approximation technique and has found use in a variety of application domains. For a general discussion of the least squares approach and its numerical nuances see Lawson's book on the subject [Lawson & Hanson 1974].

6.2 Least squares surface fitting

Least square fitting to the spring mesh is accomplished in two steps.

1. Assign parameter values to the spring points.
2. Perform a gridded data fit to the spring points.

Parameter value assignment is a trivial operation since the spring mesh is a regular grid. Values are assigned by stepping uniformly in parameter space as we move from one iso-curve to the next. The gridded data fitting to spring points now proceeds in identical fashion to what was already explained in chapter 4 (i.e. equation 4.4).

There are two notable facts about this fitting procedure. First, according to our computation the error of the least squares fit is not the cumulative closest distance of every spring point to the B-spline surface. Instead, it is the cumulative distance of each spring point to that point on the fitted spline surface which has the same parameter value i.e. it is the *cumulative parametric distance* of the spring points to the fitted B-spline surface. The mathematical expression for this error term is:

$$E_{param-dist} = \sum_i \sum_j (\| \, Spring(u_i, v_j) - Spline(u_i, v_j) \, \|)^2 \tag{6.1}$$

We call this error the *parametric distance error*. An intuition for the parametric distance error for the simpler case of a curve is provided by figure 6.1

A more traditional method for measuring the error of fit is the *closest distance error*. Since the closest distance of a point to a surface is measured along the normal to the surface this error is also known as the *normal distance error*. It is given by the expression:

$$E_{normal-dist} = \sum_i \sum_j (\| Spring(u_i, v_j) - D_{Spline}(Spring(u_i, v_j)) \|)^2 \tag{6.2}$$

where $D_{Spline}(Spring(u_i, v_j)$ is a function that returns for each spring point, the closest point from that spring point to the spline surface. The difference between parametric and normal distance error is illustrated for a one dimensional case in figure 6.1. It is worth noting that parametric distance error is an upper bound on the normal distance error. Thus, if we can make our parametric error small enough, the normal error must be smaller.

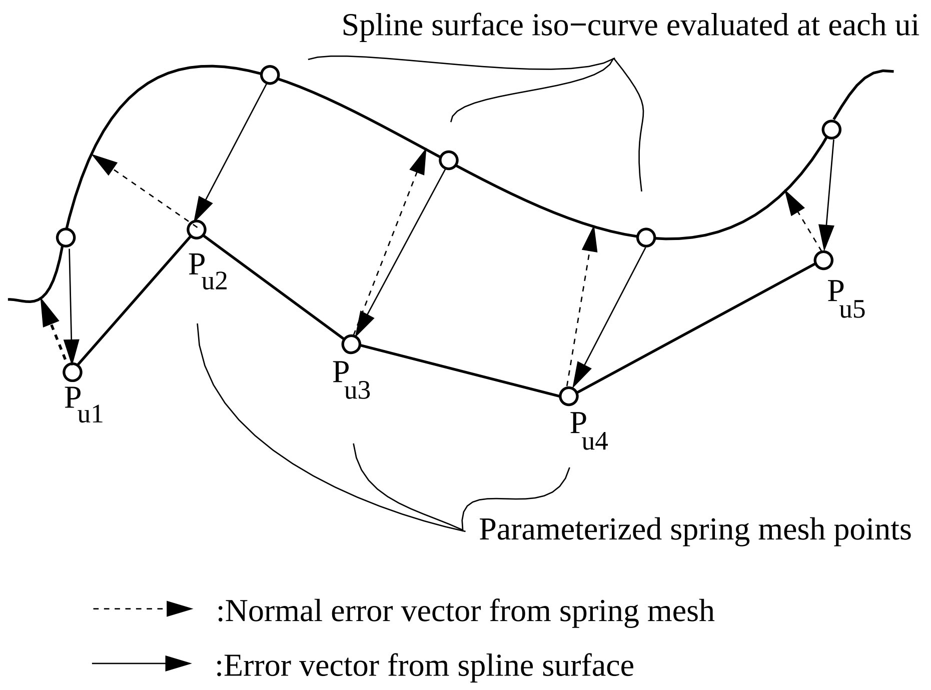
Spline surface iso−curve evaluated at each ui
P u1
P u2
P u3
P u4
P u5
Parameterized spring mesh points
:Normal error vector from spring mesh
:Error vector from spline surface

Figure 6.1. Parametric distance error. The error of our fit is measured as a sum of the magnitudes of the parametric error vectors (shown as solid arrows). An alternative error metric is the normal distance error metric. This is measured as a sum of the magnitude of vectors from a spring point to the closest point on the spline surface (shown as dotted arrows). Note that normal distance error computation can be expensive since it requires a search in parameter space for the closest point on the spline surface to the spring point.

A second notable property of our fitting procedure (that arises out of the first one) is that our least squares fitting procedure minimizes the cumulative parametric distance error only *for a given parameterization of the spring mesh*. What this translates to is the fact that once we fix a parameterization of the spring points, our fitting procedure minimizes the parametric distance error. However, it does not guarantee that there is no other parameterization of the spring points which could result in a smaller parametric distance error.

This fitting methodology (for fitting to regular gridded data) is used widely used in the literature [Forsey & Bartels 1991]. We have found it to work satisfactorily in practice. It is an efficient operation and produces B-spline surfaces that closely follow both the geometry and the parameterization of the underlying spring mesh.

If one wanted to attempt a solution that did indeed minimize the normal distance error, one possible method could be to refine the parameter values of the spring points using an iterative strategy much like the one described for point clouds (figure 4.2). The iteration would alternate between re-parameterizing our spring points based on an intermediate B-spline fit and re-fitting the B-spline surface based on the re-parameterized spring points. Note that since the spring mesh is a regular grid and since we have chosen to use gridded data fitting, each spring point along a u (or v) iso-curve must be assigned the same u value (or v) value. Therefore an iteration that re-assigns parameter values to spring points must assign the same iso-parametric value for entire iso-curves rather than just individual spring points. In the context of our pipeline, any such fitting strategy would have to address the problem of following the feature curves (and in general preserving the parameterization of the spring mesh).

6.3 Results

In summary, we fit B-spline surfaces directly to the spring mesh generated by our parameterization strategy. This fitting method has three main advantages.

- First, since the spring mesh is a regular grid, the least squares fitting process is both rapid and numerically robust even for large spring meshes.

- Second, since the spring mesh accurately re-samples the original polygonal patch, the B-spline surfaces produced by our fitting procedure are of a high quality.

- Third, because of the speed of the gridded data fitting process, the user can ask to

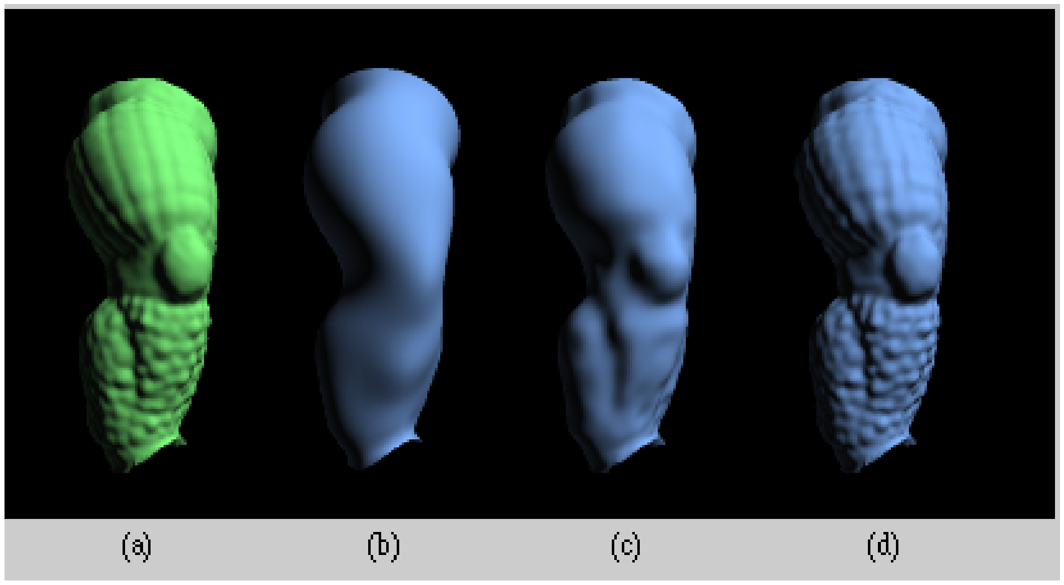

Figure 6.2. The figure shows three different resolutions of B-spline surfaces that approximate the Armadillo's leg. (a) shows the highest resolution spring mesh that is the input to our gridded data fitting step. (b), (c) and (d) show successively finer resolutions of B-spline surfaces approximating this spring mesh. The B-spline surface in (b) has 12×14 control vertices. (c) has 24×30 control vertices and (d) has 65×100 control vertices. All these approximations took under a second to compute on a MIPS 250 Mhz R4400 processor. The speed of gridded data fitting makes it useful in a interactive setting. Note that while the B-spline patch in (d) improves on capturing fine geometric detail compared to (c), it does so at the expense of a considerable increase in control vertex density. This drastic increase in the resolution of the B-spline surface is due to the underlying smoothness of the B-spline basis functions rather than any inefficiency in our fitting method. Note that any other smooth basis functions that shares the B-spline basis's smoothness properties will produce similar results. In the next chapter we discuss a more flexible method for representing fine surface detail as compared to a dense smooth surface.

change the resolution of the fitted B-spline surface and the system responds at interactive rates with the fitted results.

Figure 6.2 demonstrates these three advantages of our gridded data fitting procedure with an example. As this example shows, we may use our B-spline surface fitting techniques to capture both the coarse and fine geometry of the underlying polygonal surface (e.g. with the spline surface shown in (d)). However, as discussed earlier we represent the fine detail in our models as displacement maps. In the next chapter we explain how displacement maps can be computed from our polygonal models.

Chapter 7

Displacement maps

Using the techniques described in previous chapters, it is possible to capture the entire geometry of a polygonal model in the form of B-spline surfaces. However, we choose to separate the representation of coarse geometry and fine surface detail. We fit our polygonal models with a hybrid representation of B-spline surface patches and associated displacement maps. Our hybrid representation has several desirable properties.

The fine geometric detail in a model is usually of little significance during the modeling and animation of the coarse geometry of the model. Rather its principal use is in enhancing the appearance of the model at rendering time. Furthermore, its presence at the modeling (or animation) stage of an application may degrade the time or memory performance of an interactive system.

For example, figure 6.2 (from the previous chapter) shows the example of three different resolutions of B-spline surface that have been fit to a patch from the Armadillo's leg. The B-spline surface patch shown in figure 6.2c captures just the coarse geometry of the original polygonal patch while the B-spline surface shown in figure 6.2d captures the fine surface detail as well. While the denser patch improves on capturing the fine geometric detail in the original mesh, it does so at the expense of a considerable increase in control vertex density. This makes it unwieldy to manipulate and expensive to maintain within an interactive application. In general, a dense smooth surface representation that includes fine geometric detail is of little use for most interactive modeling or animation applications. However, it is desirable to retain the fine geometric detail in our original data so that it may be used at rendering time to enhance the appearance of the model.

