# Shaping Curved Surfaces

by

John S. Rhoades

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science

Chapel Hill

1993

Approved by:

_____
Stephen M. Pizer, advisor and reader

_____
Gary Bishop, reader

_____
Robert B. Gardner, reader

_____
Dinesh Manocha, reader

_____
Jonathan A. Marshall, reader

John S. Rhoades. Shaping Curved Surfaces (Under the direction of Dr. Stephen M. Pizer, Kenan Professor of Computer Science)

# Abstract

This dissertation presents a new tool for shaping curved surfaces, the *bending operator*. The bending operator is an interactive tool intended for use in 3-D sketching. It is based on the idea that bending a surface is equivalent to changing its normal vector field while perturbing its metric as little as possible. The user of this tool specifies a bending operator, which is a surface that indicates how the normals of a target surface should change. The bending algorithm adds the derivatives of the normal vector fields of the bending and target surfaces and integrates this sum to produce a desired normal vector field. The target surface is then reshaped using a variational technique that moves the underlying surface control points to minimize a penalty function of the target surface. After bending, the resulting surface acquires the features of the bending surface while maintaining the general shape of the original target surface.

The bending algorithm can perform a wide variety of surface shaping tasks, including bending about a cylinder axis, indenting, twisting, and embossing. The algorithm includes a positioning control used to specify the correspondence between points of the bending operator surface and target surface and a range of action selector used to restrict the bending action to a part of the target surface. The bending operator is intuitive in that a user can easily learn to predict the approximate result of a bending operation without needing a detailed understanding of the algorithm. The algorithm can be applied to any patch type that is based on control points and that is piecewise twice differentiable, including Bezier patches, B-spline patches, and NURBS. The algorithm can also be applied to a non-branching mesh of patches with smoothness constraints. The bending algorithm was implemented in an interactive prototype program using X-windows. This program performs a bending operation in seconds to minutes on a HP-730 workstation depending on the complexity of the target and bending surfaces. The dissertation also includes an outline for a joining algorithm based on variational techniques similar to those used in bending.

# Acknowledgments

I thank my advisor and committee chairman, Stephen M. Pizer, for technical and moral support during this work. One could not ask for a better advisor. Thanks to Dr. Pizer, I now have some inkling of what it takes to do a major research project and write a major technical document.

I thank the other members of my committee, Gary Bishop, Robert B. Gardner, Dinesh Manocha, and Jonathan A. Marshall, for technical advice about the dissertation and for strategic advice about doing the research. They helped me clarify my goals and limit the work to a doable amount.

I thank David Banks, David Eberly, Elaine Cohen, and Jan Koenderink for technical discussions and advice. Their feedback greatly improved the quality of this dissertation.

I thank the following projects for monetary support. NIH Grant CA 47982, Medical Image Presentation, paid my salary for a year during this work. NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, paid for a very productive week-long visit to the University of Utah.

# Table of Contents

**Page**

# List of Figures

**Figure** **Page**

# List of Abbreviations

| | |
|---|---|
| **2-D** | two dimensional |
| **3-D** | three dimensional |
| **BFGS** | Broyden, Fletcher, Goldfarb, and Shanno |
| **CAGD** | computer-aided geometric design |
| **CAT** | computer axial tomography |
| **EFFD** | extended free form deformation |
| **FFD** | free form deformation |
| **HP** | Hewlett Packard |
| **NURBS** | non-uniform rational B-spline |
| **PADL** | part and assembly description language |
| **PDE** | partial differential equation |
| **SIGGRAPH** | special interest group on graphics |

# List of Symbols

| | |
|---|---|
| $\wedge$ | wedge product of forms |
| $\times$ | vector cross product |
| $\cdot$ | vector dot product |
| $\nabla_u f$ | directional derivative of $f$ in the $u$ direction |
| $\|v\|$ | vector or matrix norm of $v$ |
| $C^k$ | parametric continuity of degree $k$ |
| $df$ | differential of $f$ |
| $G^k$ | geometric continuity of degree $k$ |
| $g_u$ | partial derivative of $g$ with respect to $u$ |
| $I$ | first fundamental form |
| $II$ | second fundamental form |
| $\Re$ | set of real numbers |
| $\Re^n$ | set of $n$-tuples of real numbers |
| $v_p[f]$ | application of tangent vector to function (directional derivative) |

# Chapter 1
# Thesis

## 1.1 Goal of this work

My goal in this dissertation is to develop and demonstrate a set of intuitive, high level operators for shaping curved surfaces. These operators are intended to be tools in a 3-D graphics modeling system. They are high level in that they permit a user to modify the shape of a surface without being concerned with patch control points or patch boundaries. The operators automatically adjust the control points to achieve the effect requested by the user. They are defined independently of patch type and thus are applicable to a wide variety of patch types, including Bezier patches, B-spline patches, and NURBS. I present two such operators, a bending operator and a joining operator. The bending operator is fully developed and implemented in a software prototype. The joining operator is defined but not implemented.

## 1.2 Prerequisites for the reader

It is possible for the reader to understand the basics of this dissertation without an advanced mathematics background. However, to understand the mathematical parts of this dissertation, the reader needs to be familiar with the fundamental concepts of differential geometry of 2-D surfaces in 3-D space. In particular, the reader should have at least an intuitive grasp of the notions of parametric patches, vector fields, forms, the metric, curvature, the Cartan structure equations, and the distinction between intrinsic and extrinsic surface properties. To this end, I include Appendix A, which is a brief introduction to this subject with pointers to the literature for more detail. I have endeavored to make the presentation comprehensible with this bare minimum of

background; however, Section 3.3, which describes the mathematical justification for the bending algorithm, cannot be completely understood unless the reader has a working knowledge of exterior differential operators and the algebra of p-forms. These parts are primarily proofs of theorems and mathematical justifications of definitions. In these cases, though, I have attempted to give intuitive explanations, with the aid of figures and diagrams, of what these theorems and definitions are saying.

### 1.3 The driving problem

Existing 3-D computer graphics modeling systems require overspecification and thus are difficult to use. The following paragraphs give background on conventional modeling approaches and explain why this problem came about.

The 3-D modeling systems available today are primarily based on four techniques: direct construction of polygon sets, generalized cylinders (surfaces of revolution), constructive solid geometry, and splines. For example, the AutoCAD system[1] and the Wavefront system[2] let the user specify simple polygon sets (boxes, cones, and spheres made of polygon meshes, as well as individual polygons) and surfaces of revolution. The Designbase system [ Chiyokura 88 ] and the Alpha_1 system[3] [ Alpha_1 90 ] allow the user to generate spline surfaces by manipulating control points. The PADL system [ Voelcker 93 ], the Designbase system, and the Alpha_1 system also allow the user to build up complex models by intersection, union, and differences of volume elements.

How do these 3-D modeling programs require the user to overspecify? When generating a model for interactive graphics applications, the user frequently wants a model that looks like some familiar object (for example, a telephone handset) but cares neither about the exact dimensions of the object, nor the exact number and arrangement of polygon vertices or spline control points. The systems mentioned above, however, require that some or all of these details be specified. This is the overspecification problem. When a model is

[1] Commercially available from Autodesk, Inc., Sausalito, CA.
[2] Commercially available from Wavefront Technologies, Santa Barbara, CA.
[3] Commercially available from Engineering Geometry Systems, Salt Lake City, UT.

being built with these systems, modifying the shape is awkward. Support for direct polygon sets is generally limited to overall scaling and positioning of subsets and movement of single vertices. Systems based on constructive solid geometry suffer from a restricted set of primitives, registration problems among the solid primitives used to generate the model, and difficulty in forming smooth joins between parts of the model. Systems that are based on splines usually allow only very local shape changes based on movement of single control points or small groups of control points.

In this dissertation I am primarily concerned with the problem of geometric modeling with curved surfaces. State-of-the-art geometric modeling software is generally based on the use of sculptured surfaces comprised of parametric surface patches. A *parametric surface patch* is a function from some simple domain in 2-space, typically a triangle or square, to 3-space. Such patches are usually defined by polynomials (Bezier patches), piecewise polynomials (B-splines), or piecewise rational functions (NURBS). The coefficients of the polynomials or rational functions are specified (indirectly) by a set of 3-dimensional *control points*. The patches are designed so that the surface is a smooth function that approximates the geometry of the control points. By design, patches usually have some nice continuity, subdivision, local control, and convex hull properties that make them easy to work with from a mathematical and programming standpoint. Objects of any complexity cannot be represented by a single patch; instead, a *mesh* of patches with certain smoothness criteria at the junctions is used. Even fairly simple objects require tens of patches with hundreds of control points for a faithful patch-based geometric model. For example, the well-known Utah teapot uses 32 Bezier patches with a total of 512 3-D control points in its definition.

The problem with control points is that they do not directly provide a very nice level of interaction from a human user point of view, however nice they may be from a mathematical or programming point of view. There are three reasons for this. First, there are just too many of them; hundreds or thousands are needed even for fairly simple shapes. Second, they are somewhat arbitrary in the sense that virtually identical surfaces can be represented in many different ways; there are choices to make about number and size of patches, degree of the polynomials defining the patch, etc. These two reasons

represent the overspecification problem. Third, the effects on the surface of manipulating a single control point are often not very intuitive to a human user. Some surface properties, such as smooth joins between patches, require complex algebraic relations to hold among several control points. Such relations are practically impossible to achieve by direct user manipulation of single control points. In short, control points can be thought of as the assembly language of shape definition.

What is needed in a good user interface is a higher level, more intuitive method of shape control. Such a method might use an underlying structure based on control points, but it would not show control points to the user unless asked. There are, of course, many possible ways to design such an interface. My idea of a useful approach is as follows. The interface would present to the user a palette of simple starting shapes and a palette of *shape modification tools*. To build a more complex object, the user interactively selects a few of the starting shapes, applies the shape modification tools repeatedly to customize them, assembles them into an overall figure, and finally joins them together, again using the shape modification tools. The starting shapes would include squares, blocks, cylinders, spherical caps, cones, etc., possibly a dozen or so shapes all told. The shape modification tools would include positioning, scaling, bending, twisting, and indenting for individual shapes and a joining operator to stick together two or more shapes. I am proposing a geometric modeler based on surface representations, not on constructive solid geometry, although one could imagine combining the two ideas.

Clearly, building a complete 3-D graphics modeling system based on this concept would be a major undertaking and is more than I propose to do in this dissertation. What I do propose is to address the problem of defining and implementing the shape modification tools.

### 1.4 Overview of results

I have developed a new concept of what bending is that is both general and intuitive. It is general in the sense that it includes not only simple bending around a cylinder axis, but also twisting, indenting, and embossing. It is intuitive in the sense that a user can easily

learn to predict the result of applying a given bending operator to a given surface. My concept is basically that bending is modification of surface normals. The *bending operator*, itself a surface, specifies how the surface normals of a *target surface* will be perturbed. The user simply applies the bending operator to a whole or part of a surface, without regard to control points. The bending algorithm automatically computes the control point movements needed using a variational method. The bending operator is a key tool in my ideal modeling system proposed in the previous section.

To design an algorithm for the bending operator, I broke down the bending process into three stages.

1) The user selects or creates a bending surface and positions it with respect to the target surface.

2) The bending algorithm computes a desired normal vector field for the target surface.

3) The bending algorithm reshapes the target surface with the dual goals of maintaining its metric and matching its true normals to the desired normals from stage 2.

Each of these stages generated a set of sub-problems that I needed to solve.

Stage 1 required the design of a positioning control, which is a method for permitting the user to identify corresponding points in the bending surface and target surface. I developed a general purpose positioning control that works for any bending surface and any target surface and that is intuitive and easy to use. The method, described in detail in Section 3.4, involves an affine transformation between the parameter space domains of the two surfaces and a rotation in the range space of the bending patch. I explained why it is necessary to work with both spaces.

Stage 2 required the design of an algorithm for combining two normal vector fields to produce a new vector field that reflects the properties of both. I explained in Section 3.3.1 why the important issue for bending is the *variation* of the normal vectors with respect to

movements in the surface. I showed how to formulate the desired normal vector field as the solution to a certain differential system. I applied the Frobenius theorem to prove that the differential system is integrable, which implies that it has a unique, global solution. I showed in Section 3.6.3 how to solve the differential system, by converting it to a system of ordinary differential equations on curves in surface parameter space and then applying a standard numerical integration algorithm to this system.

Stage 3 required the design of an algorithm for reshaping the surface to cause its true normals to match the desired normals from the second stage. I explained in Section 3.3.2 why the specification of the desired normals alone leads to an under-constrained problem – there may be many surfaces that have exactly the desired normals (this fact justifies separating stages 2 and 3). I argued that the metric (local stretch and shear) of the target surface is the geometric property that complements the desired normals to make the reshaping process well-defined. I explained how variational methods can be used to implement the reshaping algorithm. Variational methods for surfaces work by using an optimization algorithm that adjusts the surface control points to minimize a penalty function. The penalty function assigns a numerical measure to the extent to which the surface fails to attain its goals. It is usually impossible to exactly satisfy the surface goals, and the variational method is a good way to find the best approximate solution.

I constructed a composite penalty function which is a weighted sum of four component penalty functions. Section 3.3.2 gives the design criteria and the formulas for these. The first component measures the deviation between the actual and desired normals of the target surface. The second component measures deviations of the metric of the target surface from its initial value. The third component measures deviations of the curvature of the target surface from its initial value. This component was needed to implement a range of action control, which allows the user to restrict the bending action to a portion of the target patch. Experimentation with the software prototype showed that these three components were insufficient – folds and creases sometimes formed during the bending process. I added a fourth component, which penalizes collapse of the local area element, to eliminate this problem.

I have implemented a software prototype to show the feasibility of the bending operator. The prototype includes an interactive graphical user interface for setting up a bending operator and applying it to a B-spline surface patch or continuous mesh of B-spline patches and an algorithm that carries out the bending operation. The prototype runs on any system with UNIX and X-windows. The user interface provides a wireframe or shaded display of the surface being manipulated and the ability to interactively control the viewing parameters. On a desktop workstation such as the HP-730, the bending algorithm is not quite interactive; a typical bending operation takes about a minute to perform.

The other key shape modification operator in my ideal modeling system is a joining tool. I have investigated how the methodology of the bending operator could be applied to develop such a tool. I present an outline of an algorithm for smoothly joining two surface patches along a common edge.

# Chapter 2
# Previous Work

Many successful geometric modeling systems in recent years make use of a tool-based user interface. Research into design of new tools is very active. This work is enabled by the increase of available computing power in interactive systems and is encouraged by the enthusiasm of users and the proven success of the tool-based approach. In this chapter I survey the previous work on modeling tools, with special emphasis on precursors to my bending and joining operators. I give examples of such tools, discuss their strengths and weaknesses, and discuss how my bending and joining operators compare to them.

## 2.1 Tool-based geometric modelers

During the past ten years or so, the geometric modeling community has gradually come to the realization that direct manipulation of control points is an awkward and unsatisfactory user interface technique for designing with smooth, curved surfaces. From mathematical and computational viewpoints, control points are very nice, but for a model of any complexity, there are too many control points and their effects on the surfaces they define are both too subtle and too limited for convenient direct control. A *tool-based* approach for user interfaces is gaining in popularity. With a tool-based user interface, the user does not directly create the patches or other primitives that make up the model. Instead, the user is provided with a tool kit containing operators for creating or modifying the shape of such primitives. An early example of such a tool, already in use fifteen years ago, is the sweeping or skinning tool. With the sweeping tool, the user defines a curve, and then creates a surface by sweeping the curve through space.

In recent years the variety and power of the available modeling tools have increased dramatically. These tools can be categorized into roughly three classes: surface warp techniques, space warp techniques, and physically based techniques. Surface warp techniques operate directly on a surface patch or mesh without considering that the surface may be the boundary of some solid object. Examples of surface warp methods are the bending and indenting operators developed by Cobb [ Cobb 84 ] and implemented in the Alpha_1 modeling system [ Alpha_1 90 ]. The bending and joining operators described in this dissertation are in this category. Space warp techniques work by changing the shape of the ambient space in which the surface resides or by changing the shape of a solid object of which the surface is a boundary. An example of a space warp method is the Free-Form Deformation (FFD) tool pioneered by Sederberg and Perry [ Sederberg 86 ] and later extensions implemented in the ACTION3D modeling system developed by SOGITEC and INRIA [ Coquillart 90 ]. Physically based modeling techniques create or modify surface shape automatically or semiautomatically by forcing a surface or solid to obey constraints based on physics, e.g., strain energy minimization or volume preservation. Physically based methods are not exactly modeling tools, and I will not say much more about them, except to point out some similarities of implementation to the bending operator.

Tool-based modelers are attractive to users because they present an intuitive and easily learned interface. I think that this is because they mimic to some extent the real world, in which real models can be built using mainly a small class of predefined objects (bar stock, blocks of material) and shaping tools (lathes, files, drills, chisels.) Tool-based modeling systems have been successful in production use: the Alpha_1 and ACTION3D systems are examples. Chadwick et al. [ Chadwick 89 ] at Ohio State used a modeler with the FFD tool to model muscle and skin of animated characters.

As modeling tools become more sophisticated, the geometric modeling community is beginning to realize the importance of separating the underlying modeling primitives from the modeling tools. Ideally, these would be completely orthogonal issues. There are two important reasons for this. First, one would like to use a new tool with as many existing primitive types as possible and to have the possibility of using newly discovered

primitive types with existing tools. Second, one would like to avoid constraints due to limitations of the modeling primitives from affecting the result of applying the modeling tools. An example may help to clarify this second point. Consider an indenting tool being applied to a mesh of Bezier patches. The user might wish to create an indentation much smaller than the spacing of the control points would allow. Coquillart's ACTION3D system (based on the FFD tool) has no problem with this, because during rendering, the indentation is processed along with the Bezier patches. Another approach, used by Welch and Witkin [ Welch 92 ] of Carnegie Mellon, is automatic refinement of the primitives based on the needs of the modeling tool. The bending tool I describe could in principle be used in conjunction with automatic refinement, but that is not implemented in my prototype software.

In the rest of this chapter I discuss in some detail the previous work on modeling tools based on surface warp and on space warp. I give examples of such tools, discuss their strengths and weaknesses, and discuss how my bending and joining operators compare to them.

**2.2 Surface warping**

The earliest tool for surface shaping seems to be the sweep operator. The idea is to produce a surface by sweeping a curve through space. In sophisticated versions, the curve may change shape during its movement. This technique dates back at least to the 1960's, and I could not discover who should have credit for its invention. S. Coons seems to have written the first description of a computer implementation [ Coons 67 ]. Properly speaking, the sweep operator is a method for generating surfaces rather than a tool for modifying them.

The most direct precursor to my bending operator is described in E. S. Cobb's Ph.D. dissertation entitled "Design of Sculptured Surfaces using the B-Spline Representation" [ Cobb 84 ]. She described a set of "surface modification operators." These surface modification operators are now incorporated into the Alpha_1 modeling system [ Alpha_1 90 ]. Two operators are described, a bending operator and a warping operator.

The bending operator performs a cylindrical bend of a single B-spline patch about one of the coordinate axes. The restriction to a coordinate axis is not as severe as it might seem, since the patch could later be rotated arbitrarily. The warping operator produces indentations or protrusions in a single B-spline patch. The boundary of the warp is specified by a polygonal region. The result of an operation is a new patch with recomputed control points. If the control points are not sufficiently dense to support the desired curvature, the user must manually add more using the "addFlex" operator. The tools are not interactive; the user must write a script in the Alpha_1 model definition language to describe the modeling operations. Cobb later used these operators in Alpha_1 to design and automatically manufacture some shapes, including a spoon. Since my bending operator is defined by an arbitrary bending surface, it generalizes both Cobb's bending operator and warping operator. My bending operator is defined independently of patch type, although implemented only for B-spline patches and smooth meshes of B-spline patches. My implementation has the same problem with density of the control points as Cobb's, but there is no theoretical problem with adding automatic refinement to the algorithm.

Hagen et al. [ Hagen 87 ] describes a tool for automatic smoothing of surfaces. The user begins with a rough approximation of the desired surface using a mesh of polygons, Bezier patches, or B-spline patches. The smoothing operator is then used to turn the approximation into a "fair" or smooth surface, using variational techniques. Others have described alternatives and extensions to this method [ Celniker 91 ] [ Moreton 92 ]. Automatic smoothing is not, by itself, a surface shaping tool. Rather, it is a method that can be used in conjunction with shaping tools to improve the final resulting surface.

Forsey and Bartels [ Forsey 88 ] describe a technique for direct manipulation of points on a surface. Basically their scheme is a hierarchical subdivision technique for B-spline surfaces. The user can select a surface point and a range of action about that point, and then pull or push the point in any direction. The surface responds like an elastic membrane, with the flexibility limited to the range of action. The implementation involves the use of a hierarchical set of "overlays," which are smaller patches that are vectorially added to the original surface patch. The strength of this approach is the

automatic creation of more underlying control points as needed by the nature of the interaction.

Several groups have recently been working on variational approaches to surface shaping. Celniker and Gossard [ Celniker 91 ] discuss a technique for smooth deformation of curves and surfaces based on minimizing an energy function. In their technique a surface is equipped with an energy function which measures bending and stretching distortion under an externally applied "sculpting force." With this method, the user supplies a sculpting force at selected points or curves in the surface, and a minimization algorithm is run to find a distortion of the surface that minimizes the energy. The surface shape is computed by minimizing an area integral of the form

$$E_{deformation} = \int_{\sigma} \|G\|_{\alpha}^2 + \|B\|_{\beta}^2 + f,$$

where $G$ is the matrix of the first fundamental form, $B$ is the matrix of the second fundamental form of the surface, and $f$ represents the sculpting force. They approximate the integrand by a quadratic function of the patch first and second derivatives so that the minimization problem can be handled by linear methods. The main reason for this approximation is to be able to perform the minimization at interactive speeds.

Welch and Witkin [ Welch 92 ] are currently implementing an interactive surface modeler based on a similar approach. They minimize an energy function that has a form like that of Celniker and Gossard, except that there is no sculpting force. Instead, the user manipulates point and curve constraints to control the shape of the surface. Their energy function has the form

$$E_{deformation} = \int_{\sigma} \sum_{i,j=1}^{2} \alpha_{ij} D_i w D_j w + \beta_{ij} \left(D_i D_j w\right)^2 ,$$

where $w$ is the patch function and $D_i$ is partial differentiation with respect to a patch parameter. This $E_{deformation}$ is essentially a quadratic approximation to $\int_{\sigma} \|G\|_{\alpha}^2 + \|B\|_{\beta}^2$, to

permit the use of linear solution methods. They note that the approximation is not very good except near a minimum (where the higher order terms tend to zero) but that it is nevertheless a well-behaved function that produces smooth surfaces. A variation of this formula is to replace $w$ by $w$-$w_0$, where $w_0$ is the patch function for a prototype, or rest shape. In this case the energy function measures deviation from the prototype shape.

Welch and Witkin allow the user to place arbitrary point and curve constraints on the surface. They mention a "normal vector" constraint but do not explain it further. The modeling software then performs a constrained optimization in order that the surface deforms smoothly while meeting the constraints. The implementation as of early 1992 on a Silicon Graphics machine is able to perform the computation at interactive speeds. The user sees an infinitely malleable surface on which he can freely place "handles" to control he shape. At any time the user can install the current shape as the rest shape. A tensor product B-spline surface representation is used, and the modeling system does automatic refinement if the user attempts to add features that are too small to be represented by the control point mesh.

This approach is quite similar to mine. In a sense, this is an idea whose time has come (SIGGRAPH 92 has a whole session, with four papers on the subject of variational curve and surface modeling,) due to the recent availability of desktop computers that have enough power to solve such variational problems in a reasonable amount of time. I am also minimizing an energy function, but instead of using explicit sculpting forces or constraints, I replace the second fundamental form component of the integrand with a measure of the deviation between the actual and desired surface normals. The chief difference in my approach is the use of my new bending operator tool, which uses the normal vector field of a second surface to modify the normal vector field of a target surface. In my method, the integrand is not limited to a quadratic function of the control points, so I cannot use linear methods for the minimization. This means that I have sacrificed interactive speed, at least on today's workstations. What I have gained is the ability to use more complex functions to define the energy integral. This is important, since the logical choices for the energy functions for the bending and joining operators are not quadratic.

## 2.3 Space warping

The earliest published work on space warping tools is A. H. Barr's discussion of deformations of solid objects [ Barr 84 ]. His concept is to deform the space around an object and then render the object as if all its points had moved to new coordinates. He discusses global deformations, for which an explicit non-singular smooth mapping from $\Re^3$ to $\Re^3$ is given and local deformations, in which only the Jacobian derivative of such a mapping is given. In the case of local deformations, an integration process is used in the tangent planes of the object to compute the new coordinates. His examples include twisting, tapering, and cylindrical warps. Not much was done with Barr's general warps until recently due to the lack of an easy interactive way of specifying them. Recently, in 1991, the graphics modeling group at Brown University invented a tool they called "the rack" for specifying such warps. [ John Hughes, 1992, personal communication ].

Sederberg and Perry [ Sederberg 86 ] developed and implemented the Free-Form Deformation (FFD) tool. Sederberg does not claim credit for inventing the idea, but he and Perry were the first to develop it into a full-fledged, useful tool for modeling. The idea is to embed the object to be deformed into a 3-D tensor product Bezier or B-spline volume. The volume is then manipulated by moving its control points, causing the ambient space containing the model to be reshaped. Then the points of the model are rendered as if they had moved to their new coordinates in the reshaped volume, just as in A. H. Barr's approach. What was new is the way in which the global deformation function is specified. Sederberg and Perry went on to show that by placing constraints on the deformation it can be made to have certain regularities, for example, local volume preservation.

It is important to note that the FFD tool does not actually change the geometry of the underlying primitives making up the model. Instead the FFD transformation is considered for rendering purposes to be composed with the original geometry. A point in the surface is found by first locating it in the unwarped object and then applying the warp. Hence, FFD's can be composed. In practice this means that during rendering the transformation

represented by the FFD has to be inverted. Barr pointed out that one could think of the light rays near the object as following curved paths instead of the object being distorted. In other words, one can imagine looking at the model as if through a refractive medium.

Colquillart noted that the class of warps achievable with FFD was restricted due to the parallelepipedical shape of the control point mesh [ Coquillart 90 ], and he developed Extended Free-Form Deformations (EFFD) to alleviate this restriction. The extension was basically to introduce more general shapes for the 3-D mesh of control points. He developed a wedge-shaped mesh and showed how such wedges could be composed to make quite complex shapes. He incorporated EFFD's into the ACTION3D modeling system developed at INRIA-Rocquencourt. In that system, EFFD meshes are a basic modeling tool. Some predefined ones are provided by default, and a user can invent new ones and add them to the palette of tools.

**2.4 Comparison of surface and space warp**

The strength of the space warp technique compared to surface warp is that it automatically treats solid objects as solids rather than a random collection of surfaces. Hence 3-D concepts such as volume can be addressed. Because of the way it is usually applied, it is completely independent of the underlying primitive types used to implement the models, a desirable goal as mentioned earlier. This benefit comes at some cost however, as the warping function must be explicitly stored along with the model. There is also some additional cost for rendering because the warping function has to be inverted. Sederberg doesn't discuss this issue, and Colquillart states that the extra computation is not significant compared to total rendering costs.

The two main drawbacks of space warp compared to surface warp are the necessity of having an invertible mapping, and the general difficulty involved in specifying complex $\Re^3$ to $\Re^3$ mappings. There is no practical way with space warp to tie a knot in a ribbon, for example. There is not even a theoretical way with space warp to form a ribbon into a Möbius strip. Both of these operations are relatively easy to do with surface shaping tools such as my bending operator.

