Serious Putty
Topological Design for
Variational Curves
and Surfaces

William Welch
School of Computer Science
Carnegie Mellon University

Abstract

In this work we develop a new approach to designing curves and free-form surfaces on a computer. It is inspired by a style of pencil-and-paper design used for sculptured surfaces, in which the designer specifies the shapes of important curves (*character lines*) and indicates surfaces that pass through them smoothly, with no unnecessary bulges or wiggles (that is, the surfaces are *fair*). Unlike previous modeling approaches based on the notion of character lines, we allow surfaces to be cut apart and smoothly joined along arbitrary curves, so that the designer can build up complex shapes and topologies from simpler ones. Further, our surfaces are infinitely stretchy, so that the designer may add unlimited amounts of detail simply by indicating more control points and curves. Finally, portions of the surface may be made to copy externally controlled *shape tools.* This allows the designer to mix free-form and structured shapes within a single composite surface model of arbitrary topology.

This kind of conceptually simple shape description ("give me a fair surface bordered by *these curves* that passes through *those curves* while touching *that point*") may be precisely interpreted as a functional minimization problem in the calculus of variations ("give me the surface coordinate function that maximizes *this fairness integral* subject to *those geometric constraints*"). Our modeler represents curves and surfaces implicitly, as the solutions of such variational minimization problems. As the designer interacts directly with a surface, the modeler interprets these actions as changing the variational shape specification. Triangulated point sets are used to approximate these smooth variational surfaces in real time, using a novel finite-difference scheme over arbitrary-topology surface meshes along with an adaptive, interactive mesh refinement and re-triangulation scheme.

Ultimately, all of these numerical details are hidden from the designer, who sees a pristine surface that may be grabbed at arbitrary points and along arbitrary curves, and whose shape changes in simple, predictable ways. The resulting ability to interactively design variational shapes of arbitrary, mutable topology has never before been available in a geometric modeler.

Contents

Chapter 1

Introduction

In this work we develop a new approach to designing curves and free-form surfaces on a computer. Traditionally, free-form surface design tools have handled only simple surface topologies, such as panels for auto bodies. But emerging manufacturing technologies, as well as increasing demand for richer models in computer graphics and animation, motivates us to consider more ambitious approaches to free-form surface design, including support for arbitrary topologies and higher-level, structured shape control.

Our approach addresses these issues by mimicking a style of pencil-and-paper design often used for sculptured surfaces, in which the designer sketches important curves (*character lines*) and indicates that a "nice" surface should pass through them. We go beyond pencil-and-paper sketching by bringing such descriptions to life in a 3D surface modeler: a designer will control surface shape by tracing out character lines on a surface and then re-shaping them, carrying the surface along. Unlike previous modeling approaches based on this notion of character lines, we allow surfaces to be cut apart and smoothly joined along arbitrary curves, so the designer can build complex shapes and topologies from simpler ones. Further, our surfaces are infinitely stretchy, so that the designer may add unlimited amounts of detail simply by indicating more control points and curves. Finally, portions of the surface may be made to copy externally controlled *shape tools*. This allows the designer to mix free-form and structured shapes within a single composite surface model.

Our approach is based on *variational shape design*, in which shapes are specified implicitly as the solutions to optimization problems. Though the necessary mathematics have been with us since the 18th century, it is only within the past several years that computers have become fast enough to allow variational techniques to be used for interactive shape design. An important advantage of the approach is that shapes created by the designer can be viewed as templates for entire families of related shapes, parameterized by their character lines. Our ability to interactively create variational shapes of arbitrary and changeable topology, with user-specified sets of control points and curves, has never before been available in a geometric modeler.

1.1 Definitions

Before we begin a description of our approach, definition and discussion of some of the terms used here is in order. This work is concerned with the design of *free-form surfaces* — generally smooth, curved surfaces that may contain occasional creases or corners. At any given point on a surface, it may be curved in any and all local directions (that is, the surface is *doubly curved*). These are sometimes referred to as *sculptured* surfaces in the design literature.

Within this broad class of free-form surfaces, we will often be interested in what are known as *fair* shapes — surface (and curve) shapes that don't have unnecessary bulges or wiggles. Fairness is a necessarily fuzzy notion based on our perception of curve or surface quality; but it is intimately related to the distribution of curvature over the shape [Mor93]. The surface of an inflated balloon would be considered fair, whereas its deflated counterpart would likely not be.

A curve or surface's *topology* is an abstracted version of its geometric structure: the relationships that don't change when its shape is smoothly deformed. For surfaces, this includes global notions such as open and closedness, handles and holes (a cylinder is open, and has two holes and no handles; a torus is closed, and has one handle and no holes). It also includes containment and connectedness relationships between subsets of the object. Informally, if we consider a shirt crumpled on the floor as an abstract surface, the topological information includes permanent features like openings for the arms and head, and the seams that join the sleeves to the body. The seams and holes carve it up into *regions*, giving us containment relationships and boundary curves, as well as connectivity information between these *topological features*. None of this changes when you put the shirt on.

1.2 Motivation

Computer Aided Geometric Design (CAGD), a field whose principal concern is describing curves and surfaces using computers, has been a rich area of research and practical application ever since hardware for the automated machining of 3D shapes first became available in the 50's [Far90]. It is a place "where the math meets the metal." That said, we should point out that since its beginnings, which are rooted in in the development of design tools for the automotive industry, the majority of free-form surface design methods and tools have addressed simple, planar topologies — as might be used for designing stamped panels, or the separate pieces of injection molded surfaces. Computationally, dealing with arbitrary surface topologies is a *much* more difficult proposition than dealing with planar patches — enough so that issues have only recently begun to be addressed.

The attention to such issues is particularly timely, given recent advances in manufacturing technologies for rapid prototyping. It is now possible to directly fabricate free-form geometries of essentially arbitrary topology, *e.g.*, by metal deposition [MPR$^+$94]. As of this writing, we are capable of rendering in metal a wide range of surface geometries for which we have no suitable design tools! Current design tools used with these processes are based on solid modeling techniques (discussed below), and do not address the problem of direct design for free-form surfaces. More general surface design tools will ultimately allow industrial designers (who still fall back to clay models on occasion) to exploit this new fabrication capability for designing and prototyping aesthetic shapes via computer. Aside from physical realization through manufacture, such geometries are also desirable as "virtual artifacts." As computer graphics and animation progresses, so does its appetite for richer geometric models, and it is desirable to consider more ambitious design tools.

1.3 Current approaches (and their limitations)

Direct-surface modelers typically represent free-form surfaces as piecewise smooth quilt-works of surface patches [Far90]. The designer is given a mesh of control points or curves whose number and arrangement is determined by fineness of the underlying surface representation. Thus, the designer exercises local control over shape, by re-shaping elements of the control net. The drawback is that making a conceptually simple change to shape using such a control net could require moving every single element in a coordinated way, if the geometry of the change does not fit well with the given mesh structure. As if this weren't bad enough (and in fairness, recent research has aimed at better, nonlocal control of such quilt-works [CG91, WW92, Kal93]), the mathematical form of many of these patches necessarily restricts the global surface topology. Piecewise smooth patch schemes based on rectangular control nets are obviously restricted to sheets, cylinders, and tori as their only natural model topologies. Many older approaches based on smooth triangular elements are similarly limited, though the reason is more subtle (Section 2.4). Topologically general piecewise smooth representation schemes are generally more complex constructions[MLL$^+$92], and remain an active area of research[BGW88, LD90, HKD93, Loo94].

More ambitious topological modeling for manufacture has been the purview of Constructive Solid Geometry (CSG) approaches[RV82]. Here, complex composite shapes are built up by combining simple solid primitives (*e.g.*, prisms, spheres) using Boolean set-theoretic operations (*e.g.*, union and difference). While we can design unrestricted surface topologies this way, there is no direct, explicit control over surface shape; rather, it is the result of simple combinations of rigid solid primitives. Thus, even though CSG systems have been extended to handle primitives with sculptured faces[AMR83, CGP93], these solid-based methods do not address design situations where creating an overall graceful surface shape is a goal.

Another approach to topologically unrestricted design is direct volume sculpting, where local changes to the surface may be made by chipping away or adding "material" to a 3D neighborhood [GH91]. The drawback again is that global changes in shape must be formulated as sequences of local operations that touch potentially every point on the surface.

Finally, the character line approach to design described in opening paragraphs is an example of *variational* shape design, about which we will say a good deal more, below. As we shall see in Chapter 2, there has been a fair amount of CAGD research (much of it in the last 5 years) into variational shape design. The work in surface design falls into two broad categories: schemes that generate high quality surfaces of perhaps arbitrary topology, but do not run at interactive speeds; and schemes that run at interactive speeds but cannot generally produce high-quality shapes, and generally do not accommodate arbitrary topologies.

1.4 Variational shapes

In this work we focus on a class of free-form shapes that are optimal in a certain sense. In designing free-form curves and surfaces, we will often want them to take on fair shapes within a region; or we may want portions of a curve or surface to copy some other prototype shape as nearly as possible. Each of these notions can be expressed as an optimization problem: find a shape that maximizes fairness, or minimizes deviation from a given shape. Many shapes from nature can also be described in terms of optimization (soap films minimize their surface area; loaded elastic beams minimize their total curvature). A shape that optimizes some given quality measure is often referred to as a *variational* shape in the CAGD literature[HS90], because it is best described mathematically using the calculus of variations[CH37]. As we will see, though the mathematics behind such shapes can be intricate, their high-level descriptions are remarkably simple ("give me a fair cylinder with end curves shaped like *this*, passing through *that* point"); and this yields a powerful, concise vocabulary for describing curves and surfaces. A nice side-effect of this kind of shape specification is that, instead of static geometry, one has actually specified an entire family of similar shapes, parameterized by the specified control curves.

In later chapters, we will manage to escape with only a very small dose of variational calculus (just enough to let us approximate solutions to the optimization problems). Nonetheless, this technical term "variational" occurs fairly often in our discussions, for no better reason than that we need a convenient name for this class of shapes. You could safely ignore its technical meaning, and mentally substitute "optimal" or "high quality" if you prefer.

1.5 Goals

In this work we will address the problem of specifying and representing free-form shapes variationally. Some specific goals are:

- Model variational surfaces of unrestricted topology. Surfaces may be open or closed, and have any arrangement of holes and handles.
- Allow the designer to precisely control portions of the surface shape by specifying control points or curves through which the surface must pass.
- Allow the designer to stretch, shrink, or deform surface areas without restriction, and add arbitrary amounts of detail.
- Allow the designer to interactively and incrementally build up smooth topologies through surface surgery: cutting, stitching, creasing, and skinning along embedded control curves.
- Since not all shapes are best described variationally, allow the designer to incorporate explicit shapes into a variational model and thus create structured composites.

1.6 Approach

The surface behavior outlined in these goals can be precisely characterized in terms of a simply formulated (if not simply solved) variational optimization problem. User-defined control points and curves act as geometric constraints on the possible shapes; the automatic fairing and shape-copying behaviors are then realized by optimizing the shapes of user-specified topology subject to these geometric constraints. In this work, we minimize a measure of the curve or surface's total curvature as a way of making it seek a fair shape.

Unfortunately, it is not generally possible to solve such variational problems explicitly. We therefore develop a method for approximating solutions to these curvature-minimization problems using a triangulated surface mesh. This involves setting up a generalized finite-difference scheme and optimizing estimated curvatures over the mesh (a process that is complicated by the fact that we have no global parameter plane over which to formulate the computation). A dynamic adaptive mesh scheme is used to maintain a good sampling and triangulation of the surface as its shape changes.

Ultimately, we are able to treat this approximation machinery as a "black box". This allows us to build an interactive modeler that operates on variational curves and surfaces as its basic shape representation, much like a conventional modeler might operate on B-splines or Bezier patches. We interpret user interactions as modifying the variational specification for the shape, and rely on the modeler to recompute fresh approximations to the new shape at interactive speeds.

1.7 Contributions

The primary contributions of this work are:

- It is the first work to address interactive, incremental design of free-form surfaces of arbitrary topology where the user has explicit control over the topology at all times.
- It is the first interactive approach to free-form surfaces allowing the user to "slide" surface features around relative to each other.
- It includes an approximation scheme for geometric thin plate surfaces, based on triangulated surface meshes, that is speedy and robust. Some secondary results related to this scheme include:
  - an approach to computing neighborhood parameterizations for a finite-difference scheme over an arbitrary topology mesh
  - an adaptive mesh generation scheme suitable for use in interactive modeling for arbitrary-topology surfaces.

As part of this work, we built an interactive, direct manipulation surface modeler that demonstrates all of the functionality discussed above (though no one would mistake it for a full-fledged industrial design tool). The modeler itself represents a number of secondary contributions:

- It is the first modeler that uses variational curve and surface specifications as its basic shape representation (built on top of the approximation machinery above).
- It is the first *variational* modeler that allows a designer to build up surface topology in terms of "surgical operations" on the 3D surface. Traditional alternatives are to indicate edge correspondences on a single polygonal (2D) domain (*e.g.*, [FRC92]), or in 3D to connect-the-dots and fill-the-holes to build up a surface control net (*e.g.*, [CK83]). Neither of these protects the designer from specifying nonsensical surface topologies.

1.8 Road-map to the thesis

We begin with a discussion of a wide range of existing curve and surface modeling techniques (Chapter 2). The sections of this chapter are relatively self-contained, and the reader may prefer to refer back to them when directed from later chapters. We begin an overview of our approach to shape design with a sample design session (Chapter 3). A detailed discussion of our approach begins with methods of specifying and representing variational curves and surfaces (Chapter 4). This is followed by a method for approximating their shapes using triangulated meshes, based on a generalized finite-difference scheme (Chapter 5). In addition to serving as an approximate shape representation, the triangulated surface also serves as a computational mesh for these approximation calculations. In Chapter 6 we present a scheme for maintaining a quality mesh as curve and surface shapes evolve. Finally, we discuss our implementation of a modeler that uses these variational curves and surfaces as its basic representation for shape, and give additional implementation details for the machinery presented in the previous chapters (Chapter 7). The dissertation concludes with a summary of the contributions, and discussion of possible extensions to the work.

Chapter 2

Previous Work

In this chapter, we survey previous approaches to computer-aided design of free-form shapes, and review the mathematical topics and techniques that play a role in our modeling approach. There is a wide variety of free-form surface modeling approaches in the literature. None of them manages to deliver globally fair, arbitrary surface topologies at interactive speeds, but they fall short in a number of very different ways. We consider the various mathematical/computational elements that go into a free-form surface modeler, leading up to a discussion of interactive modelers in Section 2.6. This is followed by a discussion of computational techniques for triangulated surface meshes used as approximations to smooth surfaces.

2.1 Overview

[**Topology and design**] We'll begin this chapter by reviewing some concepts and terminology from topology relevant to surface modeling (Section 2.2). A fundamental aspect of our approach to modeling curves and surfaces is the separate handling of topological and positional information for a given model. Some approaches to topological specification in computer-aided design are taken up in Section 2.3. A topological description of a surface captures the ways in which various regions of a surface are connected to or contained within each other — relationships that do not change when the surface is smoothly deformed. Because a given abstract surface (*e.g.*, the topological sphere S^2) will have many possible realizations as a surface in 3D (*e.g.*, a pear, a head, a dining-room table), most of these modeling approaches also specify shape along with topology.

[**Fair surfaces**] By far, the most common methods of representing topologically interesting free-form surfaces is as collections of simpler (typically rectangular or triangular) *patches*, joined along their edges to form a piecewise smooth *composite* surface. Methods for computing the shapes of such piecewise smooth elements can be usefully classified as *local* or *global*. Local methods construct a surface one patch at a time, using geometric constructions that take information such as edge position and tangency requirements along a patch boundary and conceptually work inward to compute the shape of the patch's interior. Global schemes, which operate simultaneously on collections of patches, most often optimize some global shape objective or attempt to satisfy global area or volume constraints. In this work, we are interested in generating globally *fair* surfaces — "nice" surfaces having no extraneous wiggles. This is a bit more restrictive than simply asking for a *smooth* surface; as an example, an egg-crate mattress's surface is smooth but not fair. As will be seen, purely local schemes aren't capable of generating high-quality faired surfaces, and this will lead us to ultimately consider only global schemes.

[**Shape optimization**] We will consider some precise mathematical definitions of "fairness". A number of different characterizations have been used previously, each having strengths and weaknesses. Given a particular characterization of fairness, it is possible to cast surface shape specification as an optimization problem. Using the calculus of variations, the desired shape can be described as the solution to a variational minimization problem with the fairness measure as the objective function. We will cover some of the existing variational shape design approaches, their ways of assessing a shape's fairness, and (because such variational problems are almost never solvable in closed form) their solution approximation methods.

[**Interactivity**] Another important goal of our modeling approach is *interactivity*. Only a few of the global approaches we will discuss are appropriate for modeling at interactive speeds, recomputing curve or surface shapes quickly as the designer changes the specification. In Section 2.6 we discuss the current range of interactive variational modelers, and discuss some of the issues involved in supporting interactivity. In order to keep their computations tractable, interactive systems generally restrict the kinds of constraints and objectives that can be used in describing a surface, and unfortunately this restricts the kinds of shapes and topologies that may be effectively modeled. Overcoming these limitations will motivate many of our representational and algorithmic choices in the the following chapters.

[**Triangulated surfaces**] Finally, the approach we will develop uses triangulated surfaces. In support of the actual shape optimization computations of our approach, it will be important to compute over mesh vertices as if they were discrete samples of some underlying smooth surface. It will also be necessary to maintain good triangulations (serving as computational meshes) over surfaces. The last section of this chapter reviews some work in mesh generation for curved surfaces. Mesh generation for surfaces in 3D is a much more difficult problem than mesh generation in the plane, because of the difficulties in constructing surface triangulations over points in space.

2.2 Topology

In this work we are concerned with smooth curves and surfaces. We begin with a brief review of topological concepts, specialized to these objects. Viewing an abstract surface as a set of points, the set's *topology* defines connectivity and containment relationships among subsets of its points. Locally (that is, within a small neighborhood of a point on a curve or surface), the topology of our objects is always the same: curve neighborhoods have the 1-dimensional topology of an open interval (or half-interval, for boundary neighborhoods), and surface neighborhoods have the 2-dimensional topology of an open disc (or half-disc on the boundary). It is global topological information, rather than local, that interests us here: continuity and containment relationships among subsets, handles and holes in a surface, and the surface's *genus*. *Holes* are openings in a surface, bounded by closed

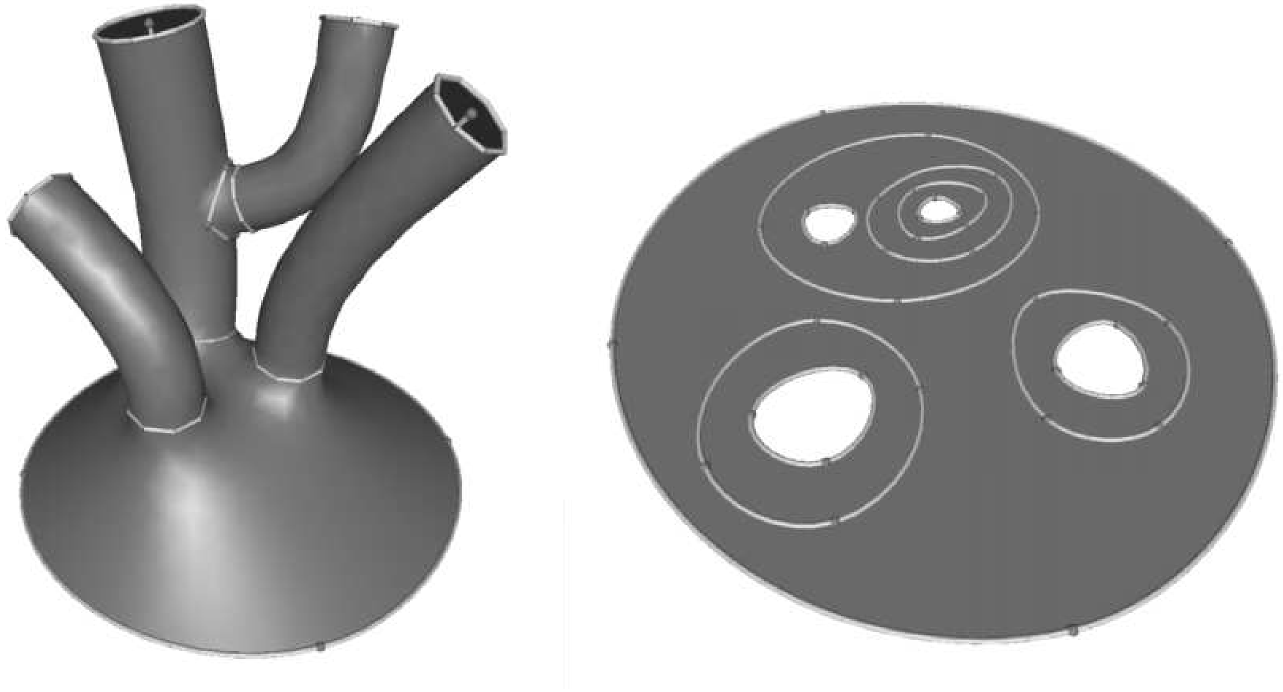


Figure 2.1: (L) Party Hat: a branching surface. (R) Party Flat: the topological regions associated with the surface on the left.

curves: topologically, a cylinder is a sphere with two holes cut in it. *Genus* is a count of the number of *handles* on a surface: a sphere is genus 0, a coffee-cup or donut is genus 1 (though it is probably too late to convince the world to speak of donut handles), and a pair of scissors has a genus 2 surface. Such topological information is largely independent of the particular shape a surface may have. Figure 2.1 shows a branching object and the topological information distilled from it.

The standard approach to representing general topological spaces builds them up as combinations of the simplest possible spaces. These simpler spaces (0D points, 1D intervals, 2D triangles, 3D tetrahedra, etc) are called *simplices*, and spaces built up from them according to a few simple rules are called *simplicial complexes*. General simplicial complexes can represent manifold topologies (curves and surfaces), and non-manifold topologies (Figure 2.2). A formal discussion of the rules that govern the construction of simplicial complexes would therefore be more precise and more general than is warranted here (see Jänich[Jän84] for a lively presentation of point-set topology leading to simplicial complexes and topological "gluing" operations, or Massey[Mas77] for a more comprehensive treatment). Informally, simplices may be glued together by having individual elements share boundary elements. Thus, 1D curve domains are built up by pairing endpoints of disjoint intervals, and 2D surface domains are built by pairing edges of disjoint triangular faces. Note that it is connectivity among simple topological elements that is being recorded here, not shape. Even though our illustrations render surface complexes as triangulated surfaces situated in 3D, topological reasoning over a complex is a combinatorial process and makes no use of such

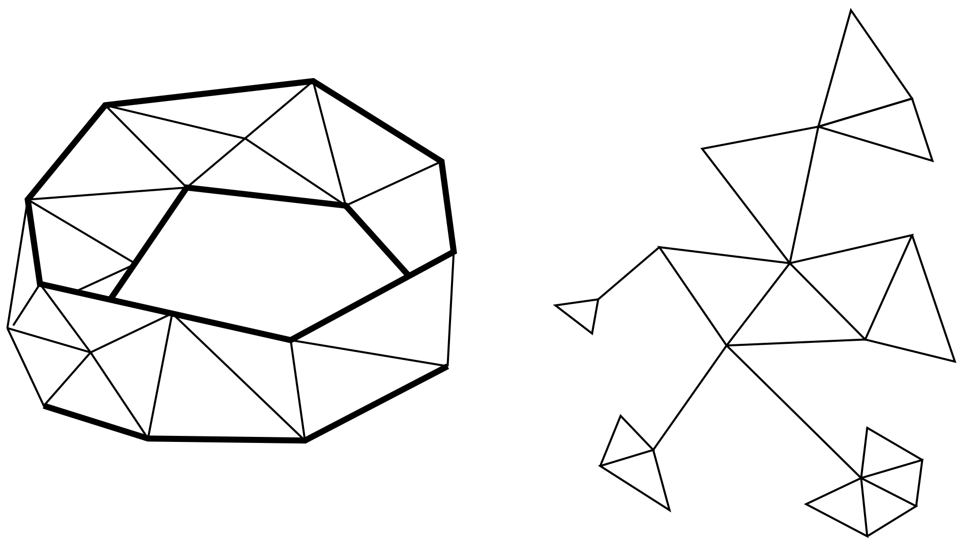


Figure 2.2: Two simplicial complexes. The one on the left represents a cylindrical surface strip. The one on the right, well.... A triangulated surface along with its nodes and edges constitutes a valid simplicial complex, but the converse is not necessarily true.

spatial coordinates. As an example, the Euler characteristic χ is a fundamental algebraic-topological quantity, and is given by F-E+V, the alternating sum of the number of faces, edges, and vertices. The Euler characteristic is related to the genus g of a surface by $\chi = 2(1 - g)$. It is invariant for a given topological surface: the genus of the surface doesn't depend on the way we triangulate it.

To take an abstract surface and associate 3D coordinates with its points — as we do in rendering our surface meshes as triangulations in 3D — is to create an *immersion* of the abstract surface. If that immersion is such that the 3D surface does not intersect itself, qualifies as an *embedding* of the surface.

In modeling applications where a surface is represented as a piecewise smooth mesh of patches, it is trivial to interpret such a mesh as a simplicial complex, or more general *cell complex* when the patches are not triangles. Having an explicit topology representation leaves open the possibility of computing over and formally reasoning about model topologies [Män83, DC91, DE93, PBCF93].

2.3 Topological Modeling

In this section, we consider approaches to surface modeling that admit variable topologies. Some of the approaches will represent topological information implicitly — it depends on and must be derived from other geometric information. Other approaches will build on explicit topological representations that may be directly modified. Of course, none of these approaches is concerned exclusively with topological specification, but also with specifying a particular shape along with the given topology. We'll take up some of these shape-related issues in more depth in subsequent sections.

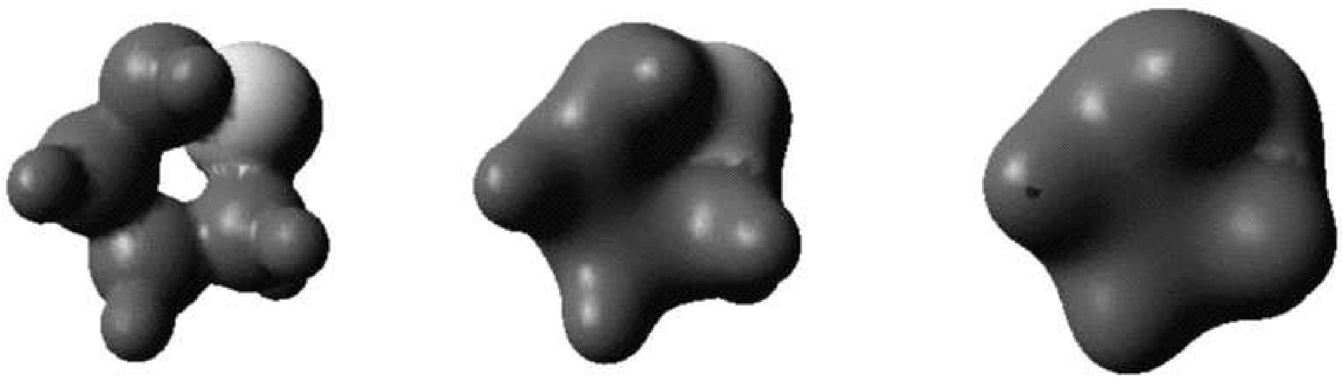


Figure 2.3: A molecular surface represented as a contour of a sum of Gaussian functions. A single bias parameter controls the "blobbiness" of the shape, and can change the topology as a side-effect.

2.3.1 Algebraic curves and surfaces

One approach to representing free-form surfaces of arbitrary genus is as contours of algebraic functions defined over $\mathcal{R}^3$ that is, the set of points

$$\mathbf{x} \in \mathcal{R}^3 \; | F(\mathbf{x}) = 0.$$

for some scalar function F. As an example, a sphere of radius r centered at point $\mathbf{p}$ could be implicitly defined as the points satisfying $(\mathbf{x} - \mathbf{p}) \cdot (\mathbf{x} - \mathbf{p}) - r^2 = 0$. This is a broad class of functions, and has seen a variety of uses in modeling and animation[Bli82, Hec94a] (Figure 2.3). A benefit of working with such a representation is that it is computationally inexpensive to decide whether a given point in space $\mathbf{x}$ is inside, outside, or on the surface (just evaluate the function and look at the sign of the result). The chief difficulty with such a representation is that it is not easy to operate directly on the surface itself, *e.g.*, to render it, or measure its shape properties within some region, or determine its topology. Such operations need explicit representations of surfaces as mappings from some 2D domain into 3D (*parametric* surfaces, discussed next). Algorithms for recovering explicit surfaces from an implicit definition involve contour tracing (*e.g.*, the popular "marching cubes" algorithm[Blo94]), or point-sampling [dFGTV92, WH94, Hec94b] when a triangulation of the surface is not required.

Although it is trivial to modify functions and their parameters to generate a wide variety of surfaces, doing so in a controlled way, so that the resulting surface satisfies given geometric constraints, is more difficult. This kind of inverse-control for implicit curves and surfaces has been accomplished through algebraic function fitting[Pra87, Baj92], and direct construction of implicit blending surfaces [MS85, HH87, Roc89]. Many of these methods construct blend surfaces of simple, fixed topology using restricted sets of basis functions. Though this does not allow the range of topologies possible with general implicit functions, one does not generally want interesting topological behavior from a rounding or fillet operation.

More general inverse control with arbitrary constraint points is also possible with direct fitting techniques [Pra87] or iterative relaxation of the parameters of a given set of shape

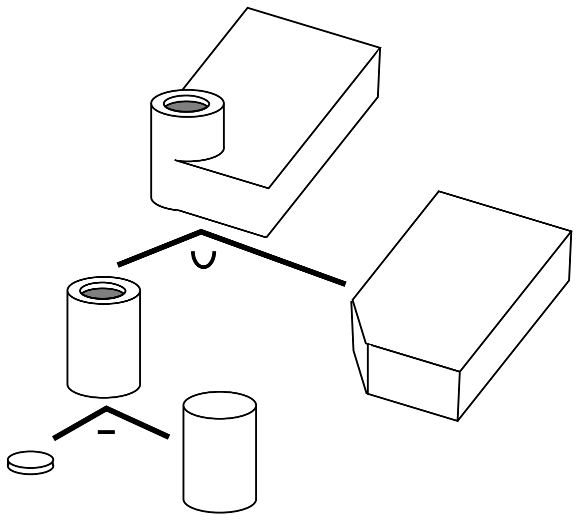


Figure 2.4: A solid model and its associated CSG tree of operations. The smaller cylinder is subtracted from the larger to make a socket in the top. The socketed cylinder is then unioned with the slab.

functions[WH94]. When a rich enough set of basis functions is used, the actual topology of the fitted algebraic surface may begin to depend on the particular values of parameters. For example, a surface generated by a sum of Gaussians [WH94] may vary between multiple disconnected regions and a single blob, or have holes and concavities depending on the settings of the Gaussians' origin, bias, and standard deviation parameters (Figure 2.3). Though it is possible to determine the topology of the fitted surface through contour tracing (though only within a given level of detail), there seems to be no general way to turn this around and constrain the topology of such surfaces to remain fixed, *e.g.*, by placing constraints on the allowable values of the parameters.

2.3.2 Particle systems

Szeliski, *et al.*[ST92, STT93] present a modeling approach that uses localized particle interactions to make the particles behave as if they were being attracted to some underlying surface. The result is evocative of a point-sampled implicit surface, but without an actual implicit function contour defining the surface. Because there is no fixed connectivity between particles, fluid "topological" changes occur when portions of the "surface" come into contact and their particles merge. The limitation for topological design is ultimately the same as with implicit surfaces: we want to prescribe topology, and change it in controlled ways. This cannot be done using such a particle system because there is no underlying surface topology. It is not sufficient to merely impose and maintain a surface triangulation over the particles (this in itself is difficult, as the discussion in Section 2.8 will show), since the particles' interactions do not take such a triangulation into account. We'll say more about particle systems and associated triangulations in the mesh generation discussions of Section 2.8.

2.3.3 Solid modeling and B-reps

Possibly the most widely-used approach to designing topologically interesting surface models is actually a method of specifying 3D solids: Constructive Solid Geometry (CSG)[RV82, Req80]. Briefly, a 3D *solid* is specified as the result of applying boolean set operations (union, intersect, complement) to solid primitives. Figure 2.4 shows an example of such a construction. A solid model is represented implicitly as a tree of boolean operations applied to a set of primitive solids. One popular way of implementing CSG schemes is in terms of an explicitly represented boundary surface, referred to as a *B-rep* solid[Req80]. These boundary representations comprise 2D faces joined by networks of boundary curves. If we consider only the connectivity information in this graph of nodes, edges, and faces, we have an abstract topological surface specified as a cell complex. In its purest form, a designer never directly interacts with these topological elements; rather, changes to the cell complexes are implied as the results of Boolean operations on bounded solids. After a boolean operation has been performed (*e.g.*, the small cylinder is subtracted from the larger in Figure 2.4), holes are cut in the B-rep surface and necessary faces are added to arrive at the boundary of the resulting solid. An advantage of this approach to design is that, when an object is being designed for subsequent manufacture, very often we are interested having a surface that does bound a 3D solid.

For the purposes of curved surface design, the approach has two main drawbacks: first, there is no direct control over the model's topology. One could imagine re-sizing or re-positioning the larger cylinder in Figure 2.4 and accidentally leaving the cylindrical socket behind, thus inadvertently changing the topology of the model as the result of a re-shaping operation. This is more of a problem in more complex models where we wish to maintain constraints that may not readily fit into the decomposition hierarchy. The second drawback is that there is no direct control over the surface *as a surface*: a designer cannot directly reshape a face (or group of faces), since they are defined as boundaries of solids, not as free-standing surfaces.

More recent solid modeling approaches have allowed one to deal with B-rep surfaces as free-standing surfaces, so-called "non-manifold" topologies (really "non-solid"), and this allows open, bordered surfaces to be represented [Bau75, Wei85, RC86, CGP93]. Their motivation is that there are certain solid operations that seem to want to leave an open surface or unadorned edge as their result. One may represent intermediate stages in construction of a boundary surface by cutting and pasting feature boundaries [RIKM93], where the intermediate forms may not make sense as surfaces or solids. These approaches argue for being able to explicitly represent and operate on surfaces, even in a modeler that ostensibly is only concerned with generating solids.

2.3.4 Domain complexes

Ferguson, *et al.*[FRC92], present a finite-element approach to modeling arbitrary smooth topologies. In the tradition of 2D finite-element mesh generation, they address the problem of topological design for sculptured surfaces by conceptually "cutting" a desired closed surface and unfolding it into a single polygon (Figure 2.5 is their double-torus example). The originally joined edges are identified with one another and are forced to maintain complementary boundary conditions during subsequent computations (the numerical difficulties of

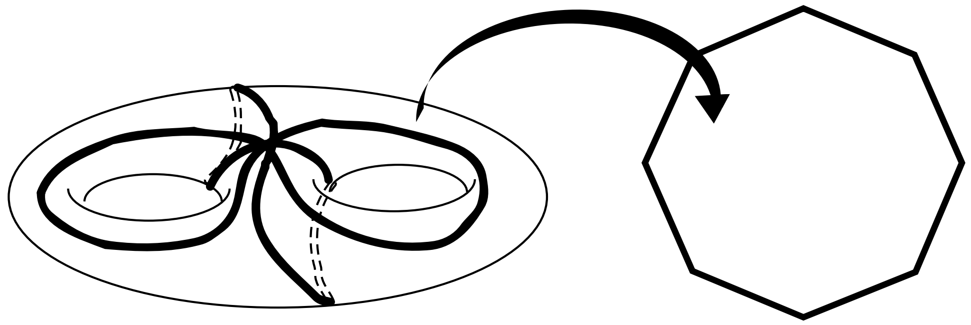


Figure 2.5: The double torus may be flattened into a single octagonal domain by making a number of cuts. In subsequent shape-related computations over this domain, these cut edges would be identified with one another using geometric continuity constraints.

actually doing this will be taken up in later sections). Note that at the time the designer specifies the domain polygon, there is not an actual embedded surface to cut apart; a finite element calculation is subsequently used to compute an immersion for the domain. Incremental approaches to specifying complex topologies are more designer-friendly, allowing one to alternately reshape portions of a model and then perform surface surgery on the current immersion. We discuss these next.

2.3.5 Changing topology by directly editing cell complexes

Intermediate between the two extremes of having topology fall out as a side-effect of geometric operations (CSG) and having to completely specify topology in the absence of geometric specification (domain complexes) are schemes that allow the designer to directly operate on topological cell representation by interacting with a 3D realization of the cell complex — as with the polyhedral shells of [CK83, Nie88], or the "connect-the-dots" approach to specifying a cell complex in [MS92]. The designer works directly with the edges and vertices of a polyhedral (piecewise-linear) shell. Later, the shell edges and faces are skinned with smooth curves and surfaces.

Shells may be built up piece by piece, by connecting points with edges, then designating loops of edges as faces; but this leaves the problem of topological consistency up to the designer. Baumgart[Bau75] proposed an approach to constructing B-rep models that uses simple operations guaranteed to preserve the topological consistency of the model. One starts with a base solid and whittles away at it with "topology-safe" local operations until the desired polyhedral boundary has been achieved. We will develop our own topology-safe operations for surface meshes in Chapter 4.

2.3.6 Higher-level geometric operations implying topological change

Preventing the designer from directly interacting with (and botching) a cell complex is an important idea. Along these lines, others have considered automating lower-level changes to the topological representation through the use of higher-level geometric operations. Toriya, *et al.*[TTSC91] propose CSG-flavored boolean operations on the skinned polyhedral shells above. They successfully avoid the expense of computing accurate surface intersections by making a coarse polyhedral approximation to the smooth intersection curve, deducing topological changes using this coarse curve, and ultimately incorporating the polyhedral curve into the boundary of the primary shape (which is then re-skinned with a smooth surface). Bonner, *et al.*[BJWK93] use higher-level operations to interactively build up surface topology for tubular structures (such as in auto body frames). Tubular structures are represented as quiltworks of patches offset from an underlying network of backbone curves. When the designer attaches one tube backbone to another, this triggers appropriate surgery on surface patches around the join to adjust the surface topology. The resulting models combine independent explicit faces (the flat sides of frame tubes) connected by dependent faired blending faces.

2.4 Parametric surface fitting

From the discussion above, we see that free-form modeling approaches that allow direct control of arbitrary smooth surface topologies do so by embedding 2D cell complexes in 3D space. This is often accomplished by constructing a piecewise smooth surface, with a separate patch for each cell of the topological domain complex. In order to treat arbitrary topologies, we will not attempt to view the surface as a single map from a triangulated subset of $\mathcal{R}^2$ to $\mathcal{R}^3$. It can be shown that in imposing a (u, v) coordinate system over an arbitrary genus surface, one cannot avoid introducing a number of singularities, this number being related to the genus of the surface. This is known colloquially as the "hairy ball" theorem, which says that you cannot comb the hair on a hairy ball without leaving a crown (singularity) somewhere (see [GP74] for a user-friendly discussion of the Poincaré-Hopf theorem, from which this derives). This disqualifies most techniques from the scattered-data fitting literature, as they depend on having a set of interpolation points all referred to a common parameter plane, and construct a surface function of the form $\mathbf{s}(u, v) = (u, v, f(u, v))$. Franke and Nielson[FN90] survey this type of surface construction.

Methods that map patches of $\mathcal{R}^2$ to $\mathcal{R}^3$ are known in the Computer Aided Geometric Design (CAGD) literature as *parametric* schemes. The general functional form we seek is vector-valued, $\mathbf{s}(u, v) = (x(u, v), y(u, v), z(u, v))$. Many particular functional forms and constructions have been developed as parametric schemes, and an excellent survey is Mann, *et al.*[MLL$^+$92]. Some additional parametric approaches not covered in the survey include subdivision surfaces[Doo78, CC78, HDD$^+$94], generalized B-splines[LD90, Loo94], and, though not exactly a parametric approach, piecewise smooth implicit surfaces that interpolate triangulated data[Baj92]. These and the surveyed approaches are all *local*; that is, each patch is constructed independently, using only data that is nearby. An important observation of Mann, *et al.*is that *none* of the surveyed local schemes produces very fair composite surfaces. That it should be so difficult to find a local construction that leads to a globally fair surface is not so surprising [1]: though we've no precise definition of fairness, and such a notion is ultimately subjective, fairness has much to do with the distribution of curvature over the entire surface[MLL$^+$92, Mor93]. Locally constructed surfaces can

[1]The surprising result was that wildly differing construction schemes all exhibited similar shape defects: curvature tended to be concentrated near patch boundaries.

be arbitrarily bad with respect to this global criterion. In these constructions, curvature information is not being communicated between adjacent patches, much less globally.