Compared to a dense smooth surface, our hybrid representation of smooth surface and displacement map allows a user to effectively separate coarse geometry (which can be used for interactive modeling and animation applications), from fine surface detail (which can be used to enhance surface appearance at rendering time). Our proposed separation of geometry into smooth surfaces and displacement maps can be used with arbitrary smooth surface representations.

Furthermore, since our displacement maps are essentially images, they can be processed, re-touched, compressed, and otherwise manipulated using simple image processing tools. This is a flexible and intuitive paradigm for manipulating fine geometric detail compared to the cumbersome interface offered by a smooth surface representation for this purpose. For example, to edit a bump or a crease on the Armadillo's leg with a B-spline surface, a user would have to first identify the control vertices that controlled that particular bump or crease then manipulate it by pulling and pushing control vertices in 3-space. In contrast, with a displacement map image the user can edit surface detail with any standard 2-D image painting tool. We demonstrate this flexibility of our hybrid representation through several examples in section 7.6. Some of these effects were achieved using Adobe Photoshop, a commercial photo re-touching program.

A variant of our approach to solving the inherent limitations of smooth surfaces for representing surface detail might be to use hierarchical basis functions such as hierarchical splines [Forsey & Bartels 1988] or wavelet based splines [Gortler & Cohen 1995]. While these bases offer a more compact means of representing complex smooth surfaces than uniform B-splines, the interface used to manipulate surface detail is cumbersome i.e. the manipulation of control vertices in three space. Therefore, while hierarchical smooth surface representations are reasonable alternatives to a B-spline surface representation for representing coarse geometry, they suffer from many of the same disadvantages that a uniform B-spline basis does for representing and manipulating fine geometric detail. We chose the B-spline basis function as our representation for coarse geometry over other parametric (including the above-mentioned hierarchical) representations because of its popularity in the CAD and animation industries.

The remainder of this chapter is organized as follows. We begin, in section 7.1 by defining a general mathematical formulation of a displacement map. Then in section 7.2 we discuss how a displacement map is computed based on our input data and approximating B-spline surface. We compute our displacement maps as vector functions and store them as color images. While vector displacement maps are useful for a variety of operations, some operations such as (displacement image) painting are more intuitive on gray-scale images. In section 7.3 we discuss some methods of extracting a gray-scale image from our displacement maps. For some applications bump maps are preferable to displacement maps for representing coarse geometry. These are functions that change the appearance of a surface without changing its geometry. In section 7.4 we discuss how bump maps can be computed from our input data. Then in section 7.5 we explain one limitation of our least squares surface fitting strategy as it relates to displacement map computation. Finally, we end this chapter (section 7.6) with some results.

7.1 Definitions

A displacement map on a surface is a function, usually defined on the domain of definition of the surface, that perturbs the position of each point on that surface [Cook 1984]. Displacement maps are widely used at rendering time to add geometrically interesting detail to smooth surfaces. They are available in several commercial renderers such as Pixar's Renderman system and Softimage's MentalRay renderer.

A displacement map can either be a scalar function or a vector function. In our discussions we use both formulations. We use the following formulation for a vector displacement map on a bivariate parametric surface $\mathbf{F}$ ($\mathbf{F} : \Re^2 \rightarrow \Re^3$):

A vector displacement map is a function $\mathbf{D} : \Re^2 \rightarrow \Re^3$ that, given an arbitrary point (u_0, v_0) in parameter space, returns a vector perturbation $\mathbf{D}(u_0, v_0)$ (or *displacement*) of the surface point $F(u_0, v_0)$.

With this definition, the displacement mapped surface is given by new function

$\mathbf{F}_{disp} : \Re^2 \to \Re^3$ such that:

$$\mathbf{F}_{disp}(u, v) = \mathbf{F}(u, v) + \mathbf{D}(u, v)$$

for every (u, v) in parameter space. Note that since there is no restriction on the directions of the vector displacements, the surface reconstructed by a vector displacement map over a planar surface is not constrained to be a height field. It may even self-intersect.

Scalar displacement maps are usually interpreted as being along the surface normal. A scalar displacement map is a function $D : \Re^2 \to \Re$ that, given an arbitrary point (u_0, v_0) in parameter space, returns a scalar displacement $D(u_0, v_0)$ of the surface point $F(u_0, v_0)$ along the normal to surface at (u_0, v_0). The displacement mapped surface $\mathbf{F}_{disp}$ is given by:

$$\mathbf{F}_{disp}(u, v) = \mathbf{F}(u, v) + D(u, v)\hat{\mathbf{N}}(u, v)$$

where $\hat{\mathbf{N}}(u, v)$ represents the surface normal at $F(u, v)$. In contrast to vector displacement maps, scalar displacement maps over a planar surface are constrained to be height fields. Therefore the geometry that can be modeled with scalar displacement maps is inherently limited.

In our system, the obvious displacement function relates points on the fitted spline surface to points on triangles of the original polygon mesh. However, computing such a function (whether scalar or vector) requires computing the closest point (or a perpendicular projection) on the unparameterized polygon mesh from a point on the spline surface. This is an expensive operation. Furthermore, our fitting procedure is premised on the assumption that the spring mesh is a faithful representation of the original polygon mesh. Therefore, we find it sufficient to define a displacement function that relates points on the spline surface to points on the parameterized spring mesh surface.

7.2 Vector displacement maps

Even given the simplification of computing displacement maps based on the parameterized spring mesh, computing a scalar displacement maps is error prone. This is because the computation involves using a perpendicular projections from the spline surface to the spring mesh. This computation can fail catastrophically if the spring mesh curves sharply away from the surface. Furthermore, a scalar displacement map by its very definition cannot represent a displacement map that is not a height field over the surface. Therefore, if the displacement map is not a height field, perpendicular projection will always result in an incorrect displacement map i.e. one that leads to an incorrect reconstruction of the surface detail. Figure 7.1 explains the problems associated with straightforward perpendicular projection. We can avoid the problems with perpendicular projection simply by defining displacements as offsets to vertices of the spring mesh from corresponding points on the spline surface. Recall that there is a natural association of the spring mesh points to parameter values: these are the same parameter values that were used for the surface fitting step. Note that the offset from a point on the spline surface to the corresponding point on the spring mesh is in general a vector. Computing one such (displacement) vector offset for each spring point gives us a regular grid of displacement vectors at the resolution of the spring mesh.

We represent each of these displacement vectors in the local coordinate frame of the spline surface i.e. we store the displacement offsets relative to the underlying spline surface. For applications that modify the underlying B-spline surface (such as animation), this choice of coordinate frames allows the displacements to move and deform relative to the surface. In general, this representation is useful for any application that stores displacement map independent of underlying surface topology (see for example figure 7.5).

Since our B-spline basis functions are continuously differentiable (i.e. C^2) so is our underlying fitted B-spline surface. Therefore, a local coordinate frame for the spline surface is easily computed by evaluating at each (u, v) position, the two surface partials $\mathbf{U}$ and $\mathbf{V}$ and the surface normal $\mathbf{N}$. This frame varies for every point (u, v) in parameter space and is given by:

$$\mathbf{U}(u,v) = \frac{\partial \mathbf{F}(u,v)}{\partial u}$$
$$\mathbf{V}(u,v) = \frac{\partial \mathbf{F}(u,v)}{\partial v}$$
$$\mathbf{N}(u,v) = \mathbf{U} \times \mathbf{V}$$

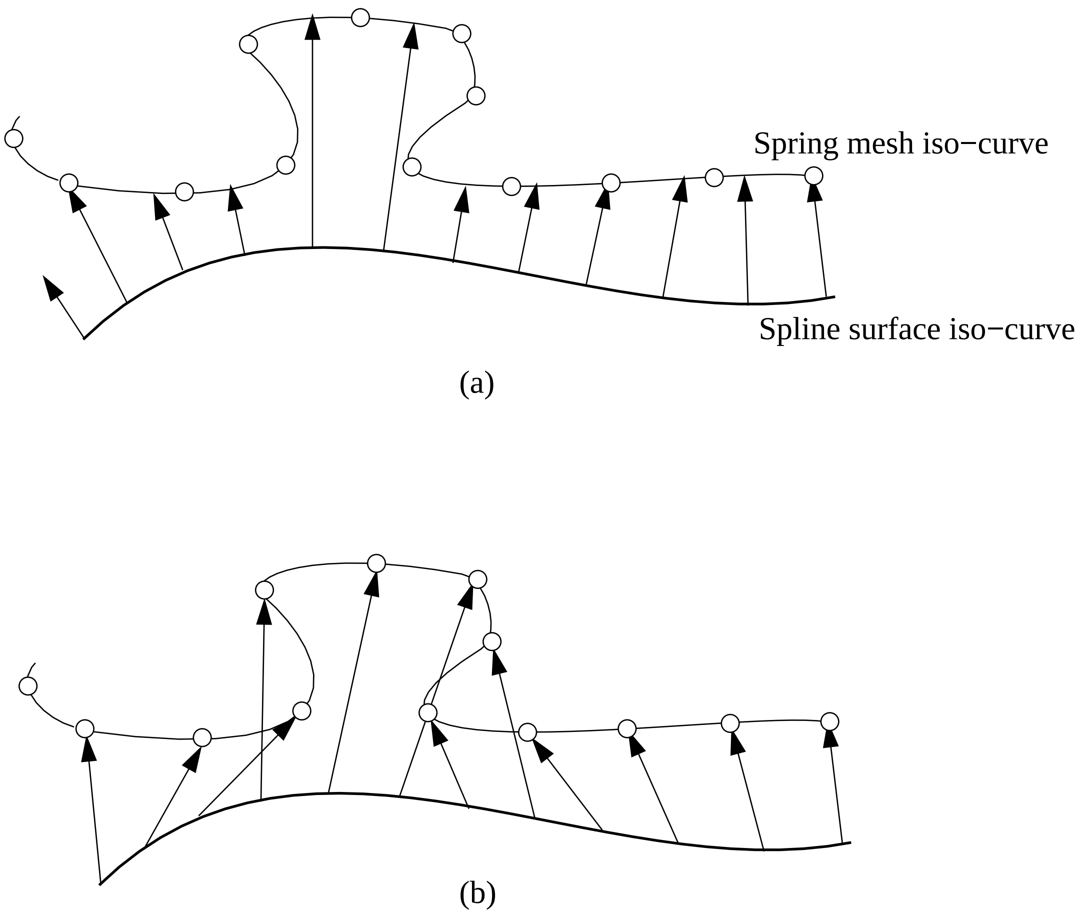

Figure 7.1. Scalar and vector displacement maps. (a) shows the computation of a scalar displacement map on an iso-curve of the spline surface. The computation involves perpendicular projection from a set of points on the spline surface to the spring mesh. Note that in general, this is a non-robust computation. For example the leftmost normal vector from the spline surface does not intersect the spring mesh surface. More importantly, note that a reconstruction of the displacement mapped spline surface based on the displacement vectors will always be incorrect. Specifically, the reconstruction misses the “overhangs” from the bump in the middle of the spring mesh. This occurs because scalar displacement maps can correctly reconstruct only height fields and the spring mesh shown is not a height field over the spline surface. (b) shows our computation of a vector displacement map on the same set of iso-curves. Note that the displacement vectors are all well-defined and correctly reconstruct the original spring mesh.