## 2.5 Relation to physically based modeling

Some of the recent variational techniques, such as those of Celniker and Welch and my bending algorithm, have a certain similarity to physically based modeling approaches. In physically based modeling of surfaces, the surface is imagined to be made of some material that obeys certain physical or pseudo-physical laws. For example, [ Weil 86 ] and [ Carignan 92 ] discuss modeling realistic cloth surfaces. Cloth resists stretching but not shearing or bending. Locally it preserves its area element. These properties can be used to create a variational model in which a cloth surface minimizes some energy functional. One can think of surface warp techniques as performing physically based modeling, but with an idealized, imaginary substance making up the surface. The physical laws governing the surface behavior are selected, not for realism, but instead for getting the model to act in ways that make the desired modeling operations work.

# Chapter 3
# Bending

My aim in this chapter is to develop a concept of *bending* that is intuitive but powerful enough to be used for a wide variety of surface shaping tasks. By intuitive I mean that a user can easily learn to predict the approximate result of applying a given bending operator to a surface without the need for extensive training or detailed understanding of the underlying algorithm. However, the bending algorithm must be powerful enough to perform not only simple bending but also twisting, indenting, embossing, and other shape modification. In addition, this concept of bending should be independent of patch type, so it can be implemented using all the common patch types such as Bezier patches, B-spline patches, and NURBS, as well as new patch types that have not been developed yet.

## 3.1 What is bending?

I claim that a good way to think about bending a surface is to consider what happens to the surface normal vectors. Mathematically, the local curvature information of a surface at some point is captured by the derivatives of the normal vector field with respect to movements of the point in the surface. So it is the *variation* of the normal vectors that is of importance in describing the surface. Bending is basically making changes to surface curvature, so we can describe bending by describing how the surface normals should change. More precisely, we describe a particular bending operation by saying how the *variation* of the surface normal vector field should change. A conceptual model of bending might work this way: Imagine that the surface is covered by a forest of little arrows representing the normal vectors, like a porcupine. To bend the surface, we grab certain of these arrows and push and pull them in different directions, causing the underlying surface to change in an intuitively predictable way. The problem is that there

are too many surface normals, so a direct implementation of this model would be too tedious to use. This is essentially the same problem as using control points.

We need a way to specify the pushing and pulling of the normals "all at once," a *bending operator*. A natural way of specifying this bending operator is by means of another surface, which I call the *bending surface*. The normal vector field of the bending surface is used to describe how the normal vector field of the target surface should be changed. In particular, the variations of the normal vector field of the bending surface are added to the variations of the normal vector field of the target surface.

Here is a simple example. The original target surface in Figure 3.1-1 is the upper half of a cylinder with the axis in the X direction. Notice that for movements in the surface in the X direction the surface normal doesn't change, but for movements in the perpendicular direction the normal falls over toward the direction of motion. In other words, the normal curvature in the X direction is zero, but in the perpendicular Y direction it is positive.

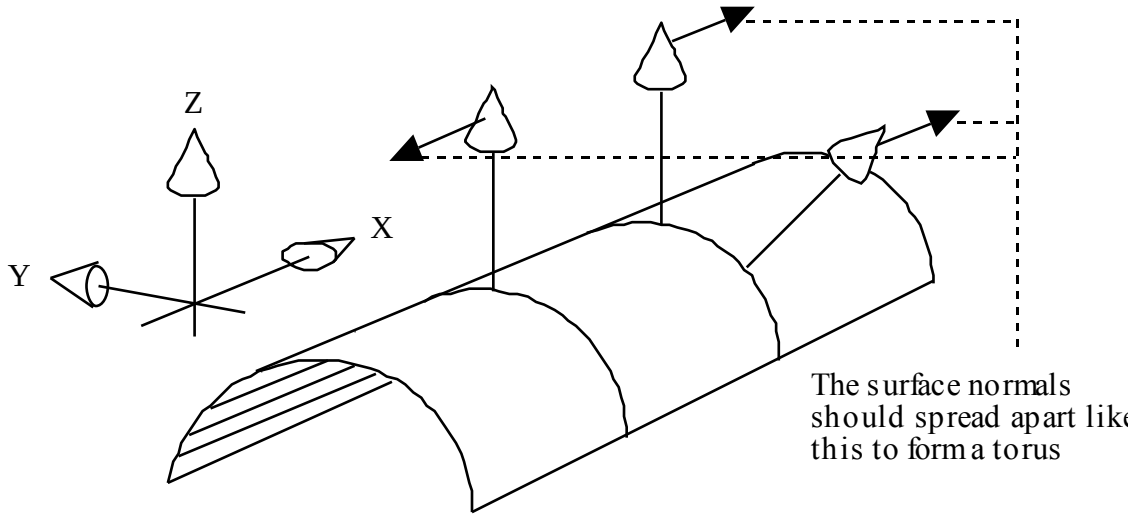



Figure 3.1-1 Original target surface

Consider how to bend this surface into a part of a torus. We need to end up with a positive normal curvature in the X direction. Thus, we use as the bending surface another piece of cylinder, shown in Figure 3.1-2, but with the axis turned 90 degrees so that the curvature, and thus the curvature change to be applied, is in the X direction.

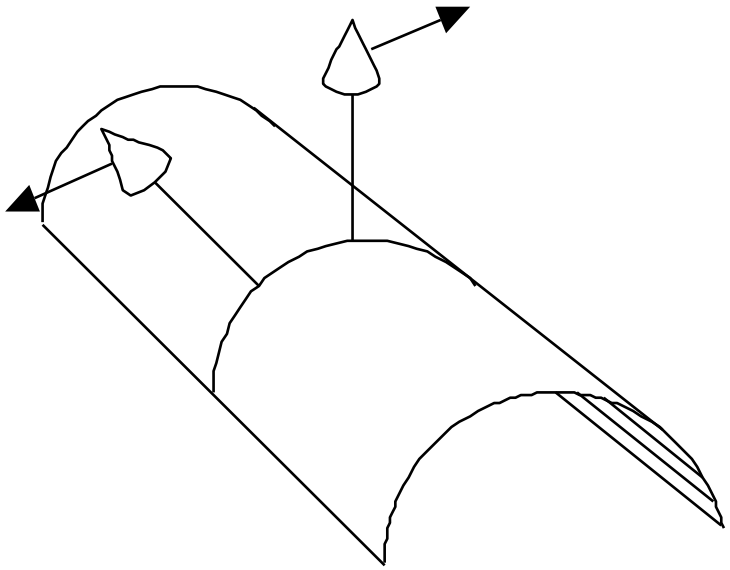


Figure 3.1-2 Bending surface

Both the target surface and the bending surface are defined on the same parameter space, the unit square, with coordinates U and V. The parameterization is such that movement in the U direction in the parameter space translates to movement in the X direction in modeling space at the exact center of parameter space. Notice that for the bending surface, movements in the U direction at the center of parameter space cause the normals to turn in the X direction, but movements in the V direction cause no change in the normals. Hence this bending surface may be interpreted as an operator that "pulls apart" the normals of the surface it is applied to, but only in the U direction in parameter space.

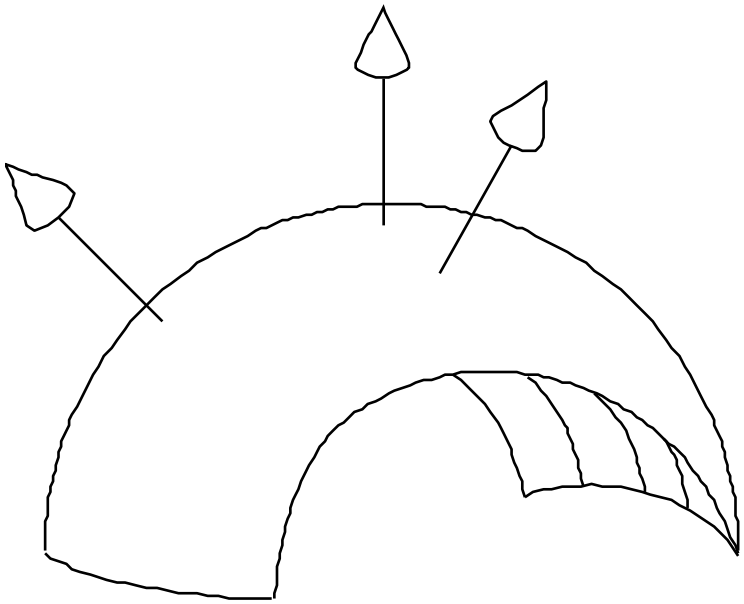


Figure 3.1-3 Final bent surface

Thus the final bent surface, shown in Figure 3.1-3, has the shape of part of a torus. For this surface, movement in the U direction at the center of parameter space causes the normal to turn in the X direction, and movement in the V direction causes the normal to turn in the Y direction. In other words, the normal variations of the final surface show the combined effects of the normal variations of the original and bending surfaces.

Since the bending surface can be of an arbitrary shape and can be oriented in an arbitrary way with respect to the target surface, the method is very powerful. For example, if the bending surface resembles a length of ribbon that is twisted about its long axis, then the target surface will be twisted. If the bending surface resembles a Gaussian height field, then a bump or dent will be placed on the target surface. If the bending surface is a bas-relief pattern, for example, the face of a coin, then that pattern will be embossed on the target surface. In all of these cases, a user of the bending operator can intuitively predict the approximate results of applying a given bending surface to a target surface.

### 3.2 Informal description of the bending algorithm

Intuitively, the bending operator works as follows: We begin with two parametrically defined surfaces, a target surface to be bent and a surface which defines the bending operator. The normal vector fields of both the bending operator surface and the target surface are differentiated to determine how the normal vectors vary with movements in parameter space. These normal vector variations are then added and integrated to determine a new normal vector field. The idea is to produce a new surface which combines properties of both the original and the bending surface. The target surface is smoothly deformed in an attempt to obtain the new desired normal vectors. During the deformation process it is generally necessary to locally deform, i.e., shrink, stretch, and shear the surface. A constraint is applied to minimize the total amount of this distortion. The surface resulting from application of the bending operator can be thought of as the result of an optimization process. There are two competing constraints: the surface normals must be close to what is prescribed by the bending operator, but the original surface must be distorted as little as possible.

### 3.3 Mathematical description of the bending algorithm

The definition of bending has two main parts: determining the desired surface normals after bending, and determining the final bent surface. The following provides motivation for the definitions, presents proofs where necessary that the definitions are consistent, and discusses some alternatives that were rejected.

#### 3.3.1 Defining the desired normals

A *patch* is a mapping of the unit square to 3-D space. In the bending algorithm, a *target* patch $c$ is bent using a *bending* patch $b$. The following definitions set the stage.

| | |
|---|---|
| $D = [0,1]^2$ | domain |
| $c: D \to \Re^3$ | target patch |
| $b: D \to \Re^3$ | bending patch. |

Note that the corresponding points of *b* and *c* are identified via a source point in *D*, i.e., via the parameterization.

It is convenient to work with complete frame fields rather than just surface normals. Let $e = (e_1, e_2, e_3)$ be an adapted frame field for the target patch *c* with $e_3$ being the normal vector, and $f = (f_1, f_2, f_3)$ be an adapted frame field for the bending patch *b* with $f_3$ being the normal vector.

Define the 1-forms, called the *structure* forms:

$$\omega_j^i = de_i \cdot e_j$$
$$\nu_j^i = df_i \cdot f_j$$
for $i = 1,2,3$ and $j = 1,2,3$.

We are interested in the pullbacks of these forms to the tangent space of *D*, but using the standard abuse of notation, I will use the same symbols for these.

In matrix form the notation becomes more compact:

$$de = \omega e$$
$$df = \nu f .$$

The $3\times3$ matrices of forms $\omega$ and $\nu$ capture the way the frame vectors turn with movements in the parameter space.

To define the bending action, we want to add the matrices $\omega$ and $\nu$, but to do this we must represent them in a common coordinate system. This is accomplished by "rotating" the matrix $\nu$, the details of which are presented shortly.

Why addition instead of some other possible way of combining $\omega$ and the rotated $\nu$? Suppose we have a part of a cylinder, and we want to bend it using another part of a

cylinder with the same axis and radius. What should happen? If the normals of one cylinder turn at some rate and a bending operator causes them to be pulled apart at the same rate, the result should have normals turning at double the original rate. If we choose an adapted frame field such that $e_1$ points along the cylinder axis, the rate of turn of the normal $e_3$ of the target cylinder in the $e_2$ direction is $\omega_{23}$. The rate that the bending operator causes the normals to be pulled a part is $\nu_{23}$ which equals $\omega_{23}$. But that means the 23 coefficient of the resulting form matrix should be $2\omega_{23}$, which is exactly the effect of addition.

Now consider the problem of rotating the structure form matrix $\nu$ into the frame of the target surface. Let $g = (g_1, g_2, g_3)$ be the frame field of the final bent surface. Define a rotation matrix $r$ by

$$f = rg.$$

Then the structure form matrix $r^t \nu r$ is $\nu$ expressed in terms of the $g$ frame. This is shown by the following calculation. Let

$$\sum a^i f_i = \sum b^i g_i$$

be a vector field on $D$. Then we need to show

$$\nu\left(\sum a^i f_i\right) = \left(r^t \nu r\right)\left(\sum b^i g_i\right).$$

In matrix form with $a = (a^1, a^2, a^3)$ and $b = (b^1, b^2, b^3)$, we have

$$af = bg,$$

and the relation that needs to be proven is

$$\nu(af) = \left(r^t \nu r\right)(bg).$$

Using linearity and the fact that $ar = b$, the following calculation proves this relation.

$$\nu(af) = a\nu(f) = a\nu(rg) = a(rr^t)\nu(rg) = br^t\nu(rg) = b(r^t\nu r)(g) = (r^t\nu r)(bg).$$

We are now ready to start defining precisely what bending means. The first step is to define a frame field $g = (g_1, g_2, g_3)$ that will specify the desired normals of the bent surface. The key idea is to transform the structure forms for the bending surface into the coordinate system of the bent surface and to add them to the structure forms of the original target surface.

Definition 3.3.1-1. Given a target patch *c* with adapted frame field *e* and structure form matrix $\omega$, and a bending surface *b* with adapted frame field *f* and structure form matrix $\nu$, and given a point of application *p* in the domain *D*, such that *b* is tangent to *c* at *p*, define the *desired bending frame field g* as the unique solution to the differential system:

$$\begin{aligned} &g(p) = e(p), \\ &dg = (\omega + r^t\nu r)g \text{ on } D, \end{aligned} \qquad (3.3.1\text{-}1)$$

where

$$f = rg \text{ on } D.$$

A proof is needed that this definition makes sense, i.e., that a solution exists and is unique. I use the Frobenius Theorem [ Warner 71 ], which can be stated as follows:

Theorem (Frobenius). Let $\phi^1, \ldots, \phi^q$ be one-forms in $\Re^n$, $n = q + s$, linearly independent at 0. Suppose the $\phi^i$ form a completely integrable system, i.e., there exist one-forms $\theta^i_j$ satisfying

$$d\phi^i = \sum_{j=1}^{q} \theta^i_j \wedge \phi^j \qquad (i = 1, \ldots, q).$$

Then there are functions $h^i_j$ and $k^j$ in a neighborhood of 0 satisfying

$$\phi^i = \sum_{j=1}^{q} h^i_j dk^j \qquad (i = 1, \ldots, q).$$

Furthermore, if $\phi^1,\ldots,\phi^q$ are linearly independent on a connected open subset $U \subseteq \Re^n$ containing 0, then the functions $h_j^i$ and $k^j$ have unique extensions to all of *U*.

The plan is to produce a completely integrable system in a suitable high dimensional space such that the frame field *g* is an integral manifold (hypersurface) of this system. Begin by putting the equation $dg = (\omega + r^t \nu r)g$ into a more convenient form. Eliminate *r* yielding

$$dg = (\omega + gf^t \nu f g^t)g = \omega g + gf^t \nu f = \omega g + g\mu \text{ where } \mu = f^t \nu f .$$

The form matrices $\omega$ and $\mu$ are independent of *g* and satisfy

$$d\omega = \omega \wedge \omega$$
$$d\mu = -\mu \wedge \mu .$$

The first is a standard property of structure forms, and the second is shown as follows, using the relations $d\nu = \nu \wedge \nu$, $\nu^t = -\nu$, and $f^t f$ is the identity matrix:

$$d\mu = d(f^t \nu f) = df^t \wedge \nu f + f^t d\nu f - f^t \nu \wedge df$$
$$= f^t \nu^t \wedge \nu f + f^t \nu \wedge \nu f - f^t \nu \wedge \nu f$$
$$= f^t \nu^t \wedge \nu f = -f^t \nu \wedge \nu f = (f^t \nu f) \wedge (f^t \nu f) = -\mu \wedge \mu .$$

Now move to the 11-dimensional space $D \times (\Re^3)^3$ with coordinates $(u, v, z_j^i, 1 \le i, j \le 3)$. Define a 3×3 matrix of one-forms on this space:

$$\gamma = dz - \omega z - z\mu .$$

I claim that $\gamma = 0$ has a unique solution and that furthermore, in the solution space, *z* is a frame field. The following calculation shows that $\gamma$ is a completely integrable system:

$$
\begin{aligned}
d\gamma &= -d\omega z + \omega \wedge dz - dz \wedge \mu - z d\mu \\
&= -\omega \wedge \omega z + \omega \wedge dz - dz \wedge \mu + z\mu \wedge \mu \\
&= \omega \wedge (-\omega z - z\mu + dz) - (dz - z\mu - \omega z) \wedge \mu \\
&= \omega \wedge \gamma - \gamma \wedge \mu
\end{aligned}
$$

Thus $\gamma$ satisfies the hypothesis of the Frobenius Theorem, so it follows that there exist functions $s_{ij}^{ki}$, $t_j^i$ such that

$$
\gamma_j^i = \sum_{k,l=1}^{3} s_{ij}^{ki} dt_k^l \qquad\qquad (i,j = 1,2,3).
$$

Abbreviate this relation as $\gamma = s dt$. Note that $\gamma$ is non-zero, so the equations $\gamma = 0$ and $dt = 0$ have the same solutions, namely the hypersurfaces $t = constant$ . Now I claim that there exists a $3 \times 3$ matrix of functions $g$ of $(u,v)$, with the prescribed initial values $g(p) = e(p)$, so that $g = z$ is an integral manifold of $\gamma = 0$, that is,

$$
dg = \omega g + g\mu .
$$

To see this, simply pick a solution hypersurface that passes through $e(p)$.

The Frobenius theorem implies that $g$ is unique and that it exists on all of $D$. What remains is to show that $g$ is a frame field. To show that $g$ is a frame field, it suffices to show that $dgg^t$ is skew-symmetric [ Flanders 89 ], p. 103. By definition,

$$
dgg^t = (\omega g + g\mu)g^t = \omega + g\mu g^t .
$$

$\omega$ is skew-symmetric by hypothesis, and the calculation
$(g\mu g^t)^t = g\mu^t g^t = g(f^t vf)^t g^t = -gf^t vfg^t = -g\mu g^t$ shows that $g\mu g^t$ is, also.

Now that we have a desired normal vector field, the next step is to produce the bent surface. Ideally, the bent surface would have as its normal vector field the desired normal vector field, but that turns out in general to be impossible. It would be nice if we could

include the position vector $x: D \to R^3$ in the differential system used to get the normal vector field by adding the equations

$$x(p) = c(p)$$
$$dx = (\omega + r^t \nu r)x,$$

and integrate this system to solve for the final bent surface in one step. This attempt fails because *dx* doesn't necessarily satisfy the integrability conditions needed to apply the Frobenius Theorem.

Even in principle, it may be impossible to construct a surface with a given normal vector field. [ Spivak 70 ] gives a simple example. There are also practical problems. First, a real modeling system will use a particular patch type, usually with only a finite number of degrees of freedom (control points), so some normal vector fields will be unattainable. Second, if we try too hard to match the desired normals, we may destroy any resemblance between the target surface and the final bent surface. Therefore, my current approach is to deform the original surface in such a way that it only approximately inherits the desired normal vector field.

### 3.3.2 Producing the final bent surface

Producing the final bent surface can be formulated as an optimization problem. I define a penalty function that measures to what extent the final surface fails to have the desired normals and to what extent the surface fails to resemble the original surface. Then the optimization problem is to find some surface that minimizes this penalty function. In practice, a patch is defined by a finite number of control points, and the optimization problem is to find a configuration of these control points that minimizes the penalty function.

How can this somewhat vaguely stated goal for the penalty function be made mathematically precise? In the differential geometry of surfaces, there is an important distinction between *intrinsic* and *extrinsic* properties. Appendix A contains a precise

definition of these terms and some examples to help the reader develop an intuitive understanding of this distinction. Roughly speaking, intrinsic properties of a surface are those properties that do not depend on how the surface happens to be embedded in 3-D space. Such properties depend only on how the surface is locally stretched and sheared, which is captured by the *metric* of the surface. Extrinsic properties of a surface are those properties that are not intrinsic, that is, they depend not just on the metric but also the position of the surface in space. It is obvious that the normal vector field of a surface is an extrinsic property, since it can be changed by a rigid rotation of the whole surface. I make use of this distinction to develop a rationale for the penalty functions. Suppose the normal vector field of a surface is fixed, i.e., the normal vectors at each point are constrained against changing their direction. What degrees of freedom are left? Consider a specific point $p$ on the surface with normal vector $v$. With the direction of $v$ fixed, all that can be done is to move the point $p$. Locally the motion of point $p$ has to stay in the plane through $p$ perpendicular to $v$, or the surface will bend, changing the direction of nearby normal vectors. But sliding a point around in the tangent plane amounts to stretching and shearing the surface, thus changing the metric. The point of this admittedly hand-waving argument is that once the normal vectors of a surface are determined, the only degree of freedom left in the surface is the metric. (It is possible to make this argument mathematically sound; see Theorem A-2 in appendix A.) Thus, the only sense in which the bent surface can resemble the original surface is to have a similar metric. It seems logical based on this argument that the penalty function should have two main components, one that measures disturbances to the metric of the original target surface and one that measures the deviation between the actual and the desired normal vector field.

Criteria for the penalty function definitions

I define the total penalty function as

$$\varepsilon = k_m \varepsilon_m + k_n \varepsilon_n,\ k_m + k_n = 1,$$

where $\varepsilon_m$ is the metric penalty function and $\varepsilon_n$ is the normal penalty function. The ratio of $k_m$ and $k_n$ determines the relative importance of metric deviations and normal deviations in the minimization. The penalty functions are functions of the initial and final target surfaces as well as the desired normal vector field. In application, the initial target surface and desired normals are fixed, so we can consider $\varepsilon$ to be a function of only the final bent surface. For patches defined by a finite number of control points, we can consider $\varepsilon$ to be a function of the control points. The definitions of the penalty functions $\varepsilon_m$ and $\varepsilon_n$ are motivated by several criteria:

1) They should separate as much as possible intrinsic and extrinsic properties.

2) They should be independent of the coordinate system, that is, invariant under translations. They should be independent of rotations in the following sense: if the desired normals and the final surface are subjected to the same rotation, then the penalty functions do not change.

3) They should be as independent as possible of surface parameterization, given that the correspondence of the normals between the bending surface and the target surface is via parameterization.

4) $\varepsilon_n$ should be zero if and only if the surface has exactly the desired normals, and should increase as the actual and desired normals deviate more.

5) $\varepsilon_m$ should be zero if and only if the surface is not locally stretched or sheared compared to the original target surface (that is, the two surfaces are isometric), and should increase as such stretch or shear increases. Furthermore $\varepsilon_m$ should be isometrically invariant, meaning that if the final surface is replaced by an isometric surface, $\varepsilon_m$ remains the same. Isometric invariance implies that $\varepsilon_m$ is independent of both parameterization and rigid motions of the final target surface.

6) They should be feasible to compute.

The intrinsic properties of a surface $c: D \to \Re^3$ are captured by the first fundamental form *I*, the matrix of which is $\begin{bmatrix} E & F \\ F & G \end{bmatrix}$ with respect to *du* and *dv*, where

$$E = c_u \cdot c_u$$
$$F = c_u \cdot c_v$$
$$G = c_v \cdot c_v .$$

I consider a time-varying surface

$$c: D \times [0,1] \to \Re^3,$$

where the final argument is time and time 0 marks the beginning of an optimization process and time 1 marks the end. At time 0, *c* is the original target surface, and at time 1, *c* is the final bent surface.

In the following definitions, subscript 0 refers to the values of variables at time 0, e.g., $c_0$ denotes the surface at time 0. With justifications that appear in the following pages, I define the metric penalty function as

$$\varepsilon_m = \min_f \int_D \left\| I_0^{-1} \cdot \hat{I} - \mathbf{1} \right\|^2 \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv \,, \qquad (3.3.2\text{-}1)$$

where $f$ is an isometry, $\hat{I}$ is the first fundamental form matrix of $f \circ c$, $I_0$ is the first fundamental form matrix of $c_0$, and $\mathbf{1}$ is the identity matrix. A discussion of the properties of the first fundamental form, also called the metric, can be found in Appendix A. The term $I_0^{-1} \cdot \hat{I} - \mathbf{1}$ is a $2 \times 2$ matrix that measures the difference of the metric of $f \circ c$ and the metric of $c_0$. I use $I_0^{-1} \cdot \hat{I} - \mathbf{1}$ instead of $\hat{I} - I_0$ to make $\varepsilon_m$ independent of scale. The term $\sqrt{E_0 G_0 - F_0^2}\, du \wedge dv$ is the area element for the surface $c_0$. Thus $\varepsilon_m$ is the minimum of the integral of the squared difference of the metrics over all possible isometries of the surface.

I define the normal penalty function as

$$\varepsilon_n = \int_D \left\| \frac{c_u \times c_v}{\|c_u \times c_v\|} - n_{des} \right\|^2 \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv. \qquad (3.3.2\text{-}2)$$

The term $\frac{c_u \times c_v}{\|c_u \times c_v\|} - n_{des}$ measures the difference of the true surface normal $n = \frac{c_u \times c_v}{\|c_u \times c_v\|}$ and the desired surface normal $n_{des}$. Thus $\varepsilon_n$ is the integral of the squared difference of the actual and desired normals over the surface.

Remark 1. Computing $\varepsilon_m$ requires minimizing over the function space of all isometries of the final surface and hence is difficult to implement in software. I address this problem in the prototype software by replacing $\varepsilon_m$ with an approximation, discussed later in this section.

Remark 2. The term $\sqrt{E_0 G_0 - F_0^2}\, du \wedge dv$ is the area element of the initial target surface $c_0$, pulled back to the domain *D*. There are two surfaces involved, *c* and $c_0$, and it is not clear *a priori* which one to integrate over. I chose to integrate over the initial surface since otherwise I would have to deal with the area element changing during the optimization process. If the initial and final surfaces are not too different according to $\varepsilon_m$, then neither are the area elements, since the area element is an isometric invariant.

Remark 3. In the definition of $\varepsilon_m$, I am trying to compare two matrices, so I need to pick a matrix norm.

$$\|A\|^2 = \left\| \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix} \right\|^2 = a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2$$

was chosen because it is easy to compute. A more logical alternative would be the matrix $\|\ \|_2$ norm induced by the vector $\|\ \|_2$ norm, but this is harder to compute since it involves

finding eigenvalues. These norms differ little however, because $\|A\| = s\|A\|_2$ where $1 \le s \le \sqrt{2}$. The following calculation shows this:

$$\text{trace}\left(A^t A\right) = \text{trace}\begin{bmatrix} a_{11}^2 + a_{12}^2 & -- \\ -- & a_{21}^2 + a_{22}^2 \end{bmatrix} = a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2 = \|A\|^2 .$$

Since $\text{trace}\left(A^t A\right)$ is the sum of the eigenvalues of $A^t A$ and $\|A\|_2^2$ is the largest eigenvalue of $A^t A$, we have $\|A\|_2^2 \le \|A\|^2 \le 2\|A\|_2^2$.

Remark 4. These definitions fully meet criteria (1)-(5). There is no getting around a certain dependence on parameterization since the correspondence of the bending and target surfaces is via parameterization.