2.5 Optimal shapes: variational curves and surfaces

The inability of local patch schemes to produce high-quality composite surfaces leads us to consider global approaches to surface patch construction. Information about surface position and shape will be shared among neighboring patches, requiring that all patch shapes be solved for simultaneously. This is much more computationally involved than a local scheme; but if we care about such global shape properties, then clearly a global scheme is indicated. A survey by Lounsbery, *et al.*[LMD92] compares some of the local schemes surveyed in [MLL$^+$92] with Moreton's high-quality global scheme[Mor93] (discussed below), and makes similar observations.

Global schemes involve geometric quality measures evaluated over the surface as a whole, rather than considered patch-by-patch. They lead to constrained optimization problems, in which some global shape measure (*e.g.*, fairness) is optimized subject to local constraints such as point or curve interpolation or global constraints on, *e.g.*, area or volume. The mathematics involved is the calculus of variations, which in its most general form allows us to pose a problem whose solution is a *function* $y = f(x)$ such than an integral objective function $\int g(x, y, y', y''...)dx$ is at a maximum or minimum subject to side conditions (constraints) on f.

The calculus of variations has its roots in several problems related to shape, including the catenary arch (an optimal load-bearing shape), the brachistochrone (an optimal shape for a roller-coaster track), and the shapes assumed by thin rods (*elastica*) under various bending moments[Boy91]. This latter is particularly relevant here, because physical splines — thin flexible strips guided through rotating sleeves called "ducks" — have long been used in drafting and design to construct fair curves that pass through specified interpolation points. The first mathematical model for such physical behavior was proposed by James Bernoulli in 1694, that the bending moment of an elastica was proportional to the radius of curvature produced. Some years later Daniel Bernoulli cast this one of the earliest variational minimum principles, suggesting that all varieties of elastica could be described as taking on shapes that minimize total bending energy $\int \kappa^2 ds$, the integral of squared curvature over their lengths. (see [Tru83] for a brief history of the elastica problem).

Elastica have long served as a basis for numerical curve fitting in CAGD, yielding fair curve functions that emulate the geometric behavior of physical splines[Far90]. More general use of minimum principles to define variational shapes[2] for design has been taken up in earnest within the past several years (see the survey in [HS90]). Variational modeling approaches are best distinguished from one another by the constraints and objective functions they use to specify shape, and the particular explicit curve and surface representations they use in constructing solutions or approximate solutions, and we will have a closer look at each of these below. Not all of the work reviewed below is cast explicitly in terms of

[2]The term "variational shape" is also used in the industrial design literature [LGL81, BS91] in a different sense, to refer to a continuum of shapes that satisfy mechanical tolerances specified as inequality constraints. This should not be confused with our usage here as the solution of a variational optimization.

variational optimization; but in the cases where the shape optimization has been formulated directly in terms of some explicit piecewise surface representation (as in, *e.g.*, [DWS93]), it is often true that a variational problem has actually been discretized, and a variational description serves to unify these disparate approaches.

2.5.1 Solving for variational shapes

Though we will discuss a variety of minimum principles and variational shape problems below, there will be no real need to discuss direct solution techniques. Though there are explicit solutions for some of the curve forms developed below, the same is not true for surfaces, for fairly deep reasons. There is a direct connection between variational surface problems and 2D systems of partial differential equations [3]. Except for the very simplest of topological domains and restricted classes of objective functions, explicit solutions cannot be found for these variational shape problems, for the same reasons that it is difficult to write down direct solutions for any but the simplest systems of partial differential equations. For the kinds of curves and surfaces we will consider here, the best we will be able to do is construct approximate solutions.

Approximations to variational curve and surface shapes are typically computed using either a finite-difference approach over a set of discrete sample points (*e.g.*, [KWT87, BCGH92, Ter86]), or a finite-element approach over a piecewise smooth polynomial curve (*e.g.*,[Nie74, CG91, Mor93]). At a high level, we will not make much of an issue over the differences between the two approximation approaches, because they can be viewed as the same basic approximation method (minimization of a residual error term) applied to different curve/surface representations. Because of the way derivatives are computed in a finite-difference scheme, it can be seen as numerically equivalent to a finite-element method using a C^0 (continuous, but not smooth) collection of polynomials, one for each data point neighborhood[ZM83]. Both approaches end up computing a "best-fit" of their piecewise curves to the given variational shape by re-expressing the variational objective as a set of algebraic equations written in terms of explicit curve and surface elements, a procedure known as *discretization*, then solving for the parameter values that minimize the discretized integral. We will take up the actual numerical details in Chapter 5. The reader may also consult Becker, *et al.*[BCO81], an excellent general finite element text.

The amount of computational effort needed to compute such approximations will be of concern to us, since we want to compute them at interactive speeds. There are several contributing factors, all related to the complexity of the algebraic equations that arise when the variational problem is expressed in terms of a specific approximating representation. These include the numerical form of the approximating representation, the complexity of the objective itself, the numerical form of constraints placed on the solution, continuity constraints imposed between neighboring elements of the representation, and the topological complexity of the domain. In reviewing some of the modeling approaches below, we will see a fundamental tradeoff: schemes that are capable of generating high quality surfaces of arbitrary topology do not compute their surface approximations at interactive speeds;

[3]They are related via the Euler-Lagrange equation, a basic tool in the calculus of variations. Strang[Str86] includes introductory chapters on the Euler-Lagrange equation, and on solution techniques, both symbolic and numerical.

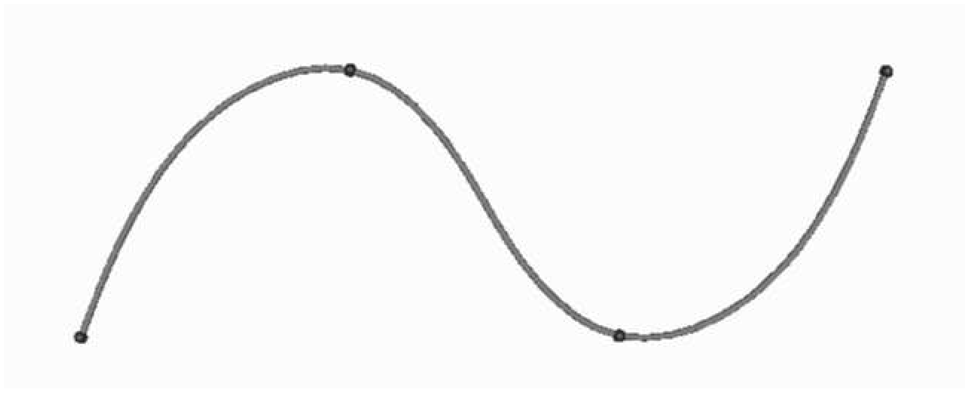


Figure 2.6: A curve that minimizes the geometric elastica functional subject to the constraint that it interpolate the given control points. This functional approximates the bending energy of a physical spline.

and schemes that run at interactive speeds cannot produce generally acceptable surfaces, or else are topologically limited. We will return to these issues after first considering current approaches to variational shape description in more detail.

2.5.2 Fairness functions

Mathematical measures of curve and surface fairness, like Bernoulli's measure of bending energy, are formulated as local measures that are integrated over an entire piece of geometry. Thus, a fairness function takes the shape of a curve or surface as input and returns a single number measuring its fairness. In our work, the particular value returned for a single curve or surface needn't be — and often isn't — meaningful in and of themselves; what we really want from the objective function is a relative ranking of different curve/surface shapes by fairness. The fairness functions discussed below all evaluate to 0 for the fairest possible shape, and greater than 0 otherwise. We may then seek the fairest shape possible by minimizing such a measure over the space of possible shapes.

Elastic curve fairness functions

As mentioned, perhaps the most venerated measure the fairness of a curve is its elastic bending energy, the integral of the squared curvature with respect to arc-length,

$$E = \int \kappa(s)^2 ds = \int (\mathbf{c}_{ss})^2 ds, \tag{2.1}$$

where the subscript indicates differentiation with respect to the arc-length parameter s, and the squared vector in the integrand is shorthand for the dot-product of the vector with itself (two conventions we use henceforth). We seek a curve $\mathbf{c}$ that minimizes E for given interpolation conditions (Figure 2.6). The usual approach is to compute an approximation by minimizing E over a given algebraic curve. In this case. It is unrealistic to expect such a curve to be parameterized by arc-length, E becomes more complicated:

$$\begin{aligned} E &= \int \kappa(s)^2 ds && (2.2) \\ &= \int \kappa(t)^2 ||\mathbf{c}_t|| dt && (2.3) \\ &= \int \frac{||\mathbf{c}_t \times \mathbf{c}_{tt}}{\mathbf{c}_t \cdot \mathbf{c}_{tt}} && (2.4) \end{aligned}$$

with respect to an arbitrary parameterization t. The value of E doesn't depend on the particular way the curve has been parameterized, since it is normalized with respect to an arc-length parameterization — that is, the measure is *geometric* [4]. We will discuss simpler *parametric* measures below, measures that do depend on parameterization. But a basic assumption of this dissertation is that geometric measures are worth their added complexity. Parameterizations are an artifice in geometric modeling — something needed at a low level in order to do calculus on a curve or surface (to measure shape properties), but not something a user should ever worry about. More importantly, when considering smooth surfaces of arbitrary topology, geometric measures are absolutely necessary for the simple reason that global parameterizations do not always exist.

E is a nonlinear function, and computing an approximate minimizer will require iterative numerical techniques, much like those used to solve systems of nonlinear differential equations, and accompanied by similar difficulties[PTVF94]. Because of the complexity of working with Equation 2.1, a variety of simplified approximations have been proposed. The simplest approximation minimizes $\int (\mathbf{c}_{tt})^2 dt$, the squared magnitude of the second parametric derivative. When a linear curve representation is used (Section 2.5.4), minimizing this version of E becomes a linear problem. This linearization will generate reasonable shapes as long as the curve parameterization resembles a scaling of arc-length parameterization (that is, as long as $||\mathbf{c}_t||$ doesn't vary much over the length of the curve).

Unfortunately, this is not a good assumption to make about the parameterization of a fitted curve. Schweikert[Sch66] considered this approximation, and noted unwanted inflection points in segments with "excessive" arc-length relative to the parameterization. He recommended reigning in the arc-length by adding a *membrane* term to the objective:

$$E = \int (\mathbf{c}_{tt}^2 + \alpha \mathbf{c}_t^2) dt, \tag{2.5}$$

which opposes large values of $||\mathbf{c}_t||$ by causing the curve to contract. The differential equations describing curves of this form can be solved directly, yielding an explicit curve known as the *exponential spline*, although it is much more common in modeling systems to approximate these shapes with composite curves[NLL90, Kal93, Cel90, WW92]. Of course, for a true arc-length parameterization the tangent magnitude $||\mathbf{c}_t|| = 1$ by definition, so having

[4]More confusion in terminology: in some of the elasticity literature cited here, such a parameterization-independent measure would be referred to as *intrinsic*. In differential geometry literature, on the other hand, *extrinsic* and *intrinsic* describe measures that do or do not depend on a manifold's embedding in ambient space. A curve's curvature at a point would be considered intrinsic by the former, extrinsic by the latter. In CAGD literature, where parameterization-independence and embedding-independence are both useful notions, *intrinsic* retains its differential-geometric meaning, and parameterization-independent extrinsic measures are called *geometric* (well, usually...).

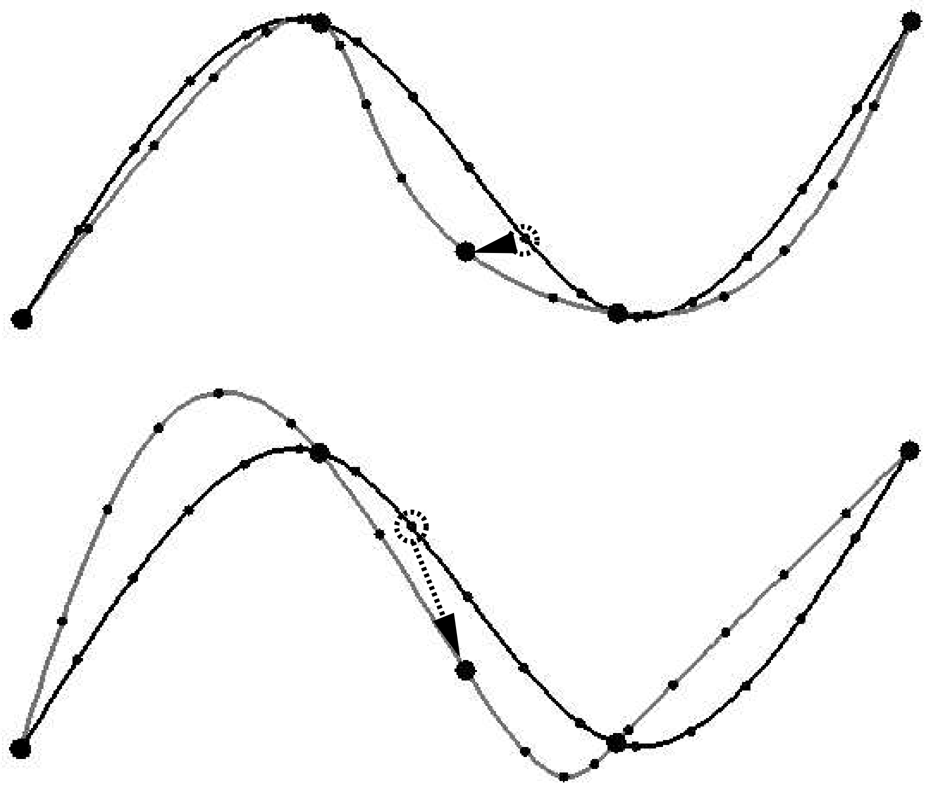


Figure 2.7: Composite B-spline curves that minimize the Schweikert functional subject to interpolation constraints (large dots), and their associated uniform parameterizations (small dots). Each black curve has had a new interpolation constraint added, placed identically in space but at different curve parameter values (the resulting shapes are shown in grey). The lower pair demonstrates how a poorly parameterized constraint point degrades the resulting global curve shape.

a membrane term drive it to 0 is not really what one wants to do. We have experimented with penalizing the deviation of $||\dot{c}||$ from 1, but this does not work well for low-degree polynomials (which simply do not seem to want to take on arc-length parameterizations).

Any linearization of E will have the drawback over the original geometric form of sensitivity to the parameterization of the fitted curve. A popular way of addressing this limitation, when fitting composite curves, is to construct the composite domain to approximate an arc-length parameterization using a *chordal approximation*[Far90, Epp76, PS83, RF89, SM91, HB91b]. As an example, consider a sequence of discrete sample points to be used in a finite-difference scheme (Figure 2.8). Rather than spacing these points at equal parametric intervals as is often done in smooth interpolation methods, a chordal parameterization approximation takes the straight-line Euclidean distance between the points as their parametric separation. A chordal parameterization converges on an arc-length parameterization in the limit as smaller and smaller parametric intervals are considered.

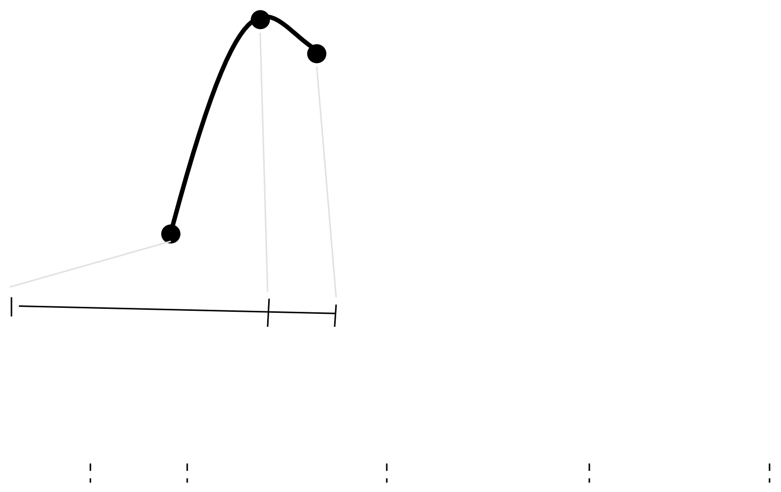


Figure 2.8: A curve the resulting chord-length parameterization of the indicated points. The parametric distance between successive points is taken to be their Euclidean distance.

This is an approximation and the actual fitted curve will of course have greater arc-length than the chordal parameter intervals (unless the fitted curve is a straight line). If the parameterization remains fixed but the curve is subsequently reshaped, this approximation may become arbitrarily bad, and the 2nd derivatives cease to be good approximations to curvature. The overall quality of the shape then decreases, as illustrated in Figure 2.7. One way of minimizing this effect is to allow constraint points to "slide" on the curve as it is re-shaped, something we consider when we take up constraints in Section 2.5.3. A different way of addressing this is to actually re-build the composite parameterization from the ground up when the curve changes shape, an approach we develop in Chapter 5.

Higher-order curve fairness functions

Finally, we should note that 2nd-order elastica are not the last word in fair curve shapes. Other authors have made use of higher-order derivatives, including 3rd-order ([Meh74, HB91b, Mor93]) and 4th-order (mentioned in [HS90]) terms. Moreton[Mor93] pointed out that minimizing bending energy alone tends to concentrate curvature near the endpoints of a faired region. He uses a curve fairness functional that measures the *variation* of curvature over the the interval, $\int (\frac{\partial\kappa}{\partial ds})^2 ds$. Minimizing this functional yields curves that seek constant curvature, and it finds circles and lines whenever they are possible as solutions to the given interpolation problem, with results that look much more "draftsman-like" than those arising from strain energy. This is quite a bit more complicated an objective function than the elastica, and although computing approximations is algorithmically similar, the calculations are correspondingly more delicate.

Thin plate surface objectives

Much of the calculus of curve shape carries straight over to surfaces through the use of cross sections. We can measure the curvature of a surface at a particular point in a particular tangent direction by slicing the surface with a perpendicular cutting plane and measuring the curve of intersection. The result is called the the *normal section curvature* in the given direction[O'N66]. One of the first results in the classical differential geometry of surfaces, due to Gauss[Spi79a], is that the sectional curvature in the neighborhood of a surface point is a smooth function of tangent direction and takes on its maximum and minimum values (the *principal curvatures* κ_1 and κ_2 — the subscripts do not indicate a derivative) in orthogonal tangent directions. The principal curvatures completely characterize the shape of the surface about a point, and in turn give rise to important geometric quantities such as Gaussian curvature ($\kappa_1\kappa_2$) and mean curvature ($\frac{\kappa_1+\kappa_2}{2}$).

The elastic curve functional of Equation 2.1 can be generalized to surfaces using principal curvatures:

$$\int (\kappa_1^2 + \kappa_2^2) du\, dv.$$

The result is commonly called the *thin plate* functional, because it approximates the strain energy of a thin elastic plate[TL25] (Figure 2.9). When expressed with respect to an arbitrary parameterized surface, this simple geometric form becomes much more complex, and we do not write it out here (see, *e.g.*, Hagen and Schulze[HS90]). This is a standard criterion for surface fairness[NR83, LP88], and has been used on a small scale as the basis for variational "twist-elimination" schemes in parametric patch constructions[HS90, KR90]. Nonetheless, Equation 2.5.2 does not seem to be widely used (if at all) to fair composite surfaces having many free parameters, perhaps because of the complexity of its parametric form and the computational difficulties of optimizing such a highly nonlinear objective function over a large number of degrees of freedom.

Much more popular are linearized forms of Equation 2.5.2 that use second-order parametric derivatives in place of principal curvatures:

$$E = \int (s_{uu} \cdot s_{uu} + s_{vv} \cdot s_{vv}) du\, dv,$$

or including a "twist term":

$$E = \int (s_{uu} \cdot s_{uu} + 2s_{uv} \cdot s_{uv} + s_{vv} \cdot s_{vv}) du\, dv.$$

As with curves, 2nd-order terms alone are not quite enough to control the surface, this time because there is no area-related feedback. Lott and Pullin [LP88] augment Equation 2.5.2 with 1st order springs to keep the surface from straying far from its original shape. Much more prevalent is the use of 1st-order membrane terms, analogous to Schweikert's curve functional[Ter86, Pot91, Nie74, CG91, WW92, HKD93]:

$$E = \int \begin{pmatrix} \alpha(s_{uu} \cdot s_{uu} + 2s_{uv} \cdot s_{uv} + s_{vv} \cdot s_{vv}) \\ + \\ \beta(s_u \cdot s_u + 2s_u \cdot s_v + s_v \cdot s_v) \end{pmatrix} du\, dv.$$

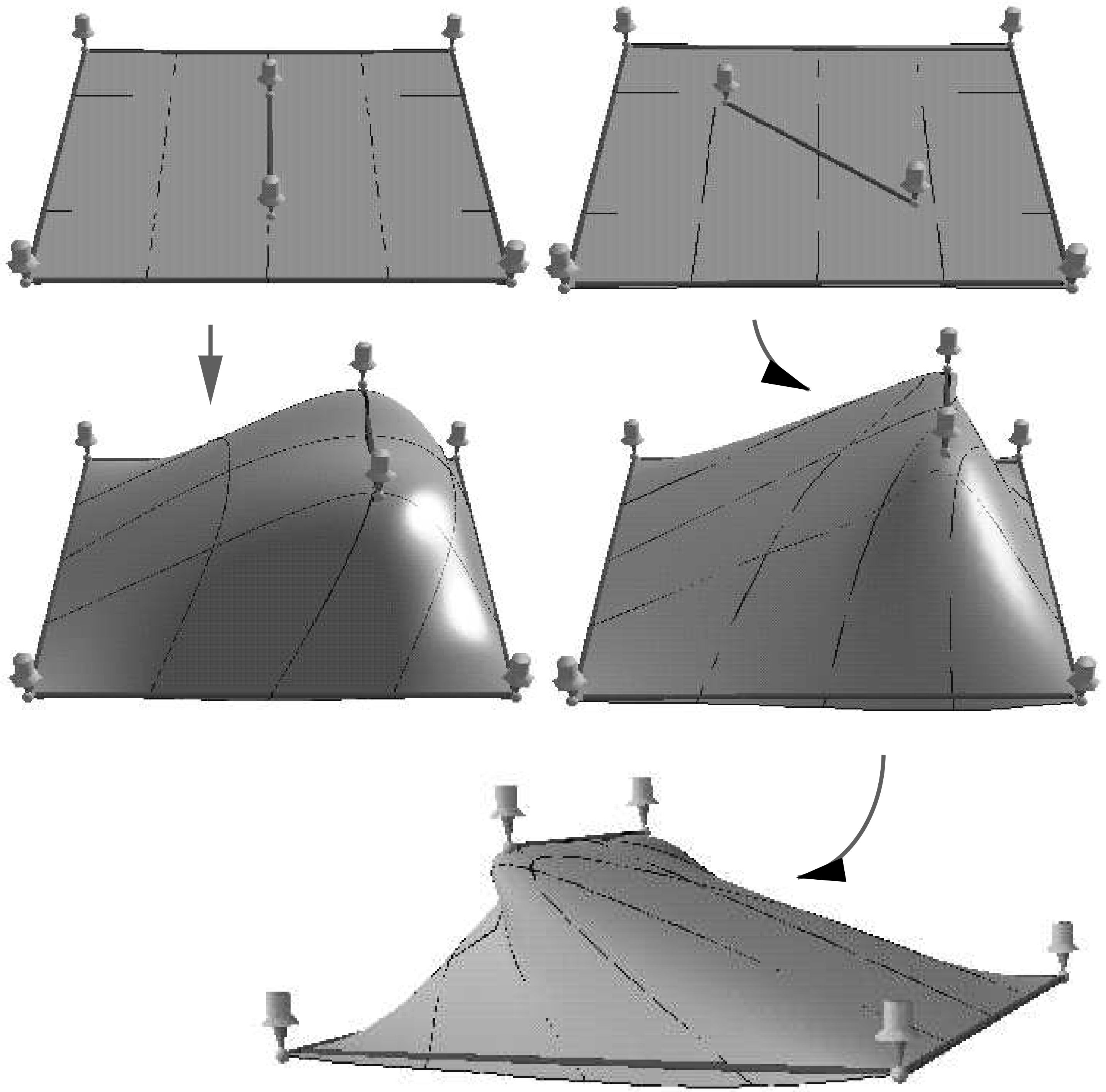


Figure 2.9: Surfaces that minimize a linearized strain energy functional can still have unpleasant shapes, because of mismatches between constraint parameterizations and the surfaces' actual metric. For the surfaces on the left and the right, a constraint curve has been added and then moved, each to a similar final position. What differs is the set of (u, v) surface coordinates associated with these constraint curves. The surface on the right suffers a serious shape defect as a result of parametric "twisting" (the bottom image is of this surface rotated to show its other side).

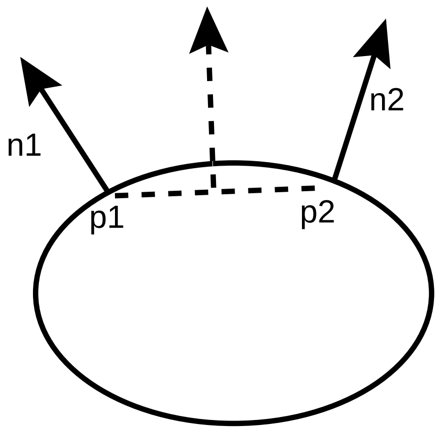


Figure 2.10: Surface points and normals used in the co-circularity potential.

This approximation has drawbacks similar to the those of the corresponding curve functional, this time when surface areas don't match a uniform scaling of their parametric areas. This can happen if interpolation points are poorly parameterized, or if the surface begins with a reasonable parameterization but is unevenly stretched in the course of re-shaping by the user, as illustrated in Figure 2.9.

Higher-order surface fairness functions

Thin plate energies, which use 1st and 2nd order information, are not the only possible fairing functions, or even the best. As with curves, these functions concentrate curvature near patch boundaries. Moreton[Mor93] proposed a fairing function that measures the variation of principal curvatures over the surface:

$$E = \int (\frac{d\kappa_1}{d\hat{e}_1})^2 + (\frac{d\kappa_2}{d\hat{e}_2})^2 dA,$$

where the $\hat{e}_i$ are the principal directions associated with the principal curvatures κ_i. Because of the explicit appearance of principal directions in this objective, it is a *much* more complicated function than its curve counterpart, and a delicate numerical optimization is needed to compute finite-element approximations of its minimizers. The method yields surfaces with an even distribution of curvature throughout, whose shapes seek circular or straight-line cross-sections. These are visually superior to surfaces produced by other published schemes, on a wide variety of interpolation problems.

Related but simpler is the *co-circularity* potential used in Szelizki, *et al.*'s oriented particle systems[ST92]. It takes position and normal vectors from two nearby points on a surface (Figure 2.10), and measures the deviation of their average direction from the normal "between" them:

$$V(\mathbf{p}_1, \mathbf{n}_1, \mathbf{p}_2, \mathbf{n}_2) = ((\mathbf{n}_1 + \mathbf{n}_2) \cdot (\mathbf{p}_1 - \mathbf{p}_2))^2 w(||\mathbf{p}_1 - \mathbf{p}_2||),$$

where the $\mathbf{p}_i$ are positions, $\mathbf{n}_i$ are normals, and $w(r)$ weights the potential by the distance between the points. This potential is 0 for points on the surface of a sphere, and at umbilic points on any surface. Unlike Equation 2.5.2, it is not 0 for cones, cylinders, or tori, because it demands that curvature be the same in all directions rather than teasing apart the surface curvature into its principal components and treating them separately. Szeliski includes a finite-element analysis of its behavior in [ST91], in which he shows that, for small deflections of a surface element represented as a graph of a function f $(s(u,v) = (u, v, f(u,v))$, the co-circularity energy over the element is approximated by

$$E \approx \int\int f_{uuu}^2 + 3f_{uuv}^2 + 3f_{uvv}^2 + f_{vvv}^2 du\, dv.$$

This is simply a 3rd-order analog of Equation 2.5.2, and the same caveats hold in applying this parametric "curvature variation" measure as an objective function.

Minimal surfaces, mean curvature, and the Laplacian operator

A different class of geometric fairing function uses the notion of a *minimal surface*, a surface interpolating a given boundary curve with the minimum possible surface area. The physical analogy here is to a soap film stretched between wire loops. Such films take on graceful free-form shapes in interpolating their given boundaries. The fundamental result in the theory of minimal surfaces is that their mean curvature, $H = \frac{\kappa_1+\kappa_2}{2}$, is everywhere 0, a straightforward exercise in the calculus of variations (see Spivak[Spi79c] for a discussion of the history of minimal surfaces in analysis).

As it happens, the published fairing approaches based on minimal surfaces are both point-based unstructured mesh schemes — a departure from our discussions so far, since we have focused on fairing functions formulated with respect to smooth surface patch parameterizations. Delingette, *et al.*[DWS93] make a geometric approximation to mean curvature over a mesh with hexagonal topology. Fairing a mesh with this geometric objective requires an iterative nonlinear optimization to drive an initial mesh to a final minimum shape.

As with elastica there is also a straightforward parametric (linearized) approximation of mean curvature. It can be shown[O'N66] that mean curvature is given by the average of the curvatures measured in any two orthogonal tangent directions at a point on the surface, not just the principal directions. Thus, given an orthonormal u, v coordinate system at a point on a surface, the condition of 0 mean curvature is described by the well-known *Laplace equation*:

$$\mathbf{s}_{uu} + \mathbf{s}_{vv} = 0$$

By integrating the square of the Laplacian $\mathbf{s}_{uu} + \mathbf{s}_{vv}$ over the surface one obtains a quadratic objective function. This is perhaps as good a rationalization as any for Mallet's proposed mesh "roughness" criterion[Mal89]:

$$E = (v^c - \sum_{i=1}^{n} v^i/n)^2,$$

where v^c is the center vertex of a mesh neighborhood and the v^i are the neighbor vertices. When this is applied to a rectangular mesh, it reduces to the finite-difference form for the squared Laplacian.

What seems to have been missed in using minimal surfaces for fair surface design is that they only make sense in situations where interpolation constraints occur along fixed boundary curves rather than at interior points. Your physical intuition about soap films should tell you that if interpolation points or curves are present in the interior, the membrane will happily crease or corner at these places. Likewise, it should be clear that one cannot directly control the orientation of the tangent plane at a point or along a curve in a membrane, as is possible with elastica (see the "tangent ribbon" discussion below). In fact, such membrane interpolation problems are standard examples of variational problems for which no smooth solutions exist[CH37]. It is not entirely clear why approaches based on minimal surfaces have actually demonstrated smooth interpolation of interior constraints. It seems safe to say that such "smoothness" must ultimately be due to approximation error.

2.5.3 Constraints

Constraints are used in variational shape specification to impose explicit control over some aspect of the shape — that it interpolate a particular point or curve, maintain tangency to some other object at a point, *etc.*. Nowacki[NLL90] surveys a variety of geometric constraints for design in conjunction with piecewise polynomial curves and surfaces. He considers local interpolation constraints involving interior points, endpoints, and tangent ribbons at surface patch boundaries; and global *integral* constraints such as surface area or volume. Similar local constraints, and also a general curve interpolation constraint are considered by Celniker and Welch in [CW92]. All this work uses parametric curves and surfaces, and the constraints themselves are *parametric*, formulated with respect to the natural parameterization of the curve or surface:

Point constraint on surface:

$$(\mathbf{s}(u_0, v_0) - \mathbf{X}_0)^2 = 0,$$

where $\mathbf{X}_0$ is a fixed point in space, and $\mathbf{s}(u_0, v_0)$ is a point on the surface with fixed u, v coordinates u_0, v_0.

Curve interpolation constraint:

$$\int (\mathbf{s}(u(t), v(t)) - c(t))^2 dt = 0,$$

where the parameter t and the surface curve $(u(t), v(t))$ associate points on s with points on the constraint curve $c(t)$. It is common to refer to the parametric coordinates $(u(t), v(t)$ and (u_0, v_0) as *material* coordinates, as if (u, v) grid lines were painted on the "material" of a deformable curve or surface. Figure 2.9 illustrates these constraints applied to a surface patch.

This need for material coordinates is an artificial constraint if one wants to specify merely that *some* point on a curve or surface coincide with a fixed point in space, or that a surface should contain a given space curve along some cross-section within a given region. As we discussed earlier, having "good" material coordinates for constraints is especially important when computing shape approximations. Chord-length constraint parameterizations are often used to address this issue for static data fitting; but clearly the approximation can become arbitrarily bad if these interpolation points are used as shape handles and manipulated by the user after being assigned fixed material coordinates.

Ideally, the material coordinates of the interpolation points would remain free, much like physical splines are left free to slide within their positioning ducks to a minimum energy shape. Something like this was considered for scattered data fitting using curves (interpolation constraints but no shape optimization) by Hoschek[Hos88, HSW89] and Sarkar and Menq[SM91], and for surface fitting by Plass and Stone[PS83] and Rogers and Fog[RF89]. The technique has been referred to as "parameter optimization," because one ends up computing an optimal set of material coordinates for the interpolation points such that overall fitting error is minimized. We'll refer to such parameterless constraints as *nonparametric interpolation constraints.*

Curve and surface tangent vectors may be constrained with formulae similar to Equation 2.5.3, using parametric derivatives of the curve and surface functions. These allow *tangent ribbons* to be controlled along the edge of a surface, and may be used to "stitch" curves or surfaces together smoothly by matching derivatives along a shared boundary. They are known, not surprisingly, as *parametric continuity* constraints[Far90], with nth-order parametric continuity typically designated C^n in CAGD literature. A more general continuity condition uses geometric rather than parametric quantities, and is therefore known as geometric continuity[DeR90]. As an example, first-order (G^1) geometric continuity equates normal vectors along a shared boundary:

G^1 surface continuity constraint:

$$\int (\mathbf{N}(\mathbf{s_1}(u_1(t), v_1(t))) - \mathbf{N}(\mathbf{s_2}(u_2(r(t)), v_2(r(t)))))^2 dt = 0, \tag{2.6}$$

where

$$\mathbf{N}(\mathbf{s}(u,v)) = \frac{\mathbf{s}_u \times \mathbf{s}_v}{\|\mathbf{s}_u \times \mathbf{s}_v\|},$$

$\mathbf{s}_1$ and $\mathbf{s}_2$ are surfaces intersecting along their respective parametric curves (u_1, v_1) and (u_2, v_2), and $r(t)$ is a reparameterization of (u_2, v_2) such that t identifies coincident points on both surface curves.

Lastly, there are the so-called *integral* constraints, such as constraints on the area contained in a closed planar curve, area of a bounded surface, or volume contained within a closed surface:

Surface area constraint:

$$\int_s \|\mathbf{s}_u \times \mathbf{s}_v\| du dv = A_0$$

These are geometric constraints, as the area of a surface, or contained volumes, do not depend on the particular parameterization used in performing the integration.

Applying constraints

In applying any of the constraints we have been discussing to a particular surface representation, as when computing an approximation to a constrained variational surface, it may happen that the given representation cannot exactly satisfy the constraint. For example, if a polynomial surface patch is constrained to interpolate a circular arc, the interpolation cannot be exact. Generally, one must be satisfied with minimizing constraint error, *e.g.*in the least-squares sense, rather than satisfying constraints exactly.

For parametric constraints — involving fixed material coordinates for point or curve positions and parametric derivatives in fixed directions — a least-squares formulation of the constraint leads to a quadratic minimization problem. As with the linearized objective functions of the previous section, this yields a set of linear constraint equations to be solved for a zero error gradient. The problem of minimizing a quadratic objective subject to linear equality constraints is a straightforward numerical optimization problem, requiring the solution of a single linear system, assuming a linear curve or surface representation is being used.

There are any number of approaches to setting up and solving this system[PTVF94, GVL89, LH74, MS78, GMW81]. Perhaps the simplest directly combines the constraint error with the shape objective as a penalty term to be minimized[WW92, MS92]. The drawback with penalty approaches is that the constraints must compete with the objective, and thus will not generally be satisfied exactly. One must heavily weight the constraint error to increase its dominance over the objective terms, and this can lead to badly conditioned equations[GVL89]. There are several approaches that enforce constraints exactly. In some curve and surface representations, constraints are "built in" so that they cost nothing to enforce, as with parametric continuity constraints for B-splines (below), or control-point interpolation constraints for Catmull-Rom splines (below). Linear constraints can also enforced by *constraint reduction*[GMW81], in which a matrix is computed that transforms the original constrained representation parameters into a smaller set of unconstrained parameters for which the constraints are built-in (see [Cel90, CW92] for applications to curve and surface fairing). Finally, the technique of *Lagrange multipliers*[Str86], which augments the representation parameters with additional degrees of freedom, one per degree of constraint, has also been used to enforce parametric interpolation and tangent constraints[WW92]. We will use constraint reduction and Lagrange multipliers in the approximation scheme of Chapter 5.

For geometric constraints, things are a good deal more complicated numerically because the constraints are typically nonlinear. For nonparametric interpolation constraints, there *will* be material coordinates for the interpolation points, and these must be left free to vary. That means gradients of the constraint error with respect to these material positions inherit the nonlinearity of the curve or surface position function. To make this distinction between the linearity/nonlinearity of parametric/geometric interpolation constraints clearer, consider a general linear parametric curve $C(t)$, which by definition may be written as a linear combination of parametric basis functions $B(t)$:

$$C(t) = \sum_i \alpha_i B_i(t),$$

where the subscripts refer to the ith element of the set (this form will be discussed in more detail in the next section). The position of the curve at a fixed parameter value t_0 is a *linear* function of the α_i. The position of the curve as t_0 varies is typically a *nonlinear* function, since the basis functions B_i are typically nonlinear in their parameter t. If we formulate error or objective gradients with respect to free t parameters rather than the α_i, they pick up this nonlinearity. Such constraints cannot generally be solved directly, but must be satisfied by iteratively minimizing constraint error.

Geometric continuity conditions suffer a different kind of nonlinearity, arising from their use of vector cross-products and division by vector magnitudes to produce unit vectors

(Equation 2.6). Additionally, there are *corner compatibility conditions* that must be satisfied where multiple patches meet at a single vertex[Pet91]. Essentially, all surfaces must meet at a 2nd-order smooth neighborhood at such a node if their 1st-order cross-boundary derivatives are to be compatible. It is most common to directly construct surfaces that satisfy these constraints ([SS87, Pet91, DeR90, LD90]). But combining such a construction with a global surface fairing scheme is daunting because the constructions typically over-constrain the surface in the name of definiteness, leaving few if any degrees of freedom available to minimize the objective (subdivision surfaces being a notable exception[HKD93], discussed below). It will generally be more practical and productive to satisfy such constraints through iterative relaxation[MS92] of an under-constrained surface.

2.5.4 Linear curve and surface representations for approximation

Several times in the previous discussion we have made reference to *linear* curve and surface representations as being desirable for approximating variational shapes. Such a curve or surface can be written as a linear combination of a set of parametric *basis functions*:

$$\mathbf{c}(t) = \sum_i \alpha_i b_i(t),$$

or

$$\mathbf{s}(u,v) = \sum_i \alpha_i b_i(u,v),$$

where the subscripts indicate the ith element of the set. There are a variety of representations in use in CAGD that satisfy this definition. We'll not go into any real detail, beyond discussing a few broad classes of basis functions — see Farin[Far90] for a survey of the mathematical particulars. Here we are interested in how linear curve and surface elements can be pieced together into piecewise smooth composites, and ultimately be used to approximate variational shapes. It worth mentioning at this point that we will not be using *any* of the piecewise smooth schemes discussed here; but we'll need this background in explaining our choices later.