Given this coordinate frame, the (vector) displacement $\mathbf{D}$ at a point $P(u, v)$ is given by:

$$\mathbf{D} = (\mathbf{D} \cdot \mathbf{U})\mathbf{U} + (\mathbf{D} \cdot \mathbf{V})\mathbf{V} + (\mathbf{D} \cdot \mathbf{N})\mathbf{N} \tag{7.1}$$

From an implementation standpoint care needs to be taken when evaluating the local frame of reference at singular points of the surface (e.g. when all the control vertices of a boundary curve are coincident). We refer the reader to any standard book on parametric surfaces for further details on the robust computation of surface partials (for example see the killer-B's book [Bartels et al. 1987]).

Since our vector displacements, as computed, are a regular grid of 3-vectors, they can conveniently be represented in the form of an rgb image. Our displacement map $\mathbf{D}(u, v)$ is therefore given by a reconstruction from this regular grid of displacement vectors. As such, any standard image reconstruction filter suffices for this purpose. We have used a bilinear filter reconstruction for the examples shown in this thesis.

Note that the displacement map as computed here is essentially a re-sampled error function since it represents the difference of the positions of the spring points from the spline surface. In the terminology of chapter 6, our displacement map is a reconstruction of the parametric distance error field.

Our image representation of displacement maps permits the use of a variety of image-processing operations such as painting, compression, scaling and compositing to manipulate fine surface detail. Figure 7.4 and figure 7.3 explore these and other games one can play with displacement maps.

7.3 Scalar displacement maps

While vector displacement maps are useful for a variety of operations, some operations such as (displacement image) painting are more intuitive on gray-scale images. There are several methods of arriving at a scalar displacement image. One method to arrive at this scalar image might be to compute a normal offset from the spline surface to the spring mesh. However, as discussed earlier, this method is both expensive and prone to non-robustness in the presence of high curvature in the spring mesh (see figure 7.1).

Instead we have used two alternative formulations. The first computes and stores at each sample location (or pixel) the magnitude of the corresponding vector displacement. In this case, modifying the scalar image scales the associated vector displacements along their existing directions. A second alternative, stores at each sample location the component of the displacement vector normal to the spline surface (i.e. $\mathbf{D} \cdot \mathbf{N}$). Modifying the scalar image therfore changes only the normal component of the vector displacement. Each of these representations offers a different interaction with the displacement map. Figure 7.4 employs normal component editing.

7.4 Bump maps

A *bump map* is defined as a function that performs perturbations on the direction of the surface normal before using it in lighting calculations [Blinn 1978]. In general, a bump map is less expensive to render than a displacement map since it does not change the geometry (and occlusion properties) within a scene but instead changes only the shading. Bump maps can achieve visual effects similar to displacement maps except in situations where the cues provided by displaced geometry become evident such as along silhouette edges. We compute bump maps using techniques very similar to those used for computing displacement maps: at each sample location instead of storing the displacement we store the normal to the corresponding spring mesh point. Thus the bump map is simply expressed as:

$$\mathbf{N}_{spline}(u, v) = \mathbf{N}_{spring}(u, v)$$

As before $\mathbf{N}_{spring}$ is stored in the local coordinate frame of the spline surface. Since the bump map, as computed, is also a regular grid of 3-vectors it can be stored as an rgb image as well. Figures 7.3d and e show a comparison of a displacement mapped spline and a bump mapped spline, both of which are based on the same underlying spring mesh. Notice how, differences are visible at the silhouette edges. Another limitation of bump maps is that while they can fake fine surface detail fairly convincingly, it is much harder to represent coarse geometry with them.

Note that an alternate method to obtain the bump map is to compute it from the displacement map image. The original bump map formulation proposed by Blinn [Blinn 1978] describes this approach. This is equivalent to our method because the displacement mapped spline surface is just the spring mesh. Therefore a bump map that is extracted from the displacement map would yield the same result as one that is extracted from our spring mesh.

7.5 A limitation of least squares fitting for displacement map computation

Within our least squares fitting process the user can choose just the resolution of the approximating B-spline surface. The fitting algorithm then automatically determines the "best possible" B-spline approximation to the spring mesh, given that resolution. The displacement map is automatically computed as a difference of the spring mesh and spline surface. This fitting strategy has certain potential drawbacks. Consider for example, the simple 1-D curve shown in figure 7.2. Our method results in the solution depicted in figure 7.2a. Note that the least squares surface passes through the bumps on the curve i.e. the smooth surface encodes part of what appears to be fine geometric detail. As a result, the displacement map itself encodes what appears to be part of the coarse geometry of the surface. Figure 7.2b shows a more intuitive result: the bumps on the spring mesh are completely offset as displacements above the fitted curve. Our least squares solution cannot produce the result shown in 7.2b. The reason for this is that a straightforward least squares solution does not have the mathematical machinery to discriminate between fine detail and coarse geometry. Note that this is a limitation of the least squares fitting process rather than a limitation of our hybrid surface representation.

A more desirable alternative to least squares fitting might be a strategy that is based on a frequency domain decomposition of the highest resolution spring mesh. The separation of geometric detail from coarse geometry could be based on this decomposition. It is worth noting that it is not straightforward to encode a users intuition (or needs) for the separation of geometric detail and coarse geometry. Therefore, any alternative fitting strategy

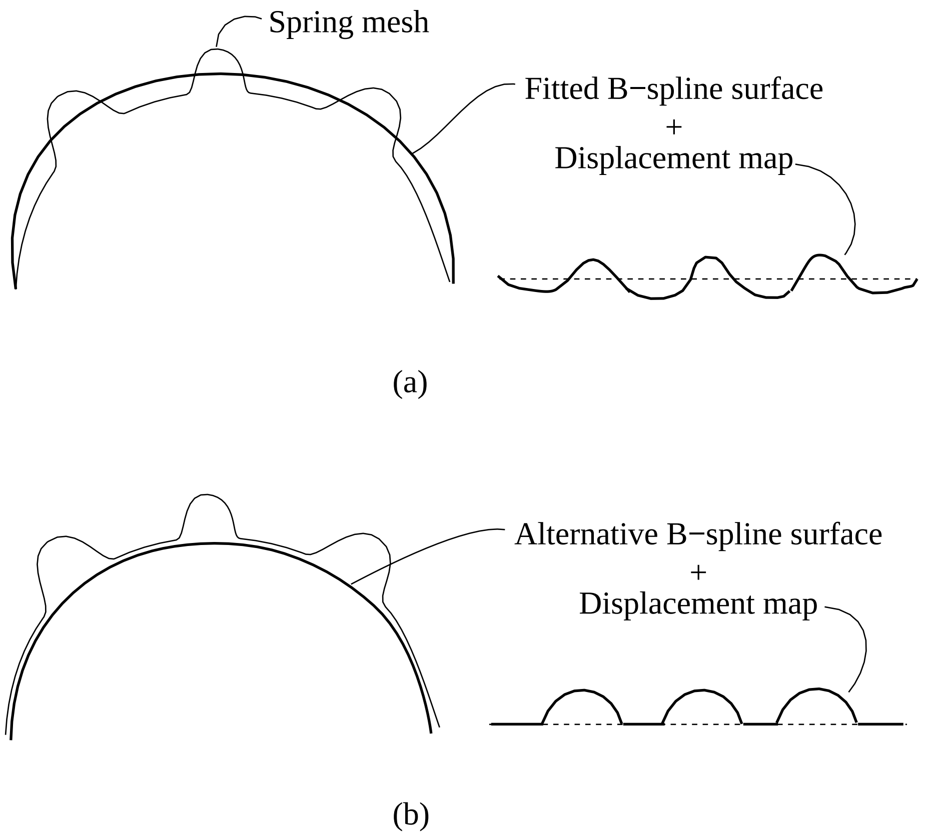

Figure 7.2. A limitation of our surface fitting strategy. (a) shows a spring mesh iso-curve and a least squares fitted spline surface using our fitting strategy. Notice that the least squares surfaces passes through the bumps on the spring mesh i.e. it encodes in the smooth surface what intuitively appears to be surface detail. The resulting displacement map is shown alongside. Note that it encodes part of what appears to be the coarse geometry of the spring mesh. (b) shows the spline surface and displacement map that a user might intuitively expect to obtain from this spring mesh. A fitting strategy that encodes this intuition as part of the surface fitting process would be a desirable extension of our approach.

should supply the user with tools to efficiently differentiate fine surface detail from coarse geometry.

7.6 Results

To summarize: displacement maps are computed as the (parametric) error between the spline surface and spring mesh re-sampled into a regular grid. Displacement maps provide a useful method for storing and manipulating fine surface detail in the form of displacement map images. We have found that this representation empowers users to manipulate geometry using tools outside our modeling system. The following pages show several games one can play with displacement maps. These games demonstrate the flexibility of our hybrid representation. Most would have been difficult (i.e. manually intensive) to duplicate with a pure smooth surface representation such as a B-spline surface.