The final bent surface is one that minimizes $\varepsilon$. It is not clear on theoretical grounds how to choose the arbitrary constants $k_n$ and $k_m$, but experience with the prototype software shows that $k_n / k_m \approx 8$ is usually a good choice. Much smaller values of this ratio tend to make the surface seem too stiff, and much larger values tend to cause the surface to expand or contract by large amounts.

From a theoretical point of view, there are a number of problems with this definition, aside from the fact that $\varepsilon_m$ is difficult to compute. First, the minimum might not be unique, or even if it is, there may be multiple local minima that are numerically indistinguishable. Second, it may be the case that radically different surface shapes have roughly the same minima. Third, the total penalty function may have stationary points, i.e., configurations of control points for which the gradient of the penalty function is zero but for which the total penalty function does not have a local minimum.

From a practical point of view these difficulties can usually be circumvented by performing the bending process incrementally. That is, the original target surface is used as a starting estimate, and the algorithm operates by altering the original normals towards the desired normals in a series of small steps and performing the minimization at each step. The advantage of using an incremental approach is that the target surface tends to

deform in a continuous manner. This behavior makes the bending operator more intuitive because it fits the user's mental model of deforming a physical piece of material. The minimization algorithm used, the BFGS algorithm (discussed in Section 3.6.4), finds a nearby local minimum of the penalty function when given a starting estimate. This local minimum will not necessarily be a global minimum. Finding a nearby local minimum rather than a global minimum is not a problem. On the contrary, it is an advantage because it tends to make the bending process continuous.

The bending algorithm can fail if the penalty function has stationary points. My experimentation with the prototype software indicates that such failures are very rare in practice. I have seen this kind of failure occasionally when using the twist operator (see Section 5.2.3 for a precise definition). The symptoms of such a failure are that the shape of final bent surface is very sensitive to the setting of the error tolerance, and that subsequent applications of bending operators cause unexpectly large changes to the shape of the surface. However, even in these failure cases, the final bent surface normal vectors are still rougly equal to the desired normals. My speculation is that the twist operator sometimes leads to an underconstrained optimization problem.

Finding a computable approximation for the metric penalty function. This is at present an open problem. The approximation I use in the prototype software is

$$\varepsilon_m = \int_D \left\| I_0^{-1} \cdot I - \mathbf{1} \right\| \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv,$$

i.e., I am using the identity map instead of minimizing over the space of isometries of the surface. In practice this works surprisingly well, considering how poor the approximation is on theoretical grounds. I speculate that this may be due to using B-spline patches with uniformly spaced knots. I think that there isn't enough freedom in the definition of such patches to permit two surfaces that are similar except for parameterization. This suggests an experiment with B-spline patches with variable knots.

Collapse resistance. In early experiments with the software prototype, occasionally creases and folds would form in the surface during the bending process. This typically occurred when there were extremely large curvatures in the bending operator. Mathematically, this behavior is undesirable because the normal vectors become undefined. To circumvent this problem, I added third component to the total penalty function. The collapse resistance penalty function is

$$\varepsilon_c = \int_D \frac{1}{\det(I)^2} \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv. \qquad (3.3.2\text{-}3)$$

The term $\det(I)^2 = EG - F^2$ measures the square of the local area magnification factor for the deformed surface. When it approaches zero, a collapse of the surface is occurring. In this case the integral will increase, causing the collapse to be penalized. The coefficient for the collapse function in the total penalty function is set to a very small value. Thus, the collapse penalty has little noticeable effect on the bending action except that creases and folds no longer occur.

Range of action. The bending action can be restricted to a neighborhood of the point of application, which I call the *range of action*, by introducing a non-negative weight function *w* into the integral that defines $\varepsilon_n$ and introducing another penalty function $\varepsilon_f$. The purpose of this new penalty function is to prevent the shape of the surface outside the range of action from changing (the subscript *f* is for "fix.") This penalty function is necessary because without it, the surface would be underconstrained in regions with *w* small, causing "ripples" to spread out from the range of action area. I define the function as

$$\varepsilon_f = \int_D \left\| I_0^{-1} \cdot (II - II_0) \right\|^2 (1 - w) \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv\,, \qquad (3.3.2\text{-}4)$$

where *II* denotes the second fundamental form of *c*. Roughly speaking, *II* measures the curvature of the surface. (Note: to be precise, the right measure of curvature is the function $I^{-1} \cdot II$, which is called the *curvature tensor* or *Weingarten map*. The

eigenvalues of $I^{-1} \cdot II$ are the *principal curvatures*, and the eigenvectors are the *principal directions*. A discussion of these properties can be found in Appendix A.) The matrix of $II$ is $\begin{bmatrix} L & M \\ M & N \end{bmatrix}$ with respect to *du* and *dv*, where

$$L = c_{uu} \cdot n$$
$$M = c_{uv} \cdot n$$
$$N = c_{vv} \cdot n .$$

The term $I_0^{-1} \cdot (II - II_0)$ captures the deviation of the second fundamental form between the initial and the bent surface. I use $I_0^{-1} \cdot (II - II_0)$ instead of $II - II_0$ to get a more accurate measure of the curvature deviation and to achieve scale independence. The weight function is not introduced into $\varepsilon_m$, since there is no harm in preserving the metric in "fixed" parts of the surface. Indeed, such preservation is needed to make the range of action work properly, since the shape of a surface depends both on its curvature and its metric. This dependence is discussed and made explicit in Theorem A-2 in Appendix A.

As it is stated, this definition of the penalty function $\varepsilon_f$ suffers from a dependence on the parameterization of the surface. To be strictly accurate, the definition should be

$$\varepsilon_f = \min_{\phi} \int_D \left\| I_0^{-1} \cdot (II \circ \phi - II_0) \right\|^2 (1 - w \circ \phi) \sqrt{E_0 G_0 - F_0^2}\, du \wedge dv,$$

where $\phi$ is a reparameterization of *D* and $II \circ \phi$ is the second fundamental form of $c \circ \phi$. As with $\varepsilon_m$, I am using an approximation to make the computation feasible.

Experience with the software prototype reveals that the range of action operator does not perfectly preserve the shape of a patch in regions where the weight function is zero. This is to be expected, since the various penalty functions are competing with one another in the optimization process. However, it appears that the normal penalty function $\varepsilon_n$ exercises better control over surface shape than does the fix penalty function $\varepsilon_f$. This effect can be somewhat compensated for by adjusting the penalty function weights but

cannot be entirely eliminated. I believe that the effect is due to $\varepsilon_f$ being a more indirect type of control than $\varepsilon_n$. By this I mean that $\varepsilon_f$ uses first and second derivatives of the patch function while $\varepsilon_n$ uses only first derivatives. This leads to the question: Why not use the original unaltered normals of the surface to define $\varepsilon_f$ rather than the second fundamental form? The answer has to do with the rather delicate nature of the differential equation used to compute the desired normals. We cannot use the original normals directly, because $\varepsilon_f$ has to be invariant under rigid motions. For example, consider bending a rod in the middle – although the ends are rigid, the normals near the ends have to rotate. Thus, the differential equation that produces the desired normals would have to be altered so that it produces a rotated form of the original normals where the range of action weight function is small. I have discovered that doing just about anything to the differential equation breaks the integrability conditions and destroys the uniqueness of the solution. I have not been able to find a way to alter the equations to get the desired effect while maintaining the integrability conditions.

### 3.4 Positioning the bending operator

This section discusses the mathematical details of positioning the bending operator patch with respect to the target patch. My goal is to permit the use of an arbitrary patch as the bending operator, so I needed to develop a general purpose method for positioning. The user's conceptual model for positioning the bending operator works as follows:

1. The user selects a point of application on the target patch and places the bending operator patch such that it is tangent to the target patch at that point.

2. The user rotates the bending operator patch about the normal vector at the point of application to achieve the desired orientation.

3. The user scales the bending operator patch along two axes to the desired size.

It may appear from this description that positioning the bending operator simply involves applying Euclidean motions and scaling to the bending operator patch, but this is not the

case. The correspondence between the target patch and the bending operator patch is via the patch parameterizations, so positioning has to include a reparameterization of the bending operator. But reparameterization alone is not sufficient, because it doesn't provide any way to handle the rotation in step 2. To see this, consider a cylindrical patch used as a bending operator. Reparameterization doesn't change the geometry of the surface defined by the patch function, so no reparameterization can change the direction of the cylinder axis for this bending operator. In the remainder of this section I develop the mathematics needed to implement the positioning control.

The bending operator placement control is designed to make it easy for the user to place the bending operator interactively. Figure 3.4-1 shows a schematic diagram of this tool.

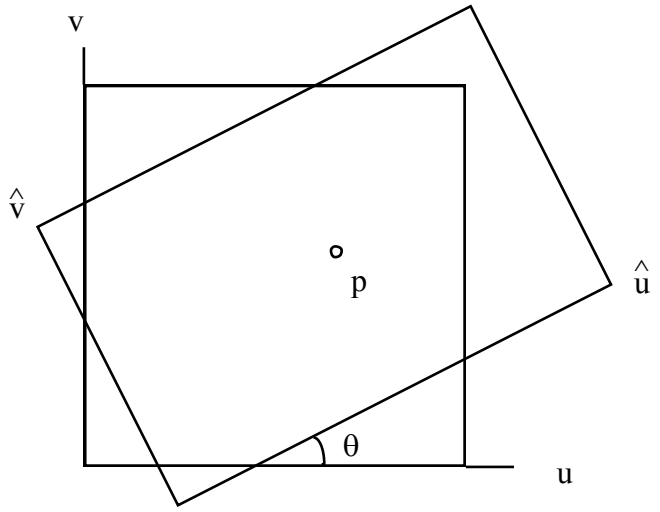



Figure 3.4-1 Positioning tool

The square with coordinates *u* and *v* represents the unit square in the parameter space of the target patch. The oblique rectangle with coordinates $\hat{u}$ and $\hat{v}$, called the control rectangle, represents the unit square in the parameter space of the bending operator. The point of application in parameter space is *p*, which has coordinates $(\hat{u},\hat{v}) = (0.5, 0.5)$ in the parameter space of the bending operator. The user selects the $(u,v)$ coordinates of *p* by moving the control rectangle. Since the control works in parameter space, the user doesn't have to explicitly manipulate the bending operator patch to make it tangent to the

target patch – this is handled automatically by the software. The user orients the bending operator by rotating and scaling the control rectangle. The final configuration of the control rectangle determines an affine transformation $(\hat{u},\hat{v}) = f(u,v)$ between the two parameter spaces by

$$\hat{u} = a_{11}u + a_{12}v + a_{13}$$
$$\hat{v} = a_{21}u + a_{22}v + a_{23}.$$

Because the control shape is a rectangle, the linear part of this transformation can be written as the product of a scaling matrix and a rotation matrix as

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix},$$

where $\theta$ is the angle between the $u$ coordinate axis and the $\hat{u}$ coordinate axis.

I am now ready to define mathematically how the positioning control works. Let $b: D \subseteq \Re^2 \to \Re^3$ be the bending operator patch, where $[0,1]^2 \subseteq D$. Note that the bending operator patch has to be defined on a subset of $\Re^2$ such that $f^{-1}(D)$ covers $[0,1]^2$, where $f$ is the affine transformation. Let $q \to r(\theta,q)$ denote a rotation of $\Re^3$ by angle $\theta$ about the normal vector of the bending operator patch at the point $b(0.5,0.5)$. The positioned bending operator $\bar{b}$ is defined by

$$\bar{b}(u,v) = r(\theta, b(\hat{u},\hat{v})).$$

This formula says that the modified bending operator patch $\bar{b}$ is obtained by a reparameterization of its domain followed by a rotation of its range. This result is not obvious and it needs to be justified. My justification has two parts. I prove for simple bending operator, namely a quadratic patch, that it works properly, and I appeal to the fact that the prototype software works as expected for more complex bending operators.

The proof for quadratic patches is as follows. To simplify the calculations, I ignore the translation and assume that the point of application is at $(0,0)$ in both the $(u,v)$ and $(\hat{u},\hat{v})$ coordinate systems. Let the (unmodified) bending operator be defined by the function $b(u,v)=(u,v,u^2+v^2)$. Suppose that the positioning operation specified by the user is

$$\hat{u} = a_{11}u + a_{12}v$$
$$\hat{v} = a_{21}u + a_{22}v,$$

and that

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

This positioning operation represents stretching the control rectangle along the $u$ direction by a factor of $1/s_1$, stretching along the $v$ axis by a factor of $1/s_2$, and then rotating the control rectangle clockwise by an angle of $\theta$. Note that stretching the control rectangle by a factor of two, say, in a given direction means that the normal vectors should turn half as fast for movements in that direction. The question is, what should the modified bending operator $\bar{b}(u,v)$ do? Intuitively, it should have the same bending effect as some quadratic patch $q(u,v)$. Having the same bending effect amounts to having the same normal vector field, since only the normal vector field of the bending operator is used in the bending algorithm. Hence, we need to determine what quadratic patch $q(u,v)$ has the normal vector field desired by the user, and show that the modified bending operator $\bar{b}(u,v)$ has the same normal vector field as $q(u,v)$. I claim that $q(u,v)$ should be the function

$$q(u,v) = (u,v,c_1u^2 + 2c_2uv + c_3v^2) = \left(u,v,\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} c_1 & c_2 \\ c_2 & c_3 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}\right),$$

where the matrix of $c_i$ coefficients has the factorization

$$\begin{bmatrix} c_1 & c_2 \\ c_2 & c_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = RSR^t.$$

It is a standard result in linear algebra [ Strang 80 ] that any symmetric matrix can be factored this way and that the columns of $R$ are the eigenvectors and the diagonal elements of $S$ are the eigenvalues. To see why this is the correct form for $q$, let us investigate the behavior of the normal vector field of $q$ near the origin. We have

$$q_u(u,v) = (1,0,2c_1u + 2c_2v)$$
$$q_v(u,v) = (0,1,2c_2u + 2c_3v),$$

so

$$q_u \times q_v = (-2c_1u - 2c_2v, -2c_2u - 2c_3v, 1).$$

This formula is easier to work with using vector notation:

$$q_u \times q_v(u,v) = -2\begin{bmatrix} c_1 & c_2 & 0 \\ c_2 & c_3 & 0 \\ 0 & 0 & -1/2 \end{bmatrix}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = -2\begin{bmatrix} RSR^t & 0 \\ 0 & -1/2 \end{bmatrix}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

Near the origin, $\|q_u \times q_v\| \approx 1$, which implies $n \approx q_u \times q_v$, where $n$ is the normal vector field. Taking partial derivatives, we have

$$(q_u \times q_v)_u = (-2c_1, -2c_2, 0) = -2\begin{bmatrix} c_1 & c_2 & 0 \\ c_2 & c_3 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = -2\begin{bmatrix} RSR^t & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ and}$$
$$(q_u \times q_v)_v = (-2c_2, -2c_3, 0) = -2\begin{bmatrix} c_1 & c_2 & 0 \\ c_2 & c_3 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = -2\begin{bmatrix} RSR^t & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

Now we examine the directional derivatives of $q_u \times q_v$ in the eigendirections $\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}$ and $\begin{bmatrix}-\sin\theta \\ \cos\theta\end{bmatrix}$. We have

$$\nabla_{\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}} q_u \times q_v = (q_u \times q_v)_u \cos\theta + (q_u \times q_v)_v \sin\theta = -2\begin{bmatrix} RSR' & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}\cos\theta \\ \sin\theta \\ 0\end{bmatrix}$$

$$= -2\begin{bmatrix} RS & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}1 \\ 0 \\ 0\end{bmatrix} = -2s_1\begin{bmatrix} R & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}1 \\ 0 \\ 0\end{bmatrix} = -2s_1\begin{bmatrix}\cos\theta \\ \sin\theta \\ 0\end{bmatrix}.$$

$$\nabla_{\begin{bmatrix}-\sin\theta \\ \cos\theta\end{bmatrix}} q_u \times q_v = -(q_u \times q_v)_u \sin\theta + (q_u \times q_v)_v \cos\theta = -2\begin{bmatrix} RSR' & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}-\sin\theta \\ \cos\theta \\ 0\end{bmatrix}$$

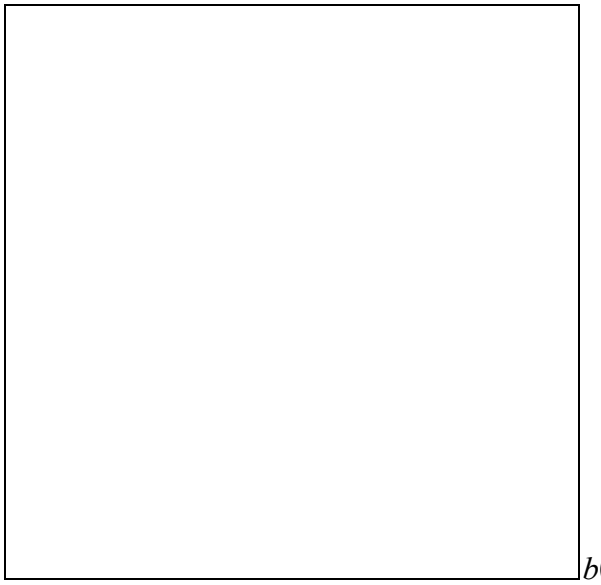
$$= -2\begin{bmatrix} RS & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}0 \\ 1 \\ 0\end{bmatrix} = -2s_2\begin{bmatrix} R & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}0 \\ 1 \\ 0\end{bmatrix} = -2s_2\begin{bmatrix}-\sin\theta \\ \cos\theta \\ 0\end{bmatrix}.$$

These equations say that small movements in the eigendirections near the origin in $(u,v)$ space cause the normal vectors to turn in the analogous directions in the range space at a rate proportional to the eigenvalues $s_1$ and $s_2$. But this is exactly the behavior that we desire from the positioning control, so $q(u,v)$ must be the quadratic patch we are looking for.

The final step is to show that $q(u,v)$ and $\bar{b}(u,v)$ have the *same* normal vector fields (note that they are *not* the same function). This is done by computing $\bar{b}_u \times \bar{b}_v$ and showing that it is a positive multiple of $q_u \times q_v$. Using the definition of $\bar{b}(u,v)$, we obtain the modified bending operator

$$\bar{b}(u,v) = r\big(\theta, b(\hat{u},\hat{v})\big) = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} b(\hat{u},\hat{v}) = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{u} \\ \hat{v} \\ \hat{u}^2 + \hat{v}^2 \end{bmatrix}.$$

We have



$$b(\hat{u},\hat{v}) = \big(\hat{u},\hat{v},\hat{u}^2 + \hat{v}^2\big)$$

$$b_u(\hat{u},\hat{v}) = \big(a_{11}, a_{12}, 2a_{11}\hat{u} + 2a_{21}\hat{v}\big)$$

$$b_v(\hat{u},\hat{v}) = \big(a_{21}, a_{22}, 2a_{12}\hat{u} + 2a_{22}\hat{v}\big)$$

$$b_u(\hat{u},\hat{v}) \times b_v(\hat{u},\hat{v}) = \big(a_{11}a_{22} - a_{12}a_{21}\big)\big(-2\hat{u}, -2\hat{v}, 1\big).$$

Since we are interested only in the normal vector, we can drop the positive multiplier $a_{11}a_{22} - a_{12}a_{21}$. Using vector notation,

$$b_u(\hat{u},\hat{v}) \times b_v(\hat{u},\hat{v}) \propto -2 \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & -1/2 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = -2 \begin{bmatrix} SR^t & 0 \\ 0 & -1/2 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix},$$

so

$$\bar{b}_u(u,v)\times\bar{b}(u,v)=\begin{bmatrix}R & 0\\ 0 & 1\end{bmatrix}b_u(\hat{u},\hat{v})\times b_v(\hat{u},\hat{v})\propto -2\begin{bmatrix}RSR^t & 0\\ 0 & -1/2\end{bmatrix}\begin{bmatrix}u\\ v\\ 1\end{bmatrix}=q_u\times q_v.$$

Therefore, $\bar{b}(u,v)$ and $q(u,v)$ have the same normal vector fields.

It seems difficult to generalize this proof because it depends on being able to produce a function analogous to $q(u,v)$ that fits our intuition of what the positioning control should do. I originally derived the mathematics of the positioning control by letting the coefficients $c_i$ of the quadratic patch $q(u,v)$ be arbitrary and then working out what they must be to give the correct bending action. For more complex bending operators I can only offer the observation that the positioning control always produces the intuitively correct behavior in my experiments.

### 3.5 Bending a mesh of patches

For the bending operator to be a useful modeling tool, it must be applicable to complex objects. In practice, complex objects are modeled by a mesh of patches. For smooth objects there are continuity requirements across the edges where the patches of a mesh are joined. The details of these continuity requirements are explored in Chapter 4. My purpose in this section is to demonstrate that the bending operator can be applied to complex objects that are represented as meshes of patches with continuity constraints. I show that the mathematics of the bending algorithm is changed very little by this adaptation. The two main difficulties that arise have to do with the topology of the mesh and the need to maintain the continuity requirements across edges. I handle the topology problem by applying the bending operator only to topologically simple parts of the mesh. I show that the continuity requirements change the unconstrained optimization problem into a constrained optimization problem. As a practical test of these ideas I have implemented a software prototype that can apply the bending algorithm to a smooth mesh of tensor product B-spline patches.

3.5.1 Adapting the bending operator to a mesh of patches

I believe it is worthwhile to proceed as far as possible without limiting this analysis to a particular patch type, and as the reader will see, that is quite far indeed. Such an approach ensures that the conclusions will be applicable to a large variety of existing patch types and perhaps to new types not yet invented. In this section, I make only two assumptions about the patch types:

1. A patch is a regular piecewise smooth mapping $c$ from a closed polygonal domain $D$ in $\Re^2$ to $\Re^3$.
2. For any patch $c$ there exists a finite set of piecewise rational *basis functions* $N_j: D \to \Re^3$ for $j = 1,\ldots,n$ such that $c(u,v) = \sum_{j=1}^{n} b_j N_j(u,v)$. The $b_j$ are called *control points* and are just points in $\Re^3$.

Note: These assumptions are not very restrictive, but they could be even further relaxed at the expense of complicating the optimization algorithm. The optimization algorithm proposed here has to have some finite set of variables to manipulate, and control point-based patches naturally provide such a set. In principle, the patches need only be piecewise twice differentiable functions of the parameter space. In this case, the optimization algorithm could approximate the patches by some finite element basis functions to provide an approximate solution to the bending problem. The approximation error could be reduced below any desired threshhold by using enough basis functions. In fact, control point-based patches *are* finite elements, but they lack the ability to add more basis functions as needed to reduce the approximation error.

I show in Chapter 4 that given the previously stated assumptions 1 and 2, the continuity conditions for a mesh of patches can be formulated as a set of algebraic equality constraints on the patch control points and a set of free real variables. That is, given a mesh of patches $c_1,\ldots,c_k$ such that patch $c_i$ has control points $b_{ij}$, there exist a finite number of real variables $t_r$ and a finite number of rational functions $f_l$ such that

$f_l(b_{11}, b_{12}, \ldots, b_{21}, b_{22}, \ldots, t_1, t_2, \ldots) = 0$. Most of the patch types in common use satisfy these assumptions, including Bezier patches, B-spline patches, and NURBS.

To apply the bending operator to a mesh, we have to be able to compute the desired normals and to evaluate the penalty functions. To satisfy the continuity requirements, we have to maintain the continuity conditions during minimization of the total penalty function. How does the use of a mesh instead of a single patch complicate these operations?

Computing the desired normals. The topology of the mesh can cause difficulties in computing the desired normals. Recall that the correspondence between the target surface and the bending operator surface is via the parameterization of the domains. This presented no problem for single patches in Section 3.3 because the domain for all patches was the unit square. With meshes, this is no longer the case. The assumption that the patches in a mesh are regular implies that it is possible to construct a piecewise polygonal domain with the same edge and vertex connectivity as the mesh. However, this is not much help since the mesh connectivity can be arbitrarily complex, with any number of branching sheets and cycles. The way out of this dilemma is to apply the bending operator only to topologically simple parts of a model.

When is a shape (i.e., a mesh of patches) simple enough to apply the bending operator to it? Let us consider for the present only the problem of computing the normal vectors. If the mesh is part of a single bounded connected sheet with no branching and no self-intersections, it is possible to redefine the set of patches that make up the mesh such that the union of the domain polygons form a single large polygon *P*, possibly containing holes. This is because the patches are regular, which implies that they are continuous mappings with continuous inverses. The technical term for such a function is *homeomorphism*. Two surfaces are said to be *homeomorphic* if there is a homeomorphism between them. As such, the patch functions preserve the topology of the mesh. This fact is proven in chapter 1 of [ Spivak 70 ]. Taking the union of these redefined patch functions produces a function from *P* whose range is exactly the union of the ranges of the original patch functions. Notice that this function satisfies assumptions 1

and 2 and hence is itself a patch. By using an affine transformation of $\Re^2$ if necessary, the domain polygon *P* can be mapped into the unit square, producing a single large patch defined on the unit square. Extending this patch function to the whole unit square produces a single patch, call it *c*, whose range contains the range of the original mesh. If there are holes in the domain polygon, extend the patch function in any smooth way to fill the holes. In order to compute normals for the patch *c*, it is necessary that the differential of *c* exists, which is equivalent to *c* having $C^1$ continuity. To summarize, it is possible to compute the desired normal vector field in the sense of Section 3.3 for any mesh of patches which is homeomorphic to a polygon (possibly with holes) in $\Re^2$ and which has at least $C^1$ continuity.

It is possible to apply the bending operator to simple closed shapes that can be mapped onto a polygon, such a cylinder or torus. Consider bending a cylinder. The cylinder can be cut and flattened into a rectangle, so it is possible to describe it with a single patch. The problem is that the normal vector field computed by the bending algorithm will not be continuous across the cut, since points on either side of the cut are not contiguous in parameter space. Thus, it is not possible to place features, such as dents, across the cut line. Also, unless continuity constraints are used to prevent it, the cylinder will break open at the cut line during bending.

Computing the penalty functions. All the penalty functions are surface integrals of certain functions derived from the basic patch function. The surface integral over a mesh is simply the sum of the surface integrals over the patches that make up the mesh. The only possible problem is that the integrals might not exist. Examining the penalty functions, we can see that all the functions being integrated are derived from first and second derivatives of the patch functions. Hence we can be assured that the penalty functions are well defined provided that the individual patches of the mesh have at least piecewise $C^2$ continuity. In fact, the assumption that the patch functions are regular and piecewise rational implies that the patches have piecewise $C^\infty$ continuity. The continuity degree of the joins between patches is immaterial because these joins have zero area and thus do not contribute to the integrals.

Performing the minimization. The main complication introduced by meshes of patches is the need to minimize the penalty functions under the constraint of satisfying the continuity equations. I show in Chapter 4 that these constraints are nonlinear in general. Most of the difficulty this causes is technical rather than theoretical. Given some random mesh of patches of possibly different types, it is a difficult problem just to write down the constraint equations explicitly. Furthermore, it is hard to write efficient software for optimization with nonlinear constraints. Such software is typically tailored to a specific problem. [ Himmelblau 72 ] is a good textbook on subject of optimization theory which discusses this point. For these reasons modeling systems that work with smooth meshes of patches generally limit themselves to one or a few patch types, for which the continuity equations have been worked out.

Implications for the user interface. We have seen that the bending operator can be applied only to single connected sheets of a model which have at least $C^1$ continuity. This raises the question of how the bending operator can be used in the context of a geometric modeling system that supports models with a more complex topology. I discussed in Chapter 1 what an ideal geometric modeling should look like to users. One of the important ideas there was that complex models should be constructed as hierarchies, with the lowest level parts being very simple shapes. A partial answer to the question is to apply the bending operator only to the lowest level parts of the hierarchy. It seems likely that the bending operator would frequently be used on such parts, but not always. This restriction is too severe, since it precludes applying the bending operator to any assemblage of parts.