Polynomial curves

Perhaps the most commonly used basis functions are univariate polynomials — B-spline, Bezier, Catmull-Rom, or the simple monomial functions $(1, t, t^2, t^3, \ldots)$. Though all generate polynomial curves, their α_i have different geometric meaning as points distributed on or about the curves, and are referred to as *control points* (except for the monomial's α_i, which are simply polynomial coefficients). The number of control points in a curve segment is always one more than the polynomial degree of the curve, just as a degree-n polynomial has $n-1$ coefficients. As Figure 2.11 illustrates, B-spline and Catmull-Rom curve segments can be chained end-to-end to make piecewise smooth composites by sharing control points between neighboring curve segments. These are "built-in" parametric continuity conditions. Geometrically continuous composites may be built up by construction, or by setting up and solving continuity-constraint equations between (no longer shared) control points of neighboring segments. Piecewise smooth Bezier curves or blends of monomial curves, with either parametric or geometric continuity, must be built by construction or though explicit constraints. There is no analog to control-point sharing here, although, given that we are

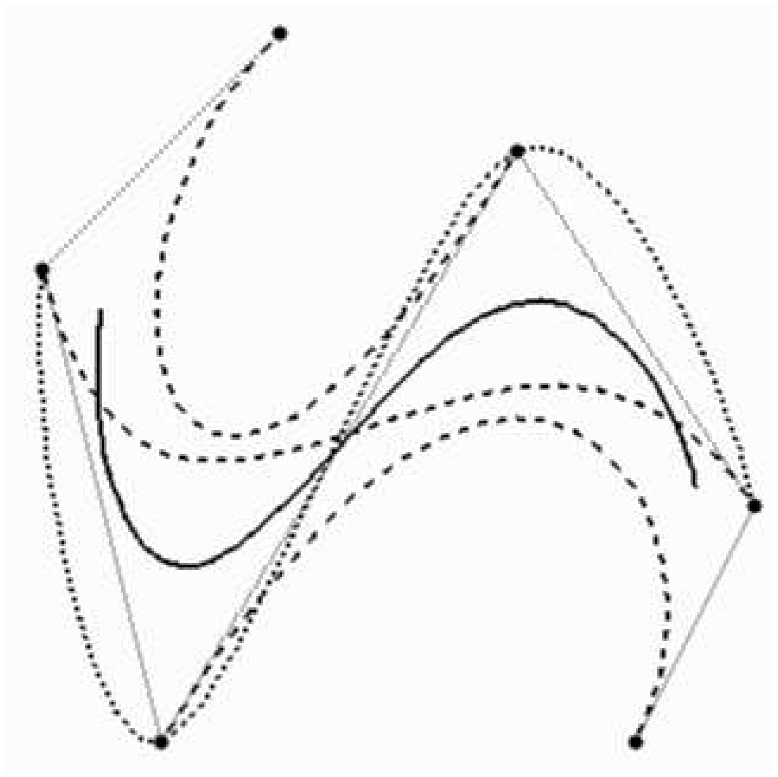


Figure 2.11: Polynomial curves: cubic Bezier (dashed), Bspline (solid), and Catmull-Rom (dotted) spline segments are computed for the same sequences of control points. Each cubic segment uses 4 successive control points, so the 6 control points yield sequences of 3 curve segments. The three Bezier curve segments are discontinuous. The three Catmull-Rom and B-spline segments are C^1 and C^2 continuous, respectively, because of control point sharing.

more interested in geometric continuity constraints, the Bezier and polynomial forms are simpler to work with in setting up the constraint equations.

Polynomial surfaces

Linear surfaces may be built up from such univariate basis functions through the *tensor product* operation:

$$\mathbf{s}(u,v) = \sum_i \sum_j \alpha_{ij} b_i(u) b_j(v),$$

where α_{ij} is now a 2D rectangular array of control points. As with the 1D forms, these patches can be stitched together into parametrically continuous sheets by control point sharing (B-spline, Catmull-Rom), or by constraints or direct construction (Bezier, monomial). Unfortunately, because patches are rectangular and can only be joined into rectangular meshes (4 patches must meet at each interior vertex), the only topologies possible with simple parametric continuity conditions are sheets, cylinders, and tori. More general topologies require interior vertices where more or fewer than 4 patches meet smoothly, or the use of other than 4-sided patches (*i.e.*, by collapsing an edge of a patch control mesh to create a "triangular" patch). Both approaches introduce degeneracies into the relationships between partial derivatives at the shared vertices, and cause the parametric continuity condition to break down. The smoothness of such neighborhoods must be ascertained using geometric continuity conditions.

Other linear surface patches do not rely on the tensor-product construction, but are instead use true 2D basis functions. Most widely used are Bezier triangles[Far90], a triangular generalization of the Bezier curve construction. Triangular patches are more convenient than rectangular ones for constructing nontrivial surface topologies, and the complexity of maintaining geometric continuity constraints or evaluating a geometric objective is essentially the same for both.

Subdivision Surfaces

A subdivision surface is defined, not as an explicit parametric function, but rather as the limit of repeated refinement of an initial control mesh[Doo78, CC78, Loo94, HDD$^+$94]. The resulting surfaces are smooth, can be of arbitrary topological type, and are linear in the initial mesh vertices (though, like their B-spline cousins, they do not directly interpolate these points). It is possible to describe these iterated subdivision steps as repeated applications of a particular linear transformation to the patch control points, and to analyze the limit behavior of the iterated transformation to determine the ultimate locations of surface points and normals corresponding to the original mesh vertices. (see [HKD93]).

Work on global shape optimization with subdivision surfaces is very recent. Hoppe, *et al.*[HDD$^+$94] attract such a surface to an unorganized set of points, minimizing the difference between limit positions of surface points and nearby data points. Halstead, *et al.*[HKD93] compute fair Catmull-Clark subdivision surfaces by minimizing the parametric thin-plate functional of Equation 2.5.2. The difficulty here is in evaluating the integral (recall that there is no explicit parametric equation for the surface). They show how to compute the integral and necessary derivatives analytically, by analyzing the eigenstructure of the subdivision matrix. It is not quite clear what it means to minimize such a parametric function over a composite, geometrically continuous (but not parametrically continuous) surface. It would appear that the objective function measures the interior of each patch independently, and any fairness across patch boundaries is a side-effect of coupling between vertices shared by neighboring patches.

2.6 Interactive variational shape design

Having discussed the variety of approaches to the individual components of a variational shape specification and its subsequent approximation, we'll now look at the several interactive variational shape design approaches[Cel90, Kal93, WW92, BB89, Fow92]. Here we are interested in approaches that compute their surface approximations in real time, as the user interacts with the variational specification by modifying constraints. All the approaches considered here make similar choices from the constraint, objective, and representation menus above. All constraint and objective functions are parametric, rather than geometric.

Bartels'[BB89] and Fowler's[Fow92] approaches allow point interpolation constraints on B-splines, but minimize no explicit fairing function. However, it is easy to show that their constraint solution technique minimizes the RMS displacement of points on the curve from their initial positions to their final positions. Celniker[CG91], Kallay[Kal93], and Welch[WW92] use the fairing function of Equation 2.5.2, and all use linear C^1 and C^2 constraints to stitch patches together (with the exception of Celniker[CG91], whose triangular patches allow almost-everywhere G^1 continuity). All offer point interpolation constraints with fixed material coordinates. Additionally, interactive curve position and tangent ribbon constraints on surfaces are developed in [CG91, CW92, WW92].

In every case, the choices of parametric functions over more versatile, higher quality geometric ones were motivated by two needs: the need to have a numerical optimization problem that could be solved at interactive speeds, and the need for robust numerics in the face of user interaction. Both of these needs are satisfied because the parametric formulations yield linearly constrained quadratic minimizations, whose solutions can be found by solving a single large linear system in a single step, rather than through iterative minimization as is required for nonlinear geometric schemes. And the solutions can be found relatively quickly: as long as no new constraints are added or old ones deleted, and as long as the representation's refinement is not changed, the matrix associated with the linear system remains constant; thus, it can be factored once, and thereafter used to solve for new surface shapes as the designer changes constraint values (*e.g.*, by moving a control point)[CG91, Fow92, CW92, WW92]. Because the solution can be found directly, it not necessary to construct a good initial guess at the solution prior to iterative minimization in order for the solution to converge. This in turn means that it is safe for the user to interactively reposition constraints *while the solver is running* without fear that the minimization process will be destabilized. Having the user's interactions with the shape specification out-pace an iterative solver's ability to keep up can be a real problem for interactive systems incorporating nonlinear optimization. In such approaches, there is an implicit dependence on the user to change parameters smoothly and at an appropriate rate, relying on visual feedback as the solution evolves to judge how quickly changes can reasonably be made[WGW90, GW91, Gle94].

The shortcomings of parametric constraint and objective functions were outlined previously. These problems are even more pronounced in an interactive modeler, where the user may severely distort a surface region while tugging on control points or curves. The designer is forced to think of the surface in terms of fixed material coordinates, and be careful not stretch the surface nonuniformly. This is exacerbated by the requirement that linear constraints also be expressed in material coordinates, preventing the surface from "sliding" across control points and curves to achieve fairer shapes. Finally, recall that those interactive modelers that rely on tensor-product surfaces[Kal93, WW92, BB89, Fow92] are incapable of representing topologies other than sheets, cylinders, and tori. In summary, current interactive schemes, though fast and stable, use simplified formulations that suffer a variety of difficulties due to their dependence on an underlying surface parameterization.

2.7 Triangulated surfaces

The surface modeling approach we develop in the next few chapters does not use smooth parametric patches as its basic representation, but rather, discrete points connected in a surface mesh. In the previous section, we mentioned some earlier mesh-based surface fairing schemes ([Mal89, DWS93]). In this section, we will review some lower-level issues involved in computing with meshes: how to estimate smooth quantities such as curvature at node vertices.

If mesh is to be treated as an approximation to smooth surface, we need to be able to estimate smooth quantities such as curvature and normals over the mesh. Simple, standard approaches to estimating gradients (tangent planes) for scattered data[Law77, Aki84, Ste84, Fra82, She68] are not adequate for our purposes, as they assume all points are referred to a single global u, v parameterization. They take as input a collection of data point triplets (u_i, v_i, f_i) and at each point compute gradients of a surface $s(u,v) = (u, v, f(u,v))$ by fitting some functional form for f to the point and its neighbors. The functions used (*e.g.*, planes, quadratics[Fra82], Shepard's surfaces[She68], and others) and the method of deciding which points are in a given point's neighborhood distinguish these methods from each other, good surveys being Nielson and Franke[NF83, FN90].

Approaches that can be applied to true 3D data are somewhat rarer. Any of the parametric surface fitting approaches discussed in Section 2.4 might be used within a mesh face, but these methods are either inappropriate or are overkill for estimating smooth quantities at mesh points only. We consider here a number of geometric and parametric fitting schemes for approximating gradient (normal field) and curvature information at 3D surface mesh vertices.

2.7.1 Mesh surface normals

Perhaps the most widely used approach to computing a surface normal at a mesh vertex takes an area-weighted average of the normals of the faces incident at the point (*e.g.*,[Ham93, MS92, Tur92, SZL93]). Variations on this idea use different weightings, *e.g.*, Akima's inverse area weighting [Aki84].

One may instead solve for the normal to a plane that best fits the points in a nearby a neighborhood, as in Hoppe, *et al.*[HDD$^+$92]. This is a standard linear regression technique, in which a set of neighborhood points Nhd and a given "center" point $\mathbf{c}$ one forms the 3 x 3 covariance matrix

$$\mathbf{S} = \sum_{\mathbf{y} \in Nhd} (\mathbf{y} - \mathbf{c})(\mathbf{y} - \mathbf{c})^T,$$

The smallest eigenvalue of $\mathbf{S}$ is the normal to the plane passing through $\mathbf{c}$ and best-fitting the points of Nhd.

2.7.2 Mesh curvature

Perhaps surprisingly, it is possible to compute the *Gaussian* curvature at a surface mesh node exactly. Maxwell[Max54] originally observed that the Gaussian definition of intrinsic curvature by means of spherical projection[Huf75] readily applies to polyhedral surfaces. Because the faces are flat and edges straight, all the intrinsic curvature must be concentrated in the vertices themselves, and it can be shown (by spherical projection) that the Gaussian curvature K at a vertex is

$$K = \frac{\beta}{A}, \tag{2.7}$$

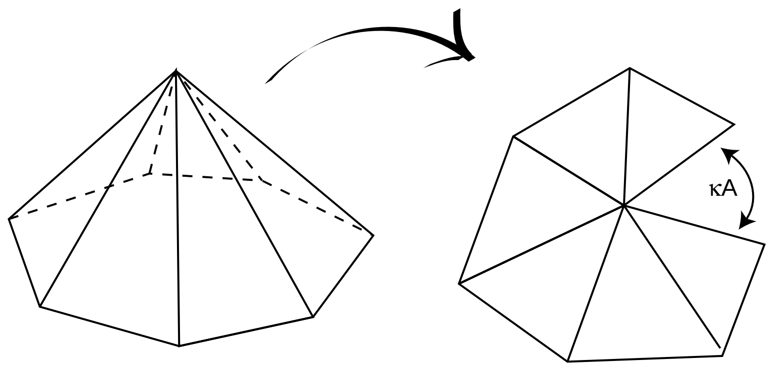


Figure 2.12: The Gaussian curvature κ at a polyhedral vertex is the "excess angle" when the neighborhood is flattened out, scaled by the neighborhood surface area.

where β is the solid angle of the vertex and A the area associated with the vertex (1/3 the area of each of its triangles). It can also be shown that β is 2π minus the sum of the vertex angles of its triangles (Figure 2.12).

In dealing with surface shape, extrinsic (geometric) curvature measures will be more useful than the intrinsic Gaussian curvature. Turk[Tur92] estimates the radius of curvature at a node by first estimating a normal at the node (face averaging) and then for each neighbor computing the radius of the sphere tangent to the line connecting the point and neighbor, and having its center on the normal ray from the point. The minimum of these radii is taken as the minimum radius of curvature at the node. Delingette, *et al.*[DWS93] fit a sphere to a tetrahedral neighborhood (all nodes are degree 3 in their meshing scheme), and claim to estimate mean curvature as a function of various relationships between the neighborhood tetrahedron and this sphere. Koenderink[Koe90] points out that the mean curvature for polyhedral surfaces can be determined exactly: just as Gaussian curvature is concentrated at the vertices, mean curvature is concentrated in the dihedral angles of the facet edges.

Estimates of smooth surface behavior at a mesh point are possible through the use of parametric fitting. Sander and Zucker[SZ90] and later Hamann[Ham93] present similar parametric fitting schemes for approximating principal curvatures at mesh nodes (and from them the gamut of intrinsic and geometric curvature measures). First, the mesh neighborhood is projected against an estimated surface normal at the neighborhood center (Sander and Zucker obtain this normal as the gradient of 3D volume data, Hamann by averaging triangle face normals). This yields a local parameterization for the neighborhood points, and after expressing a mesh neighborhood as a height-field with respect to this tangent plane a bivariate quadratic function is fit (essentially as in Lawson[Law77]). Analysis of the derivatives of this fitted function yields the desired gradient and curvature information. Stokely and Wu survey a variety of related local parameterization approaches in [SW92]. We will use a similar fitting technique in Chapter 5, and so defer further numerical details until then.

The construction of a separate parameterization in order to fit a smooth function to a mesh neighborhood seems an extra step of mathematical indirection. We experimented with fitting algebraic functions (3D quadrics) to mesh neighborhoods as a way of estimat-

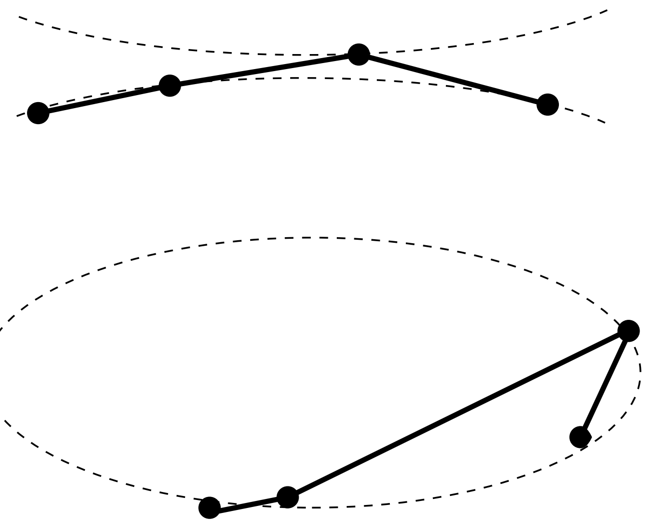


Figure 2.13: Topological pitfalls with algebraic neighborhood fitting. When fitting a general quadric, we have no reasonable way to build the desired topology into the fit, and it is quite possible to end up with topology mismatches. (Only curve fitting is demonstrated here, but the generalization to surfaces is clear.)

ing derivatives without parameterizations, a straightforward generalization of the implicit fitting techniques described by Bookstein[Boo79] and Pratt[Pra87], which are in turn generalizations of the linear regression technique discussed above. The drawback with an algebraic fitting approach is that there seems to be no inexpensive way to build *topological* constraints into the fit that would prevent algebraic projection onto the quadric surface from scrambling the neighbors' radial order (Figure 2.13). Worse, we noticed that in the case of nearly flat neighborhoods, the fitted quadric was often a hyperboloid of two sheets, with some samples on one sheet and some on the other. The only cure for this two-sheet problem is to put a nonlinear constraint on the discriminant of the quadric. This leads to an expensive iterative solution procedure rather than the direct eigenvalue computation above, and makes the technique impractical to use in our interactive modeler.

2.8 Computational mesh generation

An important part of our surface approximation scheme is mesh generation and maintenance — keeping mesh nodes distributed and triangulated so as to yield well-conditioned computations. We review here traditional grid generation approaches, and more recent work in surface sampling and triangulation.

2.8.1 Continuum grid generators and Laplace's equation

.

Numerical grid generation techniques were originally developed within the scientific computing community for the solution of partial differential equations over physical fields

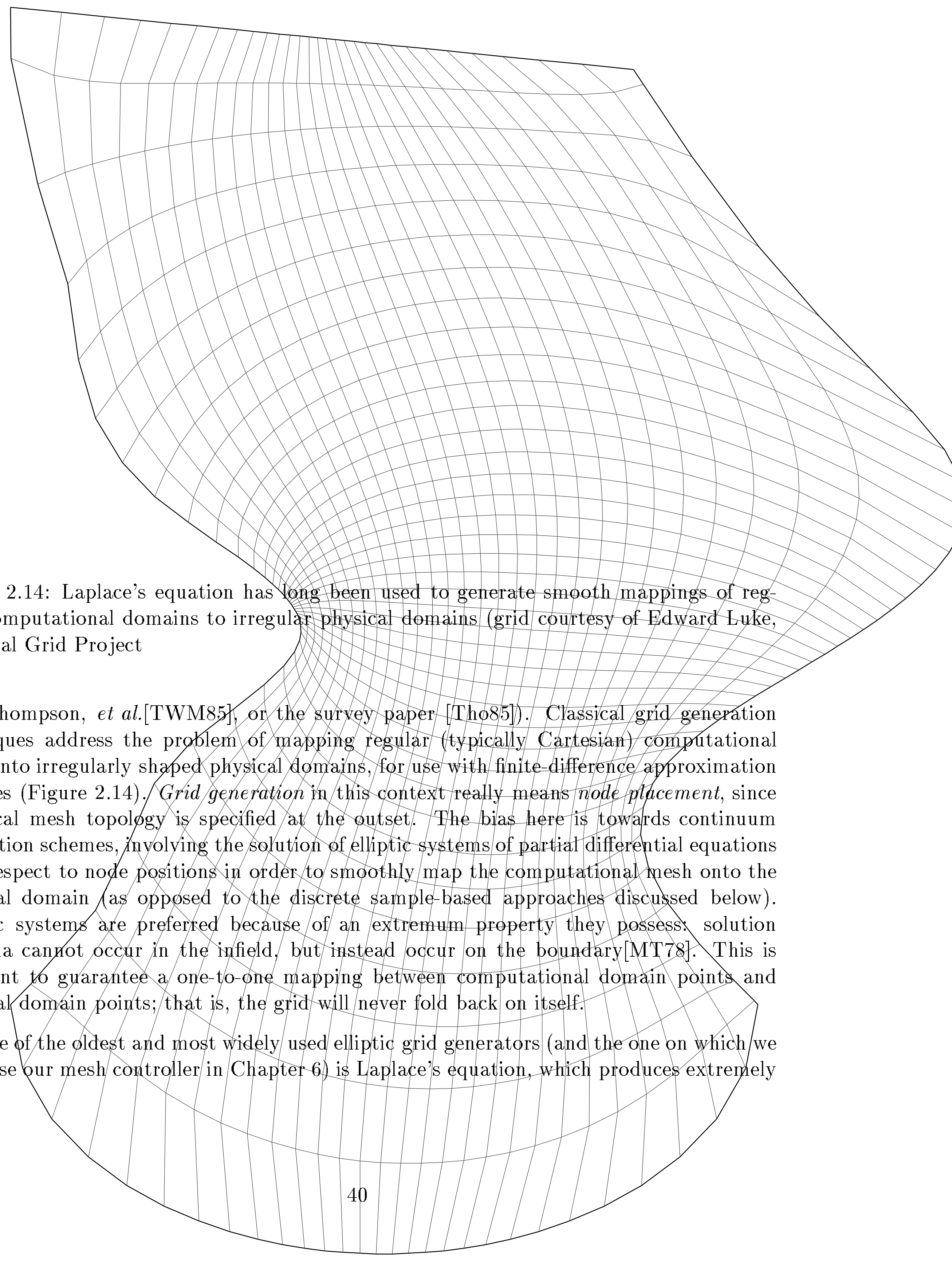


Figure 2.14: Laplace's equation has long been used to generate smooth mappings of regular computational domains to irregular physical domains (grid courtesy of Edward Luke, National Grid Project

(see Thompson, *et al.*[TWM85], or the survey paper [Tho85]). Classical grid generation techniques address the problem of mapping regular (typically Cartesian) computational grids onto irregularly shaped physical domains, for use with finite-difference approximation schemes (Figure 2.14). *Grid generation* in this context really means *node placement*, since the local mesh topology is specified at the outset. The bias here is towards continuum generation schemes, involving the solution of elliptic systems of partial differential equations with respect to node positions in order to smoothly map the computational mesh onto the physical domain (as opposed to the discrete sample-based approaches discussed below). Elliptic systems are preferred because of an extremum property they possess: solution extrema cannot occur in the infield, but instead occur on the boundary[MT78]. This is sufficient to guarantee a one-to-one mapping between computational domain points and physical domain points; that is, the grid will never fold back on itself.

One of the oldest and most widely used elliptic grid generators (and the one on which we will base our mesh controller in Chapter 6) is Laplace's equation, which produces extremely smooth coordinate maps (Figure 2.14). Its effect is easiest to explain for a 2D Cartesian grid, where we may consider two coordinate systems: the fitted curvilinear coordinate system u, v representing the regular computational domain, and the orthogonal x, y coordinate system in which the irregularly shaped physical domain is situated. Taking for the moment the (u, v) coordinate lines as functions of (x, y), the Laplacian grid generation system is

$$u_{xx} + u_{yy} = 0, v_{xx} + v_{yy} = 0.$$

Grids satisfying these equations tend to have uniform spacing away from irregularly shaped boundaries. In fact, an easy way to derive these equations is as the solution to a variational problem asking for minimum variation in coordinate spacing over the grid, an approach introduced by Brackbill and Salzman[BS82].

The Laplace system and associated variational equations for an unstructured triangle mesh in the plane are more involved, as we no longer have a single pair of (u, v) coordinate functions covering the physical domain. We will consider this in more detail in conjunction with our surface meshes in Chapter 6. We will be lead to generalize a time-worn approximation to the Laplacian grid system known as *Laplacian smoothing*[Fie84], in which nodes are iteratively moved towards the centers of their mesh neighborhoods until the distribution reaches equilibrium.

2.8.2 Surface meshes: node placement

Most of the attention in continuum grid generation literature is given to grids for 2D planar domains and 3D solid domains. Much less is said about gridding surfaces embedded in 3D, and this is certainly not helped by the unpleasant differential geometry that comes into play (see, *e.g.*, Warsi[War86, WT90] for a true debauch of indices). More recently, mesh generation for unstructured surface meshes has become active topic in computational geometry, computer aided design, and computer graphics communities. The interest here is in representing free-form surface geometries in terms of polygonal meshes — either for the pure shape representation aspect, or as a computational mesh for subsequent finite-element computations. The general approach has been to first place sample points on a surface and subsequently connect them into a surface mesh.

There are a number of schemes for establishing a fixed sampling of a surface, including random placement[Cav74], spatial subdivision methods[SZL93], or incremental "Steiner" point placement as part of a mesh improvement scheme[Che93]. We will not be interested in such static sampling schemes, because our interactive application will demand a sampling approach where point positions can vary continuously as the underlying surface changes smoothly. The near-universal time-varying approach to distributing sample points over surfaces of arbitrary topology uses the notion of *point repulsion*[RA82, Tur91, SG92b, dFGTV92, SBG93, WH94, Tur92, ST92]. Points are distributed over a physical domain by iteratively relaxing a pairwise repulsive "force" they exert on one another, inversely related to their spatial separation. At equilibrium, the points are evenly spaced over the domain. An attractive feature of this approach is the ease with which underlying features of the surface (such as curvature) can be used to control the local density of sample points and thus achieve a more efficient sampling.

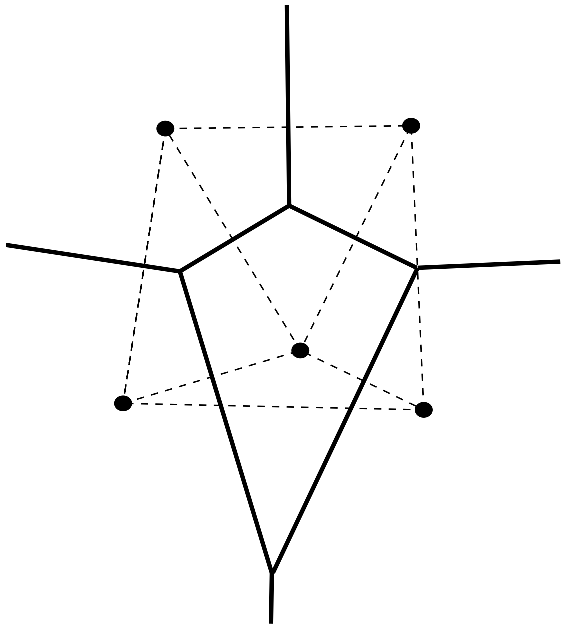


Figure 2.15: The planar Delaunay triangulation (dashed lines) is the dual of the Voronoi diagram (solid lines) of a set of points. The triangulation can be built by putting an edge between any two points whose Voronoi cells touch.

A difficulty with using point repulsion to control a dynamic surface mesh (as opposed to just sampling a surface) is that, unlike the continuum schemes, there is currently no robust way to maintain a surface triangulation over the point set as it evolves. If the triangulation is fixed, nothing prevents the mesh from from folding back on itself as points move about on the surface, and there is no penalty serving to undo such folds where they occur. Although the incremental re-triangulation techniques discussed below can go a long way towards preventing such folds from occurring, we shall see that they are stymied when a fold is accidentally introduced. Instead, current point repulsion schemes impose a triangulation over their points *a-postiori* in a variety of ways once they reach equilibrium positions. We will consider how such surface meshes might be erected over point sets after a brief detour to discuss the special case of planar triangulations.

2.8.3 Quality meshes in the plane: the Delaunay triangulation

The problem of constructing a triangulation over a set of points in the plane has been well-studied, and algorithms exist that yield triangulations optimizing a variety of quality measures. Bern and Eppstein's excellent survey [BE92] covers much of the work from within the computational geometry community for the planar triangulation problem. Perhaps the most celebrated results involve the *Delaunay triangulation* (DT), a triangulation having qualities that make it particularly desirable as a computational mesh over a set set of planar vertices. The DT maximizes the minimum included angle over the triangulation, and thus eliminates skinny triangles whenever possible, which improves the conditioning of computations over the mesh. Because of its dual relationship with the Voronoi diagram[For92]

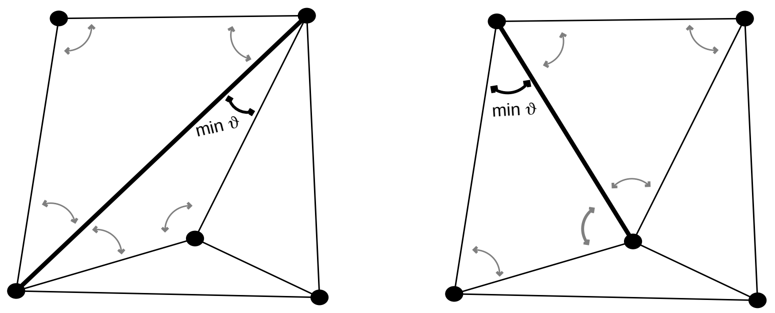


Figure 2.16: Construction of the planar Delaunay triangulation through iterative edge-flipping. The highlighted diagonal on the left is "reversed" within its quadrilateral. Flipping the edge reduces the maximum included angle, and restores the DT (right).

(Figure 2.15), the neighbor relations assigned by the DT yield an even partitioning of the plane in terms of nearest neighbor distances and relative triangle areas. There are a number of ways of constructing the (unique) DT over a set of points. One which will concern is in Chapter 6 takes advantage of the DT's max/min angle property to incrementally restore a DT from some sub-optimal triangulation through a series of edge-flips[BE92] (Figure 2.16). we will consider a generalization of this algorithm to polyhedral surfaces.

Though our ultimate interest is in a general surface triangulation, there is some related mesh generation work for the special case where a surface is erected over a single parameter plane. Here the planar DT may be used to triangulate the surface with respect to this parameterization. As an example, in the surface meshing scheme of Shimada and Gossard[SG92b, SBG93] (see also Fang and Gossard[FG92]) a point repulsion relaxation is followed by construction of a DT within the parameter plane. The price of simplicity here is that such an approach is not applicable to surfaces of arbitrary topological type.

2.8.4 Unstructured surface meshes: triangulating 3D point sets

As thoroughly solved as the planar triangulation problem may be, the situation is not nearly so rosy for general surface triangulations. The problem of taking a set of 3D points representing a sampling of some surface, and returning a triangulation of that surface is under-specified: unlike the planar case, a unique topology for the surface is not determined by the points, nor can a specific topology generally be imposed on a triangulation process (beyond the spherical topology recovered by a convex hull construction). This aspect of surface triangulation is pointed up by Edelsbrunner's *alpha shapes*[EM94], a generalization of the convex hull construction that enforces a maximum allowed edge-length (the so-called α parameter). For $\alpha = \infty$, the alpha shape is just the convex hull. For $\alpha = 0$, the alpha shape is the point set itself. Values in between allow the hull to "shrink-wrap" the point set ever tighter as the parameter decreases, resolving concavities and associated fine detail, even allowing enclosed volumes to split into disconnected pieces. A wide variety of surface topologies may systematically recovered from the *same* point set by varying this parameter. It should be noted that alpha shapes are tetrahedralizations of the space in and around a 3D point set, rather than a direct surface construction; there is no way to force all the given points to lie on the surface of the constructed alpha shape.

Szeliski, *et al.*[STT93] impose an *a-postiori* triangulation on their oriented particle systems by generalizing the empty circumcircle definition of the planar Delaunay triangulation[BE92] They look at at triangle circumspheres — the smallest sphere containing the three vertices of a given triangle, and only include the triangle in the mesh if the circumsphere is empty. We will look more at circumcircles on curved surfaces in Chapter 6. For now, it is enough to say that this "smallest sphere" test yields inconsistent results in highly curved, irregularly sampled neighborhoods, and thus cannot be relied upon to associate a unique triangulation (or surface topology) with a given point set. Chew[Che93] has developed a provably consistent generalization of the Delaunay triangulation to surfaces[Che93] that also relies on a local flatness assumption, also to be discussed further. It is not appropriate here because it requires a valid initial surface triangulation, which it iteratively improves until the surface DT definition has been satisfied. Finally, Szeliski, *et al.*'s construction includes a maximum edge length parameter, analogous to that of alpha shapes. It allows tears and holes to appear in the mesh as points move farther apart, which is a feature in their particle-based approach to recovering surface topology from unstructured CT or range data, but would be inappropriate in a modeling system that must maintain fidelity to a particular surface topology.

Hoppe, *et al.*'s surface reconstruction scheme[HDD$^+$92] may be the most successful to date at moving from an unorganized 3D surface point set to a triangulated surface. But it doesn't actually triangulate the given point set; rather, it uses the points to fit a collection of tangent planes, constructs a signed surface distance function from their union, and then triangulates a contour of this distance function.

What we take away from all of this is that, if we care about maintaining a particular surface topology as recorded in a mesh, it is not safe to discard the triangulation and then re-triangulate starting from the point set alone. In the next section we consider "topology-safe" schemes for re-triangulating a point set given an initial triangulation.

2.8.5 Unstructured surface meshes: transformation and optimization

A number of schemes for re-triangulating a given surface mesh in a topology-safe manner have been put forward. Turk's polygonal surface re-sampling scheme begins with a polygonized surface, scatters points over the surface, and re-triangulates each of the polygons to include the new scattered points in the polygonization. Nodes are iteratively deleted from the mesh, and each time the affected neighborhood is re-triangulated by projecting it onto a plane. This is not a topologically safe thing to do, especially in areas of high curvature; for this reason, topological consistency checks are performed on the resulting triangulation and the deletion is undone if it leads to inconsistent results. Similar projections and consistency checks are involved in Schroeder, *et al.*'s mesh decimation scheme[SZL93].

Much more satisfying are mesh transformation operations that do not rely on such "project-check-and-reject" tests, but are instead guaranteed to preserve the global surface topology represented by the mesh. This idea goes back to the topological work of Alexander[Ale30], who defined the mesh transforms illustrated in Figure 2.17. He shows that the order-1 moves (edge splitting and its inverse) are sufficient to transform between any two triangulations of a surface. More recently, Hoppe, *et al.*[HDD$^+$93] made one of the first principled applications of mesh transformations to the problem of surface mesh

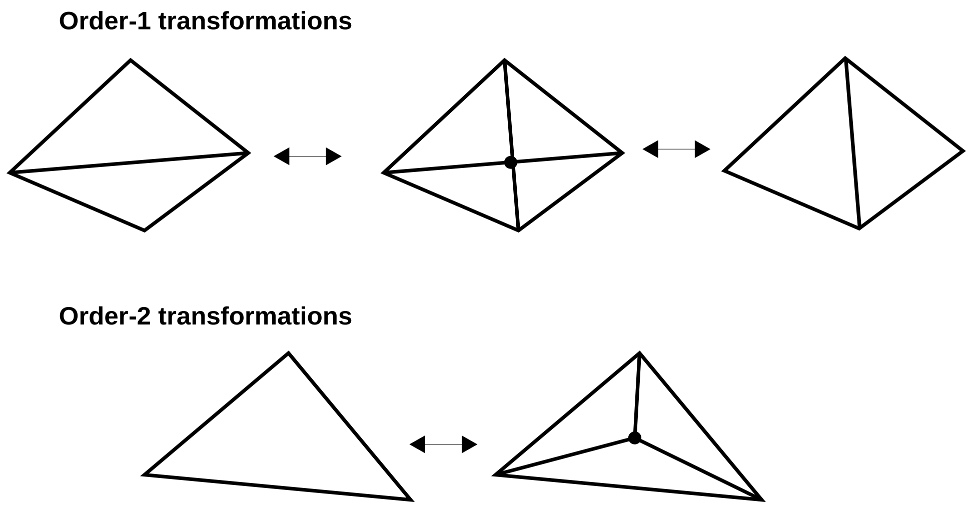


Figure 2.17: The Alexander moves. The order-1 moves operate on edges, splitting them to add nodes and collapsing them to delete nodes. The order-2 moves operate on faces, splitting them to add nodes or clipping off tetrahedral "corners" to delete nodes. The order-1 moves are sufficient to transform between any two triangulations of the same surface.

re-sampling and optimization, in the course of reconstructing a surface from scattered 3D points. They choose sequences of moves that improve the efficiency with which a mesh represents a given target shape by splitting triangles in areas of high curvature and merging triangles in flatter areas. In Chapters 4 and 6 we will need topology-safe ways of performing various bits of mesh surgery — adding and deleting nodes, and inserting p.l curves into a mesh. We will look to the Alexander moves as our basic transformation primitives, and build our mesh operations on top of these.

Chapter 3

Overview

In this chapter we give an overview of our approach to free-form shape design, beginning with the user's view. The user is presented with a very simple model for the way surfaces behave: they may be pinned down at arbitrary points and along curves, all the while maintaining globally fair shapes or locally copying externally controlled tool shapes. Detail may be added without limit, through the accumulation of additional control curves and shape tools. Additionally, surfaces may be cut up and smoothly pasted together along arbitrary curves, so that complex topologies may be built up from simpler ones.

We'll start with a construction example that shows what it is like to design a surface using these tools. We then show how such curve and surface behavior can be precisely characterized in terms of a simply formulated (if not simply solved) optimization problem. User-defined control points and curves act as geometric constraints on the possible shapes; the automatic fairing and shape-copying behaviors are then realized by optimizing the shapes subject to these geometric constraints. Shapes defined this way are sometimes called *variational* shapes [HB91a, HB93], because the resulting optimization problems are properly stated using the calculus of variations[CH37].

One of the limitations of such a variational modeling approach is that we do not generally know how to explicitly solve for optimal curve or surface shapes. This prevents us from operating directly on exact, explicit representations of the variational shapes. Instead, our modeler will construct approximations to the ideal shapes, continually updating the approximation as the user interacts with the model. In this work, we use piecewise-linear (p.l.) curves and triangulated surface meshes to build our approximations, instead of the smooth patches used in previous work, because they simplify certain aspects of our calculations and allow us to compute approximations interactive speeds.

Even though we can only present the user with approximate renderings of the shape under construction, the user will be able to create and manipulate these variational shapes directly and unambiguously, regardless of the coarseness of the approximation. This is because we will interpret each action by the user as operating on the variational specification, not the particular approximation that is being displayed.

Ultimately, we will be able to treat this representational and approximation machinery as a "black box". This will allow us to build an interactive modeler which operates on variational curves and surfaces as its basic shape representation, much as a conventional

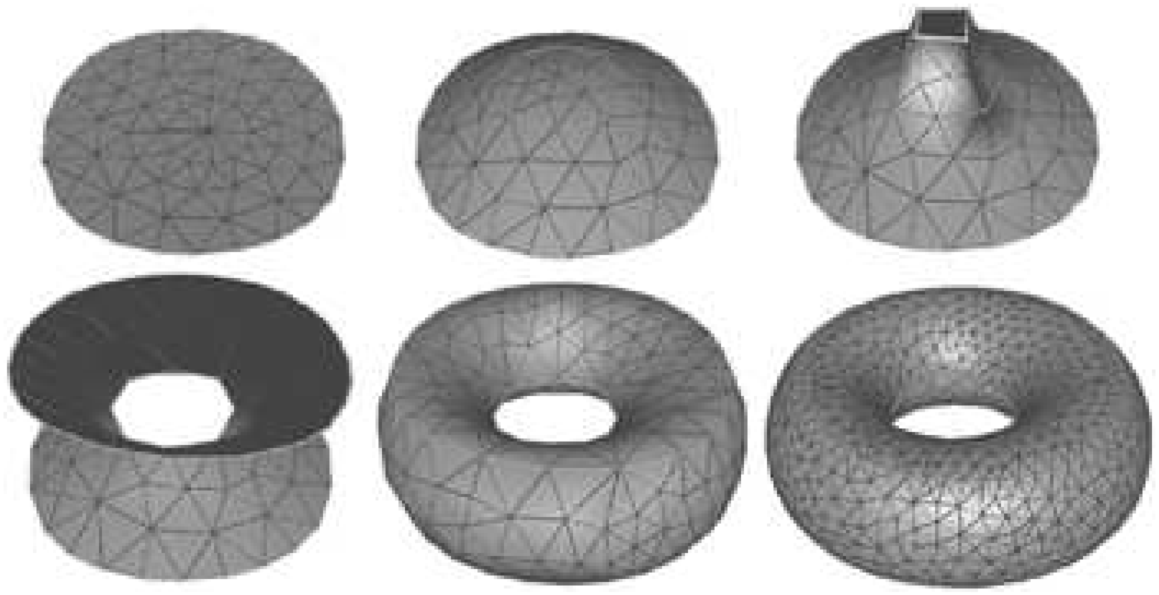


Figure 3.1: a) A closed curve is created b) the curve is skinned to make a disc. c-d) Two closed curves are drawn on the disc and elevated e) A hole is cut in center of disc and the new boundary curve elevated. f) The upper curve is expanded to match the lower. g) Two boundary curves are skinned to make a single toroidal surface passing through the three control curves.

modeler might operate on B-splines or Bezier patches. Unlike conventional modelers, this approach allows us to create and interact with with surfaces of unrestricted, mutable topology; add arbitrary amounts of detail; and incorporate a wide variety of convenient shape controls into a single structured free-form shape.