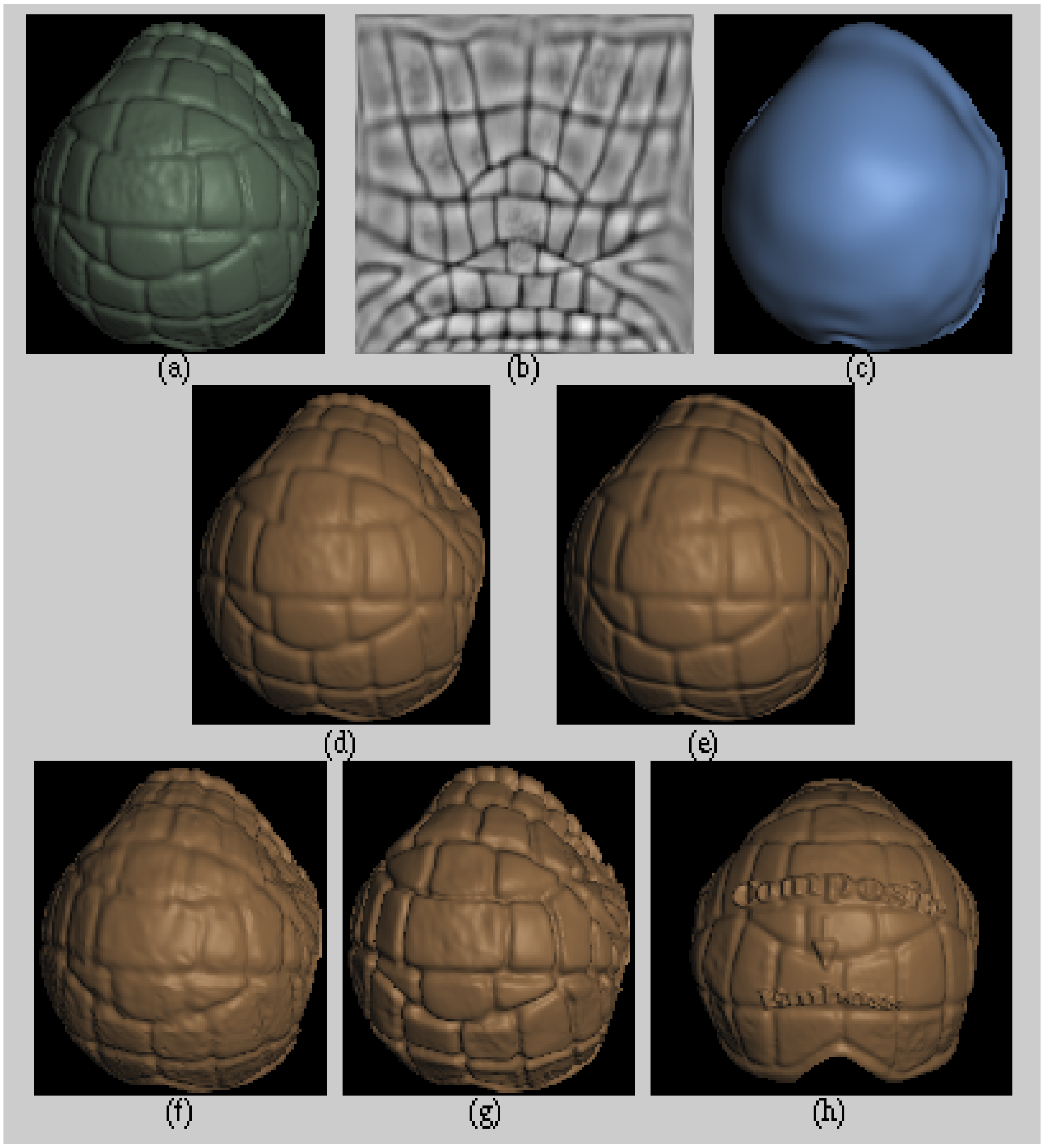

Figure 7.3. (a) shows a patch from the back of the Armadillo model. The patch has over 25,000 vertices. We obtained a spline fit (in 30 seconds) with a 15x20 control mesh, shown in (b) and a corresponding vector displacement map. The normal component of the vector displacement map, is displayed as a gray-scale image in (c). (d) and (e) show the corresponding displacement and bump mapped spline surface. The differences between (d) and (e) are evident at the silhouette edges. The second and third row of images show a selection of image processing games on the displacement map. (f) shows jpeg compression of the displacement image to a factor of 20 and (g) represents a scaling of the displacement image, to enhance bumps. (h) demonstrates a compositing operation, where an image with some words was alpha composited with the displacement map. The result is an embossed effect for the lettering.

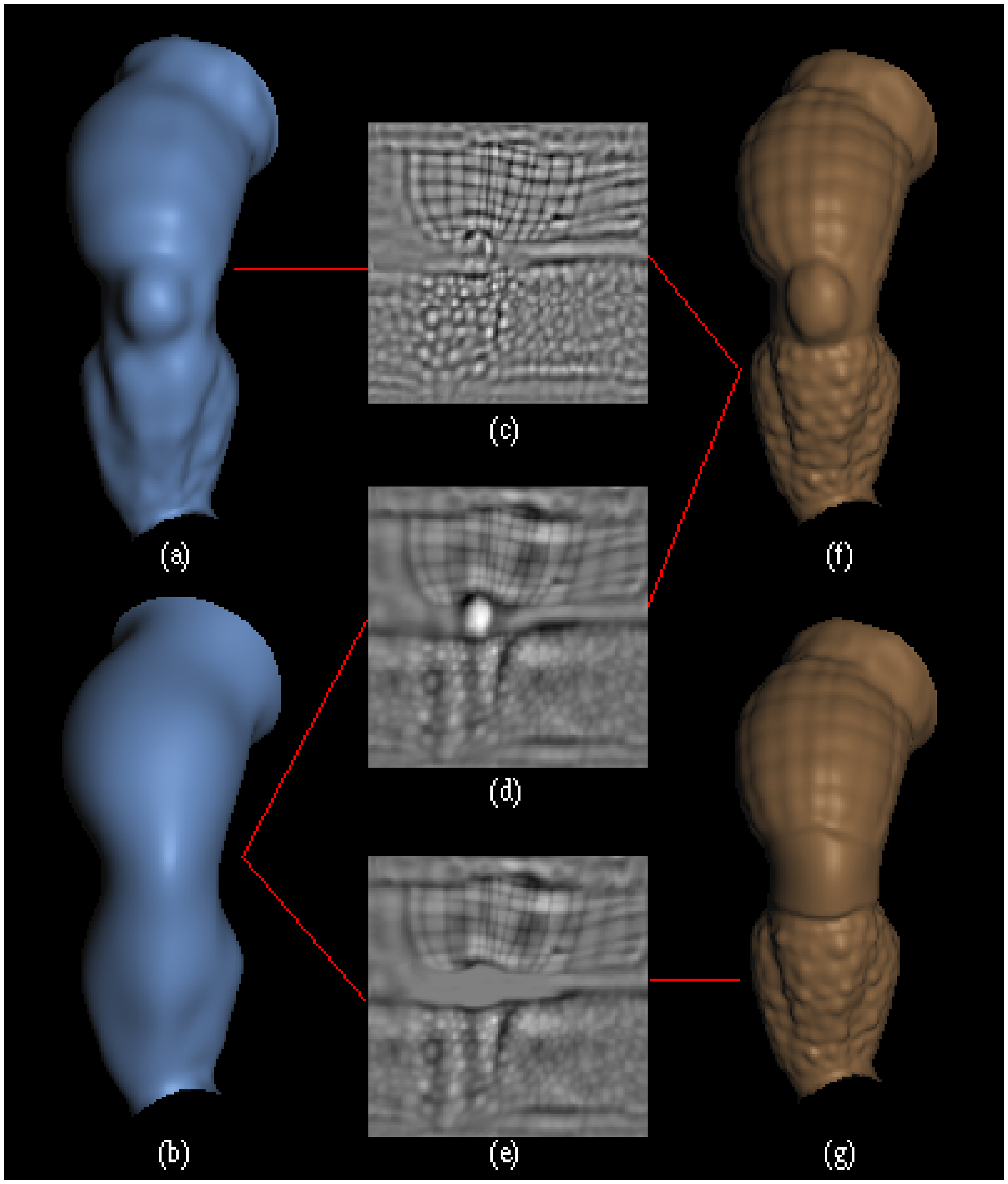

Figure 7.4. This figure explores the possibility of multi-resolution editing of geometry using multiple displacement map images. All gray-scale displacement images in this figure represent the normal component of their corresponding displacement maps. Displacement values are scaled such that a white pixel represents the maximum displacement and black, the minimum displacement. (a) shows a B-spline surface with 24x30 control points that has been fit to the leg patch from figure 5.6. (c) is its corresponding displacement image. (b) shows a B-spline surface with 12x14 control points that was also fit to the same patch. Its displacement image is shown in (d). The combination of spline and displacement map in both cases reconstructs the same surface (i.e. the final spring mesh shown in figure 5.6e). This surface is shown in (f). We observe that (c) and (d) encode different frequencies in the original mesh. For example (d) encodes a lot of the coarse geometry of the leg as part of the displacement image (for example the knee), while (c) encodes only the fine geometric detail, such as bumps and creases. As such, the two images allow editing of geometry at different scales. For example, one can edit the geometry of the knee using a simple paint program on (d). In this case, the resulting edited displacement map is shown in (e) and the result of applying this image to the spline of (b) gives us an armor plated knee that is shown in (g). Operations such as these lead us to the issue of whether multiple levels of displacement map can essentially provide a image filter bank for geometry i.e. an alternative multi-resolution surface representation based on images. Note however that the images from (c) and (d) are offsets from different surfaces and the displacements are in different directions, so they cannot be combined using simple arithmetic operations.

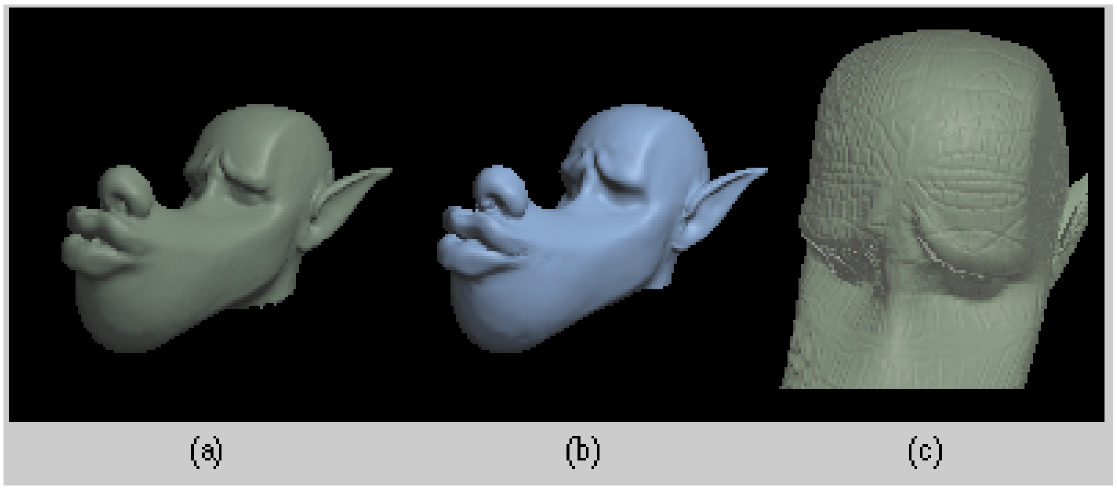

Figure 7.5. Transferring displacement maps across objects. (a) is a polygonal model of a wolf's head (under 60,000 polygons). It was fit with 54 spline patches in under 4 minutes (on a 250 Mhz MIPS R4400 processor). The splined model is shown in (b). (c) shows a close up view of a partially splined result, where we have mapped the displacement map from figure 7.3(b) onto each of 4 spline patches around the eyes of the model.

Chapter 8

Inter patch continuity

Thus far we have addressed the sampling and fitting issues connected with a single polygonal patch. Since our approximating B-spline surfaces can only model manifold deformations of a four sided planar region, polygonal meshes of arbitrary topology cannot in general be fit with just a single patch. Therefore, in our system the user specifies a network of polygonal patches that tile the surface. The techniques of the preceeding chapters are then used to fit our hybrid surface representation of B-spline and displacement map to each of these polygonal patches. By virtue of the properties of a cubic B-spline basis function individual B-spline surfaces are curvature continuous (i.e. C^2) in the interior of the patch. However we are not guaranteed this property at the shared boundaries and corners of multiple patches. We are therefore faced with the problem of keeping our patches geometrically continuous at their shared boundaries and corners. If displacement maps are used, it is desirable to keep the displacement mapped spline surfaces continuous as well. Accordingly, in this chapter we divide our discussion of inter-patch continuity into two parts. First, in section 8.1, we examine the continuity issues for B-spline surfaces. In this regards we examine two widely used continuity standards in industry: visual (or statistical) continuity and mathematical continuity (sections 8.1.1 and 8.1.2 respectively). Our surface fitting system can be adapted to either of the above paradigms. Our focus in this thesis has been on entertainment (i.e. animation) applications. In section 8.1.3. we propose a continuity solution for our system that is suited for such applications. Then, in section 8.2 we propose a method for making displacement mapped B-spline surfaces continuous. Finally, we show some fitted and stitched examples to demonstrate our solutions.