A more general method of applying the bending operator is possible, according to the following scenario. The user selects some point of application on the surface of a complex model, not on an edge at which the model branches into multiple sheets. The modeling system selects and highlights part of a connected sheet containing this point. The user can ask the modeler to shrink or expand this selection if desired. The selection can grow until it is the whole connected sheet containing the point of application. That is, the selection can grow until it reaches an edge where the model branches into multiple sheets. If the modeling system permits the user to intervene and select a branch to follow,

the selection can grow further. When the user indicates that the selection is final, the modeling system constructs a "pseudo-patch" that has the unit square as its domain and the selection as its range. The user then applies a bending operator to this pseudo-patch, possibly using the range of action tool to limit the bending action to a subset of this pseudo-patch. There are two problems that arise with this method. First is the problem of keeping the pseudo-patch properly connected to the rest of the model. The bending action must preserve the model topology and the continuity conditions between the pseudo-patch and the rest of the model. This problem is solvable by including the constraint equations for edges that connect the pseudo-patch to the rest of the model in the optimization constraints. The mesh topology will be preserved if at least $C^0$ continuity is preserved at these edges. Second is the problem of rigidity of the model outside the bending area. For example, consider a model of a telephone hand piece. If the connecting member between the transmitter and receiver is bent, the transmitter and receiver ought to be rotated with respect to each other. With my suggested method, this rotation will not occur. There does not seem to be any easy solution to the rigidity problem. To handle this problem, the modeling system would have to contain a very sophisticated representation of the model that included information about possible articulations of the parts.

### 3.6 Implementation issues

This section presents the details of the numerical methods and programming techniques used to implement the bending algorithm. A specific patch type is needed for the implementation, and I chose the tensor product B-spline patch for reasons explained later. However, the implementation of the bending algorithm is independent of the patch type. I implemented the bending algorithm by the combined use of several standard algorithms from numerical analysis. I present here the specific numerical methods I chose and the reasons for their choice. For reasons discussed in Section 3.3.2, the bending algorithm needs to be incremental, and I discuss how this is achieved.

#### 3.6.1 Patch type

The prototype software uses tensor product B-spline patches of arbitrary degree and number of control points. There is nothing special in the prototype about this kind of patch; any piecewise smooth patch type will serve. The prototype does depend somewhat on a rectangular patch domain, but it could be adapted, for example, to a triangular domain without major changes. I justify these statements later in this section. The B-spline patch is a very flexible modeling element. For example, the Bezier patch is a special case of the B-spline patch. The number of control points and the patch degree (polynomial degree) can be varied across a wide range, and quite complex surfaces can be modeled using a single patch. [ Farin 90 ] gives a definition of B-spline patches and an explanation of the deBoor algorithm for evaluating points and derivatives.

As discussed earlier, it is desirable to make the patch type and the algorithms used to do surface shaping operations as independent as possible. I achieve this independence in the prototype by using some object-oriented programming techniques. The software is implemented in the C++ language, and I use the language feature called *abstract base classes* to hide patch implementation details from the algorithms. A key feature of abstract base classes is that they define a class of objects and a set of operations for manipulating these objects without revealing any details about the internal state of the object or the implementation of the operations. I define a hierarchy of abstract base classes culminating in a class `Mpatch` which provides the operations needed to implement the bending algorithm. An abbreviated version of the class hierarchy taken directly from the source code follows.

```
class Apatch { // an abstract patch
public:
   virtual vec3 eval(double u,double v) const = 0;
   virtual vec3 deriv(double u,double v,int ndu,int ndv) const = 0;
};
```

An `Apatch` is a patch that provides the capability to evaluate points and take derivatives with respect to parameter space variables. In C++ the `eval` and `deriv` functions are called pure virtual functions, meaning that their implementation is provided elsewhere (at run time) by some concrete patch object. There are actually two concrete patch types in the prototype software. One is a B-spline patch and the other is "pseudo-patch" which

comprises a mesh of B-spline patches. The routines that use objects of type `Apatch` have no way of telling which concrete patch type they are using.

```
class Gobj { // a graphics object
public:
   virtual void draw(GW*) const = 0;
   virtual void render(GW*) const = 0;
};
```

A `Gobj` is an object that knows how to draw a 3-D projection of itself, making use of a `GW`, which is an object that encapsulates information about the graphics environment.

```
class Mpatch : public Apatch, public Gobj { // bendable patch
public:
   virtual void control_pts(const v3mat&) = 0;
   virtual const v3mat& control_pts() const = 0;
   virtual v3mat& control_pts() = 0;

   virtual double obj(const v3mat& yt) const = 0;
   virtual void dobj(const v3mat& yt,v3mat& ydt) const = 0;

   virtual void range_of_action(RGAct*) = 0;
   virtual void desired_normals(Sarb*,const mat32&,
      double u,double v) = 0;

   virtual void normal_wgt(double) = 0;
   virtual void metric_wgt(double) = 0;
   virtual void shape_wgt(double) = 0;
   virtual void collapse_wgt(double) = 0;
};
```

An `Mpatch` is a class that inherits the properties of an `Apatch` and a `Gobj` and additionally provides operations that are needed by the bending algorithm. The geometric shape of an `Mpatch` is defined by a rectangular array of 3-D control points. The function `control_pts` permits reading or writing these control points. An `Mpatch` has a penalty function that measures the deviation of the patch from some ideal shape. In the case of the bending algorithm, this is just the penalty function defined in Section 3.2. The function `obj` evaluates this penalty function for a given set of control points, and the function `dobj` evaluates the derivative of the penalty function with respect to the control points. The function `range_of_action` allows the patch access to information about the range of action, which is encapsulated in an `RGAct` object. The function `desired_normals` allows the patch access to information about the bending operator so that the desired normals can

be computed. The bending operator is specified via a prototype function encapsulated as a `Sarb` object, a 3∞2 orientation matrix encapsulated as a `mat32` object, and a point of application $(u,v)$. The functions `normal_wgt`, `metric_wgt`, `shape_wgt`, and `collapse_wgt` set the adjustable parameters of the penalty function.

The `Mpatch` class was designed to expose only the bare minimum of functionality needed to implement the bending algorithm. It was necessary to assume a few things, namely, that the patch domain is the unit square and that the geometry of the patch is defined by a rectangular array of control points. These assumptions are not very restrictive. Due to this design, any patch type meeting these assumptions can be used and the bending algorithm will work. Indeed, the source code for the bending algorithm does not even have to be recompiled to change the patch type because the linkage between the abstract patch and the concrete patch occurs at run time. As proof of this, I point out that in the prototype software, exactly the same source code is used to apply the bending operator to a single patch and to a rectangular mesh of patches.

### 3.6.2 Incremental bending

A continuous transformation from the original target surface to the final bent surface seems desirable to avoid surprising the user. The user's mental model is that a piece of elastic material is being deformed. If the deformation is not continuous, the user will be caught be surprise, and the bending operator will be less intuitive. As stated earlier, the minimization process has some mathematical difficulties. There may be undesirable local minima which represent drastic deformations of the target surface, such as part of the surface being flipped over. To avoid these, I arrange the minimization such that at any point in the process the current surface is near a local minimum. I originally thought that this would help efficiency, as the minimization algorithm would always be operating in a region in of quadratic convergence. However, experimentation shows that larger step sizes are more efficient. I do not fully understand this, but I speculate that the BFGS algorithm used (see Section 3.6.4) has such a high startup cost to attain a good estimate of the Hessian that it outweighs the savings due to quadratic convergence.

Making the algorithm incremental. The key idea is to start with a small piece of parameter space for the bending surface, and expand to the whole parameter space in a number of stages. Mathematically, if the point of application is $p = (u_0, v_0)$, the bending surface frame field at "time" *t* is

$$f(t;u,v) = f((u-u_0)t + u_0, (v-v_0)t + v_0).$$

Then *t* is increased from some small $\delta > 0$ up to 1 in a number of steps, and the minimization problem is re-solved at each step. The starting *t* cannot be 0, as this would not define a surface. The main problem with this approach is that it can introduce some "wiggles" in which the target surface bends first one way, then another.

I considered and rejected two other possible ways of making the algorithm incremental.

I tried introducing a parameter $s \in [0,1]$ into the differential system for the normal vector field, yielding

$$dg = (\omega + sr^t vr)g.$$

This attempt failed because the system no longer satisfies the integrability conditions.

I considered interpolating between the normal vector field of the original target surface and the desired normal vector field. Linear interpolation works if at no point does a normal turn 180 degrees or more. Otherwise singularities occur. I think that this constraint is too restrictive, as it precludes the use of bending operators that have drastic amounts of curvature. This possibility needs further research. There may be some other way to interpolate normals, or there may be a hybrid approach combining normal interpolation with the method I am currently using.

3.6.3 Computing the normal vector field

There are two parts: converting the differential system into an ordinary differential equation, and solving the differential equation in a systematic way. Recall that the differential system is

$$g(p) = e(p)$$
$$dg = (\omega + r^t \nu r) g \text{ on } D,$$

with

$$f = rg \text{ on } D.$$

To solve for $g(q)$ at any $q \in D$, we can apply this system to a piecewise smooth curve in parameter space

$$\alpha: [0,1] \to D, \quad \text{with } \alpha(0) = p,\ \alpha(1) = q$$

to produce an initial value problem for an ordinary first order differential equation of nine variables:

$$dg(\alpha') = (g \circ \alpha)' = (\omega(\alpha') + r^t \nu(\alpha') r)(g \circ \alpha)$$
$$(g \circ \alpha)(0) = g(p) = e(p).$$

Since $g$ is well-defined, the solution to the differential equation depends only on the end-points $p$ and $q$ and is independent of the which curve $\alpha$ is used. This path independence can be exploited to compute the values of $g$ on a regular $m \times n$ grid in parameter space. This array of values will be needed later for performing a numerical integration. The idea is to integrate from $p$ along a straight line to the nearest grid point, and then follow a pattern of straight lines to reach all the grid points, as shown in Figure 3.6.3-1.

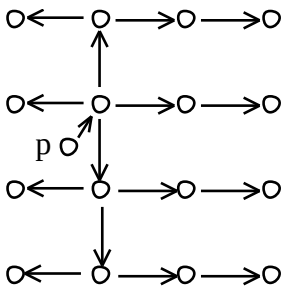



Figure 3.6.3-1 Path for integration

The computation of the desired normals is fairly expensive, amounting to about a fourth of the total bending time. Therefore, to minimize the total length of the lines integrated along, I construct a minimal spanning tree rooted at *p* that reaches all the grid points. I use the Bulirsh-Stoer method [ Bulirsch 80 ] for numerically integrating the ordinary differential equation.

3.6.4 Minimizing the penalty function

Rejected alternatives. Before looking at ways to minimize the penalty function, I consider and reject an alternative. One can apply the Euler-Lagrange equations to convert many variational problems, including this one, into a system of partial differential equations. For my penalty functions, this system turns out to be non-linear. Solving general systems of non-linear partial differential equations is a hard problem. One of the standard solution techniques is to turn the system into a variational problem, but that is what I already have.

How the penalty function is minimized. The penalty function that the bending algorithm minimizes is defined as the sum of a set of integrals over the patch surface. In practice, the integral is pulled back to the patch domain and converted to an ordinary iterated integral. The surface itself is defined as the weighted sum of a set of basis functions, the weights being the control points. It has the form

$$p(u,v) = \sum_{j=0}^{n} b_j N_j(u,v).$$

The terms that appear in the integrals are various combinations of $p$ and its first and second partial derivatives with respect to $u$ and $v$. Hence we need to minimize a function of the form

$$f(b_1,\ldots,b_n) = \int_0^1 \int_0^1 F(p(u,v), p_u(u,v), p_v(u,v), p_{uu}(u,v), p_{uv}(u,v), p_{vv}(u,v))\,du\,dv,$$

i.e., find a set of control points $b_1,\ldots,b_n$ that minimizes the integral. The minimization technique that I use requires computing first partial derivatives of $f$ with respect to the control points, which can be expressed in the form

$$\frac{\partial f}{\partial b_j} = \int_0^1 \int_0^1 \left( \frac{\partial F}{\partial p}\frac{\partial p}{\partial b_j} + \frac{\partial F}{\partial p_u}\frac{\partial p_u}{\partial b_j} + \frac{\partial F}{\partial p_v}\frac{\partial p_v}{\partial b_j} + \frac{\partial F}{\partial p_{uu}}\frac{\partial p_{uu}}{\partial b_j} + \frac{\partial F}{\partial p_{uv}}\frac{\partial p_{uv}}{\partial b_j} + \frac{\partial F}{\partial p_{vv}}\frac{\partial p_{vv}}{\partial b_j} \right) du\,dv.$$

The collection of derivatives $\left( \frac{\partial f}{\partial b_1},\ldots,\frac{\partial f}{\partial b_n} \right)$ is a vector with $3n$ components, namely, the gradient of $f$. These integrals are evaluated via a point sampling technique (Gaussian quadrature), which turns the problem into one of computing finite sums. The minimization of the penalty function proceeds as follows. The set of control points of the original surface is taken as the starting estimate. An iterative process (the BFGS algorithm, discussed later) is used to reduce the value of $f$ by varying the control points. The basic iteration step first chooses a direction vector in the space of control points and then does a line minimization along that vector to obtain a new estimate. The gradient of $f$ is used to select the direction vector. If one simply used the negative of the gradient of $f$ as the direction vector, this would be the method of steepest descent. However, the BFGS algorithm uses a more sophisticated way of picking the direction vector, which leads to a considerably more efficient algorithm.

Computing the integrals for the penalty function. I chose to do the numerical integral integration by Gaussian quadrature. This method gives order $2n-1$ accuracy with $n$ sample points (i.e., the method gives exact results for a polynomial of degree $2n-1$,) making it one of the most efficient methods known with respect to the number of function evaluations. Most of the computational cost of the bending algorithm is in the evaluation of the penalty function, so it is very important to minimize the number of evaluations. The chief drawback of Gaussian quadrature is the difficulty of obtaining an error bound. The usual practice is to compute the integral for increasing values of $n$ until the result doesn't change [ Bulirsch 80, p. 151 ]. However, for efficiency I choose $n$ as a function of patch degree and number of control points to give a reasonable accuracy, based on numerical experiments. I used Romberg integration originally, but I found that Gaussian quadrature requires about a fourth as many function evaluations for the same accuracy.

Minimization method. The main considerations in choosing a minimization algorithm are order of convergence, number of function evaluations, and the degree of derivatives required. Since the penalty functions are complicated, it is difficult to write a program to compute first derivatives with respect to the patch control points, and well-nigh impossible for second derivatives. One of my goals is experimenting with different penalty functions, so I would prefer a method that doesn't require derivatives. Unfortunately, I discovered that methods that do not use derivatives are far less efficient (by at least an order of magnitude in this application) than those that do, to the extent that experiments just take too long to run.

After examining several methods, I chose the BFGS (Broyden, Fletcher, Goldfarb, and Shanno) algorithm [ Bulirsch 80 ]. It is a quasi-Newton method, meaning that it works like Newton's method, but second derivatives (Hessian) are not computed. Instead, an estimate of the Hessian is built up as the algorithm proceeds using only first derivative information. The method has quadratic convergence near a local minimum and requires computing only function values and first derivatives. The method can make use of second derivatives to reduce the number of iterations, but some experimentation has led me to believe that the cost of computing second derivatives outweighs the savings due to fewer iterations. The basic iterative step of the BFGS algorithm is defined as follows. Let

$f:\Re^n \to \Re$ be the function being minimized, let $x_k \in \Re^n$ be the current estimate of the minimum, let $g_k$ be the gradient of $f(x_k)$, and let $H_k$ be the current estimate of the Hessian of $f(x_k)$. Then set

$$
\begin{aligned}
s_k &= H_k g_k \\
\lambda_k &= \min\left(f(x_k - \lambda s_k) \middle| \lambda \geq 0\right) \\
x_{k+1} &= x_k - \lambda_k s_k \\
p_k &= x_{k+1} - x_k \\
g_{k+1} &= \text{ gradient of } f(x_{k+1}) \\
q_k &= g_{k+1} - g_k \\
H_{k+1} &= H_k + \left(1 + \frac{q_k^t H_k q_k}{p_k^t q_k}\right)\frac{p_k p_k^t}{p_k^t q_k} + \frac{1}{p_k^t q_k}\left(p_k q_k^t H_k + H_k q_k p_k^t\right).
\end{aligned}
$$

It is shown in [ Bulirsch 80 ], Theorem 5.11.9, that if the line minimization for $\lambda_k$ is done with reasonable accuracy and $H_0$ is positive definite, $x_k$ converges quadratically to a stationary point of *f*, and $H_k$ converges to the Hessian of $f(x_k)$. $H_0$ is usually taken to be the identity matrix, although the method tends to converge faster if $H_0$ is a better estimate of the Hessian.

The big advantage of the BFGS algorithm in my application is that the number of iterations needed for reasonable accuracy is considerably less than the number of degrees of freedom of the problem. For example, an order 6∞6 B-spline patch has 36 control points, each with 3 components, yielding 6∞6∞3=108 degrees of freedom. The minimization algorithm typically converges in 15-20 iterations. Previously, I experimented with Powell's method, which is also quadratically convergent and doesn't require computing derivatives, but the minimum number of iterations required is at least twice the number of degrees of freedom. For the 6∞6 B-spline patch example, Powell's method converged in about 400 iterations. I also experimented with the most simple method, steepest descent. The literature warns that this is usually a poor method [ Bulirsch 80, pp. 308 ], with only linear convergence. I stopped the experiment when it failed to converge after 2000 iterations.

3.6.5 Implementing the bending operator for a mesh of B-spline patches

To demonstrate the feasibility of bending meshes of patches, I implemented the bending algorithm for a mesh of B-spline patches in the prototype software. My goal was only to provide a proof of concept, so the functionality is limited but sufficient to show that the main problems are solvable. The mesh is required to be a rectangular array of tensor product B-spline patches. All the patches in the array must have the same parameters, that is, the same knot vectors, control points, and patch degree. The interior edges of the patches in the array are required to join with $C^k$ continuity for $k = 0$, 1, or 2. $C^k$ continuity has the advantage of generating linear constraint equations. (Note: it would be better to use $G^k$ continuity, as discussed in Chapter 4, but $G^k$ continuity leads to non-linear constraint equations, making the optimization problem much harder.) It is also possible to join the outside top and bottom or left and right edges of the mesh with $C^k$ continuity, creating the topologies of a cylinder, a Möbius strip, a torus, a Klein bottle, or a projective plane. This implementation had to address the problems of solving for the desired normals of a large "pseudo-patch," integrating the penalty functions over a mesh, setting up the constraint equations, and performing a constrained optimization.

Creating a pseudo-patch. The individual patches of the mesh all have as their domain the unit square $[0,1]^2$. Because the mesh is rectangular, it is easy to create a combined pseudo-patch by chopping up the unit square into rectangles and assigning these rectangles as the domains of the B-spline patches. As an example, consider a $2 \times 2$ mesh. Label the patches $c_{ij}$ for $i, j = 0, 1$. Then the pseudo-patch is

$$c(u,v) = \begin{cases} c_{00}(2u, 2v) & \text{for } 0 \le u < 0.5, 0 \le v < 0.5 \\ c_{01}(2u, 2v-1) & \text{for } 0 \le u < 0.5, 0.5 \le v \le 1 \\ c_{10}(2u-1, 2v) & \text{for } 0.5 \le u < 1, 0 \le v < 0.5 \\ c_{11}(2u-1, 2v-1) & \text{for } 0.5 \le u \le 1, 0.5 \le v \le 1 \end{cases},$$

which can be differentiated by the chain rule to get the derivatives needed for computing the desired normals and the penalty functions.

Setting up the constraint equations. I do not give the complete mathematics of B-spline patches here since it is not necessary to understand this algorithm. A good reference on B-spline curves and patches is Farin's *Curves and Surfaces for Computer Aided Geometric Design* [ Farin 90 ]. Farin gives the constraint equations on the control points for $C^k$ continuity across an edge join. First, the patches have to be compatible in the direction of the edge; that is, they must have the same knot vectors, degree, and number of control points in the edge direction. Second, the corresponding rows (or columns) of control points on either side of the edge must satisfy a difference equation, as follows. Label the control points on either side of the edge as $b_{ij}$ and $c_{ij}$ for $i = 1,\ldots,n$ and $j = 1,\ldots,m$. The edges join with $C^k$ continuity if $\Delta^r c_{n-r,j} = \Delta^r c_{1,j}$ for $r = 0,\ldots,k$ and $j = 1,\ldots,m$. $\Delta^r$ denotes the *r*-th iterated forward difference and is defined recursively by $\Delta^0 a_i = a_i$ and $\Delta^r a_i = \Delta^{r-1} a_{i+1} - \Delta^{r-1} a_i$ for any sequence $a_i$. These constraint equations form a linear system in terms of the control points.

Performing the constrained optimization. I chose to use a form of continuity that generated linear constraints because the software for optimization with linear constraints is much easier to write and more efficient than for optimization with nonlinear constraints. I had already developed software to perform unconstrained minimization of the penalty functions, and my problem was to adapt it to use linear constraints. The minimization routine basically works by repeatedly doing line minimizations in various directions. The method is explained in detail in Section 3.6. I modified the algorithm such that the direction vectors were always chosen to be in the null space of the constraint equations. Thus, if the constraints are initially satisfied, they continue to be satisfied as the minimization proceeds. The constraints are initially satisfied by projecting the control points onto the null space of the constraint equations.

### 3.6.6 Summary

One of the design goals for the bending algorithm is independence from patch type. I showed that by using suitable programming techniques, it is practical to implement the bending algorithm with very few assumptions about the patch type. The main idea is the

use of abstract base classes provided by the C++ programming language to isolate the patch implementation from the patch interface. Similar techniques could be used in any standard programming language.

The problems of computing the desired normals and minimizing the penalty function were solved by using a collection of standard numerical analysis techniques. The main difficulty was choosing which of the many available techniques to use. Numerical analysis is as much an art as a science. There are often many ways to solve a given problem, and the choice of which way is best tends to be strongly problem dependent. I found that experimentation with different methods was needed to determine which worked well for the bending algorithm.

### 3.7 Strengths and weaknesses of the bending operator

The main strengths of the bending operator are its intuitiveness, power, and generality. We saw in Section 3.1 that for the simple example of bending a cylinder into a torus, the bending operator is easily understood and predictable, hence intuitive. I present further evidence of this in Chapter 5 by demonstrating the bending operator for several more complex examples. The power of the bending operator comes from its ability perform a large variety of surface shaping tasks. This ability derives from defining the bending operator by means of a bending surface that may be quite complex. We saw in Section 3.2 that the bending operator is independent of patch type and thus can be applied to a wide variety of patch types in current use. Furthermore, we saw in Section 3.5 that it is possible to apply the bending operator to more complex models made up of a mesh of patches. It is these reasons that justify calling the bending operator general purpose. The main problem with applying the bending operator to models with a complex topology is that the operator can only be applied to pieces of the model and not to the whole model at once. This tends to be a characteristic problem of surface-based modeling tools and an area where tools based on space warp excel.

A big problem is that the bending operator is slow. Since the bending operator is based on minimization of a complex and nonlinear penalty function, the computational workload is

very high, making the algorithm too slow to operate at interactive speeds on current workstations. On an HP-730 workstation a single application of the bending operator takes seconds to minutes depending on the complexity of the surface and the bending tool. Any modeling tool based on minimization techniques can be expected to have high computational requirements. Other research groups working on such tools have mostly sacrificed accuracy for speed by using linear or quadratic approximations for the penalty functions. This kind of tradeoff could be made for the bending operator also, but I chose not to do this in the current work. My reason is that the algorithm is complicated enough that I think it would be impossible to separate the effects of approximation errors from the occasionally unexpected effects of the operator itself. An example of such unexpected effects are the edge effects.

Edge effects seem to be a problem for the bending operator. These effects appear when a particular operation causes a large change of curvature near the edge or corner of a patch. The symptom is that the surface metric changes more at the edge than in the center of the patch. An example is using the bending operator to place a dent or bump in a surface. If the bump is near the center of the patch, everything is fine, but if it close to an edge or corner, the surface there tends to contract, causing the bump to be asymmetrical. Before I added a penalty function to resist collapse, such bumps even caused the surface to fold over or collapse to a point. With collapse resistance, this problem is more annoying than crippling. By this I mean that the bending operator does approximately what one would expect near edges, but with a noticeable error. I do not fully understand these edge effects, but I speculate that they occur because near an edge, there is less surface area nearby to contribute to the metric penalty function. Effectively this causes the surface to be "softer" at edges and corners. In support of this speculation, I note that no edge effects occur at interior edges and vertices of a mesh of patches. Thus one way to eliminate edge effects is to surround the target patch with extra patches. Furthermore, the range of action tool tends to reduce, though not eliminate, these edge effects.

I believe the bending operator is an important new tool for surface-based geometric modelers. Although it is slow, it can do in one intuitive step complex surface operations that might take many steps or be impossible with other tools. Although the bending

operator can only be applied to parts of complex models, it is still very useful for those parts. Since people tend to build complex models hierarchically, I expect that the bending operator will be mostly used for parts of the model at fairly low levels of the hierarchy.

# Chapter 4
# Applying the methodology to another problem: Joining

My goal in this chapter is to demonstrate that the methods used to implement the bending operator are very general and can be applied to a variety of surface shaping tasks. I do this by presenting a detailed case study of applying these methods to an important problem in 3-D graphics modeling, the joining problem. This is the problem of fitting together a set of curved surface patches into a mesh such that a specified degree of smoothness is attained across the interior edges and vertices of the mesh. The chapter has three main sections. Section 4.1 is a discussion of the general characteristics, advantages, and disadvantages of variational methods for surface shaping. Section 4.2 defines the joining problem in more detail and presents a brief review of the state of the art. Section 4.3 presents an outline of a joining algorithm based on variational techniques.

## 4.1 The role of variational techniques in surface shaping algorithms

Variational techniques are powerful because they automatically turn a local specification of a shape goal – the penalty function – into a global algorithm for shaping the surface to attain the goal. It is often easier to formulate the desirable properties for a surface locally rather than globally. A local surface goal is simply a non-negative function defined at each point of a surface. The better the surface meets the goal, the smaller the function value is at any point. In the case of bending, for example, the goal of changing the surface normals was formulated as a term in the composite penalty function that measured the difference between the actual and desired surface normals. The precise definition of the penalty function depends on the task to be achieved, and producing a good one is as much an art as a science. The penalty function is usually defined in terms of the local geometric properties of the surface. In Chapter 3 the penalty function for the bending algorithm was

defined in terms of the desired normals and the metric for the goal surface. In this chapter the penalty function for joining is defined in terms of constraint equations imposed by the requirement for smoothness across patch edges and in terms of a "fairness functional" that measures how fair, or free of unnecessary bumps and wiggles the surface is.

Algorithms based on variational techniques have two main problems. First, they take a lot of computing and thus tend to be slow. It is only in recent years with the advent of desktop workstations with substantial floating point compute power that variational algorithms have become practical at all. Except in special cases in which the penalty function is quadratic, variational algorithms are too slow for interactive use, taking seconds to minutes per operation. Quadratic penalty functions are special because they can be minimized using linear algebra methods, with speedups of orders of magnitude compared to arbitrary penalty functions. The other main problem comes about because variational algorithms are indirect – we want some global change of shape, but we specify a local penalty function. Sometimes it is hard to predict exactly what global effects a given local penalty function will produce. For example, the bending operator had unexpected edge effects. Also, it may be necessary to "tune" the penalty function by adding components to prevent undesired behavior. An example is the collapse resistance term in the penalty function for the bending operator.

### 4.2 Background of the joining problem

The joining problem is important because in practice it is impossible to build a complex graphics model from a single patch. The model may have branching surfaces or cycles, making it impossible to assign a consistent two-dimensional coordinate system to the whole object. Even in cases where it is theoretically possible to use a single complex patch, it is often more convenient to use several simpler patches. As mentioned in Chapter 2, a good modeling system should support hierarchical models in which independently designed parts are combined to produce more complex objects. Therefore, objects need to be composed of several patches. This raises the question of how to join them together. It is often required that certain smoothness criteria be met at shared

vertices and edges. For example, to avoid visible seams a minimum requirement is continuity of normal vectors across patch edges.