3.1 The user's view

We begin with a simple construction example that shows a user's view of our approach to creating free-form shapes. In Figure 3.1 a torus is built in a series of simple steps. These are perhaps not the most straightforward set of steps for specifying such a shape, but in this example we're more interested in exhibiting a number of useful tools during the intermediate stages.

In the first frame, four *control points* have been placed in roughly a square, and then a curve created that passes smoothly through them. The designer may re-shape this curve by moving the original control points, or by grabbing and re-shaping the curve at arbitrary points in-between (which then become new control points). In the next frame, a surface has been created that uses this curve as its boundary, yielding a disc. The disc's shape is thus indirectly controlled by the control points that shape its boundary curve.

In frames (c–d), a pair of curves are drawn on the surface — again, by defining control points and connecting them with smooth closed curves, but this time requiring that the curves be embedded in the surface. When the designer moves or re-shapes these *control* *curves*, the surface follows, always assuming a smooth shape that passes through them. In frame (e), the interior of the inner curve has been "burned out" to turn the disc into an annulus with the inner curve now serving as a boundary. This boundary is subsequently re-shaped (f), and then it and the original boundary are "skinned" with a new piece of surface that is smoothly joined to the original, creating a closed toroidal surface (g).

3.2 Shape design as functional minimization

In the previous example we spoke of curves that pass smoothly through control points, and surfaces that pass smoothly through control curves, or whose shapes copy parameterized shape tools. Such descriptions can be made mathematically precise by interpreting them as specifications for a shape optimization problem. For instance, the curve in Figure 3.1 is constrained to pass through its four fixed control points, in a specified order. Subject to these geometric and topological *constraints*, it takes on a shape that minimizes an *fairness function* (Section 2.5.2). The fairness function measures total curvature, and minimizing it causes the curve to iron out undesirable bulges or wiggles as it redistributes its curvature over its length. Similarly for surfaces: the surface in Figure 3.1 is constrained to pass through three control curves. Subject to this geometric constraint, and the topological constraint that it remain a torus, it also minimizes a curvature-based objective function to give it a fair shape.

This is an extremely concise way of describing a wide range of free-form shapes. For the curve above, a handful of control point positions, their topological ordering , and a curvature objective function are enough to completely determine the shape. Similarly for the surface, a handful of control curves, a given surface topology, and a surface objective function determine its shape everywhere. This is very different from what is done in conventional curve and surface modelers today, where free-form shapes are described as collections of B-splines, Bezier patches, or other explicit forms. Instead, these shapes are described *implicitly* as the solutions of variational optimizations. This abstracts away from the details of any particular surface representation, and makes it trivial to augment a shape with additional controls. They are simply stated as additional objective and constraint terms contributing to the variational form.

Our approach to representing and operating on variational specifications is developed in Chapter 4 and Chapter 7 (this work is also described in [WW94]). Briefly, a triangulated surface mesh records the surface topology, while sequences of connected edges record curve topology. Topological features within it — embedded curves and bounded regions — are each tagged with a "region controller" telling us whether to copy the region's shape from some other piece of control geometry or to solve for a region shape that blends smoothly with neighboring regions. Variational specifications will be built up incrementally and interactively by the user. The construction example above indicates some of the ways in which natural operations by the user can provide the information we need to construct and modify a variational specification; more examples of this kind of interactive specification are discussed below and in Chapter 7.

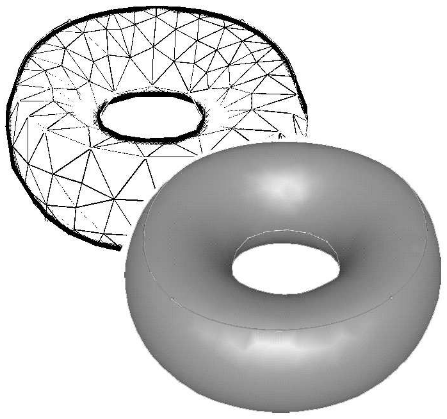


Figure 3.2: Triangulated meshes, augmented with static information about topological regions, geometric constraints and objective functions, serve as our representation of smooth variational shapes.

3.3 Approximating variational shapes

Although it is easy to pose variational problems that characterize free-form shapes, solving them is another matter. In such problems, we are solving for an optimal *function* (*e.g.*, a position function telling us where the surface is), rather than a single numerical value or discrete set of values. Techniques from the calculus of variations occasionally allow us to find explicit solutions in terms of special functions[CH37], but only for limited classes of objective functions over geometrically simple domains. Things are almost never this simple for the kinds of shape objective functions, constraints, and unrestricted domain topologies we want to consider. Even when optima are known to exist, we generally will not know how to find explicit functional forms for them (Section 2.5.1).

3.3.1 Pointwise approximation

Because we do not know how to directly solve for optimal shapes, we will instead approximate them using an explicit surface representation. Given the variational specification above, we could set up an approximation using any of a variety of smooth surface representations — piecewise smooth polynomial patches, subdivision surfaces, *etc.*, as has been done in previous work. Instead, our approach to representing approximate shapes (detailed in Chapter 5) uses the same triangulated surface meshes we that record curve and surface topology, by associating a 3D position with each node of the triangulation and thereby immersing the mesh as a piecewise linear surface.

A fair amount of machinery goes towards computing node positions to approximate variational shapes at interactive speeds, despite (or sometimes because of) the simplicity of our underlying representation. At the lowest level, we must be able to compute over the mesh as if it were a sampling of some underlying smooth surface. To do this, we use a generalized finite-difference scheme, fitting a truncated Taylor series to each nodal neighborhood in the mesh, allowing us to compute surface derivatives at the nodes. Given this local reconstruction scheme, we may then compute approximate solutions to smooth variational minimizations by evaluating the integrals numerically at mesh nodes and then solving for shapes that directly minimize this mesh objective.

3.3.2 Maintaining a quality mesh

As we will see in Chapter 5, the shape objective functions we use are really only concerned with moving surface nodes in directions normal to the fitted surface to improve the shape. They have nothing useful to say about the distribution of nodes relative to each other over the triangulated surface. As it happens, this distribution is important for the accuracy and stability of the shape approximation calculations, because the triangulated surface essentially serves as the computational mesh over which the generalized finite-difference scheme operates. This leads us to consider the problem of maintaining a uniform nodal distribution as the surface shape changes. This is posed as a variational minimization which will slide nodes around on the triangulated surface, to optimize the quality of the computational mesh. In addition to controlling the relative nodal distribution, we regulate the absolute nodal density as surface areas grow or shrink, using an automatic refinement procedure. To ensure that the mesh doesn't contain "skinny" triangles, we use a dynamic surface Delaunay triangulation scheme, so that a quality surface triangulation is present at all times as surface shape and topology changes. Incorporating such a re-triangulator into the modeler affords an unrelated but important benefit: because of the way the mesh connectivity adapts as shape changes, surface "features" are free to slide around relative to each other within the mesh (Section 7.3.4). This would allow, for example, a designer of an automobile hood to slide an air intake scoop around on the hood surface to adjust its position.

3.4 Modeling with variational shapes

The variational tools described here will be much more convenient for the designer of free-form curve and surface shapes than using fixed representation parameters like B-spline control points. The designer can create any number of control points (and control curves) and place them anywhere. Surfaces may be cut apart and stitched together into more complex global topologies. Nevertheless, even with this approach there are modeling operations that might be conceptually simple from a designer's point of view but will in fact require a coordinated set of changes to the underlying variational specification. This makes it useful to consider higher-level modeling operations expressed as simple operations on the lower-level variational surfaces. Having already developed the machinery to approximate such surfaces, we can safely hide these details in a computational black-box and use variational shape specification as the basic representation on which to build a free-form surface modeler.

In Chapter 7, we consider a number of basic modeling operations cast in terms of this variational substrate. For example, we discuss how a designer might go about specifying changes to surface topology in the course of constructing a model. The handle-attachment in Figure 7.2 is an example. The conceptually simple operation of merging the two surfaces will be automated by the modeler as a series of operations that result in an appropriately

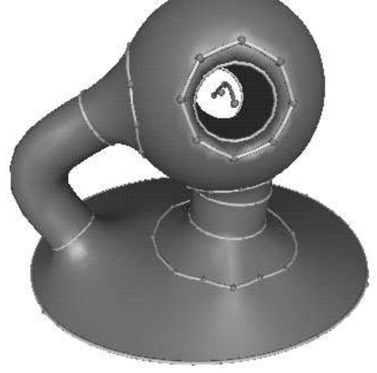


Figure 3.3: A model that mixes variational and explicitly represented shapes. A sphere tool controls a portion of the surface, to which supporting handles have been attached, and through which a hole has been drilled.

shaped hole being cut in the torus, and a blending skirt being added to join the torus and cylinder at the hole.

The cylinder tool itself demonstrates another important feature of our approach: the mixing of variational and explicit surfaces in a single model. The cylinder's shape is not defined through functional minimization, but explicitly as a constant-radius offset from a backbone curve (which is itself a variational curve, in this example). There are any number of such shapes that may properly be considered "free-form" by our broad definition, but that contain symmetries with respect to one or more defining curves, or have cross-sections that must match some specified functional form. Explicit definitions are a much more sensible way to specify such shapes, such as generalized sweeps[SK91, SK92]. Therefore, we consider ways of incorporating such explicit shapes in a variational model. The ability to smoothly join variational shapes and externally defined shapes in a common framework lets us a) "glue" these external shapes together with variational blend surfaces and b) modify their apparent topologies (independently of their explicit representations), by boring holes in them or attaching handles using trim-and-stitch operations (Figure 3.3).

3.4.1 Designing with approximations

Since we can only present a designer with approximate renderings of what are supposed to be smooth variational shapes, the question arises — how do we make sense of the designer's manipulation of these approximate shapes? The answer is that we never lose sight of the fact that the implicitly defined variational surface is the "real" surface. The user will interact only with these approximate renderings, but always with the understanding that operations will be interpreted as implicitly defining an ideal smooth shape (by modifying the definitions of the geometric constraints that frame the surface). The result is a method for directly manipulating these smooth surfaces, regardless of the coarseness of their explicit approximations.

A potential drawback of this approach is that we cannot consider modeling operations that depend on these approximations to tell us something about the exact location of the

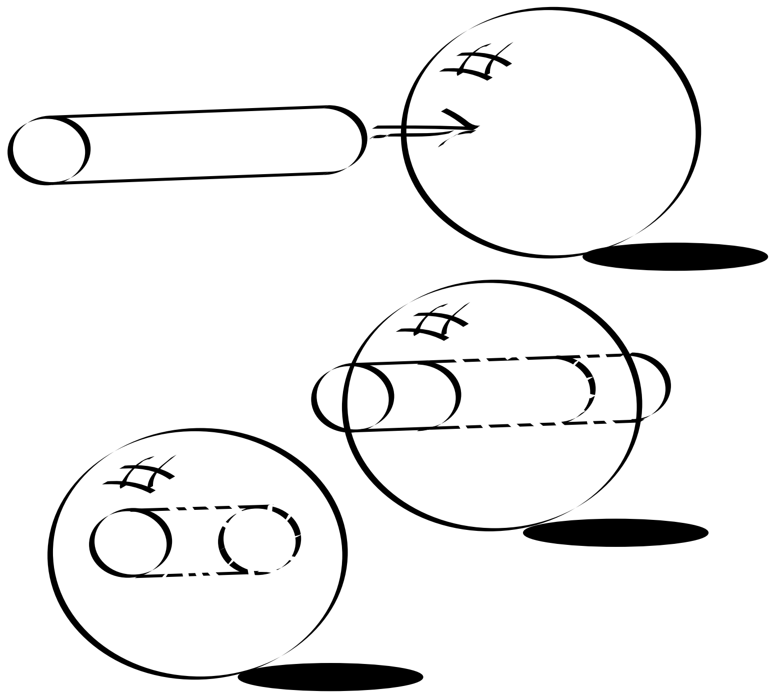


Figure 3.4: CSG-like boundary operations for surface construction: the cylinder and sphere are trimmed against their intersection curve, and then joined to make a single toroidal surface.

variational surface. For example, we shouldn't look for points of intersection between two approximate surfaces in order to answer the question, "do the variational surfaces intersect?" Because of discretization error, whether or how two approximate surfaces intersect says nothing about the true intersection topology.

This is not as big a drawback as it might seem at first. As an example, consider the curved boundary-representation operations of[Rie89], in which intersecting surfaces are trimmed against each other and joined along their intersection curves [TTSC91]. If we are using variational surfaces, we cannot generally compute their intersection, which would seem to rule out this style of trim-and-stitch construction. But the limitation disappears if instead the trimming operation is conceived as taking a snapshot of the intersection between the two approximate surfaces, then *redefining* the parent surfaces to interpolate this independent curve as their new boundary (we construct a variational curve to approximate this intersection shape by sampling the explicit mesh intersection somewhat coarsely and using these as point constraints). This has the advantage over passive surface intersection that the intersection curve, and optionally its tangent ribbon, becomes an independent control curve in the composite surface, and can subsequently be directly reshaped by the designer (Figure 3.5).

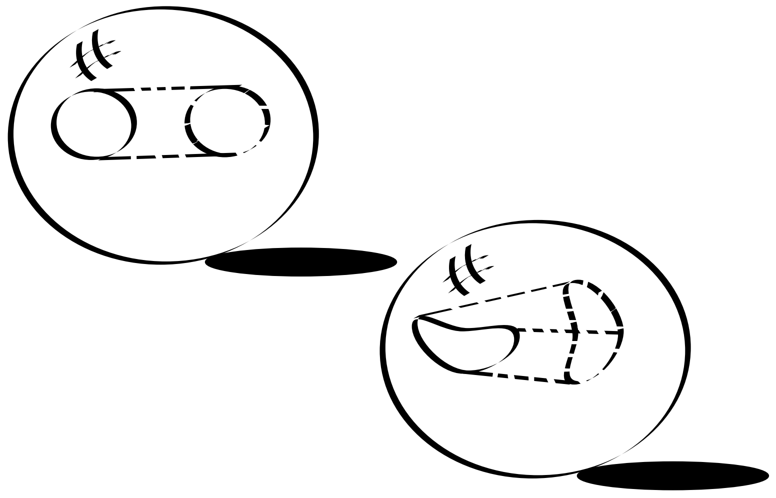


Figure 3.5: Rather than remaining dependent on the original sphere and cylinder, the original intersection curve becomes an independent control curve, and may be reshaped.

3.4.2 Topological design

We are distinguishing between topological design and shape design because it is very natural and convenient to think about these as separate phases of a 3D design process. While from a design standpoint it is not particularly interesting to point out that a donut is topologically equivalent to a coffee mug, if we want to consider designing a family of coffee mugs we might expect to fix the topology fairly early in the process, then manipulate or deform the shape while leaving the topology unchanged as we refined our mug. In this case, various disjoint regions might correspond to different components or features of the mug, each of whose shapes will be controlled in a different way. Contrast this with a volumetric sculpting process, in which mugs must be "carved" out of blocks of material. Here topology and shape are inextricably bound up together, so that in changing the shape one might accidentally change the topology (*i.e.*, gouging a hole in the mug while thinning a wall).

Summary

We have discussed what it might be like to design free-form curves and surfaces as variational shapes, and how one can build up non-trivial topologies gradually, using sequences of simple "surgical" operations. Variational shapes are described implicitly in terms of the constraints they satisfy and the quality measures they optimize. A modeler that operates on such variational specifications will need to be able to compute approximations to the optimal shapes, quickly. We touched on some of the issues involved in computing such approximations, to be fleshed out in upcoming chapters.

Chapter 4

Variational Shape Specifications

Synopsis

Variational shape specifications are skeletal recipes for smooth curve and surface shapes. The recipes include topological information, local shape information in the form of geometric constraints, and shape objective functions. We represent these specifications as meshes of connected curve and surface elements, and tag each element with an appropriate "shape controller", depending on how the element's shape is to be computed. Here we develop algorithms for building, modifying, and tagging topological meshes to represent variational specifications.

This chapter develops our basic methods of specifying variational shapes, and of representing these specifications in the computer. We are *not* concerned here with explicit curve and surface shapes. Rather, we are concerned with the information needed to specify a variational optimization that implicitly defines a particular free-form shape. Given such a specification, we will consider methods of approximating the associated surface using any of a variety of explicit curve and surface representations in later chapters.

Variational specifications have been present to varying degrees in previous variational surface modelers [CG91, WW92, MS92], implicit in the stitching-together of surface elements, the mechanics of local refinement schemes, and the objective functions used. But they have not really been considered as first-class entities unto themselves, independent of the explicit surface representations used by the modelers in question. Doing so is not particularly complicated; but it is a prerequisite to the kind of modeling we want to do. Ultimately, we will communicate with our modeler in by creating and operating on such an abstract variational specification, and thereby avoid specializing our operations to any particular piecewise surface representation.

4.1 Ingredients

Any curve or surface model can be seen as containing two fundamentally different kinds of information: that related to topology, and that related to its immersion or shape. The topological information, discussed in Chapter 2, gives us a decomposition of the model as a set of connected components. The immersion tells us where each of the model's domain points is located in space.

In this work, a model's topology will always be represented explicitly, so that we have explicit control over a model's topology. As was discussed in Section 2.2, not all shape representations allow the topology and the immersion to be treated separately. In representations where shape and topology are so intimately linked, there is no guarantee that the topology won't change out from under us when we change the shape.

Immersed free-form curves and surfaces, as discussed in Chapter 2, are most commonly represented as explicit coordinate functions that map a 1D or 2D parametric domain into space. In contrast, a variational shape specification has no such explicit immersion. Rather, the immersion is characterized implicitly, in terms of geometric constraints it should satisfy and geometric measures it should maximize or minimize subject to these constraints. A geometric constraint might take the form of a control point or curve that the shape must interpolate (*i.e.*, explicit coordinate assignments for some subset of the model); or it might require the model to have a prescribed normal at some point or (though we do not consider these here) maintain a prescribed area or volume. Geometric objective functions to be optimized subject to these constraints could include the goals of minimum surface area or curvature.

So we record three kinds of information in a variational specification: topology, geometric constraints, and geometric objective functions. Below, we consider how this information will be represented. We will work extensively with meshes, and discuss a number of mesh transformation algorithms necessary for building our topological specifications. We will not be concerned with a mesh's shape (an important aspect of the algorithms developed here is that the meshes needn't even have 3D coordinates associated with them). The problem of using these specifications to compute a variational shape will be taken up in the next three chapters.

4.2 Topology

In our modeler, points, curves, and surfaces are the only topological elements that will be needed. A natural way of representing these domains uses simplicial complexes (Chapter 2); and a natural way of representing these simplicial complexes on a computer is as lists of nodes (for 1D complexes) and triangulated meshes of nodes (for 2D complexes). With this, the seemingly abstract task of specifying and representing topology becomes the very concrete task of building a mesh. We'll look at user-level tools for creating and modifying model topologies in Chapter 7. In this section we take up lower-level representation issues — how to represent and operate on topological meshes

4.2.1 Topological representation

We begin by outlining how we represent mesh specifications, from the bottom-up, beginning with the simplest elements. There is nothing particularly novel or deep about this representation scheme; we offer it more for definiteness in the discussions to follow. Any of a variety of boundary-representation schemes from solid modeling might have been used

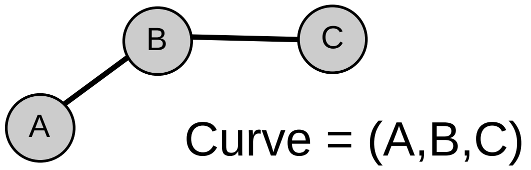


Figure 4.1: A 1D topological domain (a curve) is represented as a list of nodes.

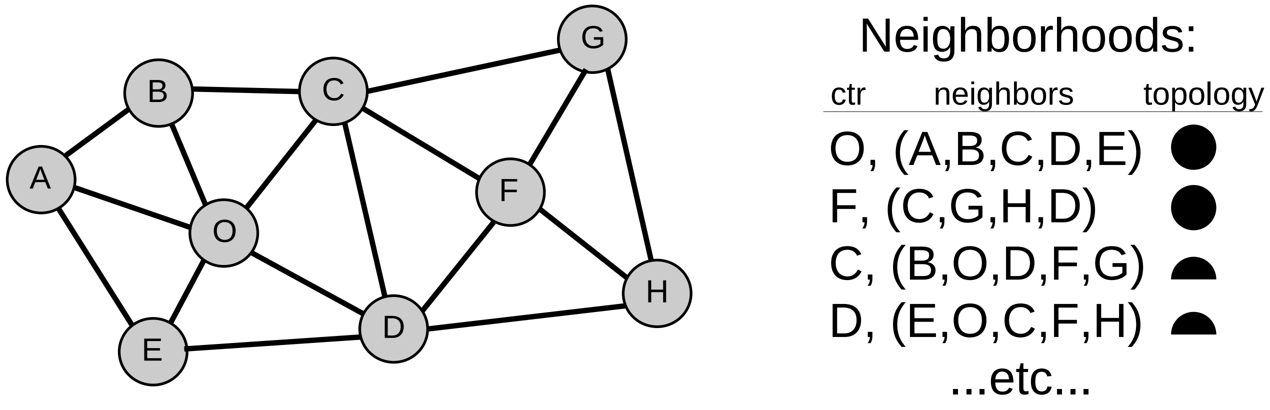


Figure 4.2: A 2D topological domain (a surface) is represented as a list of neighborhoods, each of which has a center node and a list of neighbor nodes.

here (Chapter 2), but these contain much more element grouping and incidence information than we care to use or maintain.

Primitive elements

The simplest topological element is a point (a *node*, in our data structures below). Curve and surface elements will be built up as lists of nodes.

A 1D interval (a curve segment) will be represented as a list of nodes, with edges implied between successive nodes (Figure 4.1). For open curves, the first and last nodes represent the boundary points; for closed curves, an edge is implied between the first and last nodes in the list. We will not allow more than one edge to connect the same two curve nodes (this restriction is important for embedded surface curves, below); thus, a closed curve must contain at least 3 nodes.

A 2D patch (a surface mesh) is represented as list of neighborhoods, each having a center node and an ordered list of neighbor nodes. (Figure 4.2). Triangular faces are implied between successive neighbor nodes. For nodes on a boundary of the surface, the first and last neighbors will lie on the boundary as well, and the neighborhood is isomorphic to a half-disc. For nodes in the surface interior, the neighborhood is isomorphic to a full disc, and a triangular face is implied between the first and last neighbors. Mechanically, neighbor lists work much as if they represent an (open/closed) embedded curve that bounds the (boundary/interior) neighborhood.

Curves that are embedded in a surface are represented as sequences of edge-connected

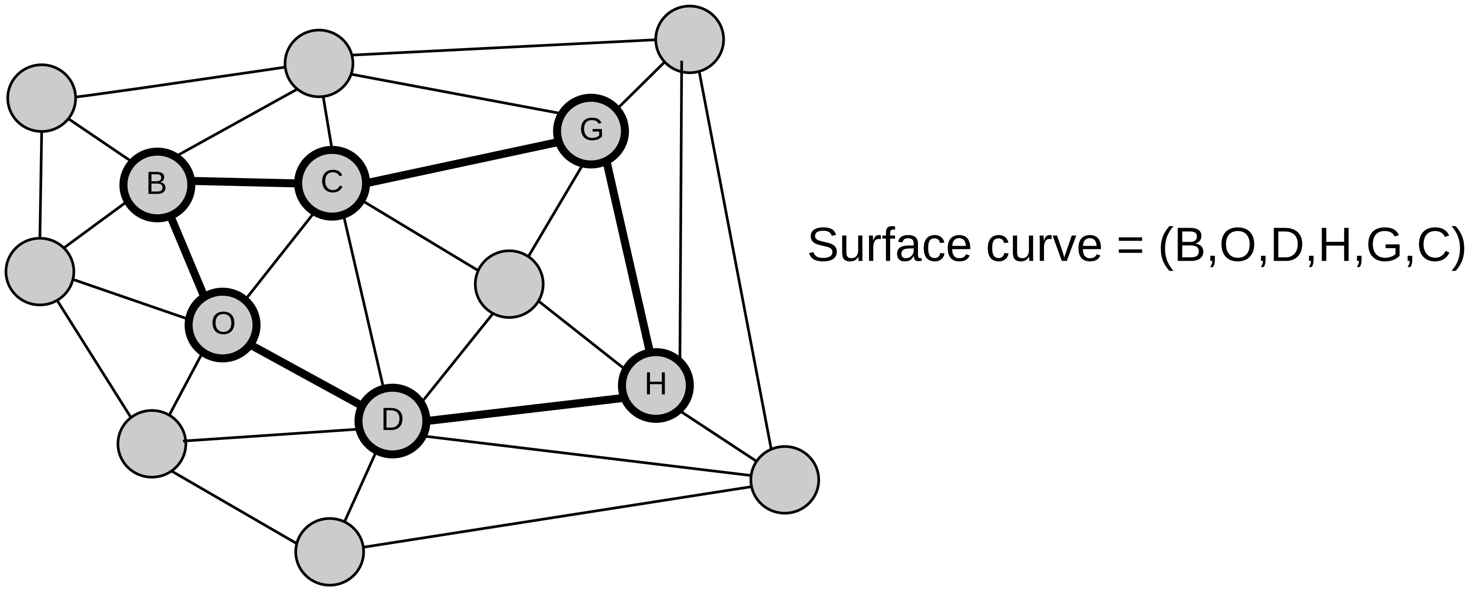


Figure 4.3: An embedded surface curve is a list of edge-connected surface nodes.

surface nodes (Figure 4.3). It is necessary that the curve nodes be connected by surface edges so that we can treat the embedded curve as a restriction of the mesh to a 1D domain. A closed surface curve must contain at least 3 nodes, so that it will enclose one or more triangles on the surface and thus divide the surface into nonempty "inside" and "outside" regions. This constraint will simplify the mesh transformation algorithms in later sections.

For the use of some of the algorithms below and in Chapter 6 we will need an explicit list of mesh edges. The edge-list is easily derived from the neighbor lists of the mesh nodes, with each node-neighbor pair corresponding to an edge. A list of triangular faces can be similarly collected. The cost of these traversals might seem to be $\mathcal{O}(nodes^2)$ because there is no limit to the number of neighbors a single node may have, but this is not so. It can be shown (using the Euler characteristic (Chapter 2) and a simple counting argument) that for a triangulated surface of genus g, the total number of edges is bounded linearly by the number of vertices:

$$E \leq 3V + 6(g-1)$$

(equality holds when the surface is closed). Further, the faces are related to the edges by

$$F = 2/3E,$$

and thus the cost of the traversal is linear in the number of nodes.

This node-based mesh representation was chosen for simplicity, and because ordered neighbor traversal will be used so often in the neighborhood-based computations of Chapters 5 and 6. Other ways of representing triangulations might also have been used here that would maintain edge and face lists at all times, thus eliminating the separate linear-cost collection steps in favor of more tedious but constant-cost bookkeeping.

Continuous collections of elements: Regions

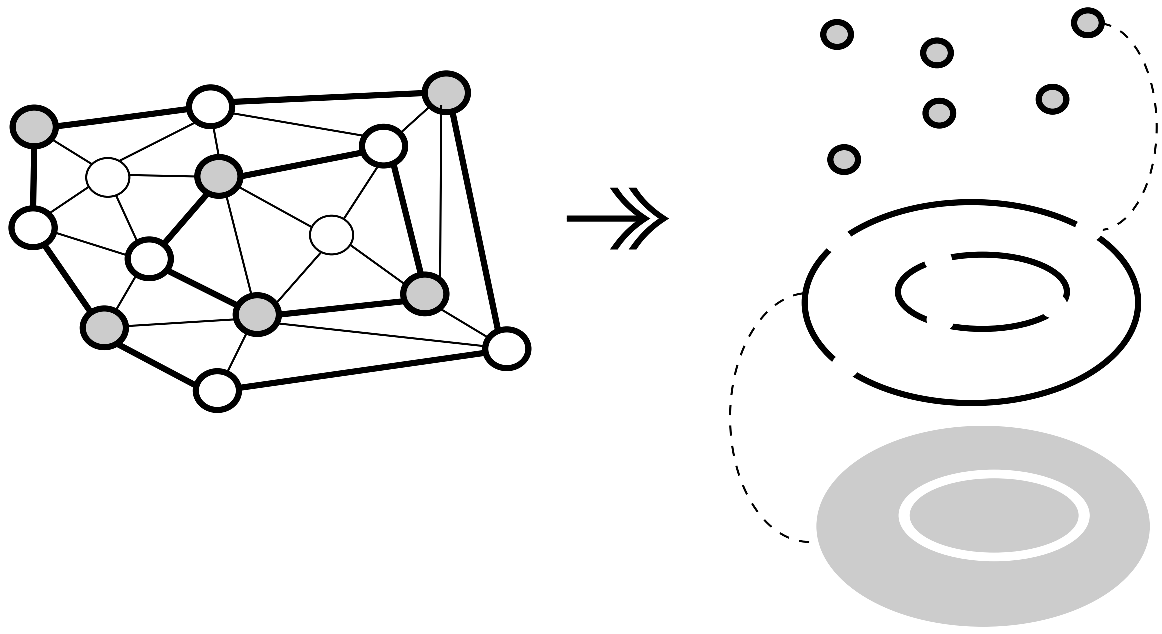


Figure 4.4: A surface mesh will be decomposed into disjoint 0D, 1D, and 2D regions.

Points on a curve break it into disjoint intervals; and closed curves on a surface break it up into disjoint patches. Such regional decompositions of curve and surface domains will be very important to us, for organizing computations and data within a mesh. A model's topological mesh will generally be decomposed into an assortment of regions of dimension 0, 1, or 2 (Figure 4.4). We will represent the decomposition by maintaining a list of regions, and by labeling nodes, edges, and faces with the region to which each belongs. Note that an element may only belong to one region; thus, a region will not generally include its boundary elements, as they are often treated separately as regions of their own. This is closely related to the topological notion of a *cell-decomposition* (Chapter 2) and is a useful abstraction because cells do not depend on the particular triangulations of their components or their relative levels of refinement.

The special nodes or edges in a mesh that define embedded curves or control points we refer to as *source* edges and nodes (a customary term in the triangulation literature). Given a mesh and a collection of source nodes and edges, we sweep through and assemble a list of regions, labeling each of the mesh elements using a standard connected-components algorithm ([CLR90]):

Algorithm: label-regions

```
1. For each source node:
    create a 0D region and label the node.

2. For each source edge:
    if neither of the edge's nodes is tagged with a 1D region,
      create a new 1D region
    else if both nodes are tagged with different 1D regions,
      union the different regions
    tag the edge and any untagged end node with the new 1D region
      (leave 0D labels in place)

3. For each surface triangle:
    if none of the triangle's edges is tagged with a 2D region,
      create a new 2D region
    else if more than one edge is tagged, with different 2D regions,
      union the different regions
    tag the face and unlabeled nodes and edges with the new region
    (leave 0D, 1D tags in place)
```

An efficient implementation of this algorithm uses a standard Union-Find data structure[CLR90] to tag and merge regions in essentially constant time (amortized over a sequence of operations). So, given the linear relationship between the numbers of edges, nodes, and faces, the cost of performing this region labeling is essentially linear in the number of mesh nodes.

4.2.2 Specifying topology by building a mesh

When it comes down to actually constructing meshes, we will sometimes do so "from scratch" and at other times operate on an existing mesh to transform it. We want to avoid ever having to repair or transform a mesh by triangulating an unadorned set of vertices, because doing something topologically appropriate can be difficult if not impossible. Likewise, we want to avoid having the user perform any kind of surgery on an existing mesh that might leave it in an inconsistent state. Therefore, we develop a set of mesh construction and transformation primitives, each of which is guaranteed to leave a topologically valid mesh in its wake.

The only meshes we directly construct are Cartesian products of an interval with either another interval or a circle. These are nothing more than planar sheets and cylinders, and they are constructed in obvious ways (Figure 4.5).

Mesh surgery

Instead of making a cylinder from scratch, we might equally well have made a sheet, identified a pair of intervals on its boundary, and stitched them together. Other, more complex topologies will be built up in exactly this way, through surgery on boundary complexes of simpler meshes. A simple example is Figure 4.6, in which the open ends of a cylinder are

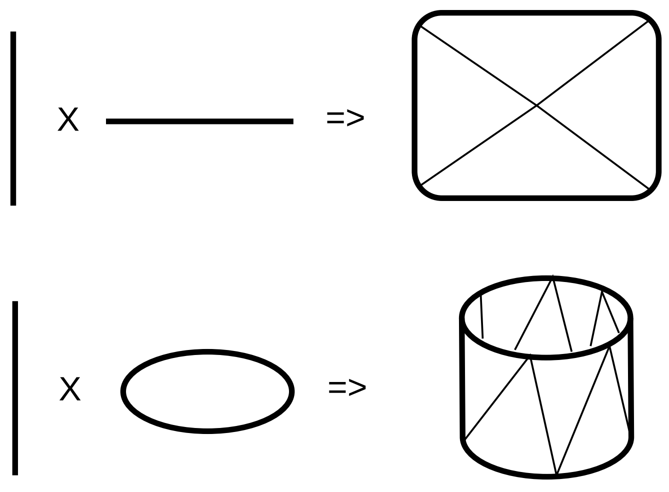


Figure 4.5: Constructing topological sheets and cylinders as products of 1-dimensional spaces

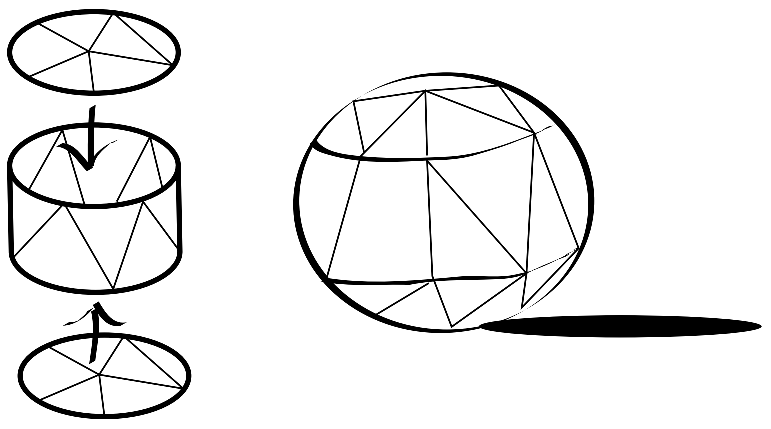


Figure 4.6: Capping the ends of a cylinder to make a topological sphere

closed by adding sheets to make a topological sphere. The basic surgical operations needed for meshes are cutting and gluing along embedded curves.

The mechanics of cutting are straightforward. Given a closed, embedded curve dividing a surface into inside and outside regions, the curve is "copied" by creating a corresponding list of new, unconnected surface nodes. Any 0D tags (control point indicators) are cloned as well, to propagate the way the curve is broken up into 1D regions. Then each of the original nodes' neighbor lists is split into inside and outside halves, and the inside neighbors are reassigned to the new curve's nodes. A similar operation is possible on an open embedded curve to cut a "slit" in a surface. In this case, only the interior nodes of the curve are copied and split, and the original curve boundary nodes are opened up into half-discs to connect the two new boundary curves into a single closed loop.

Gluing is, conceptually, the inverse of splitting. But the only time the two operations are algorithmic inverses is when we are gluing two boundary curves whose regions and their

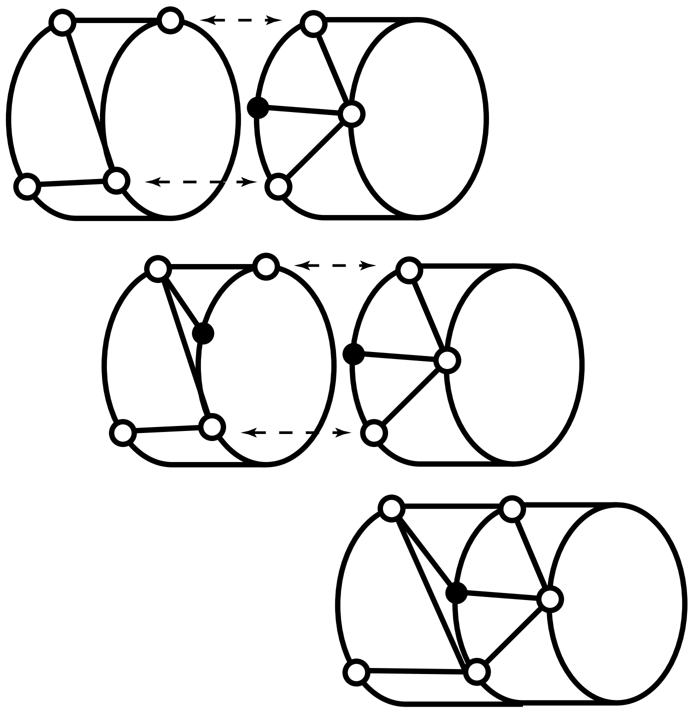


Figure 4.7: Two cylindrical surfaces are glued together after their boundaries have been made compatible through an edge-split.

nodes correspond exactly (*i.e.*, two boundary curves just created by a splitting operation). In this case, gluing is a simple matter of identifying corresponding boundary nodes, discarding one set of nodes, and adding their neighbor lists to the corresponding nodes on the other curve. The two boundary curves become a single interior curve.

The general gluing operation is a bit more complicated, because it must handle surfaces bounded by curves with different numbers of nodes (Figure 4.7). The curves must first be brought into correspondence. We assume we are given a 1-1 correspondence between source nodes on the curves, which implicitly gives us a correspondence between regions on the curves as well. Note that whatever higher-level process is calling for the merge may need to add or delete source nodes on either curve to make the 1-1 correspondence possible. Each of the regions is then brought into node-to-node correspondence, with additional nodes being created as needed through edge splitting (described below). The merger then proceeds as above.

4.2.3 Mesh transformations

In this section we discuss *mesh transformations* — operations that transform between different triangulations of the same topological surface. We will need these for purely topological reasons, *e.g.*, to insert p.l. curves into surface meshes or eliminate redundant mesh nodes. We will also use these transformations in later chapters in dealing with 3D meshes, and therefore may dwell occasionally on issues related to node positions. But the intent of each

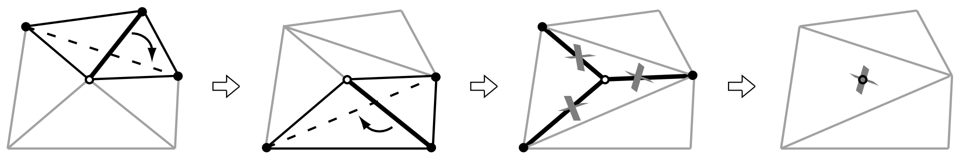


Figure 4.8: Node deletion: a sequence of edge-flips whittles away at the hollow node until an inverse face-split can be used to remove the node.

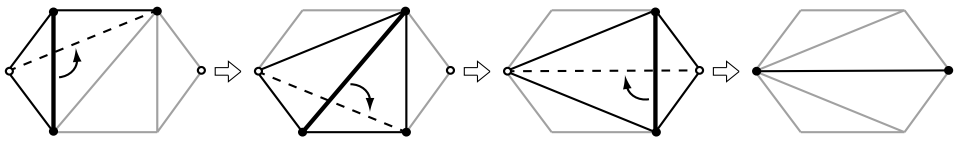


Figure 4.9: Edge insertion: a sequence of edge-flips modifies the mesh so that an edge is introduced between the hollow nodes.

of these algorithms is that it not depend on node positions in order to do its job, and that mesh topologies are always preserved or changed in controlled ways.

Our basic transformations will be the Alexander moves outlined in Figure 2.17. The order-1 refinement is more commonly known as an *edge split*, the order-2 refinement a *face split*. An edge split and an inverse edge split can be combined to exchange the diagonal of a mesh quadrilateral; this operation is commonly known as an *edge flip*, and treated as atomic. Note that boundary edges cannot be flipped (there is no quadrilateral), and edges that are part of an embedded surface curve should not be flipped or the curve's continuity will be disrupted.