8.1 Continuity of a network of B-spline surfaces

In our system the number and placement of patches are user controlled parameters. Since the user is not constrained to place the patches in any fixed juxtaposition we cannot make simplifying assumptions about patch topology. In particular, we cannot make simplifying assumptions about the number of boundary curves that meet at a patch corner.

It is interesting to note that despite the fact that B-spline surfaces are the surface representation of choice in the CAD and the animation industries very little research has addressed the issue of achieving G^1 continuity over an arbitrary B-spline surface network [Milroy et al. 1995]. A fair amount of research in the parametric interpolation literature [Lounsbery et al. 1992] has been devoted to analyzing the necessary and sufficient conditions for continuity of a number of other (mathematically more compact) surface formulations. Examples include rectangular Bezier patches [DeRose 1990], Catmull Rom patches [DeRose & Barsky 1988], triangular Bezier patches [Farin 1982, Piper 1987], combinations of rectangular and triangular Bezier patches [Shirman & Séquin 1991], Gregory patches [Shirman & Séquin 1990], quintic Hermite surfaces [Moreton & Séquin 1992], etc. Recently Eck et al [Eck & Hoppe 1996] proposed a surface spline formulation within the context of surface fitting. In this case the individual patches in the network were bicubic Bezier patches.

Unfortunately, none of the above referenced methods address the issue of how to satisfactorily deal with the admittedly harder problem of obtaining continuity within a network of B-spline surface patches. To achieve the desired level of continuity between our surface patches, we must first examine application specific requirements for continuity. There are two main kinds of inter-patch continuity that are commonly used in entertainment and CAD applications: statistical (or visual or numerical) continuity and parametric (or geometric) continuity. We examine each of these in turn.

8.1.1 Statistical continuity

The existence of statistical continuity between two patches implies that the patches are continuous only to the extent that the position and normal deviations of the surfaces at their shared patch boundaries meet certain user specified tolerances. Since this is not a precise mathematical notion of continuity it is often referred to as statistical or visual continuity. Figure 8.1 illustrates the notion of statistical continuity at a patch boundary.

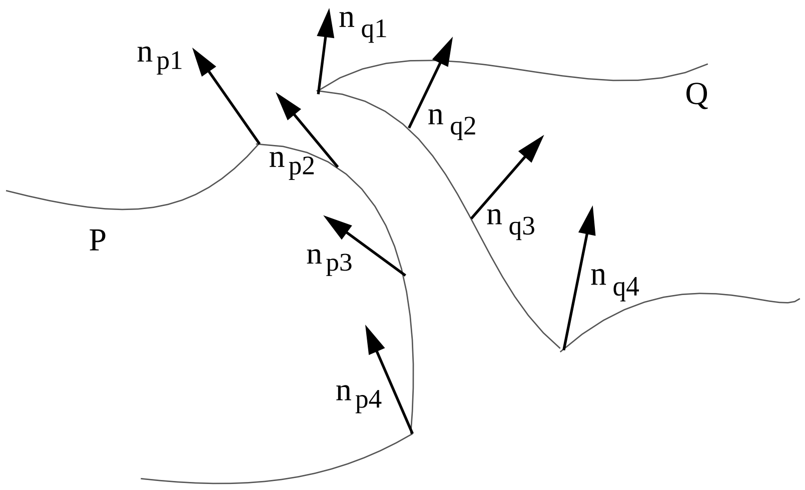

Figure 8.1. Visual continuity. The figure shows two patches P and Q with a shared boundary. Visual continuity is concerned with a statistical measure of what it means for two patches to be continuous at a shared boundary. Quantities that are typically compared are positions and normals at a set of points on shared boundary curves. One measure of tangential continuity of the two patches shown here is the difference between the normal vectors along the shared boundary. If the tangents along the boundary curves are comparable then the normals to the surface along the boundary curves must be comparable as well. A quantitative measure of tangent continuity is therefore given by: $\sum_{i=1}^{4} \parallel \mathbf{n}_{pi} - \mathbf{n}_{qi} \parallel^2$ The lower this number the more tangentially continuous the two surfaces will be at their shared boundary from a statistical perspective.

For many applications this kind of continuity is adequate. Take an example of the construction of the exterior of a car body. In this application curvature plots and reflection lines [Farin 1990] are frequently used to verify the “quality” of surfaces. In this context, even mathematically curvature continuous (C^2) surfaces might be inadequate. Furthermore, workers in the automotive industry often use trimmed NURBS and do not necessarily match knot lines at shared patch boundaries during model construction. Therefore it is not always possible to enforce mathematical continuity across patch boundaries. Therefore, often only statistical or visual continuity makes sense in this application. In the CAD industry, statistical continuity is typically obtained in one of two ways:

- Manually: by constructing trimmed surface patches that share overlapping regions at their shared boundaries.
- Through an iterative numerical optimization process [Milroy et al. 1995, Bardis & Vafiadou 1992, Sinha & Seneviratne 1993].

Recently, Milroy et al [Milroy et al. 1995] proposed a statistical stitching method to obtain visually continuous B-spline surface patches. While the results produced were better than those previously reported, the techniques outlined were computationally expensive and did not produce the surfaces of a high enough quality required of most CAD applications. Obtaining statistical continuity algorithmically (whether through numerical optimization or otherwise) remains an active area of research.

8.1.2 Mathematical continuity

In contrast to statistical continuity, geometric continuity between two surface patches implies that the patches are continuous in a precise, mathematical sense. For example C^0 continuity between two patches means that the patches are (mathematically) continuous in surface position at their shared boundary. C^1 continuity implies that the patches are both position continuous and possess a smoothly varying tangent field at their shared boundary. In general C^n continuity implies that the patches have a mathematically continuous n-th (parametric) derivative at their shared boundary. A similar but less stringent continuity requirement that is popular in the animation industry is G^1 continuity. This is a weaker form of the C^1 requirement. It requires only the direction of the tangent field to be mathematically continuous across patch boundaries rather than the actual tangent field (i.e. magnitude and direction). For more details on the subject of mathematical continuity we refer the reader to any standard text book on computer aided geometric design (e.g. see Farin's book on the subject [Farin 1990]).

A domain where statistical continuity is not acceptable and mathematical continuity is preferred is the animation industry. Typically G^1 continuity suffices for most applications in this domain. To achieve this level of continuity, the number of knots at shared patch boundaries are usually forced to be the same. This strategy has several advantages for the applications in this domain:

- Control point deformations are easily propagated across patch boundaries that are mathematically continuous.
- There is minimal distortion at boundaries in the process of texture mapping.
- The process of maintaining patch continuity during an animation becomes easier; continuity is either made part of the model definition [Ostby 1986] or is re-established on a frame by frame basis using a stitching post-process.

The strategy also has its limitations. Since adjacent patches are constrained to have the same knot resolution, regions with high surface detail that share boundaries with regions with relatively low surface detail must be carefully constructed. In particular care must be taken to ensure that there is no loss of surface detail in the high detail region and that there is not an excessive number of control vertices being used to represent the low detail region .

8.1.3 Our continuity solution

Our surface fitting system can be adapted to either of the above paradigms (i.e. either statistical or geometric continuity). It is worth noting that in our pipeline, since we are fitting to the spring mesh geometry, our B-spline surface patches typically have the same level of continuity, at a visual level, as the coarse geometry of the underlying spring meshes. These spring meshes in turn represent a faithful re-sampling of the underlying polygonal mesh (for example, see figure 5.15a). Therefore at a visual level, our B-spline surfaces tend to possess the same level of continuity as the coarse geometry of the underlying polygonal mesh.

However, our focus in this thesis has been on entertainment (i.e. animation) applications and visual continuity is not sufficient for many applications in the entertainment industry; mathematical continuity is preferred. Note that in the context of such applications, the splined model is usually deformed in the course of its use in the application. Therefore, our fitting process is merely a pre-processing step to obtain an animateable B-spline model. The original polygonal mesh and spring meshes no longer have a role in the subsequent deformations of the model. For these sorts of applications continuity solutions should work at interactive speeds (or should exist as part of the model definition) since potentially, they must be applied at every frame of an animation. As mentioned above, the applications we are focusing on typically require at most G^1 mathematical continuity. Based on these considerations, in our system mathematical continuity is enforced through a stitching post-process.

Specifically, we allow an animator to specify the level of continuity required at each boundary curve. The two options we have allowed are C^0 and G^1 continuity at boundaries. Our unconstrained, gridded data fitting to each patch leaves us with C^{-1} (i.e. discontinuous) B-spline boundaries. We use end-point interpolating, uniform, cubic B-splines for our basis functions.

For these surfaces the patch boundaries of B-spline surfaces are completely specified by just the boundary control vertices [Barsky 1982]. Therefore, to maintain C^0 continuity:

- Adjacent patches are constrained to have the same number of control points along a shared boundary.
- The shared boundary control vertices are made to have the same positions in space.

We ensure the second property by averaging just the boundary control vertices of adjacent patches and making these the new set of boundary control vertices of each of the patches. Figure 8.2a shows two adjacent patches and the associated control mesh for each patch. In this case the boundary control vertices are shown as solid circles.

Similarly, the tangent field at B-spline surface patch boundaries are completely specified by the boundary control vertices and the *tangent control vertices*. These are the vertices on the control mesh of the B-spline patch that are adjacent to the boundary control vertices. In figure 8.2a, the tangent control vertices are shown as hollow circles for two adjoining patches. The tangent field for a B-spline surface patch is identical to the tangent field for a ruled surface between two curves, each specified by the boundary control vertices and the tangent control vertices [Barsky 1982]. Therefore, G^1 continuity can be obtained by modifying the boundary control vertices such that each row of the tangent control vertices and the shared boundary control vertices of the two patches lie on a straight line, at equal spacings along that line. We accomplish this by modifying the position of the shared boundary control vertices such that the condition is satisfied. Both these operations are explained in figure 8.2a.

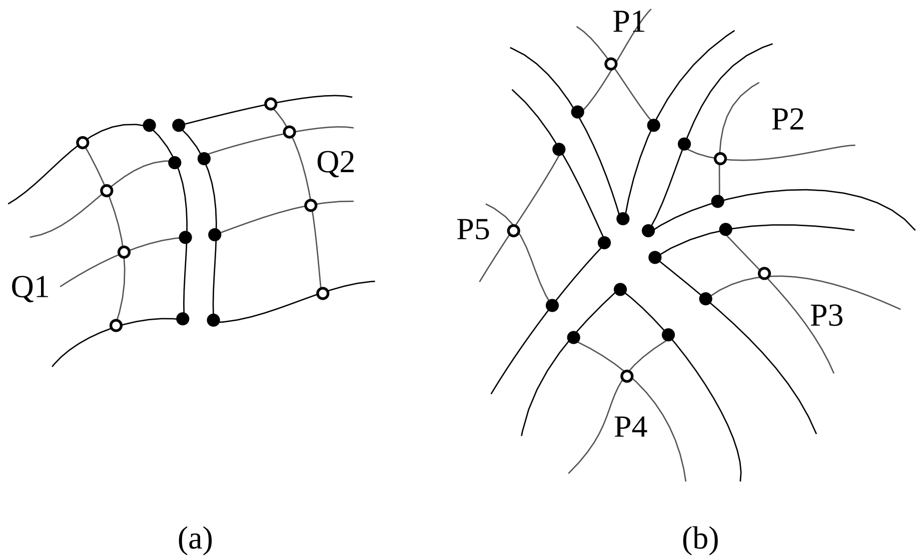

Figure 8.2. Geometric continuity at boundary curves and patch corners. (a) shows the control vertices for two adjacent patches that determine continuity at patch boundaries. The solid circles are called boundary control vertices while the hollow circles are called tangent control vertices. (b) shows the control vertices of a set of five patches that determine continuity at a shared patch corner. We call these the corner control vertices. Note that the solid circles are also boundary control vertices. The inner corner control vertex is shown as a hollow circle. This is the only interior control vertex whose position is modified by our continuity solution. See the text for details.