This section presents some definitions and results from recent research on joining. A definition is presented of what it means for patches to join smoothly, that is, to join with what is called *geometric continuity*. Producing a geometrically continuous join between two arbitrary patches requires solutions of a system of non-linear partial differential equations (PDE's) and thus is a hard problem. For a large class of patches based on control points, the PDE system can be reduced to a set of non-linear algebraic constraints on the patch control points and a set of free variables.

### 4.2.1 Definition of geometric continuity

It has been agreed in the graphics community that the correct notion for smooth joining of patches is that of *geometric continuity*. Roughly speaking, a mesh of patches joins with geometric continuity of degree *k*, denoted $G^k$, if an arbitrary point in the mesh cannot be distinguished locally from a point of a $C^k$ continuous patch. A precise definition and an explanation of why this is the right concept is given later in this section. Several authors independently defined $G^1$ and $G^2$ continuity of surfaces (cf. [ Sabin 76 ] and [ Vernon 76 ]) using geometric surface properties. T. DeRose presented the first systematic treatment of geometric continuity of arbitrary degree in his Ph.D. dissertation [ DeRose 85 ]. He gave a method for constructing the constraint equations, known as *Beta constraints*, that are necessary and sufficient for geometric continuity of arbitrary order for a mesh of patches.

The following definition for geometric continuity of a join between two surface patches is adapted from [ DeRose 85 ]. It is based on the idea that how the patches of a mesh are parameterized is irrelevant to the geometry of the resulting surface. Changing the way a patch is parameterized does not change the set of points in $\Re^3$ that comprise the surface, and in modeling it is only this set of points that we are usually concerned with. To begin, we need a new definition of a surface patch. This definition is more elaborate than the definition of a patch used in Chapter 3 to allow for domains other than rectangles. This is

needed because current modeling practice makes use of various shapes for patch domains, particularly triangles.

Definition 4.2.1-1. A *domain* is a closed, connected, and bounded subset $D$ of $\Re^2$, with a finite number of edge curves $E_i$ that can be piecewise regularly $C^\infty$ parameterized as $E_i(s)$, $s \in [0,1]$. A $C^k$ *patch* on $D$ is a regular (Jacobian has rank 2) mapping $c: D \to \Re^3$ that has continuous derivatives through order $k$ on $D$.

The regularity requirement excludes various singularities such as cusps and conical points and ensures that there is a well-defined normal vector at every point. Note that $c$ has to be defined on some open set containing $D$ in order for the $C^k$ condition to make sense.

Definition 4.2.1-2. Let $\phi$ be a function from a subset of $\Re^n$ to $\Re^n$. $\phi$ is said to be a $C^\infty$ *diffeomorphism* if $\phi$ is invertible and both $\phi$ and $\phi^{-1}$ possess continuous derivatives of all orders. $\phi$ is said to be *orientation preserving* if the Jacobian of $\phi$ has a positive determinant at all points. In the case of $n = 2$, this is equivalent to $\phi_u^1 \phi_v^2 - \phi_u^2 \phi_v^1 > 0$.

Definition 4.2.1-3. Two $C^k$ patches $c_1: D_1 \to \Re^3$ and $c_2: D_2 \to \Re^3$, defined on domains $D_1$ and $D_2$, *join with geometric continuity* $G^k$ along edges $E_1$ and $E_2$ if there exist orientation-preserving $C^\infty$ diffeomorphisms $\phi_1$ and $\phi_2$ such that $\phi_1^{-1}(E_1(s)) = \phi_2^{-1}(E_2(s))$ for $s \in [0,1]$, $\phi_1^{-1}(D_1)$ and $\phi_2^{-1}(D_2)$ overlap only on the common edge $\phi_1^{-1}(E_1([0,1]))$, and the composite map $(c_1 \circ \phi_1) \cup (c_2 \circ \phi_2)$ is a $C^k$ function at all points of the edge $\phi_1^{-1}(E_1([0,1]))$.

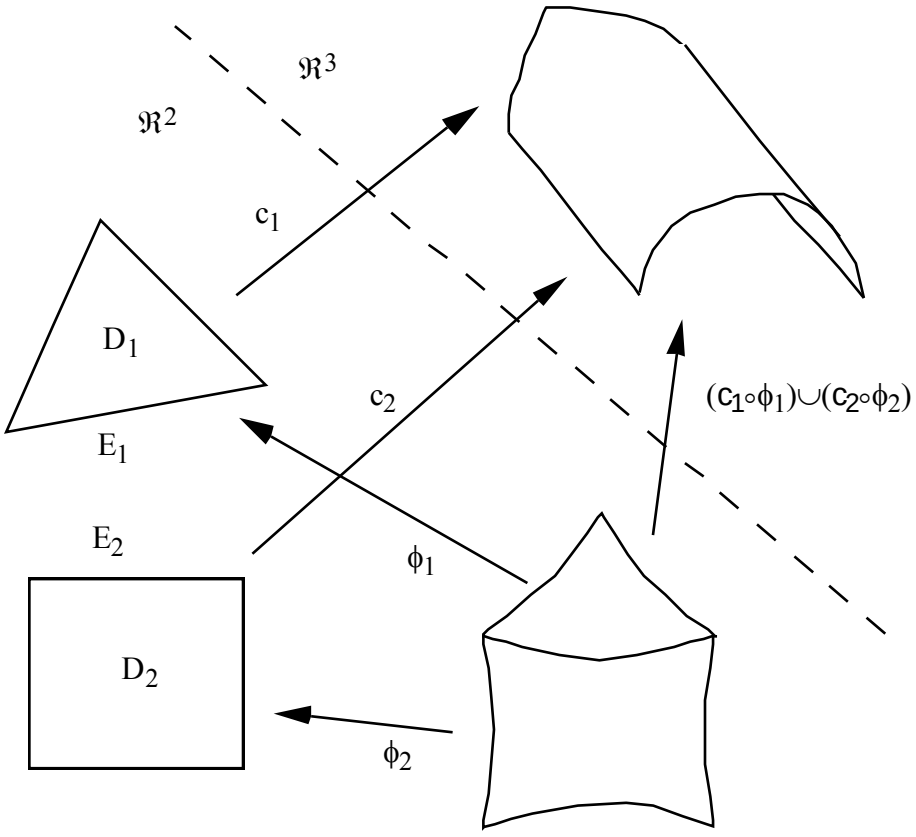



Figure 4.2.1-1 Edge joining with geometric continuity

This definition essentially says that an edge join has $G^k$ continuity if some (orientation-preserving) reparameterization of the composite surface has $C^k$ continuity. Figure 4.2.1-1 illustrates the spaces and functions involved. This is an adaptation of the definition of an oriented, two-dimensional manifold from differential geometry. The orientation-preserving property of the reparameterizations ensures that the normal vectors cannot flip over at edge joins[4].

Geometric continuity is the right concept of smoothness for geometric modeling because reparameterization of a patch does not change its geometric properties. Furthermore, there

[4] There is also a concept of weak geometric continuity in which the orientation preservation requirement is dropped, but in computer graphics, it is not very useful because it can lead to "smooth" surfaces without smooth normals. This interferes, for example, with rendering algorithms.

are benefits to be gained from taking the trouble to handle continuity of joins properly. DeRose shows $G^k$ continuity is less restrictive than $C^k$ continuity. This means that the class of possible patch types that can be use in modeling surfaces is larger. Other advantages are discussed later in this section.

4.2.2 Implications of $G^k$ continuity

Direct application of the definition $G^k$ continuity is not very useful for computer implementations because it would require working with the function space of all diffeomorphisms. DeRose presents a general procedure for deriving a set of constraint equations, the *Beta constraints*, that are necessary and sufficient for $G^k$ continuity. To state his result, some notational machinery is needed. Let us denote the mixed partial derivative of a function *f* of two variables taken *i* times in the first variable and *j* times in the second variable by $D_{i,j}f$. In the usual notation, $D_{i,j}f(u,v) = \dfrac{\partial^{i+j} f(u,v)}{\partial u^i \partial v^j}$. Repeated application of the chain rule and product rule to the composite function $f(s(u,v),t(u,v))$ yields a polynomial in terms of the mixed partial derivatives up to order $i+j$ of *f*, *s*, and *t*. This polynomial is hard to write down explicitly for the general case. A statement of the formula is given in Loomis and Sternberg [ Loomis 90 ], but we will not need it here. Denoting this polynomial by $P_{i,j}$, we have

$$D_{i,j}f(s(u,v),t(u,v)) =$$
$$P_{i,j}(D_{11}f, D_{12}f, D_{21}f, \ldots, D_{m,n}f, D_{11}s, D_{12}s, D_{21}s, \ldots, D_{11}t, D_{12}t, D_{21}t, \ldots).$$

This formula is abbreviated in the following by

$$D_{i,j}f(s(u,v),t(u,v)) = P_{i,j}(D_{m,n}f, D_{m,n}s, D_{m,n}t).$$

DeRose states and proves the following theorem (Theorem 2.2 in his dissertation).

Theorem 4.2.2-1. Let $c_1 : D_1 \to \Re^3$ and $c_2 : D_2 \to \Re^3$ be $C^k$ patches that join with geometric continuity $G^k$ along shared edges $E_1$ and $E_2$ as in Definition 4.2.1-3. Further

assume that $D_1$ has coordinates *s* and *t* such that edge $E_1$ is obtained by fixing *s* at $s_0$ and letting *t* vary, i.e., $(s_0,t)=E_1(t)$ for $t\in[0,1]$ (this can be achieved by a suitable change of parameterization). Then $c_1$ and $c_2$ meet with $G^k$ continuity along edge $c_1(E_1)$ if and only if there exist functions $\beta_k^u(t)$ and $\beta_k^v(t)$ such that the partial differential equations (PDEs)

$$\frac{\partial^i c_1}{\partial u^i}(u_0,t)=P_{i,o}\left(\frac{\partial^{j+l}c_2}{\partial u^j\partial v^l}\left(\beta_0^u(t),\beta_0^v(t)\right),\beta_j^u(t),\beta_j^v(t)\right) \qquad (4.2.2\text{-}1)$$

are satisfied for $i=1,\ldots,k$, and

$$\beta_1^u(t)\frac{d\beta_0^v}{dt}(t)-\beta_0^u(t)\frac{d\beta_1^v}{dt}(t)>0\,.$$

The functions $\beta_k^u(t)$ and $\beta_k^v(t)$ are called the *Beta constraints*. The functions $\beta_0^u$ and $\beta_0^v$ are determined by the requirement that the patch edges meet, and the functions $\beta_1^u$ and $\beta_1^v$ are constrained by the inequality in the theorem. The rest of the Beta constraints are completely arbitrary $C^\infty$ functions. The idea of DeRose's proof is to choose a diffeomorphism $\phi$ such that $c_1$ and $c_2\circ\phi$ form a composite $C^k$ function. The equation $c_1=c_2\circ\phi$ is satisfied on the edge $E_1$. The Beta constraint functions turn out to be simply the partial derivatives of the component functions of $\phi$ in the cross-edge direction. DeRose states that the PDE system of equation 4.2.2-1 is not necessarily solvable for a given patch type.

For a mesh of patches with $G^k$ continuity, there are $2k$ Beta constraints per edge. To completely define a mesh of patches that has $G^k$ continuity, the continuity at shared vertices as well as shared edges has to be defined. A detailed discussion of $G^k$ continuity at shared vertices can be found in [ DeRose 85 ] and [ Gregory 89 ]. I omit these details because they are not necessary to support my points. The main difficulty at a shared vertex is to make sure that the patches properly "wrap around" the vertex.

Solving, or even setting up the Beta constraint equations for a given patch type is a hard problem. Obtaining solutions to this problem is an open research area. A common

approach is to design new patch types such that the solvability of the continuity equations is "built in" [ DeRose 88 ] [ Loop 90 ] [ Halstead 93 ]. Another common technique is to restrict the Beta constraints to being constant functions. DeRose shows that for tensor product surfaces the Beta constraint functions are, in fact, constant [ DeRose 85 ]. Even with this restriction, there are still too many Beta constraints for direct user control of them. In practice, some of them are fixed and others are ganged together such that a single control variable determines a set of Beta constraints. An example of this technique is the Beta-spline curve [ Barsky 81 ], which exploits the extra freedom of geometric continuity by giving the user direct control over some of the Beta constraints.

4.2.3 Producing algebraic continuity constraints

For an important class of patch types based on control points the Beta constraint equations can be reduced to an algebraic system, using a well-known technique. Let *D* be a patch domain in $\Re^2$ with coordinates *u* and *v* as in definition 4.2.1-1. Suppose we have a finite number of basis functions $N_j(u,v)$ for $j = 1,\ldots,n$, such that $N_j(u,v)$ is a piecewise polynomial or rational function of *u* and *v* on *D*. Consider patches that are linear combinations of the form

$$c(u,v) = \sum_{j=1}^{n} b_j N_j(u,v),$$

where $b_j$ is a point in $\Re^3$. The $b_j$ are called *control points* and the $N_j(u,v)$ are called *basis functions*. Many commonly used patch types, including tensor product B-spline patches, tensor product Bezier patches, triangular Bezier patches, and non-uniform rational B-splines (NURBS) can be expressed in this form. Since $N_j(u,v)$ is a polynomial or rational function, so are all of its partial derivatives with respect to *u* and *v* of all orders. The partial derivatives of *c* are also polynomial or rational functions because differentiation is a linear operator. This means that all the partial derivatives that appear in equations 4.2.2-1 can be replaced by polynomial or rational functions, converting the PDE system to an algebraic system.

### 4.3 A variational algorithm for joining

In this section I describe several forms of the joining problem and show that they can be reduced to a common form. I outline how the optimization methods used in the bending algorithm can be adapted to produce an algorithm for joining patches. The requirement of inter-patch geometric continuity leads to a non-linear constrained minimization algorithm. The penalty function is chosen to make the resulting surface as fair as possible.

#### 4.3.1 Classification of joining problems

Most of the joining problems met in graphics modeling practice fall into one of the following three classes:

1. *Edge-Edge*. Join two patches by pulling together two edges that are not initially contiguous, such that the join has a specified geometric continuity of degree *k*. One or both of the patches may deform during the joining process. A precise definition of geometric continuity of degree *k* was presented in section 4.2.

2. *Blend*. Join the edges of two patches by connecting them with a third patch that fills in the gap with a specified degree of geometric continuity at both edges. The two patches being connected usually do not deform during the process.

3. *Edge-Curve*. Join two patches by gluing the edge of one to a curve in the other with a specified degree of geometric continuity at the joining curve. The patch containing the curve does not deform during the process.

I consider the Edge-Edge joining problem to be fundamental because a solution for it can be applied to solve both the Blend and Edge-Curve joining problems.

The Edge-Edge Joining Problem. The problem is to pull together the edges of two patches so that the patches join with a specified continuity across the edge. What is needed to make this a well-defined mathematical problem? Four pieces of information are needed:

1. Edge matching - how the corresponding points on the two edges should match up.
2. Softness - what parts of the patches are allowed to deform during the process.
3. Position - the initial positions and orientations of the patches.
4. Continuity - the degree of geometric continuity at the edge after joining.

The details of exactly how this information is specified is necessarily intertwined with the design of the joining algorithm. An outline for an Edge-Edge joining algorithm is given in section 4.3.2, and these details are discussed in that section.

Reducing Other Joining Problems to Edge-Edge Joining. Earlier I claimed that Blend and Edge-Curve joins can be reduced to Edge-Edge joins. This is fairly easy to see for Blend joins, in which two patch edges are to be interpolated by a third patch. Let us call the two patches to be joined A and B and the Blend patch C. Patches A and B are usually not allowed to deform during the joining process. Initially C is any patch which is reasonably close to A and B, it does not need to be touching. We simply perform two Edge-Edge joins, one from A to C and one from C to B. The trick is to assign a zero value to the softness measure of A and B, so they do not deform during the joining process. During the second Edge-Edge join B-C, the geometric continuity constraints for both the A-C and B-C edges have to be preserved. Because of this, it may turn out to be more efficient in an implementation to perform the A-C and B-C joins simultaneously.

The Edge-Curve problem is harder. Say we want to join an edge of patch A to a curve *c* lying in the interior of patch B. If a zero value is assigned to the softness of patch B, it is clear that this can be handled in the same way as the Edge-Edge case. However, there is a possible problem with the continuity constraints, namely, that there is no *a priori* reason to think that they can be satisfied for a particular type of patch. All that can be said is that if the continuity constraints are solvable then the joining problem is solvable. One way to be sure that the continuity constraints are solvable is to split patch B into two patches of the same type, such that curve *c* becomes an edge curve. If this is possible, the Edge-Curve join is possible and the problem is identical to the Edge-Edge problem.

4.3.2 Outline of an algorithm for Edge-Edge joins

It is now time to follow up the earlier suggestion that joining can be formulated as a constrained minimization problem. The objective function is a weighted sum of three penalty functions that measure various surface characteristics. One penalty function measures in some sense how "fair," or free of unnecessary bumps and wiggles, the surfaces are after the joining is completed. Another penalty function measures how much the patches being joined are distorted from their original shapes. A third penalty function measures how well the corresponding points on either side of the edge match up. The constraints are the Beta constraints discussed in Section 4.2. These are equations that, if satisfied, guarantee that the required continuity condition holds across the join. The patches are assumed to be defined by some finite number of control points, e.g., B-spline patches. These control points are the variables that the minimizer manipulates; they are not directly available to the user. Rather, the user specifies the four pieces of information needed to define the join via a graphical interactive interface, after which the joining process is automatic.

What are the penalty functions and constraint equations? I present what I think is a good way of specifying them and justify my choices. First of all, it would be desirable to specify the penalty functions and constraints without reference to the specific patch type so that the algorithm would be general purpose. This is no problem for the penalty functions but seems to be impractical for the continuity constraint equations. As shown in section 4.2, the continuity constraints are very complex. There is no practical patch type-independent way to solve them that I know of. Hence, the algorithm must be built around a patch type or types for which it is known how to solve the constraints. Furthermore, the most general form of the constraint equations were shown to involve $2n$ arbitrary functions (the Beta constrains) per edge to be joined. I make the usual assumption that these functions are constant. This assumption does cause some loss of flexibility in the resulting joined surfaces, but without it the problem has too many degrees of freedom for my proposed method to be used. A possible extension of this research would be to approximate the Beta constraint functions using finite element methods. In any event,

some technique has to be used to produce a finite number of variables for the minimization algorithm to work with.

The difficulty with the continuity constraint equations is that they must be solved exactly, at least in theory. In a computer implementation the constraint equations must be numerically satisfied to a very high accuracy. If, for example, the patch edges do not quite meet, there will be pin-holes and cracks visible in a rendered scene. These types of artifacts are very visibly objectionable. For this reason I doubt that the standard technique of converting the problem to an unconstrained minimization by turning the constraints into penalty functions would work. A high weighting factor would have to be assigned to the constraint penalty functions to get high accuracy, which might make the algorithm become numerically unstable. (I note, however, that Moreton and Sequin [ Moreton 92 ] implemented a system for filling in fair surfaces between outline curves that uses this unconstrained minimization technique. Their paper does not discuss any numerical problems or defects in the resulting joins.) Furthermore, the constraint equations must not be so restrictive that they leave no "yield" in the surface, or else the minimizer would have nothing to do, and the resulting surface might not be very fair.

We are ready to discuss how to set up the penalty functions and constraint equations. The goal is to do this in such a way that the user interface is simple and intuitive. The logical way to proceed is to organize around the four pieces of information that we need from the user. In the following discussion, assume that the user has selected two patches $c_1$ and $c_2$ to be joined along edges $E_1$ and $E_2$.

Edge matching. Suppose the two edges to be joined are parameterized as $E_1(s)$ and $E_2(s)$ for $s \in [0,1]$. Let $\phi_1(s)$ and $\phi_2(s)$ be smooth, increasing or decreasing functions from $[0,1]$ onto $[0,1]$ that are specified by the user to indicate how corresponding points on the two edges should match. Note that a decreasing function is needed to reverse the direction of one of the edges. After the joining is completed, we would ideally like to have $E_1(\phi_1(s)) = E_2(\phi_2(s))$. This requirement is too restrictive, however, as there is a danger of not leaving enough yield in the surface. Furthermore, an exact equality might not be compatible with the continuity constraint equations. Since the end goal is to

produce a seamless surface, it does not seem very important that edge matching be exact except at the end points of the edge. That is, as long as the continuity constraints are satisfied, it does not seem matter if there is some slippage *along the direction of the edge*. Finally, an exact edge match presents difficulties in the user interface – how could it be specified? For these reasons it is preferable to define a penalty function for edge matching. I define the penalty function as an integral of the deviation of the match along the edge:

$$\varepsilon_{edge} = \int_0^1 \left\| E_1(\phi_1(s)) - E_2(\phi_2(s)) \right\|^2 \frac{\left( \left\| E_1'(s) \right\| + \left\| E_2'(s) \right\| \right)}{L(E_1) + L(E_2)} ds,$$

where $L(E)$ is the arc length of edge *E*. The term $\frac{\left( \left\| E_1'(s) \right\| + \left\| E_2'(s) \right\| \right)}{L(E_1) + L(E_2)}$ is included as a weighting factor so that equal lengths of the edge curves will receive equal weight in the integral. The rationale for the weighting factor comes from the fact that integrating the magnitude of the tangent vector of a curve gives the length of the curve: $L(E) = \int_0^1 \left\| E'(s) \right\| ds$. Dividing by the curve length yields a unit weight over the curve. The factor is symmetrical between the two curves because the problem statement is symmetrical. Possibly this weighting factor is unnecessary; some experimentation is needed to decide. Note that this penalty function does not force the end points of the edge curves to meet exactly, nor does it need to, since this requirement will be enforced by the continuity constraint equations. Likewise, the continuity constraint equations keep the edge curves from pulling apart.

There remains the issue of just how the user specifies the two functions $\phi_1(s)$ and $\phi_2(s)$. Here I propose to use control points. I argued against the use of control points earlier, but that was for the direct specification of surfaces. Control points work very well for low dimensional problems, however, and this is essentially such a problem. Specifically, I propose to provide an interactive tool as shown in Figure 4.3-1.

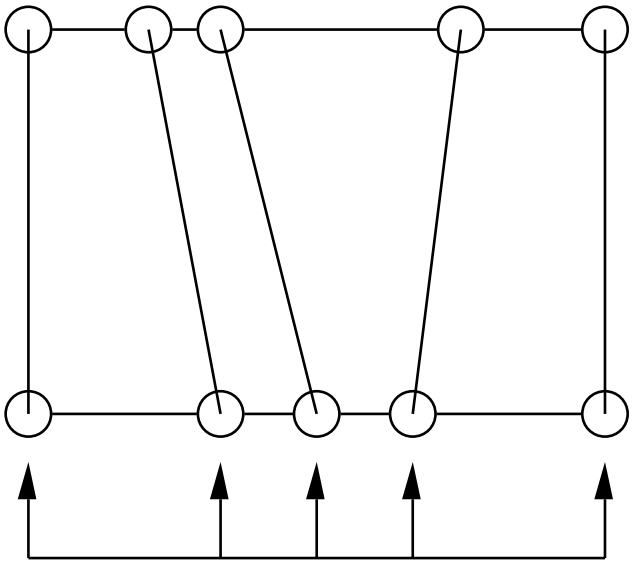



Figure 4.3-1 Edge matching tool

To use the tool the user slides the control points along the horizontal lines, which represent the parameter spaces of the two edges. The control points at the ends would not be movable. There would be a way to add or delete control points. As the control points of the tool are moved interactively, highlighted points would move correspondingly in a 3-D display of the scene reflecting the 3-D locations of the control points on the edges. The functions $\phi_1(s)$ and $\phi_2(s)$ are low (probably cubic) order B-spline functions based on these control points. Various refinements of this scheme are possible, and some experimentation would be needed to find out what works best. For example, the density of control points might be used as a weighting factor in the penalty function.

Softness. The problem here is to specify what parts of the two patches are allowed to deform and what parts are to remain fixed during the joining process. To establish the notation, suppose we are joining two patches, $c_1:[0,1]^2 \rightarrow \Re^3$ and $c_2:[0,1]^2 \rightarrow \Re^3$, and that the edges selected by the user to be joined are $E_1(s) = c_1(1,s)$ and $E_2(s) = c_2(0,s)$ as shown in Figure 4.3-2.

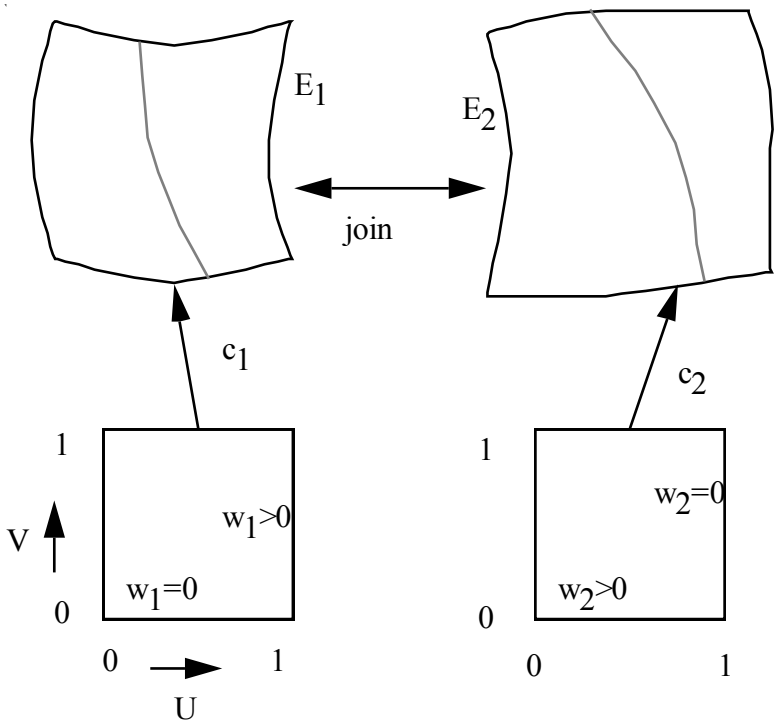



Figure 4.3-2 Softness function

Suppose the user has supplied two weight functions, $w_1:[0,1]^2 \to [0,1]$ and $w_2:[0,1]^2 \to [0,1]$. Our goal is to develop a penalty function which allows the surfaces to deform where the weights are positive and keeps the surfaces fixed where the weight is zero. The amount of deformation permitted should be roughly proportional to the weight, hence the term *softness* of the surface. The user may set one of the weight functions to zero, effectively fixing the location of one of the patches. During the joining process, the deformation should be controlled to make the resulting surface "fair", that is, in some sense the resulting surface should be as smooth as possible. We do not want unnecessary ripples and bulges to form during the bending process. Based on these considerations I define two penalty functions

$$\varepsilon_{fix} = \int_{S_1} (1 - w_1)\rho_{fix} dA + \int_{S_2} (1 - w_2)\rho_{fix} dA$$

and

$$\varepsilon_{deform} = \int_{S_1} w_1 \rho_{deform} dA + \int_{S_2} w_2 \rho_{deform} dA.$$

The function $\rho_{fix}$ measures the deviation of the patch from its original shape at a point in the patch parameter space. The function $\rho_{deform}$ is a "fairness functional" of the surface, that is, it measures the smoothness of the surface in a sense that will be defined shortly. The total penalty function is a weighted sum of the terms $\varepsilon_{edge}$, $\varepsilon_{fix}$, and $\varepsilon_{deform}$. There is some competition between $\varepsilon_{fix}$ and $\varepsilon_{deform}$, in that minimizing $\varepsilon_{fix}$ tends to preserve surface features and minimizing $\varepsilon_{deform}$ tends to remove surface features. In some applications the former may be desirable behavior, and in others the latter may be more desirable. The user-controlled weight functions $w_1$ and $w_2$ determine which behavior will dominate.