Deleting nodes

In working with triangulations in the plane, node deletion is almost always accomplished by discarding the node and its edges and then re-triangulating that portion of the mesh. In working with abstract surfaces, planar re-triangulation is not an option. Figure 4.8 illustrates our node deletion algorithm. The simplest case is an infield, non-source node: For degree-3 nodes, deletion is the inverse of the Face-split operation. For nodes of higher degree, iteratively flip away edges until the node is degree-3, then apply the inverse Face-split. It should be clear that this procedure always terminates in a degree-3 node that may then be removed.

Boundary nodes are handled similarly: they are first reduced to degree-2 by edge-flips, and are then deleted along with their associated face. In effect, the corner is snipped away, so that an edge connecting the triangle's other two nodes becomes a new boundary edge.

Deleting source nodes through which an embedded curve passes requires much more explanation, though little more actual work. Exactly two source edges meet at a source node — else the node would be a curve endpoint or curve intersection point and we would not be deleting it. Furthermore, following these two edges out to their other endpoints, we may assume these neighbor source nodes are not connected by a source edge — else, the closed curve would only contain 3 edges and again the node would not be a candidate for removal. Reduction to degree-3 proceeds as before using edge-flips (no source edges are flipped). A natural consequence of this reduction is that a (non-source) edge will be placed between the two neighbor source nodes, if one was not there to begin with. After the degree-3 node has been deleted the curve will have been disrupted. The curve's continuity may be restored by incorporating that non-source edge connecting the source nodes into the curve and marking it as a source edge to avoid future disruptions. Another way one might delete nodes uses the "edge-collapse" operation of [HDD+93], which is more general than an inverse edge split because it can be used to remove nodes having any number of neighbors. Though it is conceptually simpler than the above, we found that implementing it to correctly handle source edges is more complicated than the approach taken here.

Remark on shape preservation

So far, our algorithms have been purely topological. All references to mesh components have been through nodes and neighbor lists, and we have made no reference to actual 3D vertex coordinates. We could continue in this vein with the remaining mesh transformations. But later we will be immersing these meshes in 3D as p.l. approximations to smooth shapes. It would be nice if mesh transformation operations, in addition to being topology-preserving, would also preserve shape as much as possible. This is not an absolute requirement like topology-preservation, because it will officially be someone else's job to worry about mesh shape (Chapters 5 and 6). But it will be helpful to other calculations if the mesh transformations do not perturb the shapes needlessly, and there are some simple things we can do toward that end.

When splitting edges or faces in 3D, clearly we can compute the new node's position to center it on the element just split. Deleting a node using edge flips can cause the mesh to fold back on itself. Since it does not matter topologically which edges are flipped or in what order, we attempt to minimize folding by always flipping the edge that yields the flattest dihedral angle. This does not guarantee that the surface will keep its shape, but generally does a good job. If for some reason it was crucial that the surface shape be disrupted even less, a procedure that perturbed node positions could be devised.

We note in passing that node deletion, face splitting, and edge insertion (below) do not leave particularly "nice" triangulations when applied to 3D meshes; triangle sizes and aspect ratios can become rather uneven. Again, this is not a topological issue. If it is important that a nice triangulation be maintained over an immersed mesh (it will be for us), a re-triangulation step should follow these operations. We discuss a simple, topology-safe re-triangulator in Chapter 6. We have found it *much* simpler to apply a single universal mesh improvement algorithm to a neighborhood after any one of these transformations, rather than complicate the individual transformations by devising each to leave a nice triangulation behind.

Edge insertion

The edge-insertion operation (Figure 4.9) is a basic part of our algorithm for inserting curves into a mesh. Given two nodes in the mesh, it adjusts the mesh so that an edge connects two nodes. First, we find a sequence of abutting triangles that connect these initial and final nodes. The triangles' union is a polygonal "channel" with no interior nodes, and the edge we will insert will run down the middle of this channel. Looking down the channel from the initial to the final node, there are edges crossing the channel (like rungs of a ladder). We sweep from one end of the channel to the other, flipping each of these edges in succession. The final flip inserts the desired edge. Note that the channel must not be crossed by any source edge, because such an edge cannot be flipped out of the way (though see the discussion on curve insertion, below).

As with our node deletion algorithm, this procedure can leave small folds in a p.l. surface due to unlucky edge-flips (particularly when three or more nodes are nearly collinear). One possible solution to this problem is to change the order in which channel edges are flipped, choosing flips that minimize creasing. Dyn, *et al.*[DGR93], show that in the planar case, there is always a choice of channel and edge-flipping sequence that avoids folding. In the non-planar case, choosing the crossing edge that yields the flattest dihedral is a reasonable generalization, and given a flat mesh it reduces to the planar algorithm (assuming the channel itself is straight enough that the inserted edge won't touch or cross its boundary).

Inserting curves

In order to operate on surface curves (*e.g.*, to constrain a surface along a control curve, or prepare a mesh for surgery), we will need to explicitly embed such curves in a mesh. Given a surface in 3D, one might use a "cookie-cutter" approach, extruding the immersed curve normal to the surface to make a cutting ribbon, and intersecting this ribbon with the surface to introduce new vertices and edges. Unfortunately, this offers no insight about inserting a curve as a purely topological operation, in the absence of a 3D mesh immersion. And, even for an immersed mesh, a robust implementation of this naive approach would be plagued with the kinds of "general position" problems that complicate so many algorithms from computational geometry.

A much simpler approach uses the edge-insertion operation to connect a sequence of nodes in the mesh. These nodes can be introduced through edge- or face-splitting operations. For a p.l. surface, this allows the curve to be "drawn" on the surface as a sequence of points, and their positions can guide the channel-finding part of the edge-insertion transformations. For a purely topological version of the operation, these must be guided by structural information.

If there is no requirement that the inserted curve contain only the originally specified nodes, another way to connect these nodes begins by finding polygonal channels connecting the nodes, as with edge-insertion. But then, instead of flipping the cross-channel edges out of the way, these edges are split, creating a sequence of edges that runs through the middle of the channel to connect the initial and final nodes (Figure 4.10). This completely avoids the folding problems discussed earlier for p.l. surfaces, at the expense of creating a more densely sampled curve. It also accommodates cross-channel source edges, by creating explicit intersection points between the new curve and existing curves.

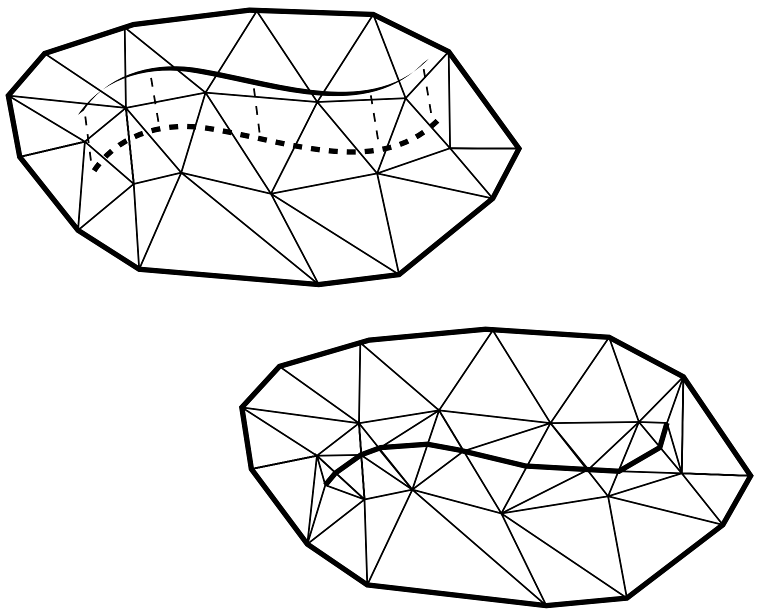


Figure 4.10: A p.l. curve is inserted into the mesh via edge splitting.

4.3 Attaching shape specifications to the mesh

The structures and operations of the previous section will be used to create a mesh representation of the "topology-part" of a free-form shape specification. In this section we consider how to represent the "shape-part" of such specifications, information related to computing an immersion for the domain.

The shape specifications we want to support are mixtures of explicit positional information (*i.e.*, specific points that curves or surfaces should interpolate) and implicit information (*i.e.*, extremize a fairness objective). A variational specification must distribute such implicit shape information over the mesh, in the form of geometric constraints, objectives, and dependencies, telling us how to compute an immersion. Think of it as a recipe for shape.

Our mesh representation includes a decomposition into 0, 1, and 2 dimensional regions. Shape information will be incorporated into the mesh by giving each region its own "shape controller", a tag indicating how an immersion for that portion of the domain is to be computed. This is a little different from a completely explicit surface modeler, where a collection of surface patches would suffice to define a surface, because we have a mix of point, curve, and surface controllers, all interacting through constraint and objective function dependencies.

4.3.1 Geometric Constraints

In this work we will consider *interpolation* and *normal* or *ribbon* constraints (Chapter 2). An interpolation constraint says that the surface must pass through a particular point or curve, or incorporate part of an explicitly defined surface region into its overall shape (external shape-copying). A normal constraint says the shape should have prescribed normal vectors

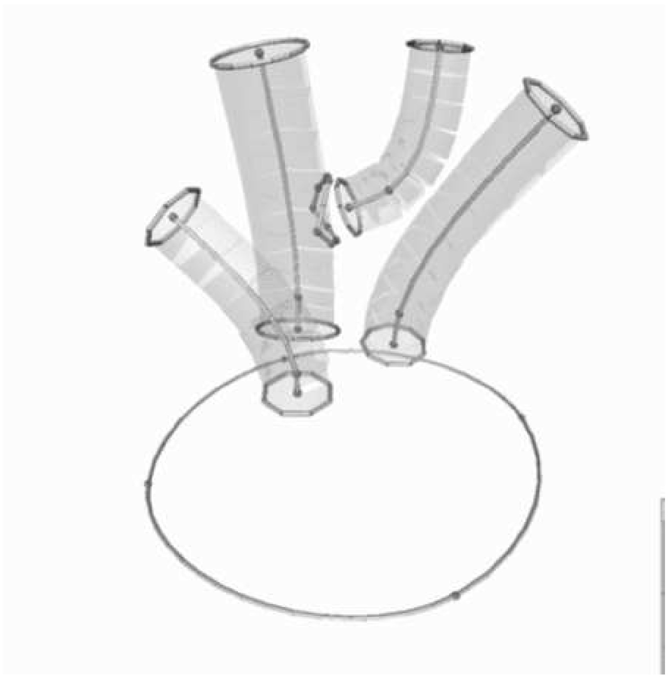


Figure 4.11: Constraint and objective skeleton for the Party Hat. The constraint curves and cylindrical offset surfaces (controlled by individual backbone curves) provide a partial specification for the overall surface shape.

within a specified region; this lets us force a curve or surface to be tangent to some other shape. We will discuss the computational issues involved in enforcing these constraints in Chapter 5. Here we are concerned with representing the constraint information rather than computing a satisfying shape.

Interpolation constraints will ultimately be implemented by copying the shape of some "source" piece of geometry. The mesh region will be tagged with a pointer to this source, for use by the process responsible for computing the region's shape. We make no restriction on the particular representation of such source geometry — it could be a piecewise smooth polynomial, an algebraic surface, or a p.l. shape represented by some other mesh. It will be the region controller's job to sample this source immersion and use it to construct a matching shape for the region. As a concrete example, surface control curves are implemented by having an embedded curve copy the position of a corresponding free-standing control curve, carrying the surrounding surface with it. The topological mesh region corresponding to the embedded curve would be tagged with a pointer to this control curve.

These kinds of one-way constraints allow us to establish a hierarchy of control, recorded in the mesh by constraint dependencies. Continuing the example above, the free-standing control curve might be a variational curve forced to interpolate selected control points. The curve nodes corresponding to these control points would be tagged with pointers to free-standing control points. Changing the position of a control point would cause the control curve to change shape, which would then cause the surface to change shape.

We only consider normal constraints anchored to point or curve constraints. The normal information is stored similarly to the positional interpolation information, as pointers to externally represented tangent planes or ribbons.

4.3.2 Specifying creases and corners

The shape computation scheme to be used in Chapter 5 assumes that regions are smooth throughout their interiors, and that discontinuities may only occur at region boundaries. We will support two kinds of continuity between faired regions: G^0 (positional continuity) and G^2 (positional, normal, and curvature continuity). A G^0 join between two curves or surfaces is a crease or corner; a G^2 join has the same degree of smoothness as the interior. The fact that this maximum continuity is G^2 and not G^1 or G^n is a function of the approximation scheme we adopt in Chapter 5, and might be different if a different approach were used.

To record this continuity information, in keeping with the spirit of labeling regions with shape-related information, we might tag region boundaries with the desired continuity. But continuity is a matter of communication between adjacent regions: for C^0 continuity, only positional information along the boundary is shared; for C^2 , normal and curvature information propagates across the boundary. It will turn out to be computationally convenient to have the mesh directly reflect this communication. Thus, we will represent a C^0 join as two independent meshes constrained to interpolate a shared boundary curve. A C^2 join will be represented simply as a single mesh with an interior embedded curve, as we have been discussing all along. Creasing a surface means splitting its mesh along the crease curve and constraining both sides to interpolate it.

4.3.3 Objective functions

The final step in constructing a variational specification is to associate an objective function with any unconstrained regions in the mesh. These are functions that measure some aspect of shape at a point of evaluation and return a "goodness" rating for that point. Integrating the measure over a piece of geometry measures the overall quality of the shape relative to other possible shapes. The modeler will use this to determine what to do with curve and surface shapes in-between constraints, by seeking shapes that minimize this measure subject to any geometric constraints that may be present.

In this work we use objective functions whose minimization yields "fair" surfaces. The functions measure total curvature over curves and surfaces, and are based on the bending energy of a thin beam or plate:

$$\mathcal{E}_{\text{curve}} = \int_{\text{curve}} \kappa^2 ds, \tag{4.1}$$

$$\mathcal{E}_{\text{surf}} = \int_{\text{surf}} (\kappa_1^2 + \kappa_2^2)\, dA. \tag{4.2}$$

Minimizing these functions distributes curvature over curves or surfaces to eliminate unwanted bulges or wiggles; they are discussed in more detail in Chapter 5. Other objective functions one might consider, though they are not addressed in this work, include surface area (Section 2.5.2) or variation of curvature. We discuss the tradeoffs between the thin-plate and other fairness measures in more detail in the next chapter.

Taken together, this information — topology, geometric constraints, and piecewise objective functions — implicitly describes a variational shape. Though we cannot solve directly for an immersion of a given topological domain, a specification like this can be used to set up approximations based on any of a variety of piecewise smooth representations. We consider this problem in the following chapter. Further implementation issues, such as how to coordinate the various shape computations with topological operations, and how to build a free-form shape modeler using this general approach, will be addressed in Chapter 7.

Summary

We have described a method of representing and constructing a variational shape specification. The basic representation is a tagged mesh: the mesh represents topological information, and the tagged regions within it indicate how the shapes of various curve and surface components are to be computed. Tagged meshes are not new — they are basic to any boundary-representation scheme in constructive solid geometry. Our intended use, however, is novel, as are our topology-safe mesh transformations for node deletion and curve insertion. These transformations may also find use beyond our application, in general polygonal mesh manipulations.

Chapter 5

Approximating Variational Shapes

Synopsis

In this chapter we present a method of approximating shapes that minimize a geometric thin-plate objective function while satisfying point and normal interpolation constraints. We use triangulated surface meshes as approximations to smooth surface shapes (the same meshes used to record topology in the previous chapter). We estimate surface curvature at mesh nodes by fitting a quadratic surface function at each mesh neighborhood. We drive the mesh towards an optimal approximating shape by minimizing a sequence of quadratic approximations to the geometric objective function (Figure 5.1).

The skeletal shape specifications developed in the previous chapter give us implicit representations of curve or surface shapes. In order to move from there to an explicit 3D representation — which we'll need to render, export, or do just about anything useful with the geometry — we must solve the corresponding variational minimization problem. But it may not be possible to solve such problems in closed form, due to the complexity of the domain, objective function, or boundary conditions[BCO81]. Generally, the best we can hope to do is compute approximations to the true solutions.

5.1 Overview of the approximation method

At an abstract level, our approach to variational approximation is typical. We've been given an infinite-dimensional problem: find an immersion for the domain curve and surface regions of the skeletal specification, mapping their points into space in a way that satisfies the given geometric conditions. We convert this into a finite-dimensional problem by choosing an explicit representation for the immersed curves and surfaces, then re-stating the constraints and objectives in terms of these representation parameters. Because of the particular constraint and objective functions we use, the discretization step will yield a multivariate optimization that is nonlinear in the representation parameters. We solve for optimal representation parameter values — thus "fitting" the curve or surface to the variational shape — by minimizing a succession of quadratic approximations to the nonlinear problem.

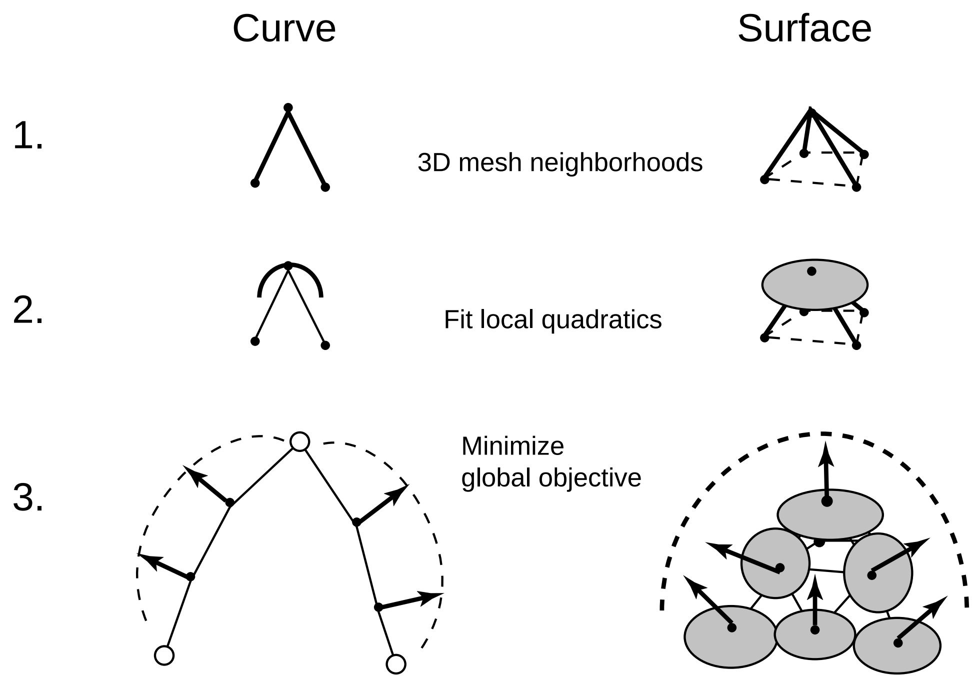


Figure 5.1: Synopsis of approximation approach: 1) Use a p.l. mesh to approximate smooth shapes. 2) Estimate curvature at mesh nodes. 3) Drive the mesh towards an optimal shape by minimizing a discretized curvature integral.

What is unusual about our approach is that we will take as our approximating representation p.l. curves and triangulated surface meshes — the same meshes used in the previous chapter to represent region topologies. Our task then is to compute a 3D position for each mesh node.

5.2 Why a mesh?

We must choose an explicit representation scheme for our curves and surfaces in $\mathcal{R}^3$. In Section 2.5.4 we discussed earlier work that used piecewise smooth curve and surface patches (tensor-product B-splines, Bezier triangles, subdivision surfaces) as representations for variational surface approximations, along with the pros and cons of each for use in an interactive modeler. In this work, we will use a triangulated mesh to represent approximate smooth surfaces. Our requirement that computations over arbitrary topologies be feasible at interactive speeds was the principal reason for this choice; if we were were willing to wait seconds or hours for a surface to be computed [HKD93, LP88, NLL90, MS92], just about any linear patch scheme would work.

Instead, we will give up smoothness and directly immerse the triangle meshes of the previous chapter to make a p.l. approximation of the free-form shape. While not as visually appealing as a smooth representation, it will be fast, and we expect that in an interactive design tool it will be OK to serve up such faceted approximations quickly, with the promise that a high-quality smooth surface can be fit to the final design in a post-processing step.

Of course, we cannot evaluate smooth functions (such as our curvature-based objective function) directly on such a non-smooth mesh. We address this in Section 5.3 by fitting a quadratic patch to the neighborhood around each node. Thus, our representation may be viewed as either a p.l. surface with curvature estimates at the nodes, or as a discontinuous union of quadratic discs, depending on your preference for finite difference or finite element methods, respectively[ZM83].

A side-effect of the combined shape/topology roles to be played by our meshes is that they will generally contain many more than the minimum number of simplices needed to express the topology. A denser mesh will be used to more accurately approximate free-form shapes by providing sample points. Nothing about our use of topological meshes will demand a minimal set of vertices, so it is alright to enrich the mesh for this purpose. When, in Section 6.3, we develop a mesh refinement scheme as part of the surface approximation machinery, we will use the mesh transformation operations of Section 4.2.3 to ensure that the underlying topology will not be changed by coarsening or refining the mesh. (this is in contrast with multi-resolution polygonal modeling schemes in which holes might disappear along with other fine detail as the model is coarsened[HG94]). Note that it will always be possible to derive a minimal mesh from a richer one by repeatedly un-refining until no legal moves are left, effectively "dehydrating" the mesh to leave its topologically essential components.

5.3 Smooth mesh neighborhoods

If we are to use a mesh to approximate variational shapes we will need to compute over the mesh as if it was a sampling of a smooth surface. To do so, in addition to sample point positions we also need to be able to evaluate surface first and second derivatives at each of these points, in order to compute curvatures and normals. If we had a rectangular mesh, we could apply a standard finite difference stencil to the node and its neighbor positions to estimate these derivatives. With irregular mesh neighborhoods, this approach breaks down since a single regular stencil cannot be used.

As it happens, the way that finite-difference stencils are constructed is by making a truncated Taylor series expansion of a low-degree polynomial fitted to a mesh neighborhood[Lan56]. This approach does generalize to irregular mesh neighborhoods[FW60]. Since we need to compute surface curvature, we will retain the first and second derivative terms of the series, effectively fitting a quadratic patch to the neighborhood. This is essentially the approach taken in several earlier scattered data approaches (*e.g.*, [Law77, SZ90, Ham93]). What is unique is our way of constructing a neighborhood parameterization with which to conduct the fitting.

5.3.1 Building neighborhood parameterizations

A garden-variety rectangular finite-difference mesh has global parameterization — the (u, v) plane. As has been discussed, global parameterizations generally do not exist for the surfaces we want to model. So instead we will construct a separate parameterization for each mesh neighborhood, and fit the neighborhood quadratic with respect to this parameterization.

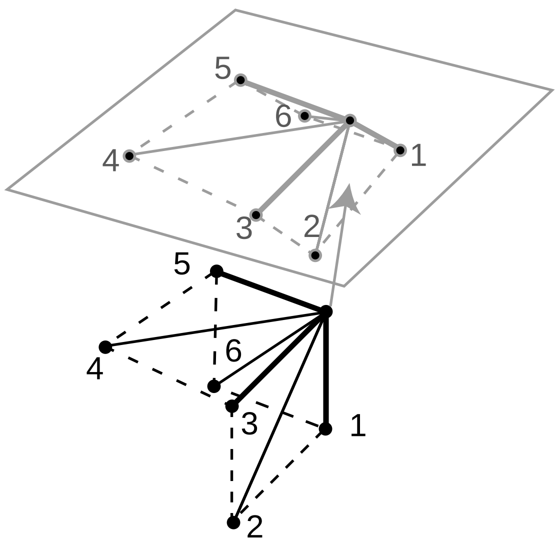


Figure 5.2: A failed neighborhood projection: projecting onto a plane does not guarantee a one-to-one relationship between surface points and plane points, and this can scramble the angular ordering of the neighbors.

As discussed, a usual way to parameterize this fit is by projection: find a normal, then project the neighborhood onto the tangent plane.

A weakness of the projection method is that it requires reliable estimates of curve and surface normals in order to determine a tangent plane on which to project the parameterization. If the normal used isn't close to the ultimately fitted normal, the expectation is the shape of the fitted surface will not be a good match to the neighborhood geometry. A Taylor expansion of a smooth surface using geometric information such as the true curvature and normal would differ from this fitted quadratic.

A more serious problem with such a projected parameterization is related to the behavior of the projection and subsequent fitting when a p.l. surface neighborhood is folded (*i.e.*, has high polyhedral curvature). In such situations, the resulting projection of the neighborhood on to the parameter plane may not be one-to-one (Figure 5.2). This has the effect of scrambling the angular order of the neighbors about the node, and thus the topology of the fitted polynomial will not match the topology of the folded mesh, and lead to erroneous derivative estimates. This folding is perhaps less disastrous (though still anomalous) when applied to the analysis of static objects (as in [SW92]), since such neighborhoods will be rare for densely-sampled surfaces. But in our interactive modeler, there is nothing to stop the user from tugging on a node and momentarily placing the surface in a badly folded configuration. Ultimately, the fairing scheme will "iron out" such transient folding, but only if the numerics honor the topology of these folded neighborhoods so that high polyhedral curvature is reflected in the smooth curvature estimate.

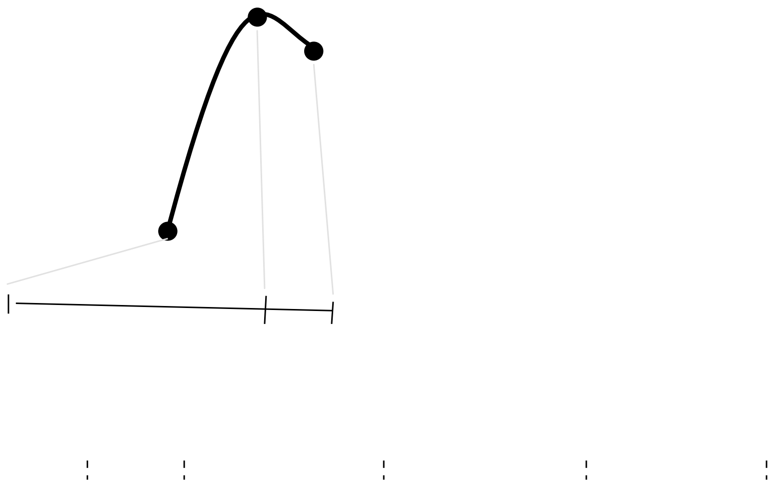


Figure 5.3: The straight-line distance between successive interpolation points is taken as an approximation of their arc-length separation along the curve.

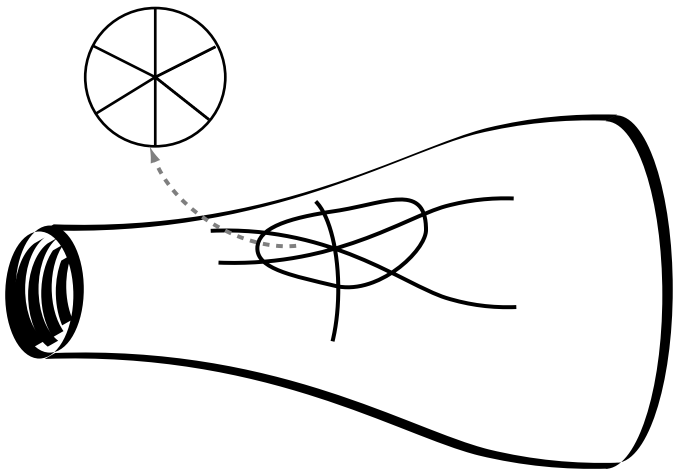


Figure 5.4: Geodesic polar map at a surface point.

Parameterizing curve neighborhoods

Rather than rely on an accurate normal projection to capture metric information (or ignore it altogether by using a uniform parameterization), we will mimic an arc-length parameterization of a curve by directly by measuring the Euclidean distances between neighbors and taking that as their parametric separation. This is the well-known chord-length approximation to an arc-length parameterization[Far90, Epp76].

Parameterizing surface neighborhoods

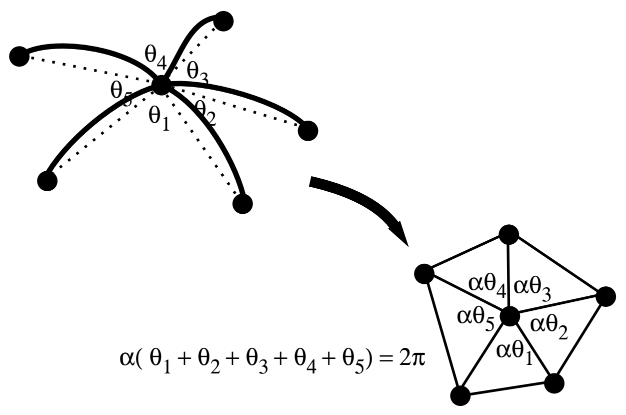


Figure 5.5: Parameterizing a surface neighborhood: angles between neighbors are measured in 3D, then scaled to sum to 2π. This acts analogously to the chord-length approximation to arc-length along a curve.

We can directly measure neighbor distances on the mesh surface we did with curve parameterization; but we also need to compute *tangent plane* angular separations between the geodesic paths to various neighbor points. This is the surface analogue of an arc-length parameterization — the so-called *geodesic polar map*[O'N66] (Figure 5.4). It is simply a generalization to geometric surfaces of polar coordinates (r, θ) in the plane, where r now indicates a distance traveled along a geodesic path away from the point, and θ indicates the initial direction to be taken. Unlike a curve, which can be given a single global arc-length parameterization, geodesic maps must be constructed separately for each point on a curved surface.

In the spirit of the chord-length approximation to arc-length, we will measure the angular separation between each successive pair of neighbors *in 3D*. We then "project" these angles onto a plane by uniformly scaling them so that they sum to 2π for nodes in the surface interior and π for nodes on the boundary (Figure 5.5). The Euclidean distance from the center node to each of its neighbors is taken as the radial parametric separation. This procedure gives us an (r_i, θ_i) for each neighbor (by convention, we place the first neighbor at $\theta = 0$). From this we compute the corresponding (u_i, v_i) by a standard polar to Cartesian transform in 2D. By analogy to the chordal parameterization for curves, we will refer to this as a *faceted parameterization* for surface neighborhoods.

5.3.2 Surface coordinate fitting

Given a suitable parameterization of the neighborhood, it is a straightforward task to fit a truncated Taylor series expansion for each of the surface x, y, and z coordinate functions about the neighborhood center. In the following, $\mathbf{p}_0$ will designate the position of the node at the neighborhood center, $\mathbf{p}_1 ... \mathbf{p}_n$ the positions of $\mathbf{p}_0$'s n neighbors, and $(u_0, v_0)...(u_n, v_n)$ their parametric coordinates $((u_0, v_0) = (0, 0))$.

For each of the coordinate functions we seek the coefficients of a quadratic:

$$s(u, v) = d_0 + d_1\, u + d_2\, v + \frac{d_3}{2} u^2 + d_4\, uv + \frac{d_5}{2} v^2 \tag{5.1}$$

$$= \mathbf{b}(u,v)\mathbf{d}, \tag{5.2}$$

where $\mathbf{b}(u,v)$ is the basis row vector $[1, u, v, \frac{1}{2}u^2, uv, \frac{1}{2}v^2]$, and $\mathbf{d}$ a column vector of coefficients. We want $s(u_0, v_0) \equiv s(0,0) = p_0$; this requires $d_0 = p_0$ (here we take p_i to mean one of p_{ix}, p_{iy}, p_{iz}, since the same fitting procedure applies to each). The remaining coefficients will be determined such that the $s(u_i, v_i)$ are a least-squares fit to the p_i.

For completeness, here are the details of the calculation: first, shift the neighborhood's 3D origin to p_0, yielding the vector of shifted neighbor positions $\mathbf{q} = [p_1 - p_0, ..., p_n - p_0]^T$. The sample matrix $\mathbf{S}$ for this shifted, center-constrained system is built by evaluating the basis vector $\mathbf{b}(u_i, v_i)$ for each of the neighbors and collecting these rows into a matrix, then deleting its first column:

$$\mathbf{S} = \begin{bmatrix} u_1 & v_1 & \frac{1}{2}u_1^2 & u_1v_1 & \frac{1}{2}v_1^2 \\ & & \vdots & & \\ u_n & v_n & \frac{1}{2}u_n^2 & u_nv_n & \frac{1}{2}v_n^2 \end{bmatrix}.$$

Then $\mathbf{S}[d_1, ..., d_5]^T = \mathbf{q}$, and the least-squares solution for $d_1 ... d_5$ is

$$[d_1, ..., d_5]^T = [\mathbf{S}^T\mathbf{S}]^{-1}\mathbf{S}^T\mathbf{q} \tag{5.3}$$

$$= \mathbf{Z}\mathbf{q}. \tag{5.4}$$

We use this shifted formulation in q instead of one in p because it reduces by one the rank of the matrix that must be inverted. The mesh's node and neighbor positions will of course be stored as absolute positions, not shifted so that p_0 is the origin. But now that we have a way to compute Z, we may recast the above more conveniently in terms of p_i instead of q_i. Let $\mathbf{P}$ represent the $(n+1) \times 3$ matrix of the $\mathbf{p}_i$'s x, y, z coordinates. The vector form for the reconstructed surface positions $\mathbf{s}(u,v)$ is then

$$\mathbf{s}(u,v) = \mathbf{b}(u,v) \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\sum Z_{1i} & Z_{11} & \cdots & Z_{1n} \\ \vdots & \vdots & \vdots & \\ -\sum Z_{ni} & Z_{n1} & \cdots & Z_{nn} \end{bmatrix} \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ & \vdots & \\ p_{nx} & p_{ny} & p_{nz} \end{bmatrix} \tag{5.5}$$

$$= \mathbf{b}(u,v)\mathbf{BP}. \tag{5.6}$$

This is the form we will evaluate when we want to compute surface derivatives in the upcoming sections.

The fitting procedure is somewhat complicated by fact that not all neighborhoods have five neighbors. And even if a node has five immediate neighbors, they might be positioned parametrically so as to make the full quadratic fit ill-conditioned (*i.e.*, if two neighbors are nearly collinear). We cannot look beyond the immediate neighbors to bring in additional nodes, as is common in planar least-squares schemes[FN90], because an angular ordering for nodes beyond the immediate neighbors is not determined by the surface triangulation.

In cases where there are not enough neighbors for a full fit, or where the full fit is poorly conditioned, we throw away some of the polynomial basis functions. For each node (of sufficient degree), an initial fit of the full basis $[1, u, v, \frac{1}{2}u^2, uv, \frac{1}{2}v^2]$ is attempted. The condition number of this fit, $c = \|\mathbf{S}^T\mathbf{S}\| \cdot \|(\mathbf{S}^T\mathbf{S})^{-1}\|$, is then computed (where the matrix norms are Frobenius norms[GVL89], $\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$).

If the fit was ill-conditioned (say, $c > 1000$), or if there were too few neighbors, a fit is attempted with the reduced basis functions $[1, u, v, \frac{1}{2}(u^2+v^2)]$ for interior nodes[1]. Boundary nodes, which will rarely have enough neighbors for a full fit, are treated specially: the parameterization is constructed so that the two boundary nodes lie on the $\pm u$ axis, and the basis functions $[1, u, v, \frac{1}{2}u^2]$ are used. This lets a surface curve along its boundary while remaining flat in the infield direction. As before, the condition number is evaluated and an ill-conditioned fit rejected. As a last resort, a planar fit (for boundary or infield nodes) is made with the basis functions $[1, u, v]$.

A shortcoming of this approach is that the somewhat arbitrary choice of basis functions could lead to instability over time. A neighborhood that is nearly well-conditioned might switch back and forth between different sets of basis functions on successive parameterization/fitting operations when used in the iterative minimization procedure later in this chapter (we have not actually observed such behavior). Better would be to consistently use the same set of basis functions and optimize some auxiliary norm in the underdetermined case. Barth's approach[Bar90] is an example: an orthogonal decomposition of $\mathbf{S}^T\mathbf{S}$ splits the range into well-conditioned and ill-conditioned subspaces. Ill-conditioned components can be left out of the fit, and then values chosen for them that minimize a separate norm, *e.g.*, coefficient magnitudes. Unfortunately, the orthogonal decomposition adds a good deal of computational expense, and it will thus degrade the overall interactivity of the modeler. We do not use such a scheme because we have not observed the instability problems it is intended to solve. Our meshing procedures (Chapter 6) do a fair job of keeping neighborhoods well-conditioned so that this borderline behavior doesn't arise.

5.3.3 Curve coordinate fitting

Fitting curve neighborhoods is so mathematically and algorithmically similar to surface fitting that we'll omit the details. A quadratic curve segment will be fit to each nodal neighborhood in the curve, parameterized as discussed earlier. The fitting procedure above is used with the monomial basis functions $b(t) = [1, t, \frac{1}{2}t^2]$. The vector form for the fitted curve position function $\mathbf{c}(t)$ is then

$$\mathbf{c}(t) = \mathbf{b}(t)\mathbf{B}\mathbf{P}, \tag{5.7}$$

where $\mathbf{B}$ is a $3\times npts$ array computed as in Equation 5.6.

5.4 Geometric objective functions

In Section 2.5.2 we discussed a geometric thin plate function that gives a measure of curves' and surfaces' total curvature. Shapes that minimize this measure are free of unwanted wiggles, creases, or bulges (*i.e.*, they are *fair*). Here we revisit curve and surface elastica,

[1]It is tempting to damp the second-order terms in the system matrix $\mathbf{S}^T\mathbf{S}$ by adding a constant on the diagonal. This will insure well-conditioning without these repeated fitting attempts. We tried this, but it lead to curvature estimation errors that noticeably degraded the fairing computations of the next section.

and with some mathematical manipulation arrive at forms that can be simply and efficiently evaluated over the quadratic mesh neighborhoods of the previous section.

5.4.1 Thin spline curve functional

Our chosen curve fairing functional mimics behavior of a physical spline by minimizing *strain energy*, approximated as the arc-length integral of curvature:

$$\mathcal{E}_{\text{curve}} = \int \kappa^2 ds \tag{5.8}$$

$$= \int (\mathbf{c}_{ss})^2 ds, \tag{5.9}$$

where, as before, $(\mathbf{c}_{ss})^2$ indicates the dot product of the second arc-length derivative $\mathbf{c}_{ss}$ with itself.

Recall that the chordal parameterization of our p.l. curves is an approximation to an arc-length parameterization. We will therefore take the fitted $\mathbf{c}_{tt}$ from Equation 5.7 as an approximation to $\mathbf{c}_{ss}$. Unlike the linearizations discussed in Section 2.5.2, this approximation inherits metric information from its special parameterization, and will not suffer the same distortions as in Figure 2.7 (the ultimate result is illustrated in Figure 8.1). This use of an approximate arc-length parameterization in order to have vvc_{tt}^2 approximate κ^2 appears in Hagen's work[HS90, HB91b].

5.4.2 Thin plate surface functional

By analogy to curve fairing, we take as the surface fairing objective function the integral of the squared principal curvatures over a smooth surface

$$\mathcal{E}_{\text{surf}} = \int_S (\kappa_1^2 + \kappa_2^2)\, dA, \tag{5.10}$$

where dA is the differential area form. This, and approximations to it, have been used as approximations the strain energy of a thin elastic plate. In order to express the thin plate functional in terms of a general parametric surface, we'll need some definitions and results from differential geometry ([Ham93, Spi79b]):

We begin with a surface $\mathbf{s}$ parameterized by u, v and immersed in $\mathcal{R}^3$,

$$\mathbf{s} = \mathbf{s}(\mathbf{u}) = (x(u,v), y(u,v), z(u,v)).$$

In the remainder of this section, all definitions and formulae assume $\mathbf{s}$ is being evaluated at a particular point $\mathbf{u}_0 \in \mathcal{R}^2$, though we will avoid notational clutter and not indicate this explicitly.