Obtaining G^1 continuity at patch corners poses a harder problem. Tackling the problem for different kinds of parametric surfaces has been the subject of much work in the literature [Peters 1990]. Figure 8.2b shows a set of patches that share a common patch corner. The control points that affect the tangent field at the patch corner are indicated in the figure. We call these the *corner control points*. For the kinds of basis functions we use, the tangent field at a patch corner of a B-spline surface is identical to the tangent field of a bi-linear surface formed by the four corner control vertices of the B-spline patch [Barsky 1982]. Therefore, we can obtain G^1 continuity at a patch corner, simply by forcing the 4 corner control points of each of the patches meeting at a patch corner to the same plane. In practice, we choose this plane to be an average of the the tangent planes at all the patch boundaries that meet at that patch corner. Note that this continuity solution potentially leaves our stitched surfaces looking overly flat at patch corners (since corner control vertices at shared corners are projected to a plane). However in practice this is not an issue since the section of the B-spline surface that is controlled by the four corner control vertices tends to be a very small surface element [Barsky 1982].

To summarize: our continuity solution moves just the boundary and corner control vertices to achieve the appropriate level of continuity over the entire network of B-spline patches. Note that the movement of these control vertices compromises our surface fit. However, since our surface fit merely recreates the coarse geometry of the underlying polygonal mesh faithfully, our stitched B-spline surfaces tend to retain the continuity and smoothness properties of the underlying model's coarse geometry. In particular, it retains this property both at the boundaries of patches as well as their interiors. Therefore in practice, our stitching solution results in a visually imperceptible change to our B-spline surfaces.

It is worth noting that an alternate method for achieving C^0 continuity of our patches, is to enforce a set of boundary constraints at the least squares fitting step. If we were able to guess a set of common boundary vertices for the two adjacent patches, this could be feasible alternative. The strategy could be extended for G^1 continuity if it was combined with an iteration that alternately established inter-patch continuity and re-fitted (with constraints) to the spring mesh data. However, we have chosen not to use this strategy; we have found that our strategy works satisfactorily in practice.

8.2 Displacement map continuity

For a network of B-spline patches, the displacement map computation is performed after the patches are stitched. Therefore, for displacement mapped splines continuity may be defined on the basis of the reconstruction filter used for the displacement maps. Recall that we generate these from the spring meshes and that we use bilinear reconstruction to display them. Displacement mapped B-spline surfaces will therefore exactly recreate the spring mesh. Note that adjacent patches can be at most position continuous with a bilinear reconstruction filter. Therefore, if the spring resolutions are the same at a shared boundary of two patches, they will be continuous by virtue of the reconstruction. However, spring mesh resolutions can differ across shared boundaries. This can result in T-joint type discontinuities in a displacement mapped B-spline surface. The problem is trivially solved by averaging the boundary rows of displacement map images of adjacent patches. This ensures that there is no cracking at the displacement mapped patch boundaries.

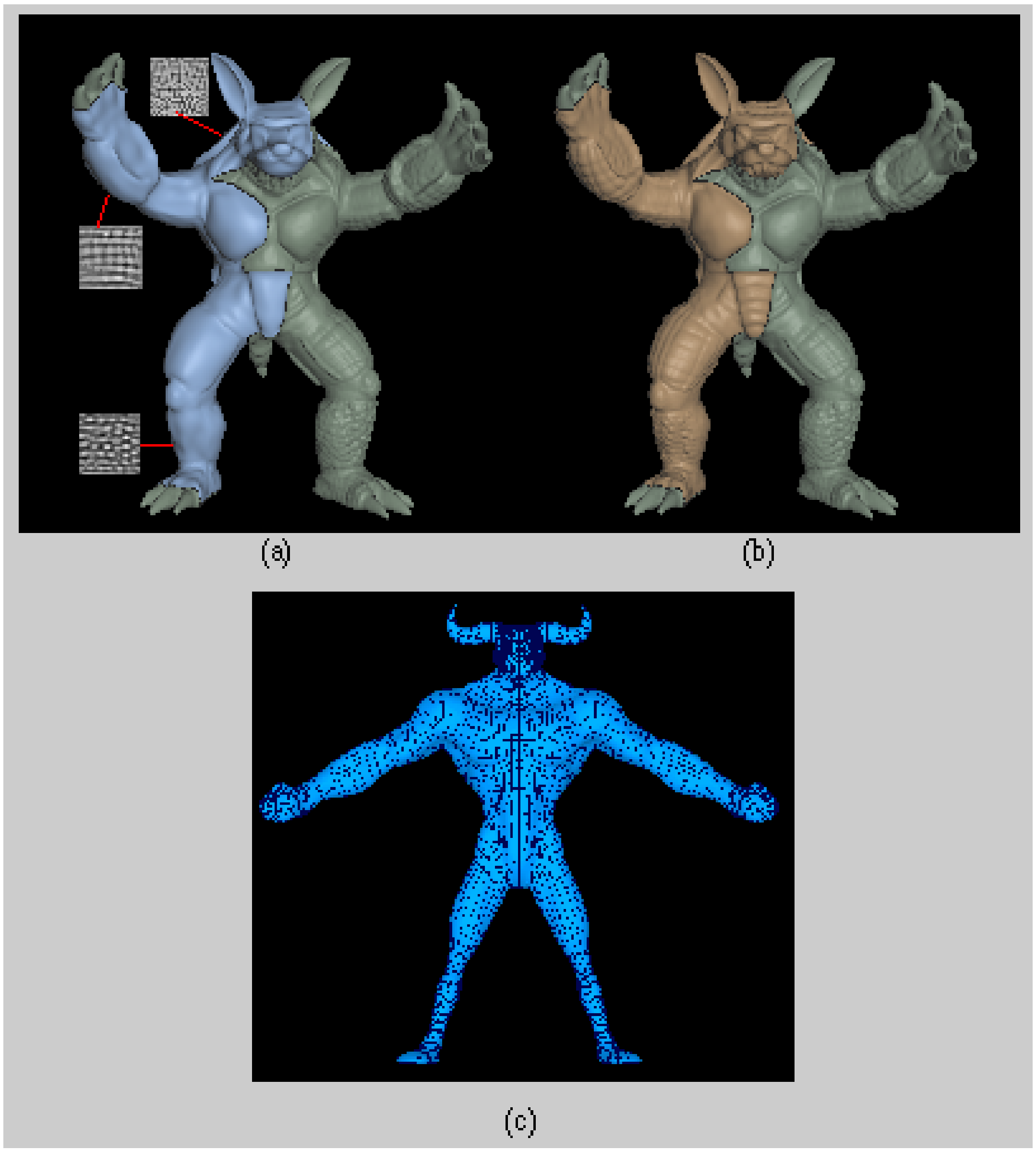

Figure 8.3. (a.) shows a split view of the Armadillo. The B-spline surfaces are shown smooth shaded on the left with the polygon mesh on the right. A few interesting displacement maps are shown alongside their corresponding patches. (b) shows a split view of the displacement mapped spline patches on the left with the polygon mesh on the right. Note that the fingers and toes of the model were not patched. This is because insufficient data was acquired in the crevices of those regions. This can be easily remedied by using extra scans or by using the hole filling techniques proposed in recent range integration techniques. The total number of patches on the Armadillo were 104 (only the left half have been shown here). The gridding stage took 8 minutes and the gridded fitting with 8x8 control meshes per patch, took under 10 seconds for the entire set of 104 patches. All timings are on a 250 MHz MIPS R4400 processor. (c) shows the stitched B-spline surfaces that were created from the Bofar model. The iso-curves of the fitted B-spline surface have been over-laid on the surface.

Chapter 9

Conclusion

In this thesis, we have presented techniques for fitting a combination of smooth parametric surfaces and displacement maps to dense, irregular polygon meshes. The algorithms we have proposed are robust, efficient and produce results that retain the fine geometric detail of the original polygonal meshes. Using these techniques, users have created some of the most complex and high fidelity smooth surface models ever created from scanned data. A selection of these models were used to demonstrate the techniques of this thesis.

A lot of exciting research has been conducted in the field of surface approximation over the last several decades [Franke & Schumaker 1986, Schumaker 1979]. Unfortunately, this research has focused mainly on surface fitting either to scattered point cloud data or to topologically simple data (e.g. height fields). While these techniques are widely applicable in a variety of problem domains, in the context of surface approximation to dense polygonal meshes of arbitrary topology they are either not applicable, inefficient or non robust (see chapter 2). A major reason for the shortcomings in related prior efforts is that they have tended to ignore the connectivity information present in dense polygonal meshes during the surface approximation process. We have used this additional connectivity information at various stages of our surface approximation process to develop algorithms that are significantly more robust and efficient than any prior efforts. In this chapter we summarize the principal contributions of this thesis (in section 9.1) and then discuss some exciting areas for future work (in section 9.2).

9.1 A summary of our principal contributions

9.1.1 Flexible surface curve editing tools

An important component of our surface fitting system is the user specification of boundary and feature curves. For most applications this is a non-automatable part of the creative process. As such, providing intuitive and efficient curve specification and manipulation tools is an important component of the surface fitting process. Providing such tools is a challenging task given the size of our data set and the fact that user might want to specify complex boundary curves.

We motivated the use of surface curves for our application rather than unconstrained space curves (such as 3-D B-spline curves). Surface curves offer a number of advantages for our application. Principal among these is that compared to space curves, surface curves offer a better intuition for the precise placement of a curve with respect to the surface.

Prior work on surface curves has focussed exclusively on formulations for curves on smooth surface representations. In this thesis we have presented a flexible formulation for surface curves on dense, unparameterized polygonal meshes: the surface snake formulation. Surface snakes are discretized controlled-continuity splines that are constrained to the polygonal surface. Our implementation is based on a coarse-to-fine computation of the energy measures of a surface snake. We showed that our implementation has an impulse complexity (i.e. the time taken for a change at one end of snake to propagate to the other end) of $O(N \log N)$ in the number of face-points in the curve. Compared to this, a static implementation of surface snakes would have an impulse complexity of $O(N^2)$.

Using the surface snake formulation we developed an efficient and intuitive interface for manipulating curves on dense polygon meshes. We also demonstrated the use of surface properties such as vertex color to assist in the curve painting and editing process. Such operations would not have been possible to duplicate using just space curve based editing.