I define $\rho_{fix}$ as the square of the deviation of the patch from its original position. Thus for patch $c_1$, denoting the original patch before bending as $c_1^0$, we have

$$\rho_{fix} = \left\| c_1 - c_1^0 \right\|^2.$$

As it is stated, this definition depends on how the patch is parameterized. Eliminating this dependence is an unsolved problem. The function $\rho_{fix}$ could alternatively be defined as a measure of the deviation of the patch from its original shape, i.e., $\rho_{fix}$ could be defined such that it was not sensitive to rotation and translation. Such a definition could be based on preservation of the first and second fundamental forms of the patch, as was done for the range of action operator described in Chapter 3. I prefer the position-dependent definition of $\rho_{fix}$ for two reasons. First, I expect the user would position and orient the patches to be joined approximately before running the joining algorithm. If the patches could then move or rotate, the effect of the joining would be less intuitive as well as less controllable by the user. Second, the position-independent definition is much simpler, so it should require less computing.

The problem of defining $\rho_{deform}$ has been well-researched [ Hagen 87 ] [ Lott 88 ] [ Celnicker 91 ] [ Moreton 92 ], and several methods have been discovered that will do a

reasonable job. A common technique for generating fair surfaces is to minimize strain energy, which can be done by minimizing local curvature. This is accomplished by minimizing

$$\int_S \left(\kappa_1^2 + \kappa_2^2\right) dA,$$

where $\kappa_1$ and $\kappa_2$ are the principal curvatures on the surface. These are defined and discussed in Appendix A. The function $\kappa_1^2 + \kappa_2^2$ is the basis of Koenderink's "curvedness" measure for surfaces.

Moreton recently showed [ Moreton 92 ] that minimizing the total *variation* of curvature rather than the total curvedness seems to produce fairer surfaces. Therefore, I adopt his fairness functional

$$\rho_{deform} = \left(\nabla_{e_1} \kappa_1\right)^2 + \left(\nabla_{e_2} \kappa_2\right)^2,$$

where $\nabla_v f$ denotes the directional derivative of a function $f$ with respect to a vector $v$ and $e_1$ and $e_2$ are the principal directions (the directions in which the principal curvatures are attained, also discussed in Appendix A).

How does the user specify the weight functions $w_1$ and $w_2$ ? I propose to provide a tool by which the user can place a pair of nested rectangles in the parameter space of each patch. Figure 4.3-3 shows a possible appearance for the tool.

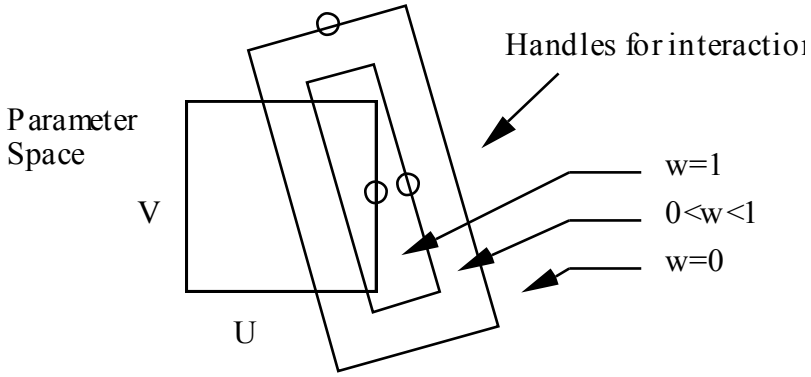


Figure 4.3-3 Softness tool

The weight function is be 0 outside the larger rectangle, 1 inside the smaller rectangle, and some smooth interpolating function in between. This is very similar to the range of action tool in the bending prototype software. Experience with the range of action tool indicates a need for a transition region where the weight function interpolates between 0 and 1. There is likely to be a noticeable seam in the final surface without this transition region. If a single rectangle doesn't give fine enough control, the tool could provide for arbitrary polygonal regions or regions bounded by smooth curves. Some experimentation will be needed to see how elaborate a tool needs to be.

Position. The position and orientation of the two patches affect the bending process via the penalty function $\varepsilon_{fix}$ which is defined in the previous section. No special tools are needed for user specification beyond the customary object space positioning tools that any 3-D modeling system would have to provide.

Continuity. The only user input needed is the value of $k$ to specify a join with $G^k$ continuity.

Putting the pieces together. The joining algorithm comprises an interface via which the user specifies the details of a join and an automatic solver that does the join. The solver has two tasks, to solve the continuity constraint equations and to minimize a penalty function while maintaining the constraints so as to produce a fair composite surface. The total penalty function is a weighted sum of the individual penalty functions defined in the previous paragraphs. The variables that the solver manipulates are the locations of the patch control points. Because of the complexity of the constraint equations, the joining algorithm will be restricted to using one or a few patch types for which it is known how to solve the constraints.

It appears be desirable that the solver operate in three stages. In the first stage an approximate solution is computed to an unconstrained minimization problem in which

the continuity constraints are replaced by penalty functions. In the second stage the constraints are solved exactly, or at least to high precision. In the third stage the full constrained minimization problem is solved. There are two reasons for this multi-stage method. First and most important, solving the constraint equations when the edges to be joined are far apart may drastically alter the shapes of the patches. Since the minimization process finds a local minimum, it might converge to a surface that has little resemblance to the original patches. Second, a constrained minimization is probably not as efficient as a unconstrained minimization. Hence, the total time will be reduced by doing part of the work as an unconstrained minimization.

I expect that there will be problems with the speed of this algorithm. I discussed in Chapter 3 that the bending algorithm is too slow on current workstations for interactive use, taking seconds to minutes to perform a single bending operation. I argued there that nevertheless, the algorithm is still useful because it can do surface shaping tasks that might take much longer or be impossible to do any other way. The penalty functions for the joining algorithm are not more complex than those of the bending algorithm, but a constrained rather than an unconstrained minimization is needed. Also, the constraints are not initially satisfied, so they have to be solved. For these reasons I expect that the joining algorithm will be slower, perhaps by a factor of 2-4, than the bending algorithm on problems of similar size. My implementation of the bending operator for a mesh of patches can be considered to be a limited form of joining, because a set of linear constraints on the patch edges has to be initially solved and maintained during the bending process. I estimate that the inclusion of the linear constraints slows down the program by 30-50%. Solving non-linear edge constraints will not be as efficient as linear edge constraints, hence the larger factor of 2-4. Unfortunately, joining problems inherently involve multiple patches and hence are likely to be bigger than bending problems, resulting in a further slowdown of the algorithm. There are three possibilities for solving the speed problem. First, it may be possible to use quadratic approximations for the penalty functions. Other researchers have achieved dramatic speed improvements for similar algorithms this way. Second, there is good potential in the algorithm for using parallelism. For example, the penalty functions get evaluated at many sample points, and

in principle this can be done in parallel. The third possibility is to simply wait a few years – the available desktop workstations still show a trend of getting much faster every year.

### 4.4 Summary and conclusions

In this chapter I presented a definition of the joining problem and an outline for an variational algorithm to solve it. There are two reasons that I thought this problem is worth addressing. First, I wanted to show by means of a non-trivial example that the optimization techniques used to solve the bending problem are quite general and applicable to a large variety of surface shaping tasks. Second, I am interested in the general problem of geometric modeling with curved surfaces, and as I explained, this problem requires the use of multiple patches, and so a way of joining patches is needed. Though there have been several recent research papers addressing special cases, nobody has yet published a fully general purpose solution. The nearest approach to date is a paper by Moreton and Sequin [ Moreton 92 ] which describes a system for filling in smooth surfaces between a set of fixed outline curves. I explain later why I think my approach could be developed into a general purpose solution with some further work.

I presented a summary of T. DeRose's definition and analysis of what geometric continuity of a join means and why it is the proper criteria for smoothness for joins. He showed that for arbitrary patches the requirement of geometric continuity leads to a system of non-linear partial differential equations that must be satisfied on patch edges and that for a large class of patch types the PDE system can be turned into an algebraic system. Even for an algebraic system, as opposed to a PDE system, the problem of solving or even setting up the equations is very hard and is an active research area in the field of geometric modeling.

The algorithm I outlined addresses the problem of joining two patches at a common edge. I argued that this is the fundamental joining problem, since the other types of joining can be reduced to this type. The user has to supply information about how the edges should match up after the join, what parts of the patches involved are allowed to deform, how the patches are positioned and oriented with respect to each other, and what degree of

geometric continuity should hold after the join is complete. This information is used to set up penalty functions to drive minimization algorithm. The main difficulty that appeared is the need to perform a constrained minimization so that the continuity constraints are met.

I believe my approach could be developed into a fairly general purpose solution except for restrictions on the patch types supported. Restrictions on patch types are due to the difficulty of setting up and solving the continuity constraints; this problem is not unique to my particular approach. To carry out further development would require addressing three problems. First, the details of handling all three types of joins, and not just edge-edge, would have to be worked out. Second, continuity constraints on vertices would have to be incorporated. Third, the problem of speed of the algorithm would have to be addressed.

# Chapter 5
# Results

This chapter describes the prototype software that I developed for the bending operator and shows some results of applying the bending operator. It comprises two main topics. First is a description of the user interface and an explanation of how the interactive controls work. Second is a series of examples showing the application of various bending operators to a single patch and to a mesh of patches.

## 5.1 Description of prototype software

The prototype software is a program that enables a user to interact graphically with the bending operator, using the X-windows system on a workstation with a color display. This program is not a complete graphics modeling system; it only provides a convenient set of functions for experimenting with the bending operator. The program creates two windows, a control panel window and a viewing window. The user directs the action of the program by manipulating widgets in the control panel using the mouse pointer. The controls are implemented using the InterViews tool kit[5]. The viewing window provides a perspective view of a 3-D scene containing the patch being manipulated. When the program is started, an initialization file is loaded that defines the patch geometry and initial values for the controls.

[5]InterViews is a public domain software package developed at Stanford University by Mark Linton.

5.1.1 The control panel

Figure 5.1.1-1 shows the control panel.

File Name : cpanel_lab.eps
Title : Control Panel with Labels
CreationDate : Fri Aug 27 13:38:38 1993
Pages : 1

Figure 5.1.1-1 Control panel

It is divided into five functional areas which are explained below. This section only summarizes the control panel; detailed instructions for using the program are in a separate programmer's manual. The functional areas are

A. mouse-sensitive widgets for viewing control
B. buttons for initiating actions and setting states
C. sliders for setting various analog parameters
D. widgets for positioning the bending operator and range of action
E. pull-down menus for selecting various options

Area A contains controls for adjusting the parameters for the viewing window. The five beveled pads are widgets that are sensitive to mouse movements and mouse button depressions. The widgets control rotation, scale, and translation of the model space, as indicated by the icons.

Area B contains push buttons that initiate some action and toggle buttons that set the state of a boolean variable. The toggle buttons have a small indicator patch which is bright for true and dim for false values. The main buttons are

1. Execute - apply the current bending operator
2. Reinit - abort the current bending operator
3. Undo - undo the last bending operation
4. Render - show a shaded surface instead of a wireframe for the patch
5. ShowCp - show the control point mesh of the patch
6. ShowGrid - show a wireframe display of the patch
7. DesNormals - show the desired normal vectors
8. RangeAct - activate the range of action positioning control
9. BendOp - activate the bending operator positioning control
10. NewBendop - make the current patch into a bending operator

The other buttons are described in later sections of this chapter where appropriate.

Area C contains sliders for adjusting parameters used by the program. The sliders are

1. Error Weight - sets the error threshold for terminating the minimization algorithm.

2. Normal/Metric - sets the relative weight assigned to the penalty functions for the desired normals and the metric. When the slider indicator is at a position $x$ units from the left on a scale from 0 to 1, the normal error gets weight $x$ and the metric error gets weight $1-x$.
3. Range of Action Weight - sets the weight multiplier of the range of action region.
4. Collapse Weight - sets the weight multiplier for the collapse penalty function.

Area D contains widgets for positioning the bending operator and the range of action with respect to the target patch. Figure 5.1.1-2 shows area D in more detail.

File Name : positioner.eps
Title : Positioner Widgets
Creator : IslandDraw for rhoades
CreationDate : Tue Aug 17 1993
Pages : 1

Figure 5.1.1-2 Positioner widgets

The large inset gray area represents the target patch parameter space. The black square in the center represents the unit square $[0,1]^2$ which is the target patch domain. The light gray (red on a color display) oblique rectangle is the positioner control for the bending

operator. The dark gray (green on a color display) oblique rectangle is the positioner control for the range of action region. Both of these controls have three small circles at the center and middles of two edges. These circles are handles that can be grabbed and moved using the mouse pointer. Moving the center handle translates the control rectangle in parameter space. Moving the side circles rotates the control rectangle or changes its size along one axis. The shape of the controls is always rectangular. A control is continuously redrawn while being manipulated. Simultaneously the image of the control in the surface patch is continuously redrawn in the viewing window to allow the user to see where the control rectangle lies in model space.

The range of action control is used to specify a weight function that determines which parts of parameter space will be included in the bending action. How the weighting function is used is described in Section 3.3. The weight function has the value 0 outside the control rectangle and the value 1 in a region around the center of the control rectangle. There is a transition region near the control rectangle in which the weight function makes a smooth transition between 0 and 1. This transition region is needed to avoid discontinuities in the penalty function. Such discontinuities are undesirable, as they cause creases, sharp changes in curvature, and other visible problems.

The bending operator positioning control is used to position, orient, and scale the bending operator patch with respect to the target patch. The positioning control can be thought of as modifying the shape of the bending operator patch; that is, the control redefines the patch function of the bending operator. The details of how this is done are complicated and have been explained in Section 3.4. The center of the control rectangle determines where the point of application of the bending operator falls in the target patch parameter space. The size and orientation of the control rectangle essentially determines an affine transformation between the domains of the bending operator patch and the target patch. The control rectangle can be thought of as the unit square in the domain of the bending operator patch.

Area E is a group of three pull-down menus for selecting which bending operator is used and for selecting parameters related to meshes of patches. Figure 5.1.1-3 shows the menu for selecting the bending operator.

File Name : bendop_menu.eps
Title : Labelled Bendop Menu
CreationDate : Fri Aug 27 13:51:06 199
Pages : 1

Figure 5.1.1-3 Bending operator menu

Six bending operators are currently implemented. They are

1. Gaussian - a Gaussian shaped bump or dent. The bending operator placement control in Area D sets the center and major and minor axes of the Gaussian. The rectangle shows approximately where the magnitude of the Gaussian falls to about 1/4 of its maximum value. The patch function for this operator is $p(u,v) = \left(u, v, he^{-u^2-v^2}\right)$, where $h$ is the height of the Gaussian. The value of h is set by the slider labeled "Gaussian Height".
2. Up Cylinder - a cylindrical bending operator. The term "Up" means that the curvature is positive, i.e., the center of the cylinder is higher than the edges as seen from the top. The bending operator placement control determines the amount of curvature and the direction of the cylinder axis. The patch function for this operator is $p(u,v) = (u, \sin v, \cos v)$.
3. Down Cylinder - a cylindrical bending operator. It is like the Up Cylinder, except that the curvature is negative, i.e., the center of the cylinder is lower than the edges as seen from the top. The patch function for this operator is $p(u,v) = (u, \sin v, -\cos v)$.

4. Twist - a twisting operator. This operator causes the patch to be twisted about an axis line in the surface. The bending operator placement control determines the direction of the axis of twist and the amount of twisting. The patch function for this operator is $p(u,v) = (u,v,uv)$.
5. Mogul - an experimental bending operator. This operator is included because it has no good quadratic approximation, to verify that the bending operator placement control is working properly. The patch function for this operator is $p(u,v) = (u,v,\cos u \sin v)$.
6. Patch - a user-defined bending operator. This operator uses a B-spline patch as the bending operator. To use this operator, the user constructs a patch, installs it using the NewBendop button, and then selects this menu item.

The menus Continuity and Weld UV are used when bending a mesh of patches. Figure 5.1.1-4 shows the Continuity menu which is used for selecting the degree of continuity for the edge joins in a mesh.

File Name : cont_menu.eps
Title : Continuity Menu
CreationDate : Fri Aug 27 13:55:13 199
Pages : 1

Figure 5.1.1-4 Continuity menu

The menu item "None" means that no continuity constraints are applied. The menu item "C n" means that during the bending process, the edges of the patches of a mesh are constrained to join with $C^n$ continuity. How this is accomplished has been discussed in detail in Section 3.5. Briefly, a constrained optimization is performed during bending, such that the continuity constraints are always satisfied. Two of the buttons in Area B are relevant when bending a mesh. Pressing the Constrain button projects the control points onto the null space of the continuity constraint equations, forcing the continuity constraints to be satisfied. This represents a crude joining algorithm. It is crude because

no penalty functions are applied to control the behavior of the joining. The Constraints toggle switch sets or clears a boolean variable that causes the constraints to be preserved during the bending process.

Figure 5.1.1-5 shows the Weld UV menu.

File Name : weld_menu.eps
Title : Weld Menu
CreationDate : Fri Aug 27 13:58:15 1993
Pages : 1

Figure 5.1.1-5 Weld UV menu

Its purpose is to force the top and bottom edges or the left and right edges of a mesh of patches to be joined together. This allows various closed shapes such as a cylinder or torus to be formed. The "U Weld" menu item causes the edges of the patch with $u = 0$ and $u = 1$ to be joined, where *u* and *v* are coordinates of the parameter space. The point $(u,v) = (0,t)$ is joined to the point $(1,t)$. The "U Reversed" menu item causes these same edges to be joined, but in the reverse direction, that is, the point $(u,v) = (0,t)$ is joined to the point $(1,1-t)$. The "U None" menu item places no constraint on these edges. There are analogous choices for the *v* coordinate. These welding operations can be applied to a single patch as well as a mesh of patches.

5.1.2 The viewing window

This window shows a 3-D perspective display of the current patch. The patch is displayed as either a wireframe or Phong-shaded surface depending on the state of Render toggle button. The wireframe display is fast enough – 10-15 updates/second – for interactive control over the viewing parameters. The shaded surface display is slower – about second

or two per update. When the BendOp toggle switch is on, an image of the bending operator positioning control appears superimposed on the surface. This display updates continuously while the bending operator control is being changed, allowing the user to see where the bending operator is positioned both in parameter space and in the object space of the patch. The same is true of the range of action control when the RangeAct toggle switch is on.

When the ShowCp toggle switch is on, the control point polygon of the current patch is shown in the viewing window. When the DesNormals toggle switch is on, the desired and actual normals are shown as a field of short line segments growing from the surface of the patch.

## 5.2 Modeling tasks handled by the prototype

This section is a demonstration of the bending operator. The purpose of this demonstration is to show the capabilities of the bending operator and to expose both its strengths and weaknesses. It begins with some examples of bending a single patch and then shows some examples using a mesh of patches.

### 5.2.1 Bending a flat sheet into a torus

This example was used at the beginning of Chapter 3 to explain the bending operator. Figure 5.2.1-1 shows a wireframe drawing of the initial flat patch. It is a B-spline patch with 7∞7 control points and degree 6∞6 (which makes it a degree 6∞6 Bezier patch). The first operation is to bend with a cylindrical bending operator with the axis parallel to the U parameter axis. The white rectangle is the image of the bending operator control rectangle, and the short white line at the center is the surface normal at the point of application. All the figures in this section were created by using the prototype software to write a Postscript file.

File Name : flat-wf-1.eps
Creator : view
Pages : 0

Figure 5.2.1-1 Flat patch with bending operator

Figure 5.2.1-2 shows the same patch after computing the desired normals but before the bending is actually performed. The actual and desired normals are indicated by short white vectors. The vectors are positioned at the points used to do the numerical integration of the penalty functions. There are 81 sample points in this example, which is enough to give good accuracy with a patch of this size, in the absence of very tiny features in the target patch or the bending operator patch. The number of sample points can be set in the initialization file.

File Name : flat-wf-dnc-1.eps
Creator : view
Pages : 0

Figure 5.2.1-2 Flat patch with original and desired normal vectors

Figure 5.2.1-3 shows the result of applying a cylindrical bending operator. Since the original patch was flat, the target patch after bending takes on the shape of the bending operator patch. In this case, the desired normals were attained almost exactly, since the metric penalty function did not contribute significantly to the total penalty function. This is because a cylinder and a flat patch are isometric. The next bending operation is another cylindrical bend, but with the cylinder axis rotated 90 degrees.

File Name : cyl-wf-1.eps
Creator : view
Pages : 0

Figure 5.2.1-3 Result of cylindrical bend

Figure 5.2.1-4 shows the desired original and desired normals resulting from applying the second cylindrical bending operator, and Figures 5.2.1-5 and 5.2.1-6 show a wireframe view and a shaded surface view of the result of the bending operation.

File Name : cyl-wf-dnt-1.eps
Creator : view
Pages : 0

Figure 5.2.1-4 Original and desired normal vectors for second bending via cylinder

File Name : torus-wf-1.eps
Creator : view
Pages : 0

Figure 5.2.1-5 Result of second bending via cylinder, wireframe

File Name : torus-sd-1.eps
Creator : view
Pages : 0

Figure 5.2.1-6 Result of second bending via cylinder, shaded

Figures 5.2.1-7 and 5.2.1-8 show the same surface with the actual and desired normals visible.

File Name : torus-wf-dn-1.eps
Creator : view
Pages : 0

Figure 5.2.1-7 Torus with original and desired normals shown

For this bending operation I set the metric/normal factor to 0.1, so that the normal penalty function contributes ten times as much as the metric penalty function to the total penalty function. This bending definitely changes the metric of the patch. The desired normals could not be attained exactly, although the discrepancy can barely bee seen in Figure 5.2.1-8. Section 5.2.4 discusses how varying the metric/normal factor affects the bending operator.

File Name : torus-wf-dn-cls-1.eps
Creator : view
Pages : 0

Figure 5.2.1-8 Torus with original and desired normals, shown close up

5.2.2 Making dents and bumps

I illustrate the use of the Gaussian bending operator in this section. The effect of this operator is to place a bump (or dent) in the surface. It is possible to produce bumps by direct manipulation of patch control points, but their size and shape is determined by the spacing of the control points. The Gaussian bending operator gives the user much better control over the size and shape of the bump. The prototype software does not perform automatic subdivision of the patch control point mesh, so there is a limit to how small features can be in the bending operator. Figures 5.2.2-1 and 5.2.2-2 show the result of bending a flat patch with 8∞8 control points and degree 5∞5 using a Gaussian bump. The image of the bending operator control is shown in Figure 5.2.2-1. Note that the image of the control rectangle has slightly curved sides. This is because straight lines in parameter space do not, in general, map into straight lines in model space for B-spline patches.

File Name : flat-bog-2.eps
Creator : view
Pages : 0

Figure 5.2.2-1 Flat patch with image of bending operator control

File Name : gauss-wf-2.eps
Creator : view
Pages : 0

File Name : gauss-sd-2.eps
Creator : view
Pages : 0

Figure 5.2.2-2 Flat patch bent by Gaussian bump

Figure 5.2.2-3 shows the desired normals versus actual normals for the lower left corner of this patch.

File Name : gauss-wf-dn-2.eps
Creator : view
Pages : 0

Figure 5.2.2-3 Desired normals for Gaussian bump

The actual and desired normals are close together near the edge and far apart (about 20 degrees) on the steeply curved side of the Gaussian bump. This is the result of the competition between the normal penalty function and the metric penalty function. The original flat patch has to be stretched considerably to add the bump, and the metric penalty function resists this stretching. Therefore, the bump is not as high as it should be to match the desired normals. For Figure 5.2.2-2 the metric/normal factor was set to 0.2.

Figure 5.2.2-4 shows the results of bending a flat patch twice, first with an elongated Gaussian bump, and second with a Gaussian dent, with the long axis in a different direction. The resulting surface resembles a difference of Gaussians, as expected.

File Name : gauss-wf-bd-2.eps
Creator : view
Pages : 0

File Name : gauss-sd-bd-2.eps
Creator : view
Pages : 0

Figure 5.2.2-4 Bending twice with different Gaussians

Figure 5.2.2-5 shows the result of placing a Gaussian dent in a cylinder. The cylinder was modeled with an 8∞8 B-spline patch with degree 7∞7. The $v$=0 and $v$=1 edges of the patch were constrained to be joined during the bending process.

File Name : dcyl-wf-2.eps
Creator : view
Pages : 0

File Name : dcyl-sd-2.eps
Creator : view
Pages : 0

Figure 5.2.2-5 Gaussian dent in a cylinder

Figure 5.2.2-6 shows the control point mesh superimposed on the resulting patch on the left and the control point mesh by itself on the right. Since this patch is of fairly high degree, the relation between the patch control points and the surface is not very intuitive. It would be practically impossible to produce this surface by direct manipulation of the control points.

File Name : dcyl-wf-cp-2.eps
Creator : view
Pages : 0

File Name : dcyl-cp-2.eps
Creator : view
Pages : 0

Figure 5.2.2-6 Gaussian dent in cylinder, with control points

One of the problems with the bending algorithm is "edge effects." Figures 5.2.2-7 and 5.2.2-8 show a Gaussian bump placed near the edge of a flat patch. Notice how the edge of the patch near the bump tends to pull inward toward the bump, giving the figure a lop-sided appearance. This problem is quite noticeable in the top view of Figure 5.2.2-8.

File Name : gauss-wf-edg-2.eps
Creator : view
Pages : 0

File Name : gauss-sd-edg-2.eps
Creator : view
Pages : 0

Figure 5.2.2-7 Gaussian bump near edge

File Name : gauss-wf-tve-2.eps
Creator : view
Pages : 0

File Name : gauss-sd-tve-2.eps
Creator : view
Pages : 0

Figure 5.2.2-8 Gaussian bump near edge, top view

5.2.3 Examples of other bending operators

There are two other built-in bending operators, the twist and the mogul. The mathematical formula for these are given in Section 5.1.1. The purpose of the twist operator is to perform twisting, i.e., to cause the surface normals to turn an amount proportional to the distance along a twisting axis. The direction of turn is perpendicular to the twisting axis. Figure 5.2.3-1 shows the twist operator, which is a hyperbolic paraboloid, applied to a flat patch. The mogul bending operator is not useful as a shaping tool; it is included in the prototype software as a testing tool for the bending operator positioning tool.

File Name : twist-wf.eps
Creator : view
Pages : 0

File Name : twist-sd.eps
Creator : view
Pages : 0

Figure 5.2.3-1 The twisting operator

5.2.4 Effect of the metric/normal factor.

The metric/normal factor is an adjustable parameter of the bending algorithm that determines the relative importance of the normal penalty function and the metric penalty function in computing the total penalty function. As mentioned in section 5.2.1, this factor is a number $x$ between 0 and 1 such that the metric penalty function is multiplied by weight $x$ and the normal penalty function is multiplied by weight $1-x$. I have found by experimentation that a value of about 0.2 usually produces good results. The bending algorithm is not very sensitive to this parameter. Large values tend to make the patch act

as if it were stiff piece of paper, i.e., the target surface before and after bending are roughly isometric. Small values lead to problems with the patch changing size during the bending process and a sort of "flowing" of surface features within the surface. In this section I demonstrate both of these effects. I decided that the metric/normal factor should be adjustable by the user, because for some bending operations the user may want to tightly constrain the normal vectors and for other bending operations the user may want to avoid the "flowing" effect.

Figure 5.2.4-1 shows bending a cylinder into a torus with the metric/normal factor set to 0.9. This means that emphasis is placed on maintaining the metric of the patch.

File Name : torus-wf-stf-3.eps
Creator : view
Pages : 0

File Name : torus-sd-stf-3.eps
Creator : view
Pages : 0

Figure 5.2.4-1 Cylinder to torus bend with metric/normal factor of 0.9

Most of the curvature seems to be concentrated into four diagonal lines that connect the mid-points of the edges. The corners and center are relatively flat. I think that the patch is approximating a shape that is isometric to a flat patch with the smallest possible normal vector deviation from that of a torus. You may want to try the following experiment. Take a piece of paper and bend it into a half-cylinder. Then try to bend again perpendicular to the cylinder axis. You will notice that the shape you get is very similar to Figure 5.2.4-1. This is because paper has the property of strongly resisting changes to its metric while being easy to bend.