The partial derivatives $\mathbf{s}_u$ and $\mathbf{s}_v$ are a basis for the tangent plane at each surface point, though they are not necessarily orthonormal. The matrix of metric coefficients

$$\mathbf{G} = \begin{pmatrix} \mathbf{s}_u \cdot \mathbf{s}_u & \mathbf{s}_u \cdot \mathbf{s}_v \\ \mathbf{s}_v \cdot \mathbf{s}_u & \mathbf{s}_v \cdot \mathbf{s}_v \end{pmatrix},$$

where $i, j \in u, v$, give rise to the *first fundamental form*, I:

$$I(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{G} \mathbf{y},$$

for tangent vectors $\mathbf{x}, \mathbf{y} \in \mathcal{R}^2$ (with respect to the tangent basis vectors $\mathbf{s}_u$ and $\mathbf{s}_v$). Similarly, the normal section curvature coefficients

$$\mathbf{L} = \begin{pmatrix} \mathbf{s}_{uu} \cdot \mathbf{n} & \mathbf{s}_{uv} \cdot \mathbf{n} \\ \mathbf{s}_{uv} \cdot \mathbf{n} & \mathbf{s}_{vv} \cdot \mathbf{n} \end{pmatrix},$$

where $\mathbf{n}$ is a unit surface normal, *i.e.*,

$$\mathbf{n} = \frac{\mathbf{s}_u \times \mathbf{s}_v}{||\mathbf{s}_u \times \mathbf{s}_v||}$$

give rise to the *second fundamental form*, II:

$$II(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{L} \mathbf{y}.$$

A basic result from differential geometry says that the eigenvalues (both real) of the matrix

$$\mathbf{A} = \mathbf{G}^{-1} \mathbf{L}$$

are the principal curvatures κ_1 and κ_2 ([Spi79b],III.51). That means a matrix $\mathbf{Q}$ exists that diagonalizes the matrix $\mathbf{A}$ by mapping $\mathbf{s}_u$ and $\mathbf{s}_v$ to two orthogonal unit eigenvectors:

$$\mathbf{QAQ^{-}1} = \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}.$$

Other important quantities related to $\mathbf{A}$ are the Gaussian curvature:

$$K = \kappa_1 \kappa_2 = det(\mathbf{A}) = det(\mathbf{L})/det(\mathbf{G}),$$

and the mean curvature

$$H = \frac{\kappa_1 + \kappa_2}{2} = trace(\mathbf{A})/2.$$

(recall that det and *trace*, like the eigenvalues themselves, are invariant under similarity transformation and thus do not depend on our choice of parameterization).

We are interested in the sum $\kappa_1^2 + \kappa_2^2$, the geometric thin plate integrand. This quantity is not so nicely related to $\mathbf{A}$, $\mathbf{L}$, or $\mathbf{G}$. We may write it as:

$$\kappa_1^2 + \kappa_2^2 = (\kappa_1 + \kappa_2)^2 - 2\kappa_1\kappa_2 = 4H^2 - 2K,$$

which leads to a rather complicated expression in terms of the elements of $\mathbf{L}$ and $\mathbf{G}$ (that is nonetheless quadratic in the elements of $\mathbf{L}$). Rather than pursue this line further, we note two facts: first, if the tangent plane basis vectors are orthonormal (that is, $||\mathbf{s}_u|| = ||\mathbf{s}_v|| = 1$ and $\mathbf{s}_u \cdot \mathbf{s}_v = 0$), as they are for a geodesic parameterization, then $\mathbf{G}$ will be orthogonal. Second, the sum of the squares of the elements of any positive definite symmetric matrix (the Frobenius norm, denoted $|| \cdot ||_F^2$) equals the sum of its squared eigenvalues (because this norm is invariant under isometry). Thus, for $\mathbf{G}$ orthogonal we may compute $\kappa_1^2 + \kappa_2^2$

directly as $||\mathbf{L}||_F^2$, since $\mathbf{Q}$ in Equation 5.4.2 must be orthogonal in this case. Then we have, in terms of $\mathbf{L}$'s elements,

$$\mathcal{E}_{\text{surf}} = \int_S ((\mathbf{s}_{uu} \cdot \mathbf{n})^2 + 2(\mathbf{s}_{uv} \cdot \mathbf{n})^2 + (\mathbf{s}_{vv} \cdot \mathbf{n})^2) dA,$$

which is also quadratic in $\mathbf{L}$'s elements. It is worth pointing out that

$$\kappa_1^2 + \kappa_2^2 \neq \frac{||\mathbf{L}||_F^2}{||\mathbf{E}||_F^2},$$

though this is a tempting-looking formula.

Recall that we take our faceted parameterizations as approximations to a geodesic maps for the neighborhoods. As with curves above, we take the derivatives of the fitted surface quadratic (Equation 5.6) as approximations to derivatives wrt such a geodesic map, taking advantage of the built-in metric information from the parameterization to justify simplifying the curvature expressions. Similar use of a local quadratic fit to estimate II, and subsequently, surface curvature at a point, has appeared in [SZ90] and [Ham93], though in that work projection onto an estimated surface normal is used to parameterize the neighborhood.

5.5 Discretized objective functions

Having selected a surface representation in the form of a collection of nodes (and associated edges/faces), we are ready to *discretize* the objective function by evaluating it only at the neighborhood centers (where our curvature estimates are best).

5.5.1 Discretized curve objective function

For a point-sampled curve, the integral in Equation 5.9 is approximated as an area-weighted sum of integrands over sample points:

$$\hat{E}_{\text{curve}} = \sum_{\text{c}\in\text{nodes}} (\mathbf{c}_{tt}(0))^2 a_c, \tag{5.11}$$

where $\mathbf{c}$ is the local curve function of Equation 5.7, and a_c the width of the curve segment associated with node c (nominally, 1/2 the distance to each neighbor). Because each of the local $\mathbf{c}$ reconstructions is quadratic, each $\mathbf{c}_{tt}$ is constant within its neighborhood. Interestingly, this means that we could just as well view the numerical integration as being exact over a collection of discontinuous quadratic curve elements, since $\int_0^a \ddot{\mathbf{c}}$ is simply $\ddot{\mathbf{c}}(0)\,a$ in this special case (this will also be true of the surface objective function, below, so that we may think of the surface as being a union of quadratic patches).

5.5.2 Gradient and Hessian of the curve objective function

Substituting (5.7) and evaluating the derivatives of the basis functions $\mathbf{b}(0)$ gives us $\hat{E}_{\text{curve}}$ as a function of the node and its neighbors' positions $\mathbf{P}$:

$$\hat{E}_{\text{curve}} = \sum_{\text{i}\in\text{nodes}} (\mathbf{B}_3\mathbf{P})_i \cdot (\mathbf{B}_3\mathbf{P})_i \, a_i, \tag{5.12}$$

where $\mathbf{B}_3$ is the third row of the nodal neighborhood basis matrix, corresponding to the curve's second-order coefficients. We will be less concerned with evaluating $\hat{E}_{\text{curve}}$ than its gradient and Hessian with respect to $\mathbf{P}$, because the absolute magnitude of $\hat{E}_{\text{curve}}$ is not important — we will know it is at a minimum when its *gradient* has vanished. As is standard, we linearize this objective function with respect to the metric information a by treating the a_i as constant, yielding an objective that is quadratic in $\bar{\mathbf{P}}$ (of course, we re-evaluate a whenever the parameterization changes). We will use the resulting gradient and Hessian to set up an quadratic approximation to the minimization problem, and ultimately solve a series of such problems to converge on the true minimum.

In actually computing the gradient and Hessian with respect to the elements of $\mathbf{P}$, we will want a global "flattened" $\mathbf{P}$, a single vector of concatenated x,y,and z values from every node in the mesh, denoted $\bar{\mathbf{P}}$. This $\bar{\mathbf{P}}$ is the vector of independent variables whose values we will solve for in the course of minimizing the objective. If, for simplicity, there was only a single neighborhood, the corresponding Hessian for $\hat{E}_{\text{curve}}$ would be a $3n \times 3n$ block-diagonal matrix, with the outer product $\mathbf{B}_3^T\mathbf{B}_3$ replicated once along the diagonal for each of x, y, and z, like so:

$$\hat{E}_{\text{curve}} = (\bar{\mathbf{P}}^T{}_x \bar{\mathbf{P}}^T{}_y \bar{\mathbf{P}}^T{}_z) \begin{pmatrix} \mathbf{B}_3 \otimes \mathbf{B}_3 & 0 & 0 \\ 0 & \mathbf{B}_3 \otimes \mathbf{B}_3 & 0 \\ 0 & 0 & \mathbf{B}_3 \otimes \mathbf{B}_3 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{P}}_x \\ \bar{\mathbf{P}}_y \\ \bar{\mathbf{P}}_z \end{pmatrix} a \tag{5.13}$$

$$= \bar{\mathbf{P}}^T \mathbf{H} \bar{\mathbf{P}} \tag{5.14}$$

When there are many nodes, on the other hand, $\bar{\mathbf{P}}$ gets correspondingly longer, and the Hessian for any particular neighborhood will be very sparse, with $3(n+1)^2$ nonzero entries scattered throughout. The global Hessian is then the sum of these very sparse neighborhood Hessians. We'll not attempt to write out an expression for the global $\mathbf{H}$, since we'll never actually have to generate and store it as a monolithic matrix. Instead, the solver (below) will only require that we compute matrix-vector products with $\mathbf{H}$, and this can be done by summing the matrix-vector product with each neighborhood Hessian.

5.5.3 Discretized surface objective function

Analogously to the curve objective, for a triangulated surface the objective integral (Equation 2.5.2) is approximated as an area-weighted sum of integrands over sample points:

$$\hat{E}_{\text{surf}} = \sum_{\mathbf{s} \in \text{nodes}} \left((\mathbf{s}_{uu}(0,0) \cdot \mathbf{n})^2 + 2(\mathbf{s}_{uv}(0,0) \cdot \mathbf{n})^2 + (\mathbf{s}_{vv}(0,0) \cdot \mathbf{n})^2 \right) a, \tag{5.15}$$

where $\mathbf{s}(0,0)$ is the fitted surface function for each neighborhood (Equation 5.6) evaluated at its center node, $\mathbf{n}$ the fitted surface normal function, and a the neighborhood's associated area (nominally, 1/3 the area of each of the triangles in its parametric neighborhood).

5.5.4 Gradient and Hessian of the surface objective function

As with curves, the derivatives in the equation above are simply dot products with rows of $\mathbf{B}$:

$$\mathbf{s}_{uu}(0,0) = \mathbf{B}_3\mathbf{P}, \mathbf{s}_{uv}(0,0) = \mathbf{s}_{vu}(0,0) = \mathbf{B}_4\mathbf{P}, \mathbf{s}_{vv}(0,0) = \mathbf{B}_5\mathbf{P},$$

where $\mathbf{B}_j$ is the jth row of the neighborhood basis matrix we fit in Section 5.3. Substituting these gives us $\hat{E}_{\text{surf}}$ as a function of $\mathbf{P}$:

$$\hat{E}_{\text{surf}} = \sum_{\text{i}\in\text{nodes}} ((\mathbf{B}_3\mathbf{P}\cdot\mathbf{n})_i^2 + 2(\mathbf{B}_4\mathbf{P}\cdot\mathbf{n})_i^2 + (\mathbf{B}_5\mathbf{P}\cdot\mathbf{n})_i^2)\, a_i. \tag{5.16}$$

The dependency of $\hat{E}_{\text{surf}}$ on the node positions is given by its gradient with respect to $\mathbf{P}$. We linearize $\hat{E}_{\text{surf}}$ with respect to the $\mathbf{n}_i$ and a_i (and implicitly, $\mathbf{G}$) by taking them to be constant when computing this gradient.

The effect of the linearization is easy to picture, if we consider only one neighborhood at a time. The dot-product against $\mathbf{n}$ means that the gradient is only sensitive to motion in the normal direction. Thus, the objective function treats the surface like a height-field with respect to the tangent plane at the neighborhood center. This suggests that we might reduce the number of free variables by allowing each sample to move only in the direction of its current normal. Instead solving directly for n new point positions, requiring that we solve a system of $3n \times 3n$ equations, we solve for n normal offsets α from the current point positions, resulting in only an $n \times n$ system:

$$\hat{E}_{\text{surf}} = \sum_{\text{i}\in\text{nodes}} ((\mathbf{B}_3(\mathbf{P}+\alpha\mathrm{N})\cdot\mathbf{n})^2 + 2(\mathbf{B}_4(\mathbf{P}+\alpha\mathrm{N})\cdot\mathbf{n})^2 + (\mathbf{B}_5(\mathbf{P}+\alpha\mathrm{N})\cdot\mathbf{n})^2)\, a, \tag{5.17}$$

where α is a vector of offsets and $\mathbf{N}$ a list of neighbor normal vectors $\mathbf{n}_j$ (ordered just like $\mathbf{P}$). Thus, $\alpha\mathbf{N}$ is a list of *offset* vectors from the $\mathbf{P}$ computed by scaling each $\mathbf{n}_j$ by its corresponding α_j. It is these α_j that we want to solve for, rather than solving for $\mathbf{P}$. To minimize $\hat{E}_{\text{surf}}$, we will compute a minimizing α using the current sample positions, normals, and areas; then $\alpha\mathbf{N}$ will be added to $\mathbf{P}$ to move the points to these new positions. To picture this more restricted situation, imagine the surface as a tent, with the samples as tent-poles arranged at various angles, propping it up to give it shape. This new $\hat{E}_{\text{surf}}$ controls shape by shortening or lengthening the tent-poles. After a length change, the poles will be re-oriented to point in the new surface normal directions. In our experience, this normal-offset scheme yields surfaces visually indistinguishable from those produced by the original unconstrained computation of $\mathbf{P}$, while requiring the solution of a much smaller system of equations. It also yields more stable behavior in conjunction with the mesh smoothing operations developed in the next chapter, because the shape optimization doesn't compete with our sample re-distribution process, which moves nodes tangent to the surface.

Noting that our linearizations with respect to $\mathbf{N}$, a, and $\mathbf{P}$ make $\hat{E}_{\text{surf}}$ a quadratic function of α, we should be able to rearrange terms to put Equation 5.17 in the form $\alpha\mathbf{H}\alpha+\mathbf{g}\cdot\alpha+C$, for use in gradient and Hessian calculations. But standard vector notation has broken down for us in Equation 5.17 because of the way α scales $\mathbf{N}$, and rearrangement will only exacerbate the problem. With apologies in advance (there really seems to be no appealing way to continue the derivation), we will re-write the summand of Equation 5.17 using index notation[2], which will then allow us to re-arrange its terms unambiguously:

$$\hat{E}_{\text{surf}} = ((B_{3i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j)^2+2(B_{4i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j)^2+(B_{5i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j))a, \tag{5.18}$$

[2]In index notation, an unsubscripted quantity is a scalar, one subscript denotes a vector, and two denote a matrix. Under the *summation convention*, the appearance of any index twice in a term implies summation, so that $M_{ij}v_j$ means $\sum_j M_{ij}v_j$, which is matrix M times vector v.

where the neighborhood index from the earlier summation has been dropped (all quantities are specific to the particular neighborhood being evaluated) and the diagonalizing constant $\mathcal{D}_{ijk} = 1$ if $i = j = k$, and 0 otherwise [3]. Now, let

$$(BB)_{ij} = B_{3i}B_{3j} + 2B_{4i}B_{4j} + B_{5i}B_{5j} \tag{5.19}$$

within each neighborhood. Some rearrangement of Equation 5.18 yields

$$\hat{E} = \sum_{\text{nodes}} n_j(N_{mj}\alpha_k\mathcal{D}_{ikm} + P_{ij})(BB)_{iq}(P_{qr} + \mathcal{D}_{qst}\alpha_s N_{tr})n_r\, a \tag{5.20}$$

$$= \sum_{\text{nodes}} \left(\begin{array}{c} \alpha_k(n_j N_{mj}\mathcal{D}_{ikm}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r)\alpha_s \\ + \\ (2n_j P_{ij}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r)\alpha_s \\ + \\ n_j P_{ij}(BB)_{iq}P_{qr}n_r \end{array} \right) a, \tag{5.21}$$

which is the form we seek. We can then read off the neighborhood Hessians and gradients of $\hat{E}_{\text{curve}}$ or $\hat{E}_{\text{surf}}$ as

$$H_k s = \frac{\partial^2 \hat{E}}{\partial\alpha_i\partial\alpha_j} = n_j N_{mj}\mathcal{D}_{ikm}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r a, \tag{5.22}$$

$$g_i = \frac{\partial \hat{E}}{\partial\alpha_i} = H_{ij}\alpha_j + 2n_j P_{kj}(BB)_{kq}\mathcal{D}_{qit}N_{tr}n_r a \tag{5.23}$$

The global Hessian and gradients are simply the sums over all neighborhoods of these local versions. A word on evaluating terms like $(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r$: $\mathcal{D}$ is a notation aid, not meant to be used explicitly in computation. Instead, we first evaluate the product $N_{tr}n_r$. The resulting vector is then used to scale the columns of $(BB)_{iq}$.

5.5.5 Geometric point and curve constraints

Point interpolation

Point interpolation constraints for a curve or surface are enforced by simply freezing the positions of their associated nodes during minimization. We might also have placed the point interpolation constraint on the interior of an edge or face, resulting in a more general linear constraint analogous to the point-constrained B-splines of [WW92]. But refinement is cheap and easy in our p.l. representation, so having to create a node for each constraint point is no hardship.

Of course, just because a node's position is frozen does not mean that the node disappears from our calculations altogether: a frozen node is no longer considered an independent variable, but it still contributes constant terms to the gradient. For curves, we split $\bar{\mathbf{P}}$ into unconstrained and constrained parts $\bar{\mathbf{Q}}$ and $\bar{\mathbf{R}}$, which partitions $\mathbf{H}$ into blocks representing cross-terms between the constrained and unconstrained points:

$$\hat{E} = [\bar{\mathbf{Q}}^T\bar{\mathbf{R}}^T] \begin{bmatrix} \mathbf{H}^{QQ} & \mathbf{H}^{QR} \\ \mathbf{H}^{RQ} & \mathbf{H}^{RR} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{Q}} \\ \bar{\mathbf{R}} \end{bmatrix} \tag{5.24}$$

[3] $\mathcal{D}_{ijk}a_i b_j c_k = a_0b_0c_0 + a_1b_1c_1 + \cdots a_nb_nc_n$. $\mathcal{D}$ is a notational trick that lets us construct a diagonal scaling matrix from a vector (and exploit the symmetries of this operation).

The gradient of $\hat{E}$ with respect to the active nodes is then

$$g = \mathbf{H}^{QQ}\bar{\mathbf{Q}} + 2\mathbf{H}^{QR}\bar{\mathbf{R}}. \tag{5.25}$$

For surfaces, because constrained α values are always fixed at 0, only the $\mathbf{H}^{QQ}$ terms need to be evaluated.

In upcoming chapters, where we develop an automatic re-triangulation scheme to run in parallel with the fairing computation, we will see how these point interpolation constraints become sliding (nonparametric) interpolation constraints.

Curve and region interpolation

Curve and region interpolation constraints, in which an embedded curve or surface region is constrained and controlled separately from the faired surface, are implemented as collections of point constraints over the affected areas. Region controllers (described in Chapter 7) will be responsible for positioning these constraint points and controlling the mesh refinement within these constrained regions.

Continuity across control regions

Recall that a surface forced to interpolate a closed embedded control curve is represented as three disjoint mesh regions — the inside, the outside, and the embedded curve itself. The nodes in these disjoint regions do communicate with each other, through the fitted neighborhood quadratics. Thus, even though the nodes in the embedded curve have their positions fixed, these nodes' derivatives incorporate positional information from nodes on either side of the curve, and the continuity across this fixed curve is thus the same (G^2 in the limit) as across any unconstrained mesh edge. If we actually want a crease in the surface along the embedded curve, we split the surface mesh along the curve as discussed in Chapter 4, and force the two independent surfaces to interpolate the same crease curve without sharing any other neighborhood information across their boundaries. If we want to control the dihedral angle at which these surfaces meet, we use a ribbon constraint (below).

Curve tangency at a point

In addition to constraining curve positions at selected points, we must be able to constrain a curve to be tangent to a given line at a given point. Suppose $\mathbf{l}(t) = \mathbf{x}_0 + t\mathbf{T}$, where $\mathbf{x}_0$ and $\mathbf{T}$ are a point and a unit vector, respectively. We force the curve $\mathbf{c}$ to be tangent to $\mathbf{l}$ at $\mathbf{x}_0$ by constraining a point $\mathbf{c}_i$ to remain at $\mathbf{x}_0$ as above, and also enforcing the following linear constraint:

$$\dot{\mathbf{c}}_i(0) = \mathbf{T} \tag{5.26}$$

$$\mathbf{B}_2\mathbf{P}_i = \mathbf{T} \tag{5.27}$$

($\mathbf{B}_2$ is the second row of $\mathbf{B}$, as in previous sections). This constraint cannot be properly enforced by freezing node positions, as was done with position constraints; instead, we will use the technique of Lagrange multipliers[Str86], to be described below.

Surface boundary ribbons

In addition to directly enforced point constraints, we consider another more complicated constraint on surface boundaries: the *ribbon* constraint, which controls cross-boundary tangents along a surface boundary curve. Recall from Section 5.3 that our local fitting scheme aligns surface boundaries in the parametric u direction. Thus, enforcing a cross-boundary tangency constraint at a point amounts to enforcing

$$\mathbf{s}_v(0,0) = \mathbf{R} \tag{5.28}$$

$$\mathbf{B}_2(\mathbf{P} + \alpha\mathbf{N}) = \mathbf{R} \tag{5.29}$$

at the boundary mesh node ($\mathbf{B}_2$ is the second row of $\mathbf{B}$). We can turn this into a ribbon if we are given tangent vectors R for each point on the boundary curve (a cross-boundary tangent function). Again, such linear constraints will be enforced through the use of Lagrange multipliers, below.

5.6 Minimizing the objective functions

The curve and surface objective functions $\mathcal{E}_{\mathbf{curve}}$and $\mathcal{E}_{\mathbf{surf}}$are nonlinear functions of the mesh node positions. We will compute a minimizing set of node positions by solving a sequence of quadratic approximations to the true nonlinear problem (*quadratic steps* in a sequential quadratic program[GMW81]).

The quadratic sub-problems are minimizations of the linearized $\hat{E}_{\mathbf{curve}}$ and $\hat{E}_{\mathbf{surf}}$ objective functions just described. Given mesh and curve shapes, we parameterize node neighborhoods, fit a quadratic function to each node neighborhood, then take a step towards the minimum of the linearized objective functions evaluated with respect to the current fitted neighborhoods. The idea behind the iteration is that near the nonlinear objective function's true minimum it can be well-approximated by a quadratic function, so that minimizing this function (which is easy) also minimizes the nonlinear version (which is hard). The assumption is that near the minimum, control parameter gradients are small, and therefore the ignored gradient terms due to our linearizations will not matter so much. This all depends on beginning our iterations with a mesh shape that is "reasonable", or the iterations will diverge and not home in on the minimum. This issue of good initial guesses is discussed further in Chapter 7.

A quadratic step involves moving towards the the minimum for our quadratic objective functions $\hat{E}_{\mathbf{curve}}$or $\hat{E}_{\mathbf{surf}}$. The minimization, subject to point constraints (Equation 5.24), is performed by solving for a value of $\mathbf{Q}$ that zeros the gradient, which means solving the linear system $\mathbf{H}^{QQ}\bar{\mathbf{Q}} = -2\mathbf{H}^{QR}\bar{\mathbf{R}}$. There may be additional linear constraints that must be maintained, such as those contributed by tangency requirements. To enforce these, we concatenate them as rows of a single matrix equation $\mathbf{C} = \mathbf{t}$, where $\mathbf{t}$ is the vector of fixed constraint values. We then use the technique of Lagrange multipliers[Str86]: an additional free variable $\mathbf{y}_i$ is added for each of the constraints, and we solve the augmented system

$$\begin{pmatrix} \mathbf{H}^{QQ} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{Q}} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} 2\mathbf{H}^{QR}\bar{\mathbf{R}} \\ t \end{pmatrix}$$

This system matrix is generally very sparse, as there is a zero for each pair of nodes not connected by an edge. The nonzero matrix entries will contain different values for every quadratic step we take (because the neighborhoods are re-parameterized each time). Therefore, rather than factor the Hessian explicitly (as in [WW92]), we solve the system using a conjugate-gradient method[Str86, She94], which only requires matrix-vector products with the system matrix, not an explicit representation. We compute these vector products by looping over the nodes, accumulating each neighborhood's contribution to the product. Technically, solving the linear system is $O(n^3)$ in the number of nodes; but we really expect the cost to scale as $O(n^2)$ on average because of sparsity arising from the local structure of the mesh (nodes rarely have more than 6 neighbors). The neighborhood-at-a-time approach to multiplication automatically takes advantage of the sparsity inherent in the global Hessian: since zeros in the Hessian correspond to node pairs that don't share a neighborhood, no time is wasted explicitly multiplying by such 0 entries.

Algorithm: Quadratic step (Figure 5.1)

1. `parameterize each nodal neighborhood (Section 5.3.1)`
2. `fit a quadratic function at each neighborhood (Section 5.3.2).`
3. `Compute new positions for the infield nodes by solving the system of linear equations in Equation 5.6. This may involve solving for more than one region simultaneously,` *e.g.*`, if interior control curves cut a surface into multiple regions that must meet smoothly at the control curves.`

At this point we should be through — we have a method of representing variational specifications, and a method of iteratively constructing an approximation to the corresponding variational shapes. Unfortunately, a look at Figure 5.6 shows that our iterations do not actually converge on a reasonable approximation. As the iterations progress, nodes tend to drift towards one another, clumping up near constraint points and curves, and leading to surfaces that bear little relation to the faired shapes we asked for.

The reason for the lack of convergence is that the relative sizes and shapes of the parametric neighborhoods in the mesh strongly influence the numerical conditioning of our calculations. As it happens, our shape functions say nothing about the distribution of nodes over a mesh, and therefore we should not be too surprised that the nodes drift about within the mesh, leaving poorly shaped triangles in their wake. Clearly, in addition to worrying about a mesh's shape, we must also be concerned with distributing and triangulating its nodes to yield a good *computational mesh.* This is taken up in the next chapter.

Summary

We have described a method of approximating thin plate surfaces of arbitrary topology using a p.l. surface mesh. A novel neighborhood parameterization scheme (*faceted* parameterizations) is used to fit a local polynomial surface at each node in an unstructured surface mesh, and thus estimate surface normals and curvatures at the nodes. This special parameterization also simplifies the evaluation of the geometric thin plate function at a node, generalizing to surfaces the technique of chordal parameterization. Approximate minimizers of the geometric elastica and thin plate fairness functions for curves and surfaces

Figure 5.6: (1) A point in the center of the disk is elevated. (2) The point is moved to the side. Notice the neighborhood around the point becoming less crowded (3) After repeatedly moving the point around and finally returning it to its original place, the node distribution over the mesh has degraded to the point that the shape approximation performs poorly. The problem will be solved by the computational mesh maintainer described in the next chapter.

are computed through a sequence of quadratic minimizations over the mesh nodes. This approximation procedure will be used as a “region controller” to compute shapes for the faired curve and surface regions of our tagged meshes.

Chapter 6

Maintaining a Quality Mesh

Synopsis

In this chapter we address three principal concerns in maintaining a quality computational mesh for the mesh approximation of Chapter 5: sample distribution, triangle shape, and node density.

1. We will keep nodes uniformly distributed by minimizing a sampling density objective function over the triangulated surface.
2. We will keep triangles well-shaped by using an incremental surface Delaunay triangulation scheme.
3. We will keep an appropriate sampling density by automatically adding or deleting mesh nodes in response to changes in curve lengths or surface areas.

These mesh optimization steps will be interleaved with the shape optimization iterations of the previous chapter, so that a good computational mesh is always present.

In Chapter 4, we used a mesh as a representation of curve and surface topology. In Chapter 5, this same mesh was used as a triangulated approximation to a variational surface shape. In doing so, implicit use was made of the mesh in a third way — as a polyhedral manifold or *computational mesh* over which the finite-difference calculations were performed. In this chapter, we consider this third (and final) role for our meshes in more detail.

The shape optimization of the previous chapter says nothing about the relative distribution of nodes over the surface. Left to themselves, the nodes tend to drift towards one another and clump — resulting in a bad computational mesh, and subsequent failure of the fairing computations (Figure 5.6). Therefore, as the surface mesh changes size and shape during a design session, we must work to maintain a *quality* computational mesh.

What is a good computational mesh?

What characterizes a good mesh, and how to construct one, is properly the concern of the field of *numerical grid generation*[TWM85, Tho85]. Informally, a good mesh is one

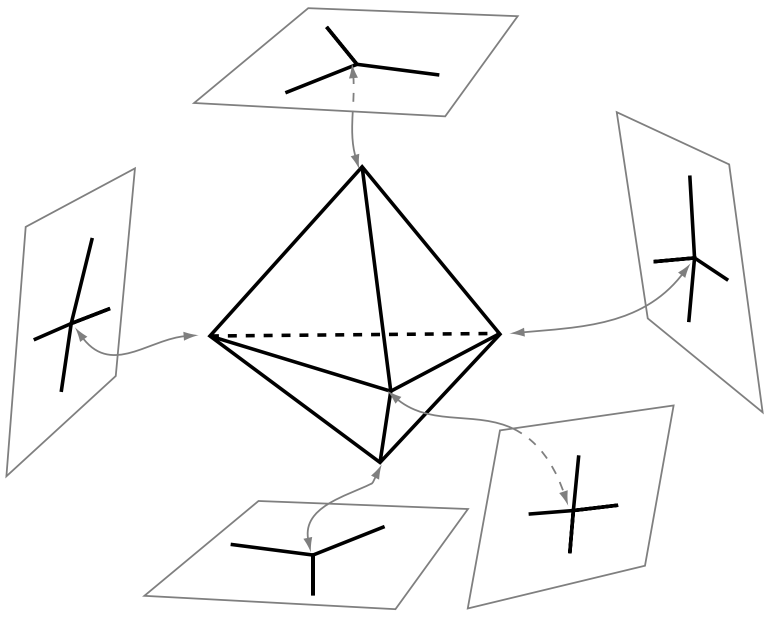


Figure 6.1: A collection of coordinate charts, one for each node-centered neighborhood.

whose sample points are spread over the surface in a smooth way, and whose triangles are well-shaped. A smooth sample distribution might mean the nodes are evenly spaced (their density is constant in all directions); or, if they are dense in some areas and sparse in others, that the spacing changes smoothly (their density has a constant gradient over the surface). Triangles are well-shaped if they are not skinny

The reason quality meshes are important for our computations is that poor meshes — ones with unevenly scattered points or skinny triangles — lead to ill-conditioned numerics and magnified truncation error (the error due to the use of truncated Taylor series approximations)[TM83, Hof82]. Schemes like the Taylor series reconstruction of Section 5.3 perform better as derivative estimators when a node's neighbor points aren't at wildly differing radial distances or angles.

In considering our triangulated surfaces as computational meshes, recall that, strictly speaking, the computations of the previous chapter do not take place directly on the polyhedral surface. They are instead scattered over the various parameterizations we construct for each of the mesh neighborhoods. The computational domain may be more properly visualized as a triangulated surface with each node bearing a flattened picture of its neighborhood (Figure 6.1). But the geometric relationship between the mesh and these neighborhood maps is very close, with well-distributed points and well-shaped triangles on the surface generally leading to well-shaped parametric neighborhoods. We'll often refer to the mesh and associated charts as the *computational mesh* (as opposed to the topological or approximating mesh) when we want to emphasize this role as domain for our smooth surface calculations.

6.1 Sample point distribution

We begin with the problem of distributing samples over a computational mesh. Because the mesh shape will be continuously changing as a result of user interaction, we rule out constructive, one-time schemes, such as those using random point placement or Steiner point insertion algorithms[BE92]. A purely constructive scheme would require that we generate a brand-new sample distribution with each iteration of the shape optimization calculation, oblivious to the existing sampling. Instead, we consider relaxation-based schemes, which can take advantage of *temporal coherency* (the fact that the mesh shape does not change much from one time-step to the next), by using the previous mesh as a starting point for computing the next mesh. In addition to being (potentially) less work, this also avoids numerical stability problems that might arise from discontinuous changes in the domain partitioning that would likely arise for a complete re-sampling of the domain.

As we discussed in Chapter 2, there are continuum methods and sample-based methods for controlling mesh vertex distribution over a physical domain. Continuum methods treat the mesh as a discretization of some continuous *computational* domain, and solve a system of grid generation equations that map the nodes into a smooth distribution over the physical domain. Generators such as the Laplacian can guarantee the mesh will not fold over on itself. Sample-based methods, on the other hand, operate on a collection of unorganized points as a discrete sampling of a computational or physical domain, with no specified interconnection topology, and use pairwise repulsion forces to distribute the points evenly over the space. As was discussed, it is unclear how to guarantee that an associated triangulation will not fold over on itself as the points move about on the surface. Nor do we wish to incur the computational cost that would accompany a wholesale re-triangulation of the points as in [Tur92]. For these reasons, we will use a node positioning approach based on Laplace's equation over the surface. Throughout this chapter we focus our attention on surfaces; the reduction of these techniques to 1D, for curve re-sampling, is simple and straightforward enough that it will not be given separate treatment.

6.1.1 Laplace's equation for planar meshes

As discussed in Chapter 2, the planar Laplace's equation may be used to smoothly map a regular computational grid onto an irregular physical domain:

$$u_{xx} + u_{yy} = 0, v_{xx} + v_{yy} = 0.$$

Notice that computational (u, v) are expressed as functions of the physical coordinates (x, y), reflecting the fact that we are concerned with node distribution in physical space. These equations can be obtained (via the Euler-Lagrange equation) as solutions to the variational minimization of the integral[BS82]

$$\iint ((u_x, u_y)^2 + (v_x, v_y)^2) dx\, dy.$$

The term $(u_x, u_y)^2$ measures the squared density of grid points as we move along the u parameter line, similarly for v. Thus, minimizing this expression yields a mapping of grid points into the physical domain whose density changes smoothly. It is also possible to view

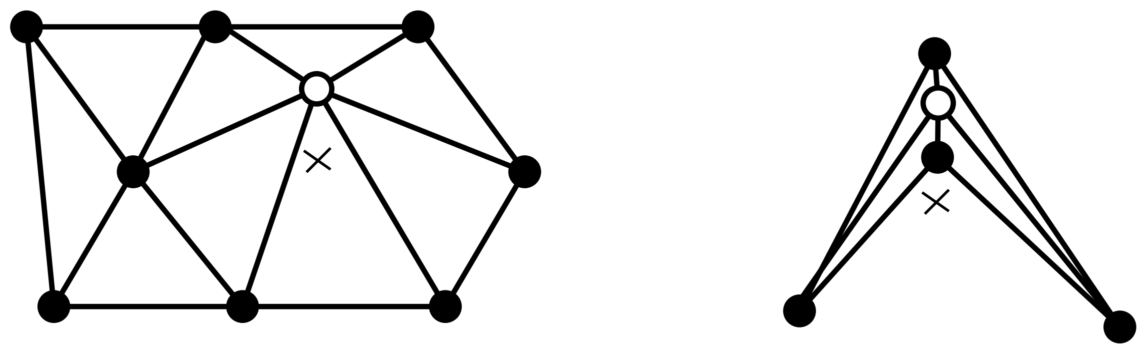


Figure 6.2: Laplacian smoothing in the plane: approximate a solution to the Laplace grid over the mesh by iteratively moving nodes to centers of their neighborhoods. This breaks down for non-convex neighborhoods, whose area centers (marked with an 'x') may lie outside the neighborhood polygon.

$(u_x, u_y)^2$ (and similarity for v) as measuring the energy in an imaginary spring connecting successive grid points. Minimizing the integral amounts to equilibrating the spring system, and again a smoothly changing mesh density results. Given a rectangular grid, it is straightforward to discretize and solve either of these by computing derivatives of the u and v functions using finite differences.

A smooth planar mesh generated by solving the variational form of the Laplacian grid generator.

For an unstructured grid, things are not so simple – we have no (x, y) coordinate lines over which to measure grid density. A popular generalization extends the spring idea to unstructured meshes, minimizing the energy of a spring network that connects each node with its neighbors. If this is implemented by iteratively moving each mesh node to the centroid of the neighbor points, it is known as *Laplacian smoothing*[Fie84]. One mildly objectionable feature of this approach is that, if there are fixed nodes closely spaced along the mesh boundary, their springs tend to "gang up" on any shared interior nodes and pull them very close to the boundary. An alternative version of Laplacian smoothing moves nodes towards the centroids of their neighborhood *polygons*, and gives similar results but does not suffer this defect (Figure 6.3). Both methods work best with convex neighborhoods, and even better in conjunction with an incremental Delaunay triangulation scheme as reported in [Fie84]. But non-convex neighborhoods that don't contain their centroids are a problem, since the smoothing operation will cause the grid to fold back on itself (Figure 6.2). We will return to this point in a moment.

6.1.2 The surface Laplacian

Evaluating the Laplacian over a surface in 3D is a rather complicated affair, involving the so-called *Beltrami derivative* from differential geometry[Spi79b, War86], which "build-in" a projection of derivatives onto surface tangent planes. That assumes we have a rectangular mesh, and have re-expressed the grid system in terms of this (u, v) parameterization. For an unstructured surface mesh, things are of course even more complicated, leading us to consider generalizations of the planar spring system above. Rather than use Beltrami

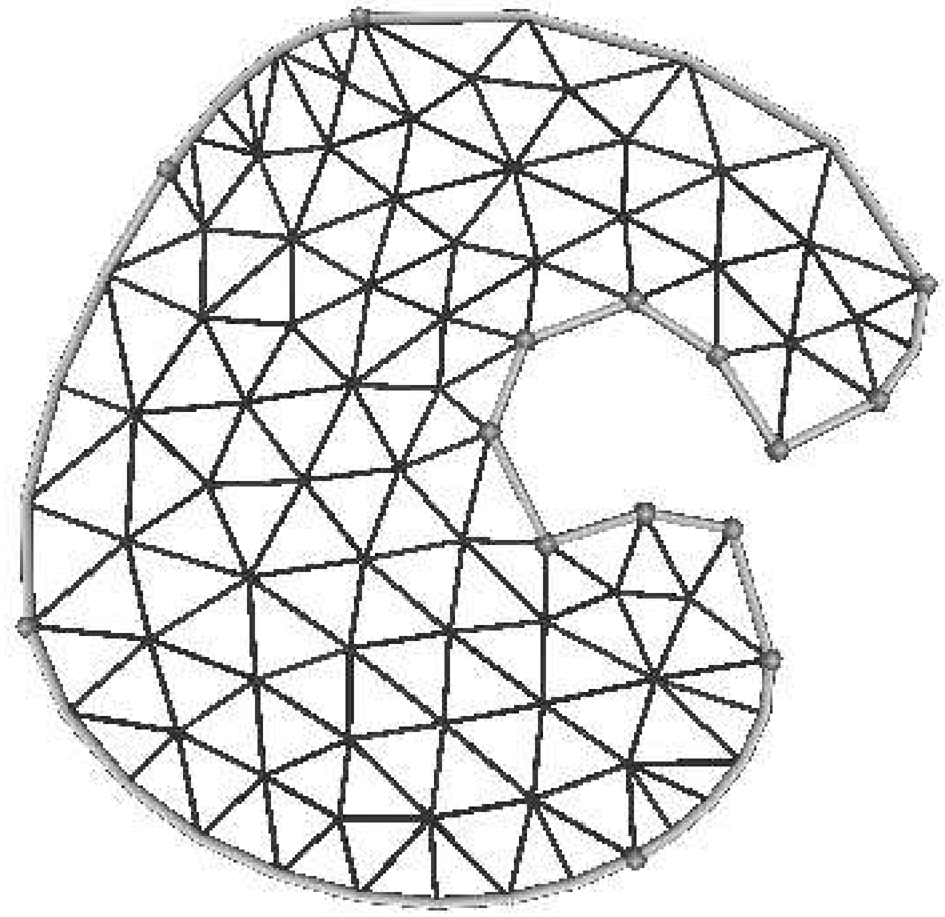


Figure 6.3: 3in

derivatives, we can solve for each new node's position in 3D and then *a-postiori* project these offsets against the nodes' tangent planes, similar to Barr, *et al.*[BCGH92]. An example of such a tangent-projected spring relaxation is illustrated in Figure 6.4.