9.1.2 Parameterizations of polygonal patches

An important component of any (parametric) surface fitting algorithm is the parameterization of the data set. A high quality parameterization is crucial for obtaining high fidelity surface approximations. In this thesis we have developed a robust, automated and customizable parameterization algorithm that is well suited for use in an interactive system. Our algorithm is based on a coarse-to-fine relaxation strategy that generates a minimum distortion surface grid called the spring mesh. The coarse-to-fine nature of our strategy allows a user to incrementally preview even large patch configurations. This is a useful property for an interactive system. In particular, it proves invaluable when modelers wish to experiment with different patch configurations for the same model.

We have supplied an implementation of our parameterization strategy that on the average works in $O(N \log N)$ time in the size of a polygonal patch. In theory this implementation can be improved to work in $O(N)$ time simply by substituting a linear time shortest path algorithm into our initialization step[1]. The only other automated parameterization scheme to date that works on polygonal meshes runs in $O(N^2)$ time [Eck & Hoppe 1996] and produces no incremental results. As such this algorithm is not a viable alternative for use in an interactive setting. Other approaches to the parameterization problem have lacked the efficiency or robustness (or both) required to successfully parameterize dense polygonal data sets.

A unique feature of our parameterization algorithm is its customizability. In our system a user can specify a number of constraints to the parameterization in the form of feature curves and the algorithm automatically “fills in” the rest of the parameterization i.e. the spring mesh smoothly interpolates feature curves. Typically, a user specifies just those feature curves that are important to a parameterization. However, in theory the user can specify an arbitrary number of feature curves and as a result patch parameterizations can be made arbitrarily complex.

It is worth noting that our surface fitting strategy solves separately for patch parameterization and for patch geometry. In our application the geometry is always fixed to be that of the polygonal mesh. However, the parameterization of the patch can be arbitrary. The

[1]Our algorithm’s time complexity is dominated by the initialization phase that involves the computation of shortest paths on the graph represented by a polygonal patch. We use Dijkstra’s algorithm which runs in $O(N \log N)$ time. Since polygonal meshes are actually planar graphs they possess a simpler structure than arbitrary graphs. As such a linear time algorithm exists to perform the shortest patch computation for planar graphs. If we were to use this algorithm our total parameterization cost would be $O(N)$. We have not heard of reports of a practical implementation of this algorithm and Dijkstra’s algorithm has performed adequately in an interactive setting.

fidelity of the spring mesh to the original polygonal geometry is maintained regardless of the particular parameterization of the data.

9.1.3 Hybrid surface approximation

We fit a hybrid of B-spline surfaces and displacement maps to our spring meshes. The B-spline surfaces represent the coarse geometry of the data while displacement maps capture the fine surface detail. Since our spring meshes are qualitatively indistinguishable from the original polygonal mesh data, our results reproduce with high fidelity the geometry of the original polygon mesh.

The tradeoff between explicitly represented geometry and displacement mapped detail is left to the user. This is enabled in part by the speed of our surface fitting algorithms. Specifically, we use gridded data fitting on the spring meshes. This algorithm runs in $O(\sqrt{N}m^2)$ time[2] where N is the size of a polygonal patch and m the average number of control vertices on a side of the approximating B-spline surface (for purposes of gridded data fitting, the spring mesh parameterization may be considered a pre-processing step). Compared to the speed of gridded data fitting, a non-linear fitting strategy that approximates the mesh vertices has a complexity of $O(Nm^4)$ per iterative step.

Our hybrid surface representation offers a flexible interface to store and manipulate the geometry of the model. We demonstrated this flexibility with surface editing operations that are trivially accomplished with our hybrid representation and yet would have been difficult to replicate with a purely smooth surface representation. In general, we have found that our hybrid surface representation empowers users to manipulate geometry using tools outside our modeling system.

9.1.4 End-to-end surface fitting system

We have presented in this thesis a practical, end-to-end interactive surface fitting system that creates smooth surfaces (and displacement maps) from dense polygonal meshes. The

[2]Assuming the patch is not overly distorted with respect to aspect ratio.

output produced by our system is in a usable form for most animation and design applications. Using this system users have created some of the most complex and high fidelity smooth surface models ever created from scanned data. Prior surface fitting systems have lacked in or more of the characteristics of our system making them an infeasible alternative for fitting to dense meshes.

9.2 Future work

Our system has several limitations. In this section we discuss directions for improving the various components and algorithms of our system as well as some exciting areas for future research.

9.2.1 Improvements to surface curve editing

From the experience of users of our surface fitting system we have found that the precise positioning of boundary curves and feature curves is an important component of the smooth surface creation process. Most of the user time is spent fine tuning the positions of feature curves and boundary curves. Furthermore, since our parameterization and fitting algorithms are rapid, once the boundary curves have been specified creating the spring mesh and approximating B-splines and associated displacement maps takes just a few minutes even for large and complex models. Since the user spends the most time positioning boundary and feature curves, tools that further assist in the curve editing process are a worthwhile area for future research. Our surface snake based tools were a first step in this direction. We have listed below several avenues where our formulation can be further improved.

- Formulating a linear time impulse propagation algorithm to make surface snake editing more efficient.

- Creating more sophisticated surface snake editing tools. One example is a tool that allows intuitive manipulation of tangent vectors at arbitrary points along a surface snake.

- Allowing for the use of templated surface curves that can be used across multiple models.

9.2.2 Improvements to parameterization and fitting

Because our parameterization strategy relies on surface walking strategies and mesh connectivity to resample polygonal patches, it breaks down in the presence of holes in the polygon mesh. However, new range image integration techniques include methods for filling holes.

Another limitation is that B-spline surface patches, our choice to represent coarse geometry, perform poorly for very complex surfaces such as draped cloth. B-splines have other disadvantages as well, such as the inability to model triangular patches without excessive parametric distortion. Despite these limitations, B-splines (and NURBS in general) are widely used in the modeling industry. This has been our motivation for choosing this over other representations.

A limitation of our surface fitting solution in the context of displacement map computation was discussed in chapter 7 (section 7.5). This limitation arises from the fact that a least squares fitting strategy does not have the mathematical machinery to allow the user fine control over the extent of geometry represented in the displacement map. Note that this is a limitation of the least squares fitting process rather than a limitation of our hybrid surface representation. Also note that any automated fitting procedure would have similar limitations.

There are a number of fruitful directions for further improving our parameterization and surface fitting processes. Straightforward extensions include:

- Improving robustness in the presence of holes in the mesh.
- Adding the ability to create triangular parameterizations and patches.
- Creating adaptive spring grids for sampling decimated meshes.
- Adding tools for fitting to the original mesh vertices.
- Allowing variable knot density in our B-spline surfaces.

- Removing the restriction of feature curves being iso-curves of a parameterization.
- Using alternative smooth surface representations such as hierarchical splines or wavelets.
- Implementing alternative (statistical or mathematical) continuity solutions.

9.2.3 Open problems

Displacement maps have long been used to create geometrical detail from 2-D images. In this thesis we have demonstrated techniques that enable the inverse of this process i.e. that enable the creation of 2-D images from geometrical detail. Thus, using our techniques users can easily switch between 2-D and 3-D representations of geometry. This ability in turn opens up possibilities for novel modeling and editing paradigms. For example, in figure 7.4 we demonstrated how multiple resolutions of displacement map images could be used to trivially effect a complex change in 3-D geometry. This same editing operation would have been significantly more cumbersome if just conventional 3-D editing tools had been used. Creating a framework for editing paradigms that use hybrid representations of images and geometry poses several challenges (e.g. see figure 7.4) and offers a variety of fruitful avenues for future research.

In the context of displacement map computation, an interesting extension of our surface fitting strategy would be that of automatically determining the separation of coarse geometry and geometric detail, perhaps based on the frequency domain characteristics of the data. A related idea is that of providing the user with efficient tools to precisely control the extent of geometry to be represented in the displacement map vs the smooth surface (also see chapter 7, section 7.5).

A natural application of the parameterization portion of our system is the interactive texture mapping and texture placement [Pedersen 1996] for complex polygonal models. Related to this idea and to the ones on displacement mapping above, is the possibility of applying procedural texture analysis/synthesis techniques [Heeger & Bergen 1995] to create synthetic displacement maps from real ones. Using our techniques such maps can be applied to objects of arbitrary topology.

Another interesting avenue for future research is that of creating an approximating surface for the entire model based on just a small collection of user specified surface curves (i.e. the curves do not have to create a patch network). Such an approximating technique could be useful for those applications where specifying the entire network of smooth surface boundary curves is not crucial, rather the only requirement is that the surface conform to a small set of potentially disconnected feature curves. Techniques in the free-form design literature have attempted such techniques but only for very small data sets. Our feature drive parameterization algorithms may be thought of as a finite difference solution to a variational parameterization problem restricted to a dense polygonal patch. Extending these approaches to work with dense, arbitrary topology polygon meshes is still an exciting open problem.

Appendix A

Face-point motion

Recall that both our surface curve and spring mesh representations consist of a set of face-points that are constrained to lie on the surface of the polygonal mesh. An operation we use often on individual face-points during our editing and relaxation operations is one of moving face-points to new locations on the surface according to editing or relaxation criteria. We call the procedure that accomplishes this *MovePointOnSurface*. Figure A.1 explains the procedure. Note that there are alternative ways of implementing this procedure on polygo-

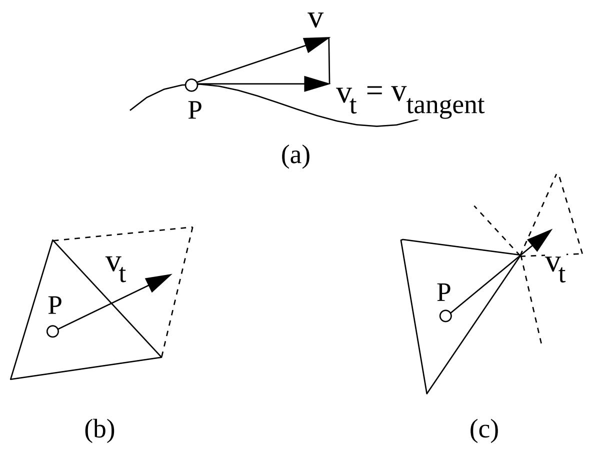

Figure A.1. (a) shows view of a face-point being pulled over the surface. The point moves along the polygonal surface until it either meets an edge or a vertex. (b) and (c) show the two cases that arise when P moves: it either intersects an edge or it intersects a vertex. In either case, we determine the appropriate triangle to move into using a mesh adjacency structure (e.g. a winged-edge representation).

nal surfaces. For example, Turk [Turk 1992] describes a scheme where points first leave the surface and are then re-projected back onto the surface. However, note that re-projection operations can exhibit non-robust behaviour in high curvature regions of the surface. Our scheme avoids this non-robustness since our face-points never actually leave the surface, rather they simply slide along it.

Bibliography

Aho, A. V. & Ullman, J. D. [1979]. *Data structures and algorithms*, Addison-Wesley.