Figure 5.2.4-2 shows the same bending operation, but the metric/normal factor set to 0.6. Notice that the lower edge curve is slightly more curved in the center than at the corners. This is another example of an edge effect. The metric penalty function appears to make the target patch more "stiff" near the corners. Actually, wireframe displays can be misleading for judging the results of the bending operator. The curves making up the display are parameter curves in the surface, and their curvature within the tangent plane of the surface (i.e., geodesic curvature) has no significance to the surface geometry.

File Name : torus-wf-mst-3.eps
Creator : view
Pages : 0

File Name : torus-sd-mst-3.eps
Creator : view
Pages : 0

Figure 5.2.4-2 Cylinder to torus bend with metric/normal factor of 0.6

If the metric/normal factor is set very low, the actual and desired normals will be very close, but there may be an undesirable "pulling in" effect on the edges. This effect is illustrated in figure 5.2.4-3 by a Gaussian bump, in which the metric/normal factor is 0.025.

File Name : gauss-wf-lnm-2.eps
Creator : view
Pages : 0

File Name : gauss-sd-lnm-2.eps
Creator : view
Pages : 0

Figure 5.2.4-3 Gaussian bump with very low metric/normal factor

5.2.5 The range of action control

The previous figures were made with the range of action control turned off. In this section I demonstrate the effects of this control. This range of action control basically causes the normal penalty function to be weighted so that there is no contribution to the total penalty function outside the range of action area. The main use of this control is to exclude parts of the target patch from the bending action. The effect of this control is explained in detail in Chapter 3. Figure 5.2.5-1 shows the result of bending a flat sheet with a cylindrical bending operator in conjunction with the range of action control. The range of action was limited to the left side of the patch. Figure 5.2.5-2 shows the state of the positioning control widget. The large dark gray rectangle shows the range of action area in parameter space. Figure 5.2.5-3 shows the desired normals superimposed on the result of the bending. The desired normals are computed for the entire patch but are ignored outside the range of action. That is why the discrepancy between the actual and desired normals is large on the right side of Figure 5.2.5-3.

File Name : cylrga-wf-4.eps
Creator : view
Pages : 0

File Name : cylrga-sd-4.eps
Creator : view
Pages : 0

Figure 5.2.5-1 Cylindrical bend with range of action

File Name : cylrga-posw-4.ep
Title : /tmp_mnt/net/molly/r
Creator : IslandDraw for rho
CreationDate : Wed Aug 18
Pages : 1

Figure 5.2.5-2 Positioning control for bend with range of action

File Name : cylrga-wfdn-4.eps
Creator : view
Pages : 0

Figure 5.2.5-3 Normals for cylindrical bend with range of action

Although the right side of the patch is completely outside of the range of action area, there is a small amount bending there. This seems to happen because there is a lesser contribution from the metric penalty function in this case than if the right side were perfectly flat. Another use of the range of action control is to limit which part of a complicated bending operator gets used. For example, a bending operator patch constructed by the user is well defined only within the unit square in parameter space. The range of action control can be used to ensure that only the well defined part of the bending operator gets applied to the target surface.

5.2.6 Bending a mesh of patches

The problem of extending the bending operator to a mesh of patches is discussed in Section 3.5. The key idea is to treat the mesh as a single large pseudo-patch. This is possible if there is no branching or looping of the 2-D sheets in the mesh. As a practical test of this idea, I implemented the bending operator for a simple rectangular array of B-spline patches. The main complication is the need to keep the edges of the patches joined together with a user-specified degree of continuity during the bending process. The conditions for the edges to be joined can be formulated as a set of continuity constraint equations on the patch control points. The bending process becomes a constrained optimization with the continuity equations as the constraints. I implemented $C^n$

continuity for $n$ = 0, 1, or 2. The prototype software determines that it is working with a mesh instead of a single patch when reading the initialization file.

I present some examples of bending a mesh of patches. In this example the mesh is a 2∞2 array of cubic Bezier patches. I show the results of bending with a Gaussian bump for several degrees of continuity. Figure 5.2.6-1 shows the result of bending with no continuity constraints. The individual patches break apart. Figure 5.2.6-2 shows the result of bending with $C^0$ continuous joins. This means that the edges stay joined during the bending but there is no constraint on the angle with which the edges join. Thus in the example, discontinuities of shading are visible. Figure 5.2.6-3 shows the result of bending with $C^1$ continuity. This implies a smooth normal vector field across the edge joins, so there is no discontinuity in the shaded images. Note that, with respect to the individual patches of the mesh, the Gaussian bump was right on some edge. However, there is no evidence of the edge effects that appeared when bending single patches. My theory is that edge effects occur because there is less "material" near edges and corners, hence less contribution to the penalty functions that control the shape. That is, edges and corners are less constrained than the center of a patch, hence more sensitive to the action of the bending operator. The lack of edge effects at interior edges of a mesh supports this theory.

File Name : mesh-wf-n.eps
Creator : view
Pages : 0

File Name : mesh-sd-n.eps
Creator : view
Pages : 0

Figure 5.2.6-1 Mesh with no edge continuity

File Name : mesh-wf-0.eps
Creator : view
Pages : 0

File Name : mesh-sd-0.eps
Creator : view
Pages : 0

Figure 5.2.6-2 Mesh with $C^0$ continuity

File Name : mesh-wf-1.eps
Creator : view
Pages : 0

File Name : mesh-sd-1.eps
Creator : view
Pages : 0

Figure 5.2.6-3 Mesh with $C^1$ continuity

It is possible to make closed figures by applying the continuity conditions to opposite edges of a single patch. In this case, the program is actually operating on a mesh of patches that contains only one patch. As discussed in Section 5.1.1, the prototype software can join the U or V parameter edges either directly or in a reversed sense. This means that it is possible in the prototype to construct several geometric shapes including a cylinder, a torus, a Möbius strip, and a Klein bottle. The cylinder shown in Section 5.2.2 was made this way. I show now how to use the bending operator and continuity constraints to construct a torus. Step 1 is to construct a cylinder. This is done by bending

a flat sheet with the cylindrical bending operator by slightly less than 360 degrees. This leaves two opposite edges almost adjacent, as shown in Figure 5.2.6-4.

File Name : cyl-open-sd.eps
Creator : view
Pages : 0

File Name : cyl-clsd-sd.eps
Creator : view
Pages : 0

Figure 5.2.6-4 Closing the cylinder

File Name : torus-open-sd.eps
Creator : view
Pages : 0

File Name : torus-clsd-sd.eps
Creator : view
Pages : 0

Figure 5.2.6-5 Closing the torus

Then enforcing a $C^1$ constraint does the join. This provides a crude joining capability in the prototype software. The cylinder is then bent using a cylindrical bending operator with the axis turned 90 degrees. The bending operations are actually the same as those shown in section 5.2.1, except that the bending angles are greater. When the ends of the cylinder almost touch, as shown in Figure 5.2.6-5, a $C^1$ constraint is imposed on the edges, causing a closed torus to be created.

5.2.7 Custom bending operators

Any patch can, in principle, be used as a bending operator. This leads to the idea of the user constructing his own building operator. The minimum requirements on the bending operator patch were discussed in Section 3.6. Basically any patch for which it is possible to take first and second partial derivatives is satisfactory. B-spline patches certainly meet this requirement. There is only one difficulty – because of the affine transformation involved in the positioning tool, it is possible that the bending operator patch will need to be defined on a larger domain than the unit square in parameter space. This difficulty can be handled by using the range of action tool. The idea is to construct the custom bending operator with a "buffer zone" around the part that is to be used, and to limit the bending action with the range of action operator to avoid using parts of the bending operator patch outside this buffer zone.

In the next few figures, I show a simple example of constructing and using a custom bending operator. Figure 5.2.7-1 shows the bending operator patch. It was made by using two cylindrical bends of opposite curvatures, limited by a range of action. It has an S-shaped profile.

File Name : wav-cb-wf.eps
Creator : view
Pages : 0

File Name : wav-cb-sd.eps
Creator : view
Pages : 0

Figure 5.2.7-1 Custom bending operator

This bending operator was applied to a target patch, shown in Figure 5.2.7-2, which is a section of a cylinder. The result of the bending operation is shown in Figure 5.2.7-3. The effect is to impose a longitudinal "wave" on the cylinder.

File Name : cyl-cb-wf.eps
Creator : view
Pages : 0

File Name : cyl-cb-sd.eps
Creator : view
Pages : 0

Figure 5.2.7-2 Original target patch

File Name : bcyl-cb-wf.eps
Creator : view
Pages : 0

File Name : bcyl-cb-sd.eps
Creator : view
Pages : 0

Figure 5.2.7-3 Target patch after bending

## 5.2.8 Making a spoon

To demonstrate a more complicated and realistic example of the bending operator, I made a model of a spoon. I started with a degree 5∞9 Bezier patch which was flat but had the right outline, shown in Figure 5.2.8-1. The prototype program doesn't have the tools for making such patches, as it is not a complete modeling system, so I made the patch offline and loaded it into the program via the initialization file.

File Name : spoon.flat-1.eps
Creator : view
Pages : 0

Figure 5.2.8-1 Spoon blank before bending

I used three bending operations to make the spoon that is shown in Figure 5.2.8-2. First, I used a Gaussian dent to make the bowl. Second, I used a cylindrical bend with a limited range of action to set the angle between the bowl and handle. Third, I used another cylindrical bend to slightly curve the handle of the spoon along its long axis. The entire operation took me less than ten minutes, not counting making the initial flat patch.

File Name : spoon.wf-1.eps
Creator : view
Pages : 0

File Name : spoon.sd-1.eps
Creator : view
Pages : 0

Figure 5.2.8-2 Completed spoon

**5.3 Summary**

This chapter demonstrates the variety of the surface shaping tasks the bending operator can do. The bending operator positioning tool allows a bending operator to be translated, scaled, and rotated. The range of action operator allows a bending operator to be restricted to a selected part of the target patch. Thus quite complex surface shaping

operations are possible with only a few bending operator functions. Nonetheless, the user can construct custom bending operators. I demonstrated that the bending operator can be applied to meshes of patches. The bending operator has some difficulties with edge effects. Edge effects are manifested by a different action of a bending operator near edges and corners of a patch. The problem is more annoying than crippling, as the bending operator still does approximately the right thing. To an extent, edge effects can be traded off for patch "stiffness" by changing the metric/normal factor. Edge effects don't occur for interior edges of a mesh, which suggests a remedy that may work in some cases. Namely, surround the target patch with some more patches, do the bending, then remove the extra patches.

# Chapter 6
# Conclusions

The key result in this dissertation is a new concept of what it means to bend a curved surface. I call this new concept the *bending operator*. I showed that the bending operator has a sound mathematical basis, and I used it to develop a new algorithm for shaping curved surfaces. This algorithm is capable of performing a wide variety of surface shaping tasks, including not only bending around a cylinder axis, but also indenting, twisting, and embossing. The bending operator is intended for use as an interactive tool within a geometric modeling system based on surface patches. I implemented the algorithm in an interactive prototype program to demonstrate its feasibility and to investigate its strengths and weaknesses. Chapter 3 discusses the bending operator in detail, and Chapter 5 presents a description of the prototype software and examples of results from this software.

The second main result in this dissertation is an investigation of the generalization of the variational approach used in bending to the problem of smoothly joining surface patches. I presented in Chapter 4 a classification scheme for joining problems, and I showed that a large class of such problems can be reduced to what I call the *Edge-Edge joining problem*. The Edge-Edge joining problem is the problem of connecting two surface patches smoothly along a common edge. The correct mathematical definition of "smoothly joining" turns out to be what is called *geometric continuity* in the geometric modeling literature. I presented a definition and some analysis due to DeRose which shows that geometric continuity for an edge-edge join is equivalent to a non-linear partial differential system of equations being satisfied on the edge curves. I presented an outline of a new Edge-Edge joining algorithm which is based on the variational techniques used in the bending algorithm.

## 6.1 The problem

Current geometric modeling systems for objects with curved surfaces are typically based on the use of meshes of surface patches. Such systems use one or a few types of surfaces patches, such as Bezier patches, B-spline patches, or Non-Uniform Rational B-spline patches (NURBS). The geometry of the surface patches is determined by a (usually small) finite number of *control points*. The characteristics of such systems are discussed in more detail in Chapter 1. The most simple form of interaction is to make the control points directly accessible to the user. Changes to the surface can be made interactively by moving the control points. Direct manipulation of control points, however, is not a good user interface technique. For complex models containing many patches there are too many control points, and the effects of moving control points on the surface are too limited and too subtle. Smooth joining of patches requires complex constraints on the control points that cannot be satisfied by direct manipulation. In short, control points can be thought of as the assembly language of surface description.

Current modeling systems often provide more intuitive, higher level methods for surface shaping. Design of such shaping tools has been a very active and successful area of research in the past few years. Chapter 2 presents a survey of the current state of the art of these tools. This tool-based approach to modeling, besides making the user's job easier, also allows a clean separation between the user interface and the underlying surface representation. A recent trend is to design the tools to be independent of the patch representation. Shaping tools can be thought of as a higher level language for surface description. Just as we should not care which particular underlying machine language is being used when programming in a higher level language, we should not care what particular underlying patch type is being used when using surface shaping tools.

## 6.2 The bending operator: A new tool for surface shaping

I claim that the bending operator is a new surface shaping tool that significantly advances the state of the art. Recall from Chapter 1 the design goals for the bending operator. First,

it should be intuitive, meaning that the user can easily learn to predict the result of a bending operation without needing a detailed understanding of the tool. Second, it should be high-level, meaning that the user can specify in a direct and simple way what action should be performed on the surface, without being concerned with control points or patch boundaries. Third, it should be general, meaning that it can be applied to a wide variety of patch types. In this section I show that the design goals have been met, and furthermore, I show that that the bending operator is capable of performing a wide variety of surface shaping tasks. However, the bending operator does have some problems, which I point out later in this section.

### 6.2.1 A new concept of what bending is

The key idea of the bending operator is that bending a surface is equivalent to changing its normal vectors. Therefore, to specify a bending operation, we can specify how we want the normal vectors to change during the bending. There are a lot of normal vectors, however, so we need some succinct way of specifying how they should change. This specification is provided by means of another surface, which I call the *bending operator surface*. The important thing about the bending operator surface is not its actual shape, but rather the way its normal vectors change with respect to movements in the surface.

### 6.2.2 How the bending operator works

A simple example of the bending operator in action may help to clarify the reader's understanding. Section 3.1 presents a detailed example of bending a piece of a cylinder into a piece of a torus. An abbreviated version of that example follows. The goal is to bend the target surface, the cylinder in Figure 6.2.2-1, into the final bent surface, the torus in Figure 6.2.2-2.

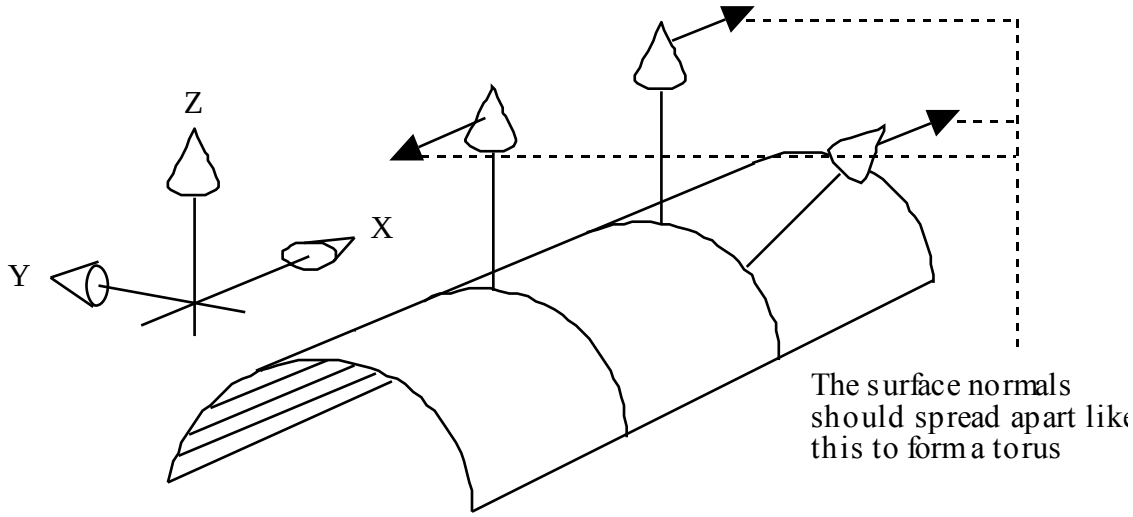



Figure 6.2.2-1 Original target surface

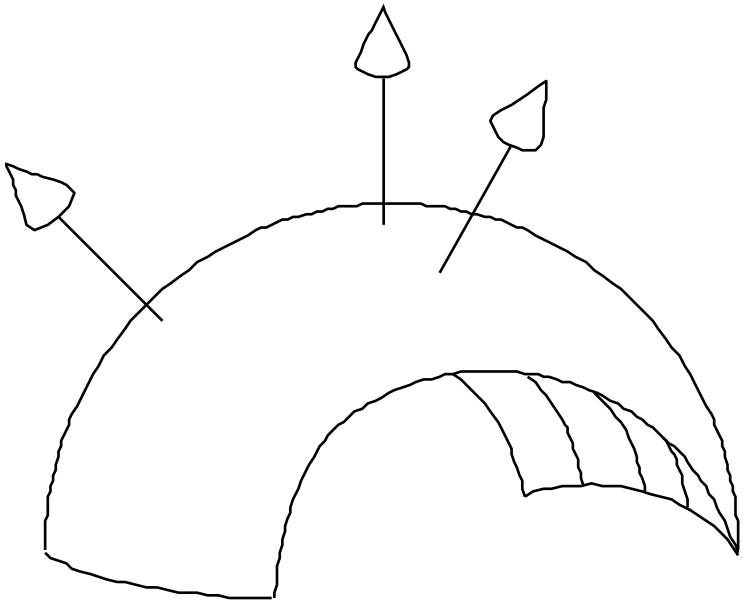


Figure 6.2.2-2 Final bent surface

Note what is different about the normal vector fields between these two surfaces. In the original target surface the normal vectors do not change for movements in the surface

along the cylinder axis (X-axis). In the final bent surface the normal vectors turn in the direction of movement for movements along the corresponding direction. For movements transverse to the cylinder axis (Y-axis) the normal vectors turn at the same rate in both the original target surface and final bent surface. Therefore, to get from Figure 6.2.2-1 to Figure 6.2.2-2, the normal vectors need to be spread apart along the cylinder axis. What surface has a normal vector field that describes this change? Another cylinder, turned 90 degrees, so that its cylinder axis is aligned with the Y-axis, as shown in Figure 6.2.2-3.

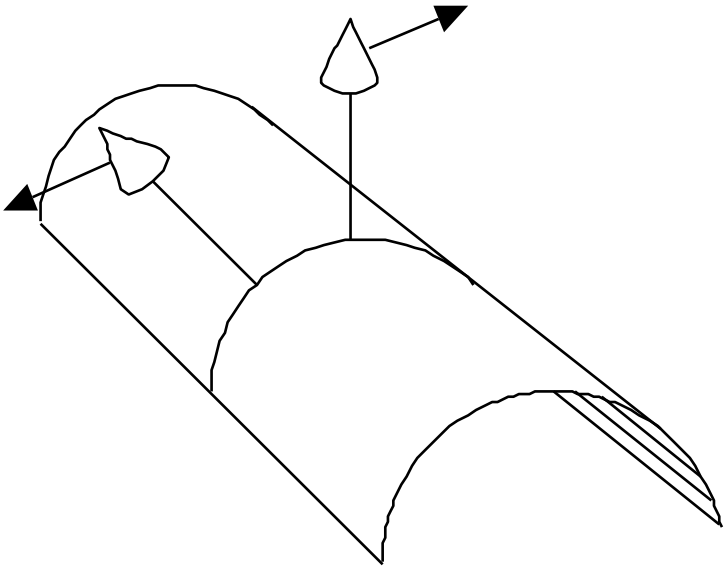


Figure 6.2.2-3 Bending operator surface

The bending algorithm adds the variations of the normal vector field of the bending operator surface to the variations of the normal vector field of the target surface to get the variations of the normal vector field of the final bent surface. Mathematically, this comprises adding the derivatives of the two normal vector fields and then integrating this sum to get the desired normal vector field of the final bent surface. The formulas and the proof that this method makes sense are given in Section 3.3.1.

The bending algorithm is a two stage process. In the first stage the normal vector field for the final bent surface is computed, as described above. In the second stage the original

target surface is warped (by varying its control points) to minimize a weighted sum of penalty functions. This composite penalty function, described in Section 3.3.2, has two main terms. The first is a normal penalty function that penalizes deviations between the actual surface normals and the desired surface normals from the first stage. The second is a metric penalty function that penalizes shearing and stretching (deviations of the metric) of the surface. This metric penalty function is needed because the normal penalty function alone does not fully constrain the surface; i.e., there may be multiple surfaces with the same desired normal vector field. There are additional penalty functions that are used to implement a range of action control and to resist folding and creasing of the surface; these are described later in Section 6.2.6. The movement of the control points to reshape the surface is handled completely by the bending algorithm; the user need not even be aware of the existence of control points. The final surface produced by the algorithm will in general not have exactly the desired normals from the first stage, because there is competition between the two terms of the penalty function, but the actual normals will generally be fairly close to the desired normals.

### 6.2.3 What the bending operator can do

Intuitively, the behavior of the bending operator is to superimpose features of the bending operator surface onto the target surface, while maintaining the general shape of the original target surface. The following figures, produced by the prototype program, shows an example of this. The original target surface, Figure 6.2.3-1, is a piece of a cylinder, and the bending operator surface, Figure 6.2.3-2, is a "wave" shaped surface. Figure 6.2.3-3 shows the result of the bending operation: a wave is superimposed on the cylinder.

| | |
|---|---|
| File Name : cyl-cb-wf.eps | File Name : cyl-cb-sd.eps |
| Creator : view | Creator : view |
| Pages : 0 | Pages : 0 |

Figure 6.2.3-1 Original target patch

File Name : wav-cb-wf.eps
Creator : view
Pages : 0

File Name : wav-cb-sd.eps
Creator : view
Pages : 0

Figure 6.2.3-2 Bending operator surface

File Name : bcyl-cb-wf.eps
Creator : view
Pages : 0

File Name : bcyl-cb-sd.eps
Creator : view
Pages : 0

Figure 6.2.3-3 Final bent surface

The power of the bending operator derives from the fact that the bending operator can be any surface. A bending operator with a complex shape will produce a complex change of shape, but the change is predictable – the features of the bending operator will be transferred to the target surface. The relation between the bending operator surface and the target surface is not symmetric because the metric of the bending operator surface is ignored. The bending operator surface is simply a concise representation of the desired shape change, as indicated by its normal vector field. With the prototype software the user can use the bending operator to make some shape, then install that shape as the next bending operator. As a practical matter, simple bending operators, such as cylinders, twists, and Gaussian bumps and dents, are used to incrementally reshape the target surface toward the user's desired final shape, with visual feedback after each step. If the result of a step is not quite right, the user can undo the operation and try again.

As a more realistic modeling example, I reproduce from Chapter 5 a model of a spoon which I made using the prototype software, shown in Figure 6.2.3-4. Starting with a flat B-spline patch, I used only three bending operations: a Gaussian dent to make the bowl of the spoon, and two cylindrical bends to shape the handle. The prototype software contains a range of action control to limit the bending action to a part of the model. This control was used to limit the bending of the handle to a region near the bowl of the spoon. It took me less than ten minutes to produce this model. I asked another person who was familiar with the theory of bending operator, but who had actually used the prototype software for less than an hour altogether, to reproduce the spoon model. He needed about 15 minutes.

File Name : spoon.sd-1.eps
Creator : view
Pages : 0

Figure 6.2.3-4 Simple spoon model

6.2.4 Strengths and weaknesses of the bending operator

Strengths. I showed in Chapter 5 and in Section 6.3.3 above that the bending operator is intuitive and high level. I pointed out that because the bending operator can be any surface, and because the range of bending action can be limited to part of the target surface, the bending operator is powerful. Examination of the equations for the penalty function in Section 3.3.2 shows that only first and second partial derivatives of the target surface and bending surface are used. The optimization algorithm that determines the final surface operates by varying the locations of the control points. Therefore, the bending operator can be applied to any patch type that is twice piecewise differentiable and described by a finite set of control points. These restrictions are very light, and virtually all of the patch types in common use satisfy them. Furthermore, I demonstrated in Chapter 3 that the bending algorithm can be applied to a mesh of patches, if the mesh has no branching sheets or loops. Therefore, the bending operator is general purpose. The current implementation of the bending operator requires patches defined by control points, but I discussed in Section 3.5.1 a possible extension to further generalize the algorithm to arbitrary $C^2$ patches.

Weaknesses. The bending operator has some problems, however. These are discussed in detail in Chapters 3 and 5. My implementation of the bending operator is slow. A single bending operation takes seconds to minutes on current generation workstations, such as the HP-730. Most of the running time is spent performing the minimization of the penalty function, and this penalty function is quite complicated and non-linear. The bending operator has problems with edge effects. This problem is manifested by asymmetrical effects near edges and corners; i.e., the same feature in the bending operator will produce different results at the center of a patch than at the edge. Examples are shown in Chapter 5. Finally, the bending operator can only be applied to surfaces that can be flattened out into a single sheet. This restriction is discussed in detail in Section 3.7. This means that the bending operator is primarily useful for working with the simple parts of a complex 3-D model. I discuss later in this chapter a possible way of removing this restriction.

### 6.2.5 Variational techniques for surface shaping

The bending operator is a new example of a variational technique for surface shaping. The basic idea of the variational method is to define a penalty function that measures numerically how much a surface differs from some shape goal. An optimization algorithm is then performed to vary the surface to minimize the penalty function. Variational methods for implementing surface shaping tools have gained in popularity in recent years; several examples are discussed in Chapter 2. My joining algorithm can best be viewed as an extension of the work of Welch and Witkin [ Welch 92 ], who are developing an interactive modeling system with which a user can change the shape of a surface by placing "handles" on the surface and manipulating the handles. The handles they describe are point and curve constraints that the surface has to satisfy. They minimize a penalty function on the surface that measures deviation from "fairness" or smoothness. Their algorithm can also minimize the deviation of the surface from a specified "rest" shape. My bending operator extends their work because it is more general than their handle constraints. It is more general because it exercises control over the shape of the whole surface, not just a point or curve in the surface. On the other hand, the bending operator cannot be used to explicitly force the surface through a particular point or curve, so the two approaches can be thought of as complementary.

### 6.2.6 Key issues in the design of the bending operator

I discuss in this section the important design decisions that I had to make during the development of the bending algorithm. In designing mathematical algorithms there are tradeoffs that have to be made between generality and complexity. One of my design goals is to allow any surface to be used as a bending operator. This goal had a major effect on design of the positioning control and the range of action control, which are the first two topics of this section. There are also issues of accuracy and robustness that are forced on the designer by the nature of the algorithm. These issues led me to address the last three topics of this section, making the algorithm incremental, making the target surface resistant to collapse, and making the penalty functions independent of patch parameterization.

Positioning the bending operator. The bending operator derives much of its power from the fact that the bending surface can be *any* surface. This flexibility comes at some cost in the complexity of the design. One aspect of this cost is the need to develop a general method of positioning an arbitrary bending operator with respect to the target patch. For any particular bending operator, such as a cylinder or Gaussian dent, it is fairly easy to construct a positioning control that works for that bending operator. But this piecemeal approach would cause adding a new bending operator to require significant coding changes to the bending operator software. I chose instead to develop a general approach that worked uniformly for all bending operators. One advantage is that adding new bending operators to the prototype software is easy – a prototype function for the new bending operator needs to be written and a menu item needs to be added. A more important advantage is that the general approach permits the user to design custom bending operators. In the prototype software, the user can at any time install the current target patch as a bending operator. The details of this method are discussed in Section 3.4 and outlined in the next paragraph. The execution time penalty of this general solution is small, about 6-7 percent of the total running time for the bending algorithm.