Analogously to the planar case, we can also consider a *surface* Laplacian smoothing scheme in which nodes are moved towards their neighborhood centroids. We will accomplish this without any kind of projection operations, by using our faceted parameterizations from the previous chapter. A smoothing iteration consists of, for each mesh node, computing the area center of its faceted parameterization, mapping this back to a point on the polygonal surface, and sliding the node toward this point. The results are qualitatively similar to the spring system, but do not rely on our having fitted quadratics to evaluate tangent planes for the nodes. This also has an advantage over the planar Laplacian smoothing scheme: convex boundary neighborhoods, like that in Figure 6.2, are opened up into half-discs by the faceted parameterization. For internal non-convexities, however, we must look to the re-triangulation scheme below.

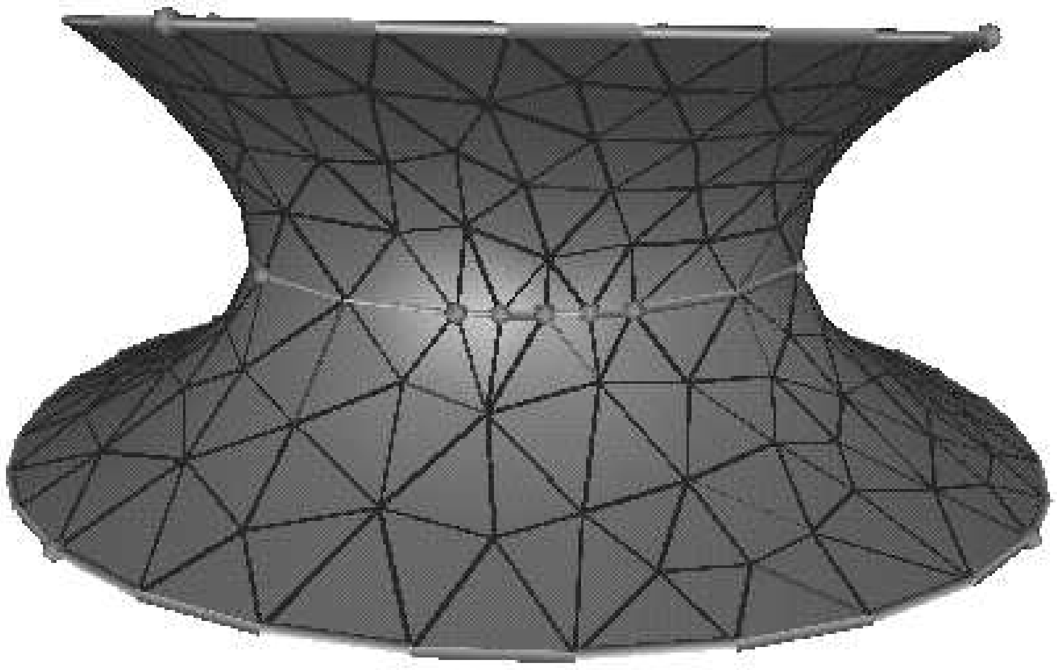


Figure 6.4: Laplacian smoothing over a surface triangulation.

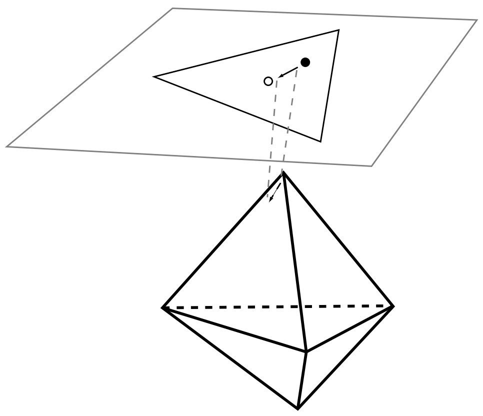


Figure 6.5: Depiction of the surface Laplacian smoothing technique for a single neighborhood.

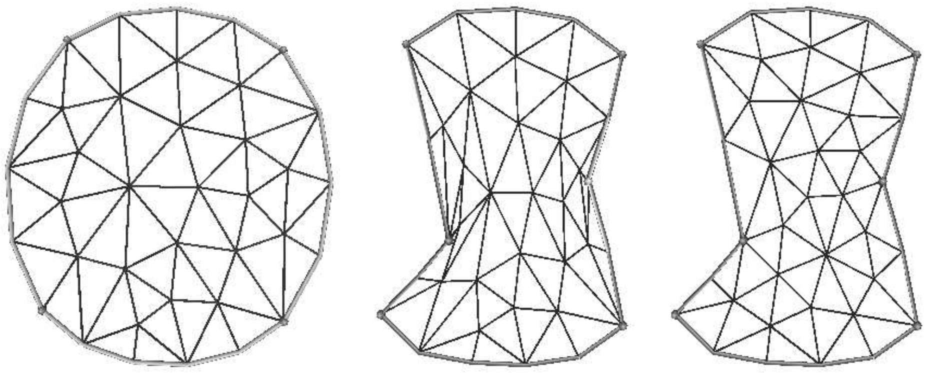


Figure 6.6: Why sample positioning isn't enough: in each figure, sample positions are controlled using Laplacian smoothing. When the original mesh (left) is deformed while its triangulation remains fixed (middle), badly shaped triangles cannot be avoided. Maintaining a Delaunay triangulation over the points (right) improves the mesh dramatically.

6.2 Surface triangulation

The sample distribution scheme above moves nodes around within their respective triangulated neighborhoods, assuming a fixed mesh connectivity. Although this does improve the quality of the triangulation by equilibrating edge lengths somewhat, it is not sufficient to yield the best possible surface triangulation over a given set of points with given topology. For example, Figure 6.6 shows a mesh whose sample distribution minimizes $\hat{\mathcal{E}}_{\mathrm{smooth}}$, yet which has many skinny triangles. The figure also shows a Delaunay triangulation (DT) over the nodes. As discussed in Chapter 2, the planar DT's max/min angle property makes it attractive for computational mesh generation, because it eliminates skinny triangles whenever possible.

In this section we outline a scheme for dynamically maintaining a surface DT over the nodes as they change position during surface re-shaping. The max/min angle property of the planar DT will carry over into the surface DT. As with the parameterization procedures of Section 5.3.1, mesh maintenance will be formulated so that it remains faithful to the given mesh topology, and never relies on the kind of projection and consistency testing found in [Tur92, SZL93].

6.2.1 The surface Delaunay triangulation

The classical DT is defined over a planar set of points, and generalizing it to 3D surfaces is not particularly straightforward. Recent work by Chew[Che93] generalizes the empty circumcircle characterization of the DT to triangulations over surfaces (Figure 6.7). It turns out that a naive generalization using geodesic distances to inscribe circumcircles on surfaces is undesirable, because it admits strange situations like self-intersecting circumcircles (and would be expensive to test, as well). Instead, Chew defines the circumcircle of a triangle on a curved surface as the surface's intersection with the sphere that includes the three triangle vertices and whose center lies on the surface (Figure 6.8). If a local flatness assumption holds

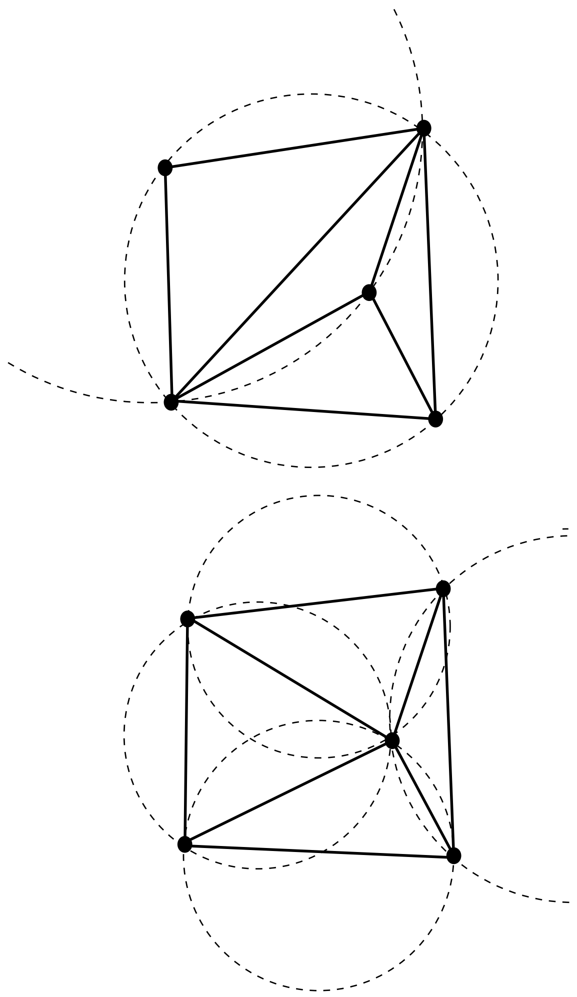


Figure 6.7: One characterization of the DT is that its triangles' circumcircles are all empty. The upper triangulation contains an edge that is not in the DT, and its associated circumcircles contain other vertices. This is not true of the lower figure, the DT of the points.

(the surface normals within the union of circumcircles associated with a given quadrilateral vary by less than $\pi/2$), it can be shown that this definition shares a consistency property with the planar circumcircle that makes it a reasonable generalization of the planar DT. The result is a unique triangulation that maximizes the minimum included angle in 3D.

An important consequence of this consistency property is that a surface DT can be incrementally recovered from a valid initial surface triangulation by iterative edge-flipping. One repeatedly tests quadrilaterals in the mesh to see if flipping an edge within the quadrilateral will improve the triangulation (by increasing the minimum angle), and continues flipping such edges until the DT has been restored (Figure 6.9, algorithm below). This is just like the standard planar DT edge-flipping algorithm, but angles are measured in 3D. The edge-flips preserve the topological type of the surface mesh, and thus are a topologically safe way to perform re-triangulation over a surface.

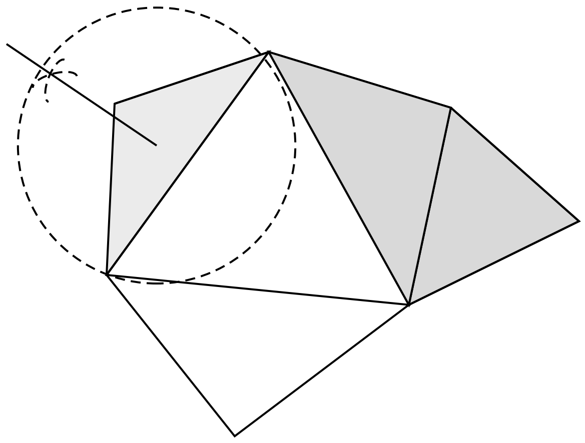


Figure 6.8: Circumcircles on a surface: for a given triangular face, there is a family of spheres through the three vertices, and their centers all lie on a line. The intersection of this line with the surface (not necessarily the given face) defines the center of a "circumsphere", and the intersection of this sphere with the surface satisfies an important consistency property satisfied by circumcircles in the plane.

6.2.2 Constrained triangulation

In constructing the surface DT from an initial surface triangulation, there will be edges that must not be disturbed, such as those that are part of embedded control curves. A scheme that incorporates these so-called *source edges* is referred to as a *constrained Delaunay triangulation*, or CDT [DFFP85]. It enjoys the same max/min-angle property as the DT (when restricted to consider only triangulations *that include the source edges*), and thus an incremental edge-flipping restoration for the CDT is also possible. One simply never considers flipping a source edge.

6.2.3 The flatness assumption

In order for edge-flipping to terminate, producing the unique surface DT, the surface must satisfy the local flatness assumption that no dihedral angle exceeds $\pi/2$. Rather than enforce this requirement by refining the triangulation in highly curved neighborhoods, as suggested in [Che93], we have found that it works well in practice to relax the requirement by maintaining only an approximate DT. We temporarily freeze edges with sharp dihedral angles, rather than allowing them to be flipped, and this preserves the algorithm's termination guarantee.

6.2.4 An incremental surface CDT

For a valid surface triangulation, and an edge e not on the boundary, let Q_e be the quadrilateral formed by taking the two triangles on either side of e (Figure 6.9). We say that Q_e is *reversed* if e forms a smaller minimum angle with the outside edges than the other diagonal does.

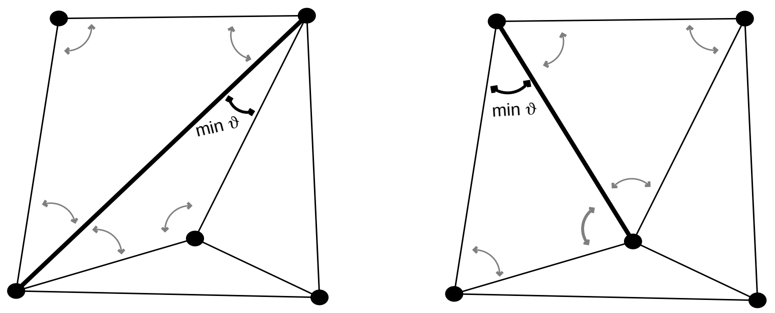


Figure 6.9: Construction of the planar Delaunay triangulation through iterative edge-flipping. The highlighted diagonal on the left is reversed within its quadrilateral. Flipping the edge increases the minimum included angle, and restores the DT (right).

Algorithm: Restore-CDT

```
1. Put all non-boundary, non-source edges into a queue.

2. Remove the first edge $e$ from the queue.

3. If $Q_e$ is reversed, and the dihedral angle across $e$ does not exceed $\pi/2$,
   and flipping $e$ does not introduce a dihedral angle exceeding $\pi/2$, replace
   $e$ in the mesh with $Q_e$'s other diagonal.  Add the non-boundary, non-source
   edges of $Q_e$ to the queue if not already present.

4. Continue removing, checking, and possibly flipping edges from the queue.
   When the queue is empty, the CDT has been restored.
```

This is a very convenient and inexpensive way to dynamically maintain a CDT over a gradually changing set of vertex positions. Although the algorithm formally terminates in $\mathcal{O}(edges^2)$ flips, it is more often the case that only a few edges will need to be flipped at any one time assuming the changes in vertex positions are small and given that we began with a DT over the original positions. We have found it desirable to introduce a bit of hysteresis by only flipping edges if they increase their local minimum-angle by some small minimum. This produces an approximate CDT by making edges somewhat more reluctant to flip, but it reduces "chattering".

6.3 Controlling the number of samples

The sample distribution and re-triangulation procedures above let us make optimal use of a given number of sample points. But as surface area grows and shrinks we would like to adjust the absolute number of samples so that the sampling density remains relatively fixed. Refinement/re-zoning schemes in finite element computations can be fairly complex. But given the machinery we already have in place, this last task is very simple for us. We measure edge lengths in 3D, and trigger an edge-split if any two neighbors are too far apart. Similarly, if any node is too close to each of its neighbors, the node is destroyed using the

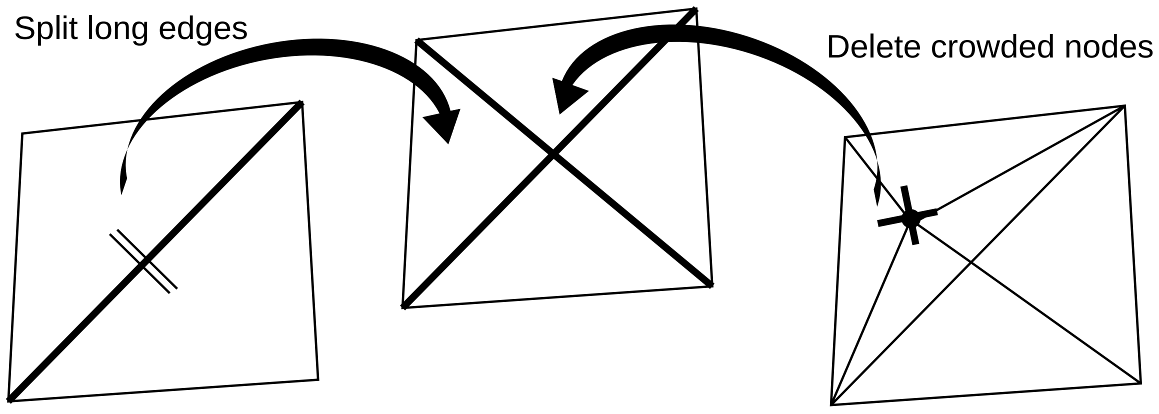


Figure 6.10: Adaptive mesh refinement: split edges that are too long, delete nodes that are too crowded. The triangulator improves the mesh afterwards.

node deletion algorithm from Chapter 4 (Figure 6.10). The sampling distribution and re-triangulation procedures above take care of restoring a quality mesh after one of these mesh transformation operations.

6.4 Mesh improvement algorithm

The techniques of the previous section can be packaged in a single "mesh improvement" step, to be applied to p.l. curves and surfaces. As with the quadratic shape steps of the previous chapter, the following steps will be iterated so that the mesh converges on a good distribution of nodes with well-shaped triangles (we discuss the interleaving of mesh shape and mesh improvement steps in the next chapter):

Algorithm: Mesh improvement step

```
1. Mesh refinement:  split long edges and delete crowded nodes.

2. Re-triangulation (surfaces only):  restore the surface DT by iteratively
   flipping edges to eliminate skinny triangles

3. Mesh smoothing:  adjust the distribution of the nodes by performing a curve
   or surface Laplacian smoothing step.
```

Ordering the operations this way seems to leave the best mesh at the end of the step. The re-triangulator gets a chance to clean up after the simple edge-splitting refinement operations, and the mesh smoother then gets to optimize neighborhood shapes after the triangulator has had its say.

Summary

We have described a novel method of maintaining a good computational mesh over a triangulated surface of arbitrary topology. The method is iterative and incremental, making it appropriate for use in an interactive surface modeler in which shapes change gradually over time. Nodes are kept evenly distributed over the surface by solving a version of Laplace's equation restricted to the surface. A surface Delaunay triangulation maintains well-shaped triangles over the nodes. A simple enrichment/depletion scheme adds nodes to sparse areas of the mesh, and deletes nodes from crowded areas.

Chapter 7

Implementing the Modeler

Synopsis

We combine the shape approximation and mesh optimization computations of the previous chapters. The combination serves as a black box that, given a smoothly changing variational specification (stored as a tagged topological mesh), outputs shape approximations in real time. We discuss a variety of region shape controllers that can be plugged into the mesh to control the shapes of constrained regions, and we consider other ways in which this "variational substrate" can be built upon to create interactive modeling application programs.

In previous chapters we have been through good deal of mathematics, and have devoted some discussion to solution and minimization techniques for our curve and surface fairing equations. But little has been said about how to combine all this to yield a working modeler. In this chapter we discuss the implementation of a variational surface modeler. The first four sections are devoted to algorithms and data-structures that let us wrap the mathematical machinery of the previous chapters into a "variational substrate." The goal is a black box that will accept variational surface specifications as input and serve up mesh approximations as output, thus encapsulating the details of mesh transformations, shape approximations, *etc.*.

We begin with a discussion of shape computations, describing how the shape and sampling computations of the previous chapters fit together and are applied to our mesh structures. Next is a brief a discussion of rendering techniques for meshed surfaces. Following this is a user (programmer) description of the variational substrate — what kinds of data and operations one works with in building a modeler on top of this substrate. The remaining sections of the chapter discuss examples of higher-level modeling operations specified in terms of this programmer's abstraction, drawn from a prototype direct-manipulation surface modeler that uses variational curves and surfaces as its basic free-form shape representation. Just as other modelers operate directly on B-spline or Bezier patch parameters, and leave it to lower-level routines to actually render the patches, our variational modeler operates directly on a variational specification (stored as a tagged topological mesh), and leaves it to the approximation machinery to maintain a good mesh representation of the shape.

7.1 Computing variational mesh and curve shapes

As discussed in Chapter 4, our tagged curve and surface meshes are decomposed into a disjoint union of regions, and each of the regions is assigned its own shape controller. Shape controllers for variational mesh regions are computed by iteratively minimizing a sequence of quadratic approximations to the true nonlinear objective function, as discussed in Chapter 5. On the other hand, *constrained* mesh regions will typically be tied to *explicit* controllers. Explicit controllers will write their positional information directly into the mesh points they control. A controller could be as simple as an interpolation point tracking a moving mouse position, or as complicated as the examples in Section 7.4.1. We consider here at a low level how to orchestrate the computations of these various controllers to produce an interactive modeler.

7.1.1 Basic shape iteration

```
1. For each constraint point (on curves or surfaces), invoke its region controller
   to update its position.

2. For each curve segment with an explicit shape controller, invoke the controller
   to compute a new curve shape.  The controller may depend on previously computed
   constraint point positions (from step 1).

3. For each faired curve segment, execute a step towards the minimum shape
   (Section 5.6), followed by a curve mesh improvement step (Section 6.4).

4. For each surface region with an explicit shape controller, invoke the controller
   to compute a new region shape.  The controller may depend on points or curves
   computed in steps 1, 2, or 3.

5. For each faired surface region, execute a step towards the minimum shape,
   followed by a surface mesh improvement step.
```

In the computations above, we always compute faired geometry after updating explicitly controlled geometry, so that faired elements may adjust their shapes in response to changes in the explicit elements.

In experimenting with this combination of shape and mesh improvement iterations, we tried leaving out various components of the quadratic and mesh improvement steps, or performing the components at differing frequencies. For example, we tried reparameterizing neighborhoods every few iterations instead of every time, and similarly for refinement and re-triangulation phases. The best overall behavior, in terms of stability and speed of convergence, resulted from performing each of the steps once each time through the approximation loop. For added speed and stability, we have found it helps to turn off re-parameterization, re-triangulation, and edge refinement whenever an "upstream" shape is changing rapidly (*e.g.*, when a user is tugging on a control curve). This is because a fast-moving constraint point typically causes the attached curve or surface to momentarily elongate, and automatic refinement at this point would likely be spurious and needlessly slow down the fairing computation. Keeping the mesh topology fixed at such times allows the shape to respond quickly to gross changes in the constraints. Once the user has stopped manipulating the control, surface refinement and re-triangulation may resume at a more leisurely pace.

7.1.2 Mesh improvement for other region controllers

The mesh improvement iteration above is general enough that it may also be used with other region controllers, not just variational patches. For example, in our implicit surface shape controller below (Section 7.4.1), we only worry about attracting the mesh nodes to the implicit surface, and rely on the mesh-improvement iteration above to keep nodes dispersed over the surface and nicely triangulated. In order for this to work well, the region controller should not attempt to move nodes in tangential directions when adjusting surface shape, but only move nodes along their current surface normals, as was discussed in Section 5.5.4. Tangential motion could compete with the mesh smoothing step, and delay or even prevent convergence. Of course, a region controller is always free to manage its own node distribution if desired.

7.1.3 Convergence

In considering the formal convergence of these iterations, there is little we can say; but in our experience the minimization has been well-behaved over a wide range of configurations. As with almost any nonlinear optimization posed as a sequence of quadratic subproblems, a caveat is that "reasonable" initial surface shapes must be used. It has not been necessary to precisely characterize what constitute good starting values for the optimization: because of the interactive nature of our system, changes to shape are generally incremental, with the endpoint of one iteration serving as the starting point of the next. It is certainly possible for a malicious user to make such a drastic change to a constraint in such a small interval of time that the computation diverges; but in practice the visual feedback from the modeler (in the form of animated, smoothly-evolving surface shapes) has been adequate to enable the user to change constraint shapes at a safe and reasonable speed. The place where we must generate reasonable starting shapes *a-priori* is when new surfaces are created through skinning operations. Our system uses only two such skinning operations, that construct cylinders and sheets, and we have found that linearly interpolating between their defining boundary curves works well.

Finally, it should be noted that curvature minimization, while well-behaved in a wide range of surface configurations, is not always the most desirable shape objective function. There are some configurations, such as narrow cylinders, in which it performs poorly (cylinders pinch in at the waist, and may even collapse). We re-visit some other possible fairness functions (and their difficulties) in Section 8.3.

7.1.4 Solver speeds

As discussed in Chapter 5, the equations to be solved when fairing curves and surfaces are very sparse. We use a conjugate gradient method to solve the associated linear systems, because it takes advantage of this sparsity without any additional work on our part (*i.e.*, in the form of special sparse matrix data structures). Technically, this requires $O(n^3)$

operations, where n is the number of nodes; but we really expect performance to scale better on average because of sparsity arising from the local structure of the mesh. We can't argue for this bound *a-priori* as we did in Chapter 4 using the Euler characteristic. The worst case surface mesh is a disc with a single central neighborhood, for then the Hessian does have $O(n^2)$ entries and solution takes $O(n^3)$ operations. But the re-triangulator tends to break surfaces up into evenly-sized neighborhoods (because of the relation of the DT to the Voronoi diagram), typically having 5–7 neighbors, and solver behavior scales more like $O(n^2)$.

Nonetheless, for large meshes ($>$ 100 nodes), solution times are still much too long for interactive modeling, where we can afford no more than 100ms per iteration (assuming we will re-render the mesh between each quadratic step). We therefore take advantage of another feature of the conjugate gradient solver: rather than viewing it as a "black box" linear system solver, we note that is actually an iterative solver, with successive iterations closing in on better solutions to the supplied linear system. Rather than run the method to full convergence, we allow the conjugate gradient solver only a fixed number of iterations (10–20) per solve/redraw cycle. This slows global convergence somewhat, but allows us to redraw the mesh often enough for the user that we maintain the illusion of a smoothly deforming surface over time. We have worked comfortably with models approaching 1000 nodes, running on a Silicon Graphics Indigo (R4000).

A side-effect of bailing out of the conjugate gradient solver early is that the intermediate stages we render are not themselves true minima, and the intermediate surfaces look as if they are moving through a viscous medium as the solution converges over time. It takes anywhere from 1 to 5 seconds for a surface to reach quiescence after a large change in a constraint position (intermediate shapes are continuously rendered during this time).

7.2 Rendering

7.2.1 Fast rendering

In between each shape iteration, we will re-draw the mesh shape for the user. To quickly render a mesh as a "smooth" surface, we evaluate normals at mesh nodes (using the fitted node quadratics), use these to compute a lighted color for each vertex (Phong shading[FW94]), and render each triangular facet by interpolating its vertex colors across the facet. This kind of rendering has hardware support on the graphics workstations we use (Silicon Graphics Indigos), and surfaces thus rendered are of passable quality for all but the coarsest of refinement levels. The technique has been used in all of the "smooth" renderings in this dissertation.

7.2.2 High-quality rendering

If speed is not an issue, a higher-quality (and accordingly more complex) approach to rendering uses the mesh's vertex/normal (and possibly curvature) data to fit a network of smooth patches. One of the simplest schemes is Nielson's G^1 triangle[Nie87], but as we discussed earlier, this and other purely local fitting approaches can yield unattractive curvature discontinuities across patch boundaries. Moreton's MVS interpolator[MS92], or

Halstead, *et al.*'s subdivision surface fairing[HKD93] are examples of globally smooth surface interpolation schemes that could be applied to a network of patches that would fill in the triangular facets of our mesh. Alternatively, if the cost of a global scheme is acceptable, we could use the variational specification encoded in the mesh to drive a high-quality smooth patch scheme, using the same objectives and constraints that we used in the discrete mesh approximation.

7.2.3 Orienting surfaces

An issue that arises when using a fast Phong shading scheme for lighting is that vertex normals must be consistently oriented "outward" across the surface. Our choice of normal orientation (or its consistency) doesn't affect the shape or sampling calculations, or affect lighting calculations that take the time to determine which side of a surface is facing the viewer, so this is only an issue for quick rendering. The orientation of a mesh node's computed normal (as given by the cross-product of u and v tangent vectors) is implicitly determined by the radial ordering of the neighbors about the node; reversing the order of the neighbor list reverses the direction of the computed normal but has no other effect. In choosing an orientation for a vertex normal, we are not helped by the fact that with bordered surfaces it is not always clear which side is the "out" side, and with non-orientable surfaces like the Möbius strip a consistent choice of normal direction isn't even possible. If the surface is actually orientable, it is straightforward to maintain a consistent orientation over various mesh transformation operations, or to impose an orientation *a-postiori* by orienting a single node and then propagating the orientation to the rest of the mesh (similar to [HDD+92]). For non-orientable surfaces, more careful rendering techniques must be used[1].

7.3 A programmer's view

Having exposed the inner workings of our variational substrate in some detail, it is time to see how much of that detail we can hide. In this section we'll discuss a simple user/programmer model of this computational machinery that we used in a simple modeler. The idea is to create and manipulate variational shape specifications as first-class objects while concealing the details of mesh-based shape approximation and mesh surgery.

7.3.1 Objects

The basic objects available to the programmer are:

- **point**: a 3D position
- **c-point**: a 3D point on a curve (corresponding to a node in the p.l. curve representation).
- **curve**: a sequence of **c-points** corresponding to a smooth (curvature-continuous) curve.

[1]Any non-orientable surfaces illustrated in this document have had their normal-field discontinuities carefully moved to the back-sides.

- **c-region**: a continuous subset of a curve, with boundary c-points
- **s-point**: a 3D point on a surface (corresponding to a node in the surface mesh).
- **surface**: a continuous collection of s-points and a surface triangulation over them, representing a curvature-continuous surface.
- **s-curve**: an embedded surface curve (open or closed), represented as a sequence of connected s-points.
- **s-region**: a continuous subset of a surface, with boundary s-curve.

These are created and modified by various topological operations, below, and their shapes are controlled by various region controllers.

7.3.2 Topological operations

We want the programmer to be able to create and make controlled changes to curve and surface topology, without having to worry about maintaining the consistency of the underlying mesh representation. We use the following topological operations, implemented in terms of the mesh constructs of Section 4.2.3:

- **curve(***point list***)**: create a free-standing curve, open or closed.
- **sheet(***boundary curve***)**: given a closed boundary s-curve or free-standing curve, fill in the "hole" with a surface sheet. In the case of a boundary s-curve, the sheet will meet the existing surface in a crease.
- **sweep(***c-region1, c-region2***)**: given a pair of s-curve or curve regions (or entire curves) of matching topology, create a swept surface between them, meeting any existing surface in a crease.
- **s-loop(***s-point list***)**: create an embedded surface curve.
- **burnout(***s-region***)**: destroy the surface region, leaving its boundary curves intact. If the region is a subset of some larger surface, the region's boundaries become border s-loops for that surface. This operation may actually split the surface into multiple unconnected regions.
- **split(***s-region***)**: split the surface along the region's boundary. This produces two independent surfaces, and clones the region boundary curve to make two independent control curves (in the case of a non-orientable surface such as a Möbius strip, there will be only one resulting surface).
- **crease(***s-region***)**: like split, but the control curve is not cloned and instead both surfaces remain attached to the original control curve. This relaxes the continuity of the surface by introducing a crease along the curve. The operation is not defined for a non-orientable surface region.
- **smooth(***loop***)**: The inverse of a crease operation, this takes two surfaces sharing a boundary control curve and merges them into a single curvature-continuous surface with an internal control curve where the crease used to be.

7.3.3 Shape control and region shadowing

But these are more than just collections of point constraints: presumably there is some separate, smooth curve or surface element whose shape we are trying to approximate with the constrained region. Without such an externally defined reference shape, it would be impossible to perform mesh refinement, UN-refinement, and optimization (Chapter 6) within these constrained regions, since we would not know how to reposition constrained sample points. In Chapter 7 we will discuss special *region controllers*, whose job it is to take mesh curves or regions and position their points in a sampling of the desired shape, in Chapter 7. Some care must be taken to coordinate global refinement and re-sampling operations with such controllers, especially when they themselves are variational shapes.

Shape control is organized in terms of topological regions and region controllers. Regions are created as side effects of higher-level user actions — *e.g.*, defining c-points on curves, and s-curves on surfaces (which act as region boundaries).

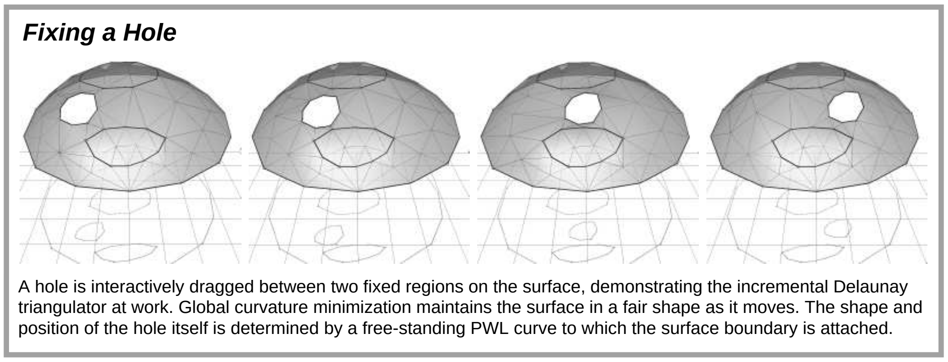
- **set-region-shaper(***region, shaper***)**: shaper may be `fair` for fairness-optimization, `shadow` to constrain region to follow another region, or the programmer may build an external shape controller to be applied to mesh regions, which will be called continuously to update the region's shape along with the built-in controllers.

- **make-curve-control-pt(***curve, vertex***)**: Create a free-standing point, initially located at the given curve vertex, and cause the vertex to shadow it.

- **make-surf-control-curve(***surf, vertex list***)** Create an s-curve and a free-standing control curve that overlays it, and cause the s-curve to shadow the control curve.

Interpolation constraints provide a partial specifications of curve and surface shapes, acting as a skeleton over which the topological skin will be draped (Figure 7.4). Whenever a user grabs a surface point during interaction, an appropriate constraint will be created and added to this skeleton so that the surface will be forced to follow the user's motions. A number of Higher-level shaping tools will interact with the specification by applying geometric constraints and manipulating them in coordinated ways.

Variational control curves are free-standing variational curves attached to corresponding s-curves. A convenient way to handling such situations uses a special *region shadowing controller*. Region shadowing lets us attach p.l. surface curves (sequences of mesh edges) to free-standing p.l. control curves. It is responsible not only for copying the the positions of its source nodes into those of the constrained shadowed nodes, but also for maintaining a one-to-one correspondence between nodes in the the source region and the shadow region. Whenever the source region undergoes refinement, the shadow region is refined as well, to preserve the correspondence. Thus, whenever a control curve edge is split, a shadow controller must perform a split on the corresponding edge of the embedded surface curve. This has a nice effect of allowing the control curve to determine the density of *surface* sampling in the neighborhood of the curve constraint. Though we have only applied shadowing to curves, and in a degenerate sense to control points on curves (the correspondence problem is trivial), shadowing could apply equally well to surface regions, *e.g.*, to apply an "embossing stamp" to a surface.



A hole is interactively dragged between two fixed regions on the surface, demonstrating the incremental Delaunay triangulator at work. Global curvature minimization maintains the surface in a fair shape as it moves. The shape and position of the hole itself is determined by a free-standing PWL curve to which the surface boundary is attached.

Figure 7.1:

7.3.4 Nonparametric interpolation and sliding features

In addition to the numerical conditioning benefits that derive from a good triangulation, interleaving mesh improvement steps with the fairing computation has the (somewhat surprising) effect of allowing nodes and edges to migrate across neighborhoods. Surface features like bounded subregions and embedded curves are free to slide around relative to each other within the surface triangulation, without disrupting the mesh's global surface topology (Figure 7.1). These are the nonparametric interpolation constraints discussed in Section 2.5.3. We avoid a difficulty experienced by smooth patch-based surface modelers, which must maintain parametric or *material* coordinates for such embedded curves, describing the (u, v) path of the curves across the patch. For a parametric surface, having a curve slide across the surface means adjusting its material coordinates to track its projected physical shape, and this is a messy nonlinear problem.

7.3.5 Mesh surgery revisited

Sometimes we will want to glue together surfaces along two unrelated boundary curves — curves that have different distributions of nodes, and may not even lie atop one another prior to the gluing operation. The curves must be brought into node-to-node correspondence and superpositioned before they can be glued together. One "high road" to solving this correspondence problem casts it as a minimum edit-distance problem [CLR90], and uses a dynamic programming scheme to transform one edge loop to match the other through a sequence of edge-splits, edge-collapses, and node repositionings[SG92a].

Though we implemented a similar scheme, we subsequently found it preferable (and much simpler) to avoid the superposition problem altogether, and simply create a cylindrical blend surface that bridges the space between the two boundary curves — assuming the curves are reasonably close to one another (Figure 7.2). After attaching the blend to the boundary curves, the boundary curves become interior loops within a single unified surface, and they may be retained as control curves or freed to yield an unconstrained surface. If both curves are retained as constraint curves, they may be used to control the tightness and continuity of this blend region. On the other hand, if one or both of the curves is freed, the

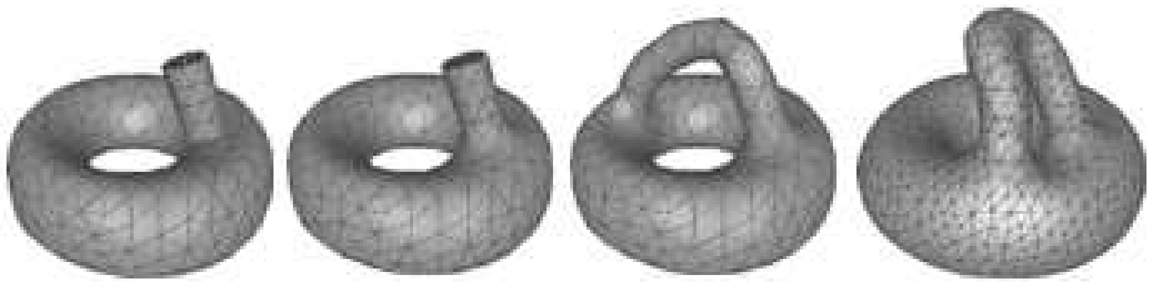


Figure 7.2: Attaching a handle: (1) one end of a cylinder is brought near the torus surface. (2) The user indicates that a *merge* is desired, and the modeler cuts a hole in the torus and attaches the cylinder with a blend surface. (3) The other side of the cylinder is similarly attached. (4) The user grabs the far handle attachment and slides it to the front, demonstrating nonparamteric interpolation constraints.

result is ultimately the same as if both curves had first been superimposed and then zipped together to join the surfaces.

7.3.6 Picking

We allow curve and surface points to be created and selected geometrically with picking operations:

- **ray-intersect-curve(***ray, curve***)**: find the point on the curve closest to the ray, locally refine the curve to place a c-point there, and return the c-point.
- **ray-intersect-surface(***ray, surface***)**: find the point on the surface closest to the ray, locally refine the surface to place a s-point there, and return the s-point.

7.4 Building on top of the substrate

Our intent is that machinery thus far described be used to manage the details of mesh maintenance and shape approximation on behalf of a higher-level modeler. This modeler in turn would offer more convenient ways of specifying and interacting with geometry. In this section we demonstrate some ways in which an external client of the substrate might extend its functionality.

7.4.1 External shape tools

One approach to using our free-form surfaces combines them with standard surface shapes like generalized cylinders, spheres, etc. Our surfaces act as blends to transition between portions of these structured surfaces, but with much more topological flexibility than traditional blend surfaces.

We can incorporate such structured shapes into a free-form model by writing region controllers that manage mesh approximations to the shapes. Given a triangulated collection

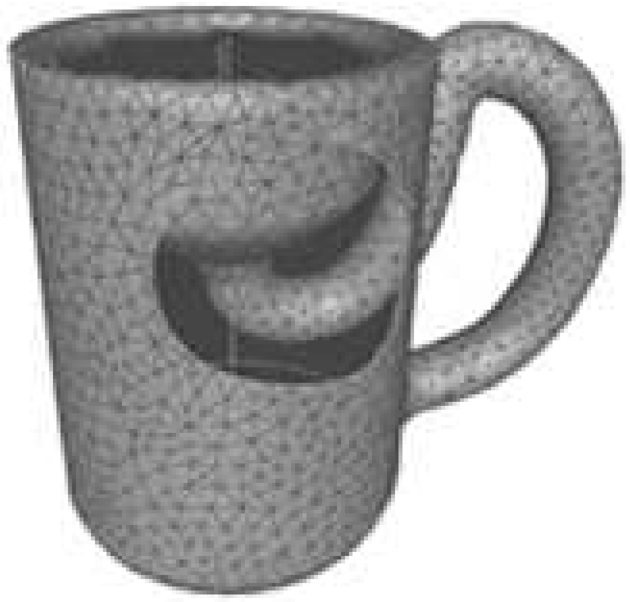


Figure 7.3: A *Klein mug*, a variation on the famous 1-sided surface. The handle and sidewalls are controlled by cylinder tools. Variational blend surfaces join the cylinders into a globally smooth, non-orientable surface.

of nodes (of suitable surface topology), a region controller's job is to position these nodes on its own separately-represented surface. The region's boundary points will also lie on this surface managed by the controller, so that they act as "trim-curves". The end effect is that we may punch holes in or extend simple structured shapes by trimming them and attaching variational blend surfaces smoothly along the trim curves.