Bajaj, C., Bernardini, F. & Xu, G. [1995]. Automatic reconstruction of surfaces and scalar fields from 3D scans, *Proceedings of SIGGRAPH '95 (Los Angeles, CA, Aug. 6-11, 1995)*, ACM Press, pp. 109–118.

Bardis, L. & Vafiadou, M. [1992]. Ship-hull geometry representation with b-spline surface patches, *Computer Aided Design* **24**(4): 217–222.

Barr, A. H., Currin, B., Gabriel, S. & Hughes, J. F. [1992]. Smooth interpolation of orientations with angular velocity constraints using quaternions, *in* E. E. Catmull (ed.), *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, pp. 313–320.

Barsky, B. A. [1982]. End conditions and boundary conditions for uniform b-spline curve and surface representations, *Computers In Industry* **3**.

Bartels, R., Beatty, J. & Barsky, B. [1987]. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Palo Alto, CA.

Bennis, C., Vezien, J. & Iglesias, G. [1991]. Piecewise surface flattening for non-distorted texture mapping, *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, pp. 237–246.

Blinn, J. F. [1978]. Simulation of wrinkled surfaces, *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, pp. 286–292.

Celniker, G. & Gossard, D. [1991]. Deformable curve and surface finite elements for free-form shape design, *in* T. W. Sederberg (ed.), *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, pp. 257–266.

Chen, J. & Han, Y. [1990]. Shortest paths on a polyhedron, *Proc. 6th Annual ACM Symposium on Computational Geometry*, pp. 360–369.

Cook, R. L. [1984]. Shade trees, *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, pp. 223–231.

Curless, B. & Levoy, M. [1996]. A volumetric method for building complex models from range images, *Computer Graphics (SIGGRAPH '96 Proceedings)*.

DeRose, T. [1990]. Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces, *Computer Aided Geometric Design* **7**(1-4): 165–180.

DeRose, T. D. & Barsky, B. A. [1988]. Geometric continuity, shape parameters, and geometric constructions for catmull-rom splines, *ACM Transactions on Graphics* **7**(1): 1–41.

Dierckx, P. [1993]. *Curve and Surface Fitting with Splines*, Oxford Science Publications, New York.

Dietz, R., Hoschek, J. & Jüttler, B. [1993]. An algebraic approach to curves and surfaces on the sphere and on other quadrics, *Computer Aided Geometric Design* **10**(3): 211–230.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M. & Stuetzle, W. [1995]. Multiresolution analysis of arbitrary meshes, *Computer Graphics (Proceedings of SIGGRAPH '95)*, pp. 173–182.

Eck, M. & Hoppe, H. [1996]. Automatic reconstruction of b-spline surfaces of arbitrary topological type, *Computer Graphics (Proceedings of SIGGRAPH '96)*.

Farin, G. [1982]. A construction for visual C1 continuity of polynomial surface patches, *Comput. Graphics and Image Process. (USA)* **20**: 272–282.

Farin, G. [1990]. *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press.

Finkelstein, A. & Salesin, D. [1994]. Multiresolution curves, *Computer Graphics (SIGGRAPH '94 Proceedings)*, pp. 261–268.

Foley, T. A. & Nielson, G. [1989]. *Knot selection for parametric spline interpolation*, Academic Press, pp. 261–271.

Forsey, D. R. & Bartels, R. H. [1988]. Hierarchical B-spline refinement, *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, pp. 205–212.

Forsey, D. R. & Bartels, R. H. [1991]. Surface fitting with hierarchical splines, *Topics in the Construction, Manipulation, and Assessment of Spline Surfaces, SIGGRAPH course 25*, pp. 7–0–7–14.

Franke, R. [1982]. Scattered data interpolation: test of some methods, *Math Computation* **38**: 181–200.

Franke, R. & Schumaker, L. [1986]. A bibliography of multivariate approximation, *in* C. Chui & L. Schumaker (eds.), *Topics in Multivariate Approximation*, Academic Press.

Gabriel, S. A. & Kajiya, J. T. [1985]. Spline interpolation in curved space, *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*.

Golub, G. H. & Loan, C. F. V. [1993]. *Matrix Computations*, The Johns Hopkins University Press.

Gortler, S. J. & Cohen, M. F. [1995]. Hierarchical and variational geometric modeling with wavelets, *Symposium on Interactive 3D Graphics*, pp. 35–42.

Grimm, C. M. & Hughes, F. J. [1995]. Modeling surfaces of arbitrary topology using manifolds, *in* R. Cook (ed.), *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 359–368. held in Los Angeles, California, 06-11 August 1995.

Halstead, M., Kass, M. & DeRose, T. [1993]. Efficient, fair interpolation using Catmull-Clark surfaces, *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 35–44.

Heeger, D. & Bergen, J. R. [1995]. Pyramid-based texture analysis/synthesis, *Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 229–237.

Hoppe, H., DeRose, T. D., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J. & Stuetzle, W. [1994]. Piecewise smooth surface reconstruction, *Computer Graphics (SIGGRAPH '94 Proceedings)*, Vol. 28, pp. 295–302.

Hoschek, J. [1988]. Intrinsic parameterization for approximation, *Computer Aided Geometric Design* **5**: 27–31.

Hoschek, J. & Lasser, D. [1993]. *Fundamentals of Computer Aided Geometric Design*, AK Peters, Wellesley.

Kass, M., Witkin, A. & Terzopoulos, D. [1988]. Snakes: Active contour models, *Int. J. of Computer Vision* **1**: 321–331.

Klein, P., Rao, S., Rauch, M. & Subramanian, S. [1994]. Faster shortest-path algorithms for planar graphs, *Proceedings of 26th Annual ACM Symposium on Theory of Computing. STOC '94*, ACM Press, pp. 27–37.

Krishnamurthy, V. & Levoy, M. [1996]. Fitting smooth surfaces to dense polygon meshes, *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 5-9 1996)*, ACM Press, pp. 313–324.

Lawson, C. L. & Hanson, R. J. [1974]. *Solving Least Square Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Loop, C. [1994]. Smooth spline surfaces over irregular meshes, *in* A. Glassner (ed.), *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 303–310. ISBN 0-89791-667-0.

Lounsbery, M., Mann, S. & DeRose, T. D. [1992]. Parametric surface interpolation, *IEEE Computer Graphics and Applications* pp. 45–52.

Ma, W. & Kruth, J. P. [1995]. Parameterization of randomly measured points for least squares fitting of b-spline curves and surfaces, *Computer Aided Design* **27**(9): 663–675.

Maillot, J. [1993]. Interactive texture mapping, *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 27–34.

Milroy, M. J., Bradley, C., Vickers, G. W. & Weir, D. J. [1995]. G1 continuity of b-spline surface patches in reverse engineering, *Computer-Aided Design* **27**: 471–478.

Mitchell, J. S. B., Mount, D. M. & Papadimitriou, C. H. [1987]. The discrete geodesic problem, *SIAM J. Comput.* **16**: 647–668.

Moreton, H. P. & Séquin, C. H. [1992]. Functional optimization for fair surface design, *in* E. E. Catmull (ed.), *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, pp. 167–176.

Nielson, G. & Foley, T. [1989]. *An affinely invariant metric and its applications*, Academic Press, pp. 445–467.

Nielson, G. M. [1993]. Scattered data modeling, *IEEE Computer Graphics and Applications* pp. 60–70.

Ostby, E. [1986]. Describing free-form 3d surfaces for animation, *Workshop on Interactive 3D Graphics*, pp. 251–258.

Parke, F. I. & Waters, K. [1996]. *Computer Facial Animation*, A K Peters.

Pedersen, H. K. [1995]. Decorating implicit surfaces, *Computer Graphics (Proceedings of SIGGRAPH '95)*, pp. 291–300.

Pedersen, H. K. [1996]. A framework for interactive texturing on curved surfaces, *Computer Graphics (Proceedings of SIGGRAPH '96)*.

Peters, J. [1990]. *Fitting smooth parametric surfaces to 3D data*, Ph.d. thesis, Univ. of Wisconsin-Madison.

Piper, B. [1987]. Visually smooth interpolation with triangular Bézier patches, *in* G. Farin (ed.), *Geometric Modeling: Algorithms and New Trends*, SIAM, Philadelphia, pp. 221–233.

Reeves, W. T. [1990]. Simple and complex facial animation: Case studies, *State Of The Art In Facial Animation, SIGGRAPH course 26*, pp. 90–106.

Rogers, D. F. & Fog, N. G. [1989]. Constrained b-spline curve and surface fitting, *Computer Aided Geometric Design* **21**: 641–648.

Sarkar, B. & Menq, C. [1991]. Parameter optimization in approximating curves and surfaces to measurement data, *Computer Aided Geometric Design* **8**: 267–290.

Schmitt, F. J. M., Barsky, B. A. & hui Du, W. [1986]. An adaptive subdivision method for surface-fitting from sampled data, *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, pp. 179–188.

Schumaker, L. [1979]. Fitting surfaces to scattered data, *in* G. Lorentz, C. Chui & L. Schumaker (eds.), *Approximation Theory II*, Acedemic Press, pp. 203–268.

Sclaroff, S. & Pentland, A. [1991]. Generalized implicit functions for computer graphics, *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, pp. 247–250.

Shirman, L. & Séquin, C. [1990]. Local surface interpolation with shape parameters between adjoining Gregory patches, *Computer Aided Geometric Design* **7**(5): 375–388.

Shirman, L. & Séquin, C. [1991]. local surface interpolation with Bezier patches: errata and improvements, *Computer Aided Geometric Design* **8**(3): 217–222.

Sinha, S. S. & Schunck, B. G. [1992]. A two-stage algorithm for discontinuity-preserving surface reconstruction, *IEEE Trans. On Pattern Analysis and Machine Intelligence* **14**(1): 36–55.

Sinha, S. S. & Seneviratne, P. [1993]. Single valuedness, parameterization and approximating 3d surfaces using b-splines, *Geometric Methods in Computer Vision 2* pp. 193–204.

Strang, G. [1986]. *Introduction to Applied Mathematics*, Wellesley-Cambridge Press.

T. Poggio, V. T. & Koch, C. [1985]. Computatational vision and regularization theory, *NATURE* **317**: 314–319.

Terzopoulos, D. [1986]. Regularization of inverse visual problems involving discontinuities, *IEEE Trans. On Pattern Analysis and Machine Intelligence* **8**: 413.

Terzopoulos, D. [1988]. The computation of visible-surface representations, *IEEE Trans. On Pattern Analysis and Machine Intelligence* **10**(4): 417–438.

Thompson, J. F. [1991]. *The Eagle Papers*, Mississippi State University.

Turk, G. [1992]. Re-tiling polygonal surfaces, *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, pp. 55–64.

Turk, G. & Levoy, M. [1994]. Zippered polygon meshes from range images, *Computer Graphics (SIGGRAPH '94 Proceedings)*, pp. 311–318.

Weinstock, R. [1974]. *Calculus of variations with applications to physics and engineering*, Dover Publications, Inc., 180 Varick Street, New York, N.Y. 10014.

Welch, W. & Witkin, A. [1994]. Free-form shape design using triangulated surfaces, *Computer Graphics (SIGGRAPH '94 Proceedings)*, Vol. 28, pp. 237–246.