Parametric curves and surfaces

The simplest region controllers are for surfaces that may be represented as functions mapping (u, v) surface coordinates to positions in $\mathcal{R}^3$, such as swept surfaces. The controller assigns a fixed (u, v) surface coordinate to each node in its region. Updating the region's shape is then a simple matter of evaluating the surface function for each node. We assume that the controller will do its own mesh management — in (u, v) space — using any of a variety of standard mesh generation techniques.

Algebraic curves and surfaces

Limited kinds of algebraic curves and surfaces (Chapter 2) can also be used as region controllers. Unlike parametric surfaces, it is not clear how to implement an algebraic region controller in a completely general way, because of the intrinsic difficulty of sampling and rendering contours of algebraic functions. If the topology of the algebraic surface itself is allowed to change interactively, as in [WH94], we would be faced with the nightmare of tracking this change and triggering mesh surgery as needed.

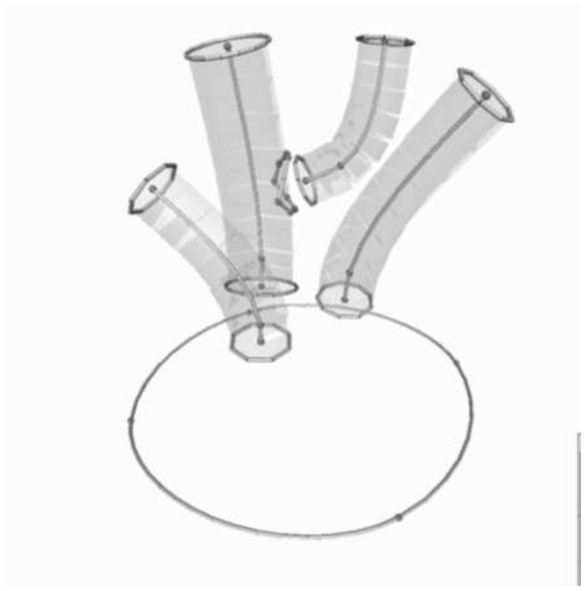


Figure 7.4: Control skeleton for a structured shape. The cylinder backbone curves, cylinder radii, cylinder attachment points and blend widths are all available to the designer as shape controls.

This didn't stop us from experimenting with a few algebraic controllers of known, fixed topology. The cylinder tool used in many of the figures is a mixture of parametric and implicit controllers. It is an offset surface from a variational backbone curve (the backbone is controlled like any other variational curve). For such a restricted surface, it is possible to create an initial mesh of appropriate topology and shape, and attract the nodes to the implicit surface using a constraint technique similar to [WH94]. This works almost acceptably in practice; there are problems if the algebraic surface moves too quickly for the node position controller to track its changes differentially, and it seems advisable to implement such controllers as parametric surfaces whenever possible.

7.4.2 Deformation hierarchies

Another useful way to add structure to a free-form surface model is to group constraints and move them in coordinated ways. Deformation hierarchies may be used to this end, as in [BJWK93]. For example, if a variational curve shape is controlled by a number of constraint points, we might choose to scale, translate, or otherwise transform some or all of these control points at once in order to simplify the re-shaping of the curve. A surface model structured this way could offer the designer a small set of parameters to control the size and positioning of major components, and rely on variational curves and surfaces to keep a smooth skin stretched over the whole assembly (Figure 7.4).

7.4.3 Auto-merge

Finally, we consider an example of a high-level modeling operation that makes simultaneous changes to shape and topology using basic operations above. One of the charming behaviors of Szeliski, *et al.*'s oriented particle systems[ST92] is the way two separate particle surfaces will automatically "melt" into each other when brought in close proximity. As we discussed in Chapter 2, one drawback here is that there is no explicit control over the topological change that occurs (*i.e.*, it is not clear how to bring two particle surfaces close together *without* having the merge take place).

We implemented a controlled version of this proximal merging; Figure 7.2 shows the user's view. As the two surfaces are moved close to one another, a transparent tube lights up along the line connecting their closest points, to show where an auto-merge would take place. If the user chooses to proceed, the modeler pierces each of the surfaces at its intersection with this line. Each intersection point is then "opened up" by first surrounding it with a closed s-curve, then burning out the s-curve's interior, and finally expanding the s-curve to a user-specified radius. A tubular surface is then swept between these two closed boundary curves, smoothly joining the two surfaces.

Summary

We have shown how to combine the variational specifications of Chapter 4, the mesh-based thin plate approximation scheme of Chapter 5, and the mesh maintenance scheme of Chapter 6 in an interactive surface modeler. The approach allows us to mix explicit shape controllers with implicit region fairing to make structured free-form models. A simple programming abstraction lets us hide the details of the mesh approximation machinery in a "variational substrate" that takes tagged topological meshes as input and renders approximate shapes in real time. The example surfaces throughout this document have been produced with this modeler. Our initial implementation of these ideas is a modeler that runs at interactive speeds on a Silicon Graphics Indigo class workstation, for surface models of several hundred to a thousand nodes. The models illustrated in this document range from 500 to 1200 nodes each. Each was created in only a few minutes time (by the author), using operations similar to the construction sequences in Figure 3.1 and Figure 7.2.

Chapter 8

Conclusion

In the previous chapters, we have outlined an approach to interactive modeling for fair surface shapes of arbitrary topological type. We must confess that the focus has been on the substantial mathematical and algorithmic hurdles to be cleared moreso than on practical application of the techniques. Nevertheless, at this point it is appropriate to compare this approach with some of the modeling approaches we originally surveyed in Chapter 2. This will be followed by a discussion of general limitations of our approach, and opportunities for future research. We conclude with a summary of the work's contributions.

8.1 Comparison with other free-form surface modeling approaches

- **Polygon meshes with locally smooth skins:** As was discussed, approaches to surface design involving local skinning of point or curve networks do not produce globally fair shapes. Additionally, there is no principled way to refine and unrefine

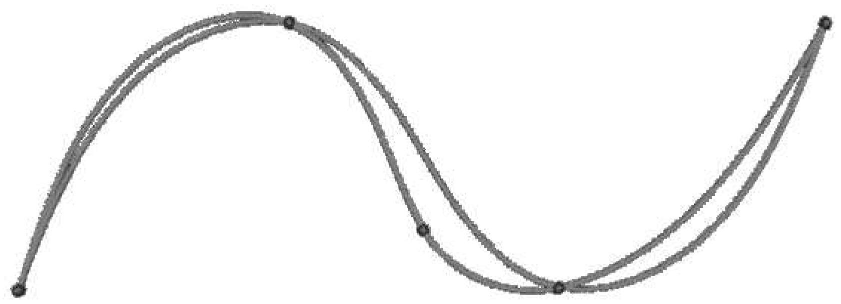


Figure 8.1: Revisiting a curve interpolation problem of Chapter 2: even though the user has grabbed and dragged a point near one of the original interpolation points, the shape remains fair. The success is due to the combination of our geometric thin plate function and adaptive curve sampling.

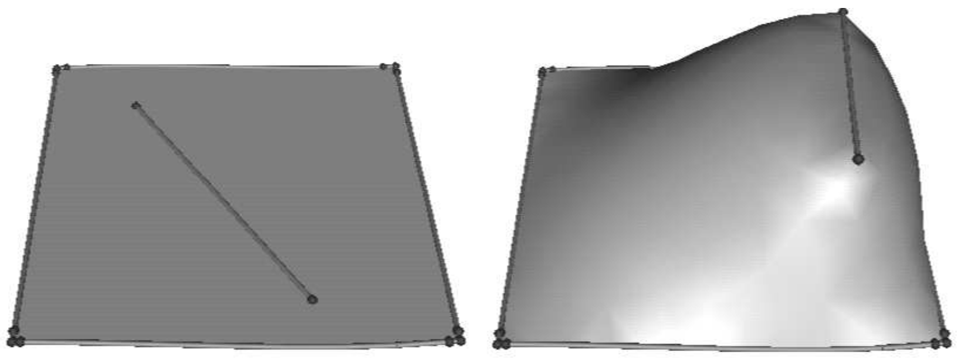


Figure 8.2: Revisiting a surface interpolation problem of Chapter 2: even though the user has grabbed and dragged a control curve in a twisting motion as before, the mesh adapts to let the control curve slide relative to the surface, yielding a fair surface shape.

such representations. Reference cannot be made to some invariant idealized shape in deciding where to place new points or reposition existing ones (the exception being the placement of new points on a subdivision surface). Our approach allows one to change mesh densities on-the-fly, while remaining faithful to an ideal variational shape.

- **Linearized thin-plate modelers:** The main drawback of previous interactive variational modelers — resulting from their use of a linearized thin-plate objective — was that surface shapes depended on an underlying surface parameterization. This parameterization easily becomes warped or stretched in the course of interactive reshaping, and surface shapes degrade as a result. Earlier, we displayed a number of defective curve and surface shapes suffering this effect (Figure 2.7, Figure 2.9). We have addressed this problem by our way of approximating the geometric thin plate objective, essentially by recomputing a chordal parameterization each time a curve or surface shape changes. Illustrations of our approach's performance on these "problem" interpolations appear in Figure 8.1 and Figure 8.2.

- **Volumetric modeling:** We have spoken of variational models as *structured shapes*, because a single specification can capture an entire family of related shapes. It is instructive to compare this with unstructured, volumetric sculpting approaches. To use an analogy from 2D drawing, comparing structured and unstructured shape models is like comparing drawings produced by MacDraw and MacPaint, respectively (though MacDraw does not allow one to set up dependencies between geometric elements as we can in our structured models). For complex, highly detailed sculpted shapes (*e.g.*, a gargoyle), structured modeling may be the wrong approach: much more effort will likely be involved in designing a control-curve model than if we were to directly sculpt the object — and the resulting parameterized model may be so complex as to not be a useful representation of a family of similar objects. This is when direct volume editing (or, more likely, honest clay modeling) makes the most sense. On the other hand, there is much parameterizable structure in everyday manufactured objects. Variational design (especially when mixed with local explicit control as we have done here) is more appropriate for describing families of shapes that may have simple parametric relationships to one another while not necessarily having simple shapes or topologies.

- **Generalized sweeps and cross-sectional design:** Sweeps are appropriate when the surface topology is simply parameterized and there is one or more symmetry axis. One may then specify profile and backbone curves instead of entire surfaces. Our variational approach is more appropriate for models that have no such inherent symmetries. Then again, the generating curves in a swept surface might be best described as variational curves; and a swept surface model could be stitched into a more general variational models using the shape-copying objective, giving us the best of both worlds.

8.2 Limitations

8.2.1 Solver complexity

The most expensive computational step in our shape approximation is the large linear system that must be solved at each time-step. Even at $O(n^2)$ complexity (as discussed in Chapter 7), this will prevent us from scaling up model sizes into the thousands of nodes while maintaining interactive performance. One possible way around this is to put portions of a large, complex model to "sleep" while the user is interacting with a smaller portion. It is reasonable to expect that the extremities of such a model would not change drastically in response to modeling operations that are not nearby (when we pull on the tail-fin of a 747, we do not expect the nose to wiggle). Disregarding nonlocal features (at least during intense local sculpting operation) is rather like putting a drop-cloth over the portions of a room that are not undergoing renovation, and would allow us to reign in the size of the linear systems we must actually solve.

8.2.2 The fairness objective

As we mentioned in Chapter 7, although the squared curvature objective function we use to measure surface fairness is a vast improvement over previous interactive modelers, it is not the highest-quality objective one might consider. When applied to a long, narrow tube, it causes the sides of the tube to bow in. In most other situations, the minimum is reasonable shape. But Moreton showed (using smooth patches) that even then, curvature tends to concentrate at region boundaries[Mor93]. This is perhaps counterintuitive, since a minimization of $\int \kappa^2 ds$ over a curve would seem to imply that curvature will be distributed evenly. In fact, this is only true when the curvature integration is performed over a *fixed-length* arc with suitable boundary conditions. Presumably, our curves are decreasing the magnitude of the global curvature integral by decreasing their total length, at the expense of uneven curvature distribution; surfaces behave analogously. We will consider other, higher-order fairness objectives that do not suffer this defect, in Section 8.3.

8.2.3 Self-intersection

A potentially important kind of constraint we do not address is one that prevents surfaces from self-intersecting. This would be a nice property to have in a modeler, *e.g.*, if the surfaces were intended as boundaries of homogeneous solids. A model that depends on prevention of self-intersection in order to define its shape is unusual: a knot in a rope that has been pulled tight is an example of the kind of sliding contact constraints involved. Note that the fact that the rope is or is not knotted isn't recorded in the rope's domain topology (which is simply a cylinder), but rather is a function of its immersion in space. Unfortunately, this is a rather difficult requirement of an interactive modeler, and it is not clear how to incorporate such a constraint into our approach while maintaining interactivity.

8.2.4 The triangulated surface representation

It is perhaps a little unsatisfying that our approach to smooth surface design never actually produces a smooth surface, but only point-wise approximations. Actually, when it comes down to exporting a model, there are many situations in which a triangle mesh is a desirable representation, provided it is fine enough. Needless to say, refining a mesh to a specified export resolution is straightforward with our approach.

If an explicit smooth surface is needed for any reason, it is possible to fit a surface to the triangulated surface points in a post-processing step. We can use the variational specification to set up an approximation using smooth parametric patches rather than a p.l. mesh. This is exactly the computation we ruled out in Chapter 2 as being too expensive; but interactivity is not an issue here. The mesh shape can supply a starting shape for the approximating surface, and thus speed the method's convergence (coming up with good initial values is both important and nontrivial in this kind of computation). This will be a more expensive computation than the constrained point-skinning above, but should yield higher-quality results for comparable mesh resolutions.

8.2.5 The approximation procedure

If we are to consider exporting the variational specification itself as a representation of a surface shape, so that another modeler could construct its own approximate shape using its own representation, we must wonder how well these two shapes will agree. When the specification is subsequently handed off to a more accurate, non-interactive shape approximation scheme, might we find that the approximated surface is no longer where the user intended? But this is exactly where geometric constraints come into play: if the surface is supposed to be positioned in some precise way, there should be constraints in place to do this. Given that variational shapes can only be approximated, it does not seem unreasonable to offer only qualitative assurances about the behavior of faired shapes away from explicit constraints.

8.3 Future Work

Though the limitations discussed above are not likely to be relaxed by any straightforward modifications to our approach, there are other useful improvements that are more within reach.

Objective function

Using variation of curvature as our objective function, as in [MS92] would lead to even higher-quality shapes than our current curvature minimization scheme delivers. A big difficulty with implementing such a scheme over our p.l. meshes is that we need 3rd derivatives. We cannot directly extend the finite-difference scheme to 3rd-order since we don't generally have enough neighbors to fit the additional coordinates. One possibility is to explicitly pull on normal vector alignments using a co-circularity objective as did Szeliski[ST91]. The difference between the directions of neighboring normal vectors acts as a kind of 2nd derivative (this goes back to the differential-geometric definition of sectional curvature). As discussed in Chapter 2, minimizing the variation between neighboring normal alignments is equivalent to minimizing a function of the 3rd derivatives over a patch.

Control curve networks

Though we formulate surface shape control using interpolated control curves, our scheme does not yet accommodate intersecting control curves. A compatibility condition[Pet91] demands that when control curves meet at a point, they must all fit a common quadratic surface form; otherwise, no there can be no smooth interpolating surface in the neighborhood of the intersection. What is needed is a special quadratic intersection node (essentially, a surface node) that enforces this compatibility constraint on curves meeting there, like the hub of an umbrella.

Curvature-adaptive sampling

It may be worthwhile to consider a curvature-sensitive scheme for distributing sample points across the surface. The error of our objective function integration in a neighborhood is related to the neighborhood's total curvature, and an adaptive scheme would would tend to distribute this error more evenly across nodes. This requires a modification of the mesh resampling scheme, since the centroid-based technique does not generalize well to nonuniform spacing. We actually experimented with a curvature weighting scheme, but the results were not particularly satisfying: mesh densities didn't decrease gradually away from the high-curvature areas, but rather changed rapidly to a uniform distribution. This is to be expected, given the Laplacian's aggressive smoothing characteristics, and a different smoothing function might be better suited for solution-adaptive meshing.

Smooth surface patches

It would be interesting to try these techniques with a smooth, geometrically continuous (or approximately continuous) surface patch scheme such as subdivision surfaces or Celniker triangles[Cel90]. This would shift the local shape computations from being node-based to being a patch-based finite-element scheme. We would need to give up the use of our version of the geometric thin-plate functional (which depended on faceted parameterizations to yield a quadratic minimization problem), and consider more a general, geometric objective function. Still, it is worth asking whether one could use substantially fewer patches than nodes to obtain a given approximation accuracy, and thus reduce the overall computational burden to a comparable level.

8.4 Review of contributions

The primary contributions of this work are:

- It is the first work to address interactive, incremental design of fair free-form surfaces of arbitrary topology. The user has explicit control over the topology at all times, and builds up complex topologies from simple ones through "surface surgery."
- We developed an approximation scheme for geometric thin plate surfaces, based on triangulated surface meshes, that is suitable for use in an interactive modeler: *i.e.*, it is *speedy and robust.*
- Included as a part of our approximation approach, but useful in its own right, is a robust method of computing neighborhood parameterizations for neighborhoods of an arbitrary shape/arbitrary topology p.l. mesh. (faceted parameterizations).
- We developed a mesh improvement scheme for arbitrary topology surface meshes. Meshes are automatically refined to a prescribed sampling density, nodes are smoothly distributed across surfaces, and a nice triangulation is maintained. Again, the approach is speedy and robust enough for use in an interactive modeler.
- We developed an approach to "sliding" interpolation constraints, as a side-effect of our curve and surface mesh improvement schemes.

As part of this work, we built an interactive, direct manipulation surface modeler that demonstrates all of the functionality discussed above (though no one would mistake it for a full-fledged industrial design tool). The modeler itself represents a number of secondary contributions:

- It is the first modeler that uses variational curve and surface specifications as its basic shape representation (built on top of the approximation machinery above).
- It is the first *variational* modeler that allows a designer to build up surface topology in terms of "surgical operations" on the 3D surface.

Summary

In this work we have developed a new approach to designing curves and free-form surfaces on a computer. Unlike related modeling approaches based on the notions of character lines and fair surfaces, we allow surfaces to be interactively cut apart and smoothly joined along arbitrary curves, so that complex shapes and topologies can be built up from simpler ones. Unlimited amounts of detail may be added to a model simply by indicating more control points and curves. Finally, portions of a curve or surface may be made to copy externally controlled *shape tools.* This allows us to mix free-form and structured shapes within a single composite surface model of arbitrary topology.

References

[Aki84] Hiroshi Akima. On estimating partial derivatives for bivariate interpolation of scattered data. *Rocky Mountain Journal of Mathematics*, 14(1), 1984.

[Ale30] James Alexander. The combinatorial theory of complexes. *Ann. Math*, 31:292–320, 1930.

[AMR83] J. Alander, M. Mantyla, and T. Rantanen. Solid modeling with parametric surfaces. In P. J. W. ten Hagen, editor, *Eurographics '83*, pages 71–83. North-Holland, 1983.

[Baj92] Chadrajit L. Bajaj. Smoothing polyhedra using implicit algebraic splines. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[Bar90] Timothy Barth. Higher order solution of the euler equations on unstructured grids using quadratic reconstruction. In *28th Aerospace Sciences Meeting*. AIAA-90-0013, 1990.

[Bar94] Ken Bartels. Rewarming hypothermic animals with microwaves. *Veterinary Forum*, pages 28–29, March 1994.

[Bau75] B. Baumgart. A polyhedron representation for computer vision. *AFIPS Nat. Conf. Proc.*, 44:589–596, 1975.

[BB89] Richard H. Bartels and John C. Beatty. A technique for the direct manipulation of spline curves. In *Proceedings, Graphics Interface*, 1989.

[BCGH92] Alan H. Barr, Bena Currin, Steven Gabriel, and John F. Hughes. Smooth interpolation of orientation with angular velocity constraints using quaternions. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[BCO81] Eric Becker, Graham Carey, and Tinsley Oden. *Finite Elements, an introduction*. Prentice-Hall, 1981.

[BE92] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In F.K. Hwang and D.Z. Du, editors, *Computing in Euclidean Geometry*. World Scientific, 1992. also appears as XEROX PARC report CSL-92-1, March 1992.

[BGW88] Chanderjit Bajaj, Thomas Garrity, and Joe Warren. On the applications of multi-equational resultants. COMP TR88-83, Department of Computer Science, Rice University, November 1988.

[BJWK93] D.L. Bonner, M.J. Jakiela, M. Watanabe, and N. Kishi. Pseudoedge: nonintersected parametric quilt modeling of multiply connected objects. *Computer Aided Design*, 25(7):438–452, July 1993.

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1, July 1982.

[Blo94] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[Boo79] F.L. Bookstein. Fitting conic sections to scattered data. *Computer Graphics and Image Processing*, 9:56–71, 1979.

[Boy91] Carl B. Boyer. *A history of mathematics*. John Wiley and Sons, Inc., 1991.

[BS82] J.U. Brackbill and J.S. Saltzmann. Adaptive zoning for singular problems in two dimensions. *J. Comp. Phys.*, 46:342–368, 1982.

[BS91] M. Boyer and N.F. Stewart. Modeling spaces for toleranced objects. *Intl. J. Robotics Research*, 1991.

[Cav74] J. C. Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *Int. J. for Numerical Methods in Engineering*, 8:679–696, 1974.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[Cel90] George Celniker. *ShapeWright: Finite Element Based Free-Form Shape Design.* PhD thesis, Department of Mechanical Engineering, MIT, 1990.

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite elements for free-form shape design. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 257–266, July 1991.

[CGP93] J.M. Chen, E. L. Gursoz, and F.B. Prinz. Integration of parametric geometry and non-manifold topology in geometric modeling. In J. Rossignac, J. Turner, and G. Allen, editors, *Proceedings, 2nd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 53–64, 1993.

[CH37] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume I. Wiley, 1937.

[Che93] Paul Chew. Guaranteed quality mesh generation for curved surfaces. In *Proceedings of the ACM Symposium on Computational Geometry*, 1993.

[CK83] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17, July 1983. (Proceedings Siggraph '83).

[CLR90] Thomas Corman, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[CW92] George Celniker and William Welch. Linear constraints for nonuniform B-spline surfaces. In *Proceedings, Symposium on Interactive 3D Graphics*, 1992.

[DC91] B.R. Donald and D.R. Chang. On the complexity of computing the homology type of a triangulation. In *Proc. 32nd Annual Symposium on Foundations of Computer Science*, pages 650–661. IEEE, IEEE Comput. Soc. Press, October 1991.

[DE93] C.J.A. Deflinado and H. Edelsbrunner. Incremental algorithm for betti numbers of simplicial complexes. Technical Report UIUCDCS-R-1787, U. Ill. at Urbana-Champaign, January 1993.

[DeR90] Tony D. DeRose. Necessary and sufficient conditions for tangent plane continuity of Bezier surfaces. *Computer Aided Geometric Design*, 7:43–55, 1990.

[DFFP85] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics, and Image Processing*, 32:127–140, 1985.

[dFGTV92] Luis Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulous, and Luiz Velho. Physically based methods for polygonization of implicit surfaces. In *Proceedings, Graphics Interface '92*, pages 250–257, 1992.

[DGR93] Nira Dyn, Ifat Goren, and Shmuel Rippa. Transforming triangulations in polygonal domains. *Computer Aided Geometric Design*, 10:531–536, 1993.

[Doo78] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[DWS93] Herve Delingette, Yasuhiko Watanabe, and Yasuhito Suenaga. Simplex based animation. In N.M. Thalman and D. Thalman, editors, *Models and Techniques in Computer Animation.* Springer-Verlag, 1993.

[EM94] H Edelsbrunner and E P Mücke. Three-dimensional alpha shapes. *Transactions on Graphics*, 13(1):43–72, 1994.

[Epp76] M. Eppstein. On the influence of parameterization in parametric interpolation. *SIAM J. Numer. Anal.*, 13:261–268, 1976.

[Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, 1990.

[FG92] L. Fang and D. C. Gossard. Recostruction of smooth parametric surfaces from unorganized data points. In *Curves and surfaces in computer vision and graphics III*, volume 1830. SPIE, 1992.

[Fie84] D. A. Field. Laplacian smoothing and delaunay triangulations. *Comm. Appl. Numer. Methods*, 4:709–712, 1984.

[FN90] Richard Franke and Gregory Nielson. Scattered data interpolation and applications: a tutorial and survey. In Hans Hagen and Dieter Roller, editors, *Geometric Modeling: methods and applications.* Springer-Verlag, 1990.

[For92] Steve Fortune. Voronoi diagrams and Delaunay triangulations. In F. K. Hwang and D.Z. Zu, editors, *Computing in Euclidean Geometry.* World Scientific, 1992.

[Fow92] Barry Fowler. Geometric manipulation of tensor-product surfaces. In *Proceedings, Symposium on Interactive 3D Graphics*, 1992.

[Fra82] Richard Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38(157), January 1982.

[FRC92] Helaman Ferguson, Alyn Rockwood, and Jordan Cox. Topological design of sculptured surfaces. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[FW60] G. Forsythe and W. Wasow. *Finite Difference Methods for Partial Differential Equations*, chapter 19, pages 179–182. John Wiley and Sons, 1960.

[FW94] Frederick Fisher and Andrew Woo. R.E versus N.H specular highlights. In Paul Heckbert, editor, *Graphics Gems IV*, pages 388–400. Academic Press, Boston, 1994.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 267–274, July 1991.

[Gle94] Michael Gleicher. *A Differential Approach to Graphical Interaction.* PhD thesis, Carnegie Mellon University, November 1994.

[GMW81] Phillip Gill, Walter Murray, and Margaret Wright. *Practical Optimization.* Academic Press, New York, NY, 1981.

[GP74] Victor Guillemin and Alan Pollack. *Differential Topology.* Prentice-Hall, 1974.

[GVL89] Gene Golub and Charles Van Loan. *Matrix Computations.* Johns Hopkins University Press, 1989.

[GW91] Michael Gleicher and Andrew Witkin. Differential manipulation. to appear in *Graphics Interface '91*, June 1991.

[Ham93] B. Hamann. Curvature approximation for triangulated surfaces. *Computing/Suppl*, 8:139–153, 1993.

[HB91a] H. Hagen and G. Bonneau. Variatinal design of smooth rational Bezier curves. *Computer Aided Geometric Design*, 8(5):393–400, 1991.

[HB91b] H. Hagen and G.P. Bonneau. Variational design of smooth ratinoal bezier curves. *Computer Aided Geometric Design*, 8:393–399, 1991.

[HB93] H. Hagen Hahmann and G.-P. Bonneau. Variational surface design and surface interrogation. In R. J. Hubbold and R. Juan, editors, *Eurographics '93*, pages 448–459, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[HDD$^+$92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):55–64, July 1992. (Proceedings Siggraph '92).

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of Siggraph 93*, July 1993.

[HDD$^+$94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of Siggraph 94*, July 1994.

[Hec94a] Paul Heckbert, editor. *Graphics Gems IV*. Academic Press, Boston, 1994. comes with either MAC or DOS floppy.

[Hec94b] Paul S. Heckbert. A minimal ray tracer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 375–381. Academic Press, Boston, 1994.

[HG94] Paul Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proceedings, Graphics Interface '94*, pages 43–50, 1994.

[HH87] C. Hoffmann and J. Hopcroft. The potential method for blending surfaces and corners. In G. Farin, editor, *Geometric Modelling*. SIAM, 1987.

[HKD93] Mark Halstead, Michel Kass, and Tony DeRose. Efficient, fair interpolation using catmull-clark surfaces. In *Proceedings of Siggraph 93*, 1993.

[Hof82] Hoffman. Relationship between truncation errors of centered finite difference approximation on uniform and nonuniform meshes. *J. Comp. Phys.*, 42:469–474, 1982.

[Hos88] J. Hoschek. Spline approximation of ofset curves. *Computer Aided Geometric Design*, 5:33–40, 1988.

[HS90] H. Hagen and G. Schulze. Variational principles in curve and surface design. In Hans Hagen and Dieter Roller, editors, *Geometric Modeling: methods and applications*. Springer-Verlag, 1990.

[HSW89] J. Hoschek, F. Schneider, and P. Wassum. Optimal approximate conversion of spline surfaces. *Computer Aided Geometric Design*, 6:293–306, 1989.

[Huf75] David A. Huffman. Curvature and creases: a primer on paper. In *Proc. Conf. Computer Graphics, Pattern Recognition, and Data Structures*. IEEE, May 1975.

[Jän84] Jänich. *Topology*. Springer-Verlag, 1984.

[Kal93] Michael Kallay. Constrained optimization in surface design. In *Modeling in Computer Graphics*. Springer Verlag, 1993.

[Koe90] Jan. J. Koenderink. *Solid Shape*. MIT Press, 1990.

[KR90] Michael Kallay and Bahram Ravani. Optimal twist vectors as a tool for interpolating a network of curves with a minimum energy surface. *Computer Aided Geometric Design*, 7:465–473, 1990.

[KWT87] Michael Kass, Andrew Witkin, and Dimetri Terzopoulos. Snakes: Active contour models. *International Journal Computer Vision*, 1(4), 1987.

[Lan56] Cornelius Lanczos. *Applied Analysis.* Dover, 1956.

[Law77] C.L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.

[LD90] Charles Loop and Tony DeRose. Generalized B-spline surfaces of arbitrary topology. *Computer Graphics*, 24(4), 1990. (Proceedings Siggraph '90).

[LGL81] V. C. Lin, D. C. Gossard, and R. A. Light. Variational geometry in computer-aided design. *Computer Graphics (SIGGRAPH '81 Proceedings)*, 15(3):171–177, August 1981.

[LH74] C.L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, 1974.

[LMD92] Michael Lounsbery, Stephen Mann, and Tony DeRose. Parametric surface interpolation. *IEEE Computer Graphics & Applications*, September 1992.

[Loo94] Charles Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[LP88] N. J. Lott and D. I. Pullin. Method for fairing b-spline surfaces. *Computer-Aided Design*, 20(10), 1988.

[Mal89] Jean-Laurent Mallet. Discrete smooth interpolation. *ACM Transactions on Graphics*, 8(2):121–144, 1989.

[Män83] M. Mäntylä. Topological analysis of polygon meshes. *Comput. Aided Des.*, 15:228–234, July 1983.

[Mas77] William S. Massey. *Algebraic topology, an introduction.* Springer-Verlag, 1977.

[Max54] James Clerk Maxwell. On the transformation of surfaces by bending. *Transactions of the Cambridge Philosophical Society*, 9:455–469, 1854. (Reprinted, 1890, in *The Scientific Papers of J.C.Maxwell, Cambridge University Press)*.

[Meh74] E. Mehlum. Nonlinear splines. In R. E. Barnhill and R. F. Reisenfeld, editors, *Computer Aided Geometric Design.* Academic Press, 1974.

[MLL+92] Steve Mann, Charles Loop, Michael Lounsbery, D. Meyers, J. Painter, Tony Derose, and K. Sloan. A survey of parametric scattered data fitting using triangular interpolants. In H. Hagen, editor, *Curve and Surface Modeling.* SIAM, 1992.

[Mor93] Henry P. Moreton. *Minimum Curvature Variation Curves, Networks, and Surfaces for Fair Free-form Shape Design.* PhD thesis, University of California, Berkeley, 1993.

[MPR+94] R. Merz, F. B. Prinz, K. Ramaswami, M. Terk, and L. E. Weiss. Shape deposition manufacturing. In *Proceedings of the Solid Freeform Fabrication Symposium.* U.T. Austin, 1994.

[MS78] B.A. Murtagh and M.A. Saunders. Large-scale linearly constrained optimization. *Math. Prog.*, 14:41–72, 1978.

[MS85] A.E. Middleditch and K.H. Sears. Blend surfaces for set theoretic volume modeling systems. *Computer graphics*, 19(3):161–170, 1985.

[MS92] Henry Moreton and Carlo Séquin. Functional minimization for fair surface design. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[MT78] Wayne Mastin and Joe Thompson. Elliptic systems and numerical transformations. *J. Math. Anal. and Appl.*, 62(52), 1978.

[NF83] Gregory Nielson and Richard Franke. Surface construction based on triangulation. In Robert Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 157–162. North-Holland, 1983.

[Nie74] Gregory Nielson. Some piecewise polynomial alternatives to splines under tension. In R.E Barnhill and Risenfeld R.F., editors, *Computer Aided Geometric Design*, pages 209–235. Academic Press, 1974.

[Nie87] Gregory Nielson. A transfinite, visually continuous, triangular interpolant. In Gerald Farin, editor, *Geometric Modelling*, pages 235–246. SIAM, 1987.

[Nie88] Gregory Nielson. Interactive surface design using triangular network splines. In *Proc. 3rd International Conf. Enginerring Graphics and Descriptive Geometry*, volume 2, pages 70–77. ASEE, 1988.

[NLL90] Horst Nowacki, Dingyuan Liu, and Xinmin Lu. Fairing bezier curves with constraints. *Computer Aided Geometric Design*, 7:43–55, 1990.

[NR83] Horst Nowacki and Dirk Reese. Design and fairing of ship surfaces. In Robert E. Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer-Aided Geometric Design*, pages 121–134. North-Holland, 1983.

[O'N66] Barrett O'Neill. *Elementary Differential Geometry.* Academic Press, 1966.

[PBCF93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(3):56–102, January 1993.

[Pet91] Jörg Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–246, 1991.

[Pot91] Helmut Pottmann. Scattered data interpolation based on generalized minimum norm networks. *Constructive Approximation*, 7:247–256, 1991.

[Pra87] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):145–152, July 1987.

[PS83] M. Plass and M. Stone. Curve fitting with piecewise parametric cubics. *Computer Graphics*, 7:229–239, 1983.

[PTVF94] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C.* Cambridge University Press, 1994.

[RA82] M. M. Rai and D. A. Anderson. Application of adaptive grids to fluid-flow problems with asymptotic solutions. *AIAA J.*, 20:496–502, 1982.

[RC86] A. A. G. Requicha and S. C. Chan. Representation of geometric features, tolerances, and attributes in solid modelers based on constructive geometry. *IEEE J. Robotics and Autom.*, RA-26(3):156–166, September 1986.

[Req80] A. A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *Computing Surveys*, 12(4):437–64, Dec 1980.

[RF89] D. F. Rogers and N. G Fog. Constrained b-spline curve and surface fitting. *Computer Aided Design*, 21(10), December 1989.

[Rie89] R.F. Riesenfeld. Design tools for shaping spline models. In Tom Lyche and Larry Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 499–519. Academic Press, 1989.

[RIKM93] Mervi Ranta, Masatomo Inui, Fumihiko Kimura, and Martti Mäntylä. Cut and paste based modeling with boundary features. In J. Rossignac, J. Turner, and G. Allen, editors, *Proceedings, 2nd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 23–34. ACM Press, 1993.

[Roc89] Alyn Rockwood. The displacement method for implicit blending surfaces in solid models. *ACM Transactions on Graphics*, 8(4):279–297, 1989.

[RV82] A. A. G. Requicha and H. B. Voelcker. Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, 1982.

[SBG93] K. Shimada, B. Balents, and D.C. Gossard. Automatic mesh reconstruction for feature-based sculpting of deformable surfaces. *Geometric Modeling for Product Realization. IFIP TC5/WG5.2, Working Conference on Geometric Modeling; Rensselaerville, NY, USA; IFIP Transactions B (Applications in Technology)*, B-8:165–188, October 1993.

[Sch66] D.G. Schweikert. An interpolation curve using a spline in tension. *Journal of Math and Phys.*, 45:312–317, 1966.

[SG92a] Thomas Sederberg and Eugene Greenwood. A physically based approach to 2D shape blending. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[SG92b] K. Shimada and D.C. Gossard. Computational methods for physically-based fe mesh generation. *Human Aspects in Computer Integrated Manufacturing. IFIP TC5/WG 5. 3 Eighth International PROLAMAT Conference. Man in CIM; Tokyo; IFIP Transactions B (Applications in Technology); vol.B-3*, B-3:411–420, June 1992.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings 23rd ACM National Conference*, pages 517–523, 1968.

[She94] Jonathan Richard Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1994. Available by anonymous FTP to WARP.CS.CMU.EDU (128.2.209.103) as quake-papers/painless-conjugate-gradient.ps.

[SK91] Yoshiuki Shiroma and Yukinori Kakazu. A generalized sweeping method for csg modeling. In J. Rossignac and J. Turner, editors, *Proceedings, 1st ACM/IEEE Symposium on Solid Modeling and Applications*, pages 149–157. ACM Press, 1991.

[SK92] John Snyder and Jim Kajiya. Generative modeling: a symbolic system for geometric modeling. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[SM91] Biplab Sarkar and Chia-Hsiang Menq. Parameter optimization in approximating curves and surfaces to measurement data. *Computer Aided Geometric Design*, 8:267–290, 1991.

[Spi79a] Michael Spivak. *A Comprehensive Introduction to Differential Geometry*, chapter II:3, page 99. Publish or Perish, Inc., 1979.

[Spi79b] Michael Spivak. *A Comprehensive Introduction to Differential Geometry.* Publish or Perish, Inc., 1979.

[Spi79c] Michael Spivak. *A Comprehensive Introduction to Differential Geometry, Volume 4.* Publish or Perish, Inc., 1979.

[SS87] L. Shirman and C. H. Séquin. Local surface interpolation with Bezier patches. *Computer Aided Geometric Design*, 4:279–295, 1987.

[ST91] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. Technical Report 91/14, Digital Equipment Corporation, Cambridge Research Lab, December 1991.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[Ste84] Sarah E. Stead. Estimation of gradients from scattered data. *Rocky Mountain Journal of Mathematics*, 14(1), 1984.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[STT93] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 82–87, New York, New York, June 1993.

[SW92] Ernest Stokely and Shang You Wu. Surface parameterization and curvature measurement of arbitrary 3-D objects: Five practical methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(8), August 1992.

[SZ90] P. T. Sander and S. W. Zucker. Inferring surface trace and differential structure from 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):833–854, September 1990.

[SZL93] William Schroeder, Jonathan Zarge, and William Lorensen. Decimation of triangle meshes. In *Proceedings of Siggraph 93*, July 1993.

[Ter86] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-8:413–424, 1986.

[Tho85] Joe F. Thompson. A survey of dynamically-adaptive grids in the numerical solution of partial differential equations. *Applied Numerical Mathematics*, 1:3–27, 1985.

[TL25] S. Timoshenko and J. M. Lessells. *Applied Elasticity*, chapter 10, page 270. Westinghouse Technical Night School Press, East Pittsburgh, PA, 1925.

[TM83] Joe F. Thompson and Wayne Mastin. order of difference expressions of curviliear coordinate systems. In *Advances in grid generation, ASME Fluids Engineering Conference*, 1983.

[Tru83] C. Truesdell. The influence of elasticity on analysis: the classic heritage. *Bulletin of the AMS*, 9(3):293–310, 1983.

[TTSC91] H. Toriya, T. Takamura, T. Satoh, and H. Chiyokura. Boolean operations for solids with free-form surfaces through polyhedral approximation. *Visual Computer*, 7:87–96, 1991. polyhedral approximation to intersections, then re-skin surfaces to meet smoothed approximate curve. Booleans are reversable (see toriya-1986).

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):289–298, July 1991.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, July 1992. (Proceedings Siggraph '92).

[TWM85] J.F. Thompson, Z.U.A. Warsi, and C.W. Mastin. *Numerical Grid Generation: Foundations and Applications*. North-Holland, 1985.

[War86] Z.U.A. Warsi. *J. Comput. Phys.*, 64(82), 1986.

[Wei85] K. Weiler. Edge-based data structured for solid modeling in curved-surface modeling environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. (Proceedings, 1990 Symposium on Interactive 3D Graphics).

[WH94] Andrew Witkin and Paul Heckbert. Using particles to sample and control implicit surfaces. *In these proceedings*, July 1994.

[WT90] Z.U.A. Warsi and Joe. F. Thompson. Application of variational methods in the fixed and adaptive grid generation. *Comp. Math. Applic.*, 19(8):31–41, 1990.

[WW92] William Welch and Andrew Witkin. Variational surface modeling. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[WW94] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. *Proceedings Siggraph '94*, July 1994.

[ZM83] O.C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation.* John Wiley and Sons, 1983.