Positioning of the bending operator is performed by the user via an interactive set of controls, explained in Section 5.1.1. The user chooses a point of application, i.e., a point in the target surface about which the bending will occur, and then scales and rotates the bending surface about this point to determine what parts of the target surface will be affected by what parts of the bending operator surface. The correspondence between the target surface and the bending surface is determined by the surface parameterizations. Mathematically, both the target surface and the bending surface are defined as patches, that is, functions from the unit square in $[0,1]^2 \subseteq \Re^2$, called *parameter space*, to $\Re^3$, called *model space*. A point in the target surface and a point in the bending surface correspond if they are the image of the same point in parameter space. The effect of the positioning control is to redefine the patch function of the bending surface. There are two parts to this redefinition. The first part is an affine transformation of the parameter space of the bending surface so that correspondence of points between the bending and target surface is what the user desires. The second part is a rotation of the normals of the bending surface in model space. This rotation is necessary because changing the parameterization of the bending surface does not affect the directions of its normal vectors. Section 3.4 gives the mathematical details.

Range of action control. The range of action control permits the user to restrict the action of the bending operator to a selected subset of the target surface. This capability greatly increases the utility of simple bending operators, such as cylinders and Gaussian bumps and dents. It also makes possible the use of custom user-designed patches as bending operators. A custom patch might not be well defined on the whole parameter space of the target patch, or it might have features that the user wants to ignore. The range of action control can pick out only the part of a custom patch that the user wants to use.

Implementing the range of action control required some modifications to the penalty functions and an additional interactive user interface for specifying the range of action region. The interactive control is a rectangle in patch parameter space that the user can move, rotate, and scale to the desired position. Only parts of the target patch inside the rectangle are subject to the bending operator. The normal penalty function needed to be modified and a new penalty function had to be added. The details of these penalty

functions are described in detail in Section 3.3.2 and summarized here. The modification to the normal penalty function was to add a weighting factor that is 1 inside the range of action and 0 outside. Therefore, discrepancies of the normals outside the range of action do not contribute to the normal penalty function. A new penalty function, the *fix penalty function*, had to be added to constrain the surface outside of the range of action. It basically measures deviations between the curvature of the original and bent target surface outside the range of action area. It is needed because the metric penalty function by itself does not fully constrain the surface.

Incremental bending. The bending algorithm works incrementally so that it generates a continuous deformation from the initial target surface to the final target surface. Discontinuities in this deformation process would surprise the user and hence make the bending operator less intuitive. The basic cycle of computing desired normals and warping the surface is repeated several times, with the desired normals in each iteration gradually changing from the normals of the initial target surface to the desired normals for the final bent surface. The penalty functions used in the warping algorithm might have several local minima, some of which represent drastic deformations of the target surface, such as part of the surface being flipped over. With incremental bending, the warping algorithm will find a local minimum of the penalty function, but not necessarily the global minimum. Finding the global minimum is a much harder numerical problem, and it is actually not a good idea anyhow. Consider the user's mental model of what happens during bending. The user imagines a material surface undergoing a deformation process. This process can be thought of as defining a one parameter family of surfaces, the first of which is the original target surface and the last of which is the final bent surface. Nearby surfaces in this family should have similar shapes, or the user will be surprised and the intuitiveness of the bending process will be lost. The minimization algorithm requires a starting estimate and tends to be attracted to nearby local minimum. The incremental approach causes the starting estimate used by the minimization algorithm to be near a local minimum of the penalty function that represents a small change to the target surface, since the target surface doesn't change greatly during each increment. Therefore, the incremental approach guides the warping process to produce a continuous family of surfaces. I originally thought that the incremental approach would improve efficiency

since the minimizer ought to work faster with a better starting estimate, but timing measurements showed that efficiency was not significantly affected.

Having decided to use an incremental approach, I needed a method for generating several intermediate sets of desired normal vectors, with a smooth transition from the normals of the original surface to the desired normals of the final bent surface. The obvious method of interpolating between the normals of the original surface and the desired normals of the final surface can produce discontinuities if the normals vary greatly. The method I chose is to start with a small piece of the bending surface centered at the point of application, and gradually expand that piece to include the whole bending surface. Mathematically, if the point of application is $p = (u_0, v_0)$ and $f(u,v)$ is the bending surface selected by the user, the bending surface used for "time" $t$ is

$$f(t;u,v) = f\big((u-u_0)t + u_0, (v-v_0)t + v_0\big).$$

The parameter $t$ is increased from some small $\delta > 0$ up to 1 in a number of steps, and the bending algorithm is applied using $f(t;u,v)$ as the bending surface at each step. Section 3.6.2 discusses the incremental algorithm in more detail, including some alternatives that were considered and rejected.

Collapse resistance. During early experiments with the bending operator, I found that bending operators with high curvatures sometimes caused creases and folds to form in the target surface. This seems undesirable from the user's point of view. Furthermore, at such singularities the normal vectors become mathematically undefined, with disastrous effects on the numerical algorithms. To circumvent these problems, I added a collapse penalty function. This penalty function, described in detail in Section 3.3.2, measures the inverse of the size of local area element integrated over the surface. If any non-zero area of the surface attempts to contract to zero area during the surface warping phase of the bending algorithm, the collapse penalty function will become large, causing the collapse to be resisted by steering the minimization algorithm away. The collapse penalty function is given a low weight in total penalty function and thus does not significantly affect the results of the bending operator except that creases and folds are prevented.

Independence of parameterization for the penalty functions. It is desirable that the penalty functions be independent of the parameterization of the target surface. This is because in geometric modeling we are interested only in the geometry of the surface, which is shown in Chapter 3 to be independent of its parameterization. There has to be some dependence on parameterization because the correspondence between the bending and target surfaces is determined by the parameterization. However, this dependence should appear only in the normal penalty function. I discuss in Section 3.3.2 what measures are needed to define the metric penalty function and the fix penalty function in a parameterization independent way (the collapse penalty function turns out to be naturally independent of parameterization). These measures involve minimization over the function space of all isometric warpings of the target surface. How to implement such a minimization is a problem I have not solved, although I think it could be done by finite element techniques. Although my definitions are independent of parameterization, my implementation uses an approximation to avoid doing the minimization, so the prototype software is not independent of patch parameterization.

## 6.3 An algorithm for joining using variational methods

I present two new results about the joining problem in this dissertation. The general joining problem is to connect several surface patches such that the resulting composite surface has a specified degree of smoothness across the joined edges. My new results are a classification scheme for joining problems and an outline for a joining algorithm. I addressed the joining problem in this dissertation for two reasons. First, I wanted to show that the variational methods used to implement the bending algorithm have a wide utility for surface shaping tasks and are applicable to the joining problem. Second, my long term goal is to develop a sufficiently complete set of high level surface shaping tools to build a full function 3-D graphics modeling system. The joining problem turned out to be unexpectedly difficult for reasons explained in Section 6.3.1. I developed the theory and an outline for a joining algorithm, but I did not implement the algorithm.

### 6.3.1 Why joining is a hard problem

In Chapter 4 I presented a definition and some analysis due to DeRose [ DeRose 85 ] of the concept of *geometric continuity of degree k* (usually denoted $G^k$ in the literature). This concept derives from the theory of differentiable manifolds. The basic idea is that the composite surface obtained by joining some patches with $G^k$ continuity at the edges and corners should be indistinguishable from a single patch that has degree *k* continuity. In particular the property of $G^k$ continuity is independent of how the individual patches are parameterized. DeRose showed that $G^k$ continuity is the correct way to define smoothness of joins and that it is a more general property than $C^k$ continuity (which can be destroyed by changing the patch parameterization). DeRose showed that $G^k$ continuity implies that a system of non-linear partial differential equations must be satisfied on the edge curves of the patches in the composite. For a large class of patch types based on control points the non-linear PDE system can be reduced to a non-linear algebraic system of equations in terms of the control points and a finite number of arbitrary functions (the Beta constraints). The problem is that these non-linear systems of constraints are in general very hard to solve and, in fact, may not be solvable at all for certain patch types. This problem is well-known in the geometric design field, and one of the common techniques for dealing with it is to design patch types around these constraints such that solvability is built in. It is this problem that makes my solution to the joining problem not fully general over patch types.

6.3.2 Classification of joining problems

I presented in Chapter 4 a new classification scheme for joining problems. This scheme has three categories:

1. *Edge-Edge*. Join two patches by pulling together two edges that are not initially contiguous. One or both of the patches may deform during the joining process.

2. *Blend*. Join the edges of two patches by connecting them with a third patch that fills in the gap. The patches being joined usually do not deform.

3. *Edge-Curve*. Join two patches by gluing the edge of one to a curve in the other. The patch containing the curve does not deform.

Most of the joining problems encountered in practice seem to fall into one of these categories. I showed that, under certain conditions, detailed in Section 4.3.1, the Blend and Edge-Curve joining problems can be reduced to the Edge-Edge joining problem.

6.3.3 Outline of an algorithm for Edge-Edge joining

I presented in Chapter 4 an outline of a variational algorithm for performing Edge-Edge joining of two patches with $G^k$ continuity. The joining algorithm is similar to the bending algorithm in that a penalty function of the two surfaces is minimized to produce the smoothest possible final surface. The important difference is that the joining algorithm requires a constrained minimization and the bending algorithm requires an unconstrained minimization. The constraint equations for the joining algorithm are the non-linear algebraic equations mentioned earlier in Section 6.3.1. The joining algorithm is restricted to using patch types for which it is known how to set up and solve these constraint equations. The penalty function has two components: one measures how smooth the surface is, and the other measures how well the corresponding points on the edge curves match up. I did not implement the algorithm, but I am confident that with the restrictions on patch types mentioned earlier, it would work because it is quite similar to the bending algorithm for a mesh of patches. As discussed in Chapter 3, the bending algorithm for a mesh of patches has to preserve a set of continuity constraints on the joined edges of the patches in the mesh. I only wanted to demonstrate a proof of concept in the prototype software, so I used $C^k$ continuity (which yields linear constraints) rather than $G^k$ continuity. Because of the non-linear constraints, the joining algorithm would be more difficult to implement than the bending algorithm, and I expect it would be slower to execute.

## 6.4 Future work

There are several directions in which this work could be extended. There are the obvious extensions of implementing the joining algorithm, improving the speed of the bending algorithm, and incorporating the bending algorithm into a complete graphics modeling system. There is also an interesting and not so obvious extension of the bending operator to solid objects rather than surfaces.

### 6.4.1 Making the bending algorithm faster

The bending algorithm spends most of its execution time minimizing the penalty function. Given an arbitrary, non-linear penalty function, there is not much that can be done to improve efficiency further. I am already using one of the most efficient known numerical optimization algorithms, the BFGS algorithm, described in Section 3.6.4. However, if the penalty function were quadratic in terms of the control points, there is a far more efficient algorithm. The gradient of the penalty function would then be linear in terms of the control points. The minimum could be found directly by setting the gradient to zero and solving the resulting linear system for the control points. This ought to provide a two or three order of magnitude speed improvement for the minimization. Several researchers working with variational techniques for surface shaping have addressed this same problem; see Chapter 2 for the details. Welch and Witkin [ Welch 92 ], for instance, expanded their penalty function as a Taylor series in the control points and discarded the terms of greater order than quadratic. This, of course, yielded a very poor approximation of the penalty function, but amazingly, their surface shaping algorithm still worked well. Because I am using a quite similar penalty function, it seems worth trying the same trick for the bending algorithm.

### 6.4.2 Incorporating the bending operator into a 3-D modeling system

By itself the bending operator does not provide a complete set of modeling capabilities. For example, it cannot be used to stretch or shear (change the metric) of a surface. In the spoon example I had to make the initial, flat patch for the spoon outside the prototype

program because the patch had to be wide where the bowl was to be made and narrow where the handle was to be made. Incorporating the bending operator into a full function geometric modeling system would be useful for further experimentation. A logical candidate system is the Utah Alpha-1 system. It is a patch-based system that uses B-spline patches as the basic modeling element. It is a modular system in that specialized shaping tools can be and are implemented as stand-alone programs that communicate via sockets with the rest of the modeling system.

### 6.4.3 Bending a solid object via the symmetric axis

It may be possible to apply the bending operator to a solid object by bending the object's symmetric axis. In fact, this was my original plan for the dissertation, but figuring out how to do bending of a surface turned out to be a big enough problem in itself.

The symmetric axis of a solid object is the set of points inside the object and equidistant from two or more points on the boundary surface of the object. The symmetric axis of a solid object turns out to be a possibly branching two-dimensional manifold. At each point on the symmetric axis there is a radius function that is the distance to the nearest boundary surface point. Given both a symmetric axis and a radius function, it is possible to reconstruct the object. Thus a possible shaping algorithm for solid objects would work as follows: find the symmetric axis and radius function for the object, bend one of the sheets making up the symmetric axis, and reconstruct a new object from the modified symmetric axis and old radius function. A variation of this algorithm would to also allow the radius function to change (by an analog of bending).

There is a theoretical problem and a practical problem with this proposed algorithm. The theoretical problem is that a sufficiently large bending of the symmetric axis might cause it not to be any longer the symmetric axis of any object. There are certain inequalities that must hold between the derivatives of the symmetric axis and radius function, and the bending process might violate these. The bending algorithm could possibly be modified to respect these constraints or to change the radius function to preserve these constraints. The practical problem is that the symmetric axis of a complicated object is itself a

complicated object, possibly containing loops and branches. As discussed in Chapter 3, the bending operator can only be applied to parts of a mesh of patches that can be flattened out, i.e., single, non-branching sheets.

There is an interesting potential application of this idea to visualization of 3-D data, for example, volume rendered medical patient CAT scans. A big difficulty in viewing volume rendered data is obscuration of interesting parts of the data set by uninteresting parts. It is often not possible to remove the uninteresting parts by classification, because they may have virtually the same density as the interesting parts. Consider, for example, looking for a tumor. What is needed is a range of interest selection tool, that is, a solid shape that can be used as a key for whether or not to remove parts of the data set. To be useful, such a range of interest tool would need to be very malleable so that it could be interactively shaped to select the interesting parts of the data set. The bending operator applied to the symmetric axis of the range of interest tool might provide the necessary ease of shape control.

# Appendix A
# Tutorial on differential geometry of surfaces

The purpose of this tutorial is to provide definitions and explanations of the concepts of the differential geometry of surfaces in 3-D space. This presentation does not comprehensively cover the subject; only concepts actually needed to read the dissertation are included. Some important ideas that are omitted here are the theory of space curves, arc length, the Frenet equations, and the algebra of *n*-forms and exterior differential forms. Some references to the literature on differential geometry are given at the end of this appendix. The reader is assumed to be familiar with linear algebra and calculus of several variables.

The reader who isn't familiar with the notation of modern differential geometry may become dismayed and confused by the tremendous number of definitions and the great variety of spaces used. I assure the reader that these definitions and spaces are not arbitrary but that they have been worked out carefully over more than half a century. They are the right tools for the job. Before jumping into the details I offer the following bit of advice, which helped me a great deal when I was first learning the subject. When things get confusing, try drawing a diagram of nodes connected by arrows. Label the nodes with the spaces and the arrows with the functions and operators involved.

The setting for this topic is 2-D and 3-D Euclidean space, i.e., real vector spaces of dimensions 2 and 3 with a notion of length and dot products of vectors. Many of the concepts are independent of the actual dimension.

Definition A-1. Euclidean n-space $\Re^n$ is the set of all ordered n-tuples of real numbers. Such an n-tuple $p = (p_1,\ldots,p_n)$ is called a *point* of $\Re^n$. The *distance* between two points $p$ and $q$ is given by the Pythagorean formula

$$\|p - q\| = \sqrt{(p_1 - q_1)^2 + \ldots + (p_n - q_n)^2},$$

and the *dot product* of two points $p$ and $q$ is given by

$$p \cdot q = p_1 q_1 + \ldots + p_n q_n .$$

Euclidean n-space becomes a real vector space by defining addition and scalar multiplication componentwise.

Tangent Vectors. Intuitively, the notion of a vector in $\Re^n$ can be thought of as a directed line segment or arrow between two points. In differential geometry both the starting point and the arrow are important. A vector $v_p$ is defined by giving the starting point $p$ and the change, or vector $v$, necessary to reach its end point $p+v$. Strictly speaking, $v$ is just a point of $\Re^n$.

Definition A-2. A *tangent vector* $v_p$ to $\Re^n$ consists of two points of $\Re^n$, its *vector part* $v$ and its *point of application* $p$.

The word "tangent" is included to emphasize that the point of application is important: two tangent vectors $v_p$ and $v_q$ are not the same if $p$ is not equal to $q$. In this dissertation, all vectors are tangent vectors with a specific point of application although sometimes the word "tangent" or the subscript $p$ is omitted for brevity.

For a given point $p$ in $\Re^n$, the totality of all tangent vectors $v_p$ with point of application $p$ form an $n$-dimensional vector space by applying addition and scalar multiplication to the vector parts, called the *tangent vector space* of $p$, or *tangent space* of $p$. A *tangent vector field*, or *vector field* on $\Re^n$ is obtained by selecting one tangent vector $v_p$ from the

tangent vector space for each $p$ in $\Re^n$. Such a vector field is defined to be continuous or differentiable if its vector part is a continuous or differentiable function of the point of application.

Tangent vectors can also be thought of as derivative operators, in the following sense. Let $f$ be a real-valued function on $\Re^n$. For a given tangent vector $v_p$ one can think of taking the directional derivative of $f$ at $p$ in the $v$ direction. To be specific, we define the application of a tangent vector $v_p$ to a function $f$ by

$$v_p[f] = \lim \frac{f(p+tv) - f(p)}{t} \text{ as } t \to 0 .$$

Consider the operator $\frac{\partial}{\partial x}$ in $\Re^2$. By definition

$$\frac{\partial}{\partial x}\Big|_p f(p) = \lim \frac{f(p+t(1,0)) - f(p)}{t} \text{ as } t \to 0 .$$

Thus $\frac{\partial}{\partial x}\Big|_p$ acts just like the tangent vector $(1,0)_p$ when applied to functions. Thus the familiar partial derivative operators can be considered to be tangent vectors.

We can extend the concept of tangent vectors to two-dimensional surfaces in $\Re^3$. We define tangent vectors $v_p$ of a two-dimensional surface $S$ in $\Re^3$ as the subset of tangent vectors of $\Re^3$ with point of application $p$ in $S$ and vector part $v$ tangent to $S$ at $p$. It can be shown that the set of tangent vectors at a point of a surface forms a two-dimensional vector space. In this dissertation, we are interested in surfaces that are the range of a single patch. We define patches as follows.

Definition A-3. A *two-dimensional patch c* is a smooth (differentiable) function $c:[0,1]^2 \to \Re^3$ which is *regular* (its Jacobian has rank 2) at each point of $[0,1]^2$. Letting $u$

and $v$ be the usual coordinates of $[0,1]^2$ the *outward normal vector* of the patch is $\frac{c_u \times c_v}{\|c_u \times c_v\|}$, where $c_u$ and $c_v$ denote the partial derivatives of $c$ with respect to $u$ and $v$.

We require patches to be regular so that there is always a well-defined normal vector. Not every surface can be represented as the image of a single patch. The proper generalization of a two-dimensional surface is a two-dimensional differentiable manifold, which is roughly speaking, a surface that can be covered by overlapping patches. The main difficulty is to make sure that all of the geometrical objects defined on a manifold agree on the overlap between patches and are thus independent of patch coordinates.

1-Forms. The concept of 1-forms is very useful in the differential geometry of surfaces. 1-forms are equally as important as tangent vectors, but they are harder to visualize and less widely known. One can think of a 1-form as a measuring device for tangent vectors. If $\phi$ is a 1-form and $v_p$ is a tangent vector, then $\phi(v_p)$ is a real number. At a given point of application $p$, the function from the vector part of $v_p$ to the reals is required to be linear.

Definition A-4. A 1-form $\phi$ on $\Re^n$ is a real-valued function on the set of all tangent vectors of $\Re^n$ such that $\phi$ is linear at each point $p$, i.e.,

$$\phi(av_p + bw_p) = a\phi(v_p) + b\phi(w_p)$$

for any real numbers $a$, $b$ and any tangent vectors $v_p$, $w_p$ at the same point $p$ of $\Re^n$.

The sum of 1-forms on $\Re^n$ is defined pointwise

$$(\phi + \psi)(v_p) = \phi(v_p) + \psi(v_p).$$

Likewise multiplication by a real-valued function on $\Re^n$ is defined pointwise.

$$(f\phi)(v_p) = f(p)\phi(v_p).$$

At a given point $p$, the 1-forms $\phi_p = \phi(p)$ form an $n$-dimensional real vector space due to the linearity property. This is just the dual space of the tangent vector space. If $\phi_p$ is not zero, a standard theorem of linear algebra guarantees the existence of a tangent vector $v_p$ such that $\phi(v_p) = 1$. If $w_p$ is another tangent vector at $p$, then $\phi(w_p) = v \cdot w$. In particular, if $v_p$ and $w_p$ are orthogonal, then $\phi(w_p) = 0$.

One way of visualizing a 1-form $\phi$ is to picture for each $p$ a stack of parallel plates centered at $p$. Let $v_p$ be the tangent vector for which $\phi(v_p) = 1$. The plates are thought of as being perpendicular to $v$ with a separation of the length of $v$. Hence $\phi_p$ is seen as an oriented ruler that measures the component of tangent vectors at $p$ in the $v$ direction, with a scale such that $v$ is the unit length. To measure the size of a tangent vector $w_p$ using $\phi$, count the number of plates crossed by $w$. For 1-forms on $\Re^2$ picture line segments rather than plates.

Differentials. The symbols *df*, *dx*, and *dy* that appear in the familiar equation

$$df = \frac{\partial f}{\partial x}\,dx + \frac{\partial f}{\partial y}\,dy$$

and which are often not rigorously defined in introductory calculus courses, turn out to be 1-forms. Let $f$ be a real-valued function on $\Re^n$. Then $df$ is the 1-form which measures the change of the value of $f$ at a point $p$ for movements along tangent vectors $v_p$. By definition $df(v_p) = v_p[f]$. It is instructive to use this formula to compute the differentials *dx* and *dy* in $\Re^2$. Define $x$ as the coordinate function from $\Re^2$ to $\Re$ which picks out the first component of a point $p$. That is, $x(p) = x(p_1, p_2) = p_1$. Hence,

$$dx(v_p) = v_p[x] = \lim \frac{x(p+tv) - x(p)}{t} = \frac{p_1 + tv_1 - p_1}{t} = v_1.$$

In other words, *dx* throws away the point of application and returns the first or *x* component of the vector part of $v_p$. Using the visualization technique, we picture a family of line segments parallel to the y axis with a unit separation.

2-Forms. The concept of 1-forms can be generalized to *n*-forms for any non-negative integer *n*. We will need 2-forms to define integration on surfaces. One can think of a 2-form as a measuring device for area. A 2-form operates on pairs of tangent vectors and gives the signed area of the parallelepiped spanned by the two vectors.

Definition A-5. A *2-form* $\omega$ on $\Re^n$ is a mapping from pairs of tangent vectors of $\Re^n$ to the reals which satisfies the following two properties.

1. At a given point *p*, $\omega$ is linear in each component, that is, if $u_p$, $v_p$, and $w_p$ are tangent vectors at *p* and *a* and *b* are real numbers then

$$\omega(au_p + bv_p, w_p) = a\omega(u_p, w_p) + b\omega(v_p, w_p)$$

and

$$\omega(w_p, au_p + bv_p) = a\omega(w_p, u_p) + b\omega(w_p, v_p)$$

2. $\omega$ is *alternating*, that is,

$$\omega(u_p, v_p) = -\omega(v_p, u_p).$$

For consistency, 0-forms are defined to be ordinary real-valued functions. It turns out that on $\Re^2$ and on two-dimensional surfaces, all forms of degree 3 or higher are zero. We will only be concerned with 0-, 1-, and 2-forms in this tutorial.

A 2-form $\omega$ applied to a pair of identical tangent vectors yields zero due to the alternating property: $\omega(u_p, u_p) = -\omega(u_p, u_p) = 0$. 2-forms are closely related to determinants as the following theorem shows:

Theorem A-1. Let $\omega$ be a 2-form on $\Re^2$, and let $u_p$ and $v_p$ be tangent vectors of $\Re^2$ at a point $p$. Let $e_{1p}$ and $e_{2p}$ be a basis for the tangent space of $\Re^2$ at $p$ and $\alpha_{ij}$ be real numbers such that $u_p = \alpha_{11}e_{1p} + \alpha_{12}e_{2p}$ and $v_p = \alpha_{21}e_{1p} + \alpha_{22}e_{2p}$. Then

$$\omega(u_p, v_p) = (\alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21})\omega(e_{1p}, e_{2p}) = \det\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix}\omega(e_{1p}, e_{2p}).$$

This is easily shown by expanding the left hand side using the linearity and alternating properties. Thus a 2-form on $\Re^2$ is completely determined at any point by its value on a basis of the tangent space at that point. Using componentwise addition and scalar multiplication, the 2-forms can be made into a vector space at each point. Theorem A-1 shows that this space is one-dimensional. Hence if we have any non-zero 2-form $\omega$ on $\Re^2$, we can write any other 2-form $\phi$ on $\Re^2$ as a scalar multiple of $\omega$. That is, given $\phi$ there exists a function $f$ such that $\phi = f\omega$.

There is a natural way to multiply $n$-forms called the wedge product, which we will need to investigate integration on surfaces.

Definition A-6. Let $\omega$ and $\phi$ be 1-forms, and let $f$ be a 0-form (ordinary function) on $\Re^n$. Let $u_p$ and $v_p$ be tangent vectors of $\Re^n$. The *wedge product of $\omega$ and $\phi$*, denoted by $\omega \wedge \phi$, is the 2-form defined by

$$\omega \wedge \phi(u_p, v_p) = \omega(u_p)\phi(v_p) - \omega(v_p)\phi(u_p) = \det\begin{bmatrix} \omega(u_p) & \omega(v_p) \\ \phi(u_p) & \phi(v_p) \end{bmatrix}.$$

The *wedge product of $f$ and $\omega$*, denoted by $f \wedge \omega$, is the 1-form $f\omega$, the scalar product of $f$ and $\omega$.

It is easily shown that if $\omega$ and $\phi$ are 1-forms and $f$ is a 0-form then $\omega \wedge \phi = -\phi \wedge \omega$ and $\omega \wedge f = f \wedge \omega$. It is interesting to note how often the determinant appears when dealing

with *n*-forms. One way of thinking of *n*-forms is that they are generalizations of the determinant. Indeed, on $\Re^2$ the function det can be thought of as a 2-form.

We define *n*-forms on a surface *S* by restricting *n*-forms of $\Re^3$ to the surface. It can be shown that the 1-forms at a point of the surface form a 2-dimensional vector space and that the 2-forms at a point of the surface form a one-dimensional vector space.

Derivative Maps and Pull-backs. Tangent vectors and *n*-forms can be transferred from one surface to another by a smooth mapping. Suppose we have a function *f* that smoothly maps a two-dimensional surface *M* in $\Re^3$ to a two-dimensional surface *N* in $\Re^3$

$$f: M \to N .$$

We define a function from tangent vectors of *M* to tangent vectors of *N* as follows.

Definition A-7. Let $v_p$ be a tangent vector of *M*. Let $(f^1, f^2, f^3)$ be the components of *f*. The *derivative map* $f_*$ is defined by

$$f_*(v_p) = (v_p[f^1]\, v_p[f^2]\, v_p[f^3])_{f(p)}.$$

Another way of saying this is that $f_*$ at *p* is the linear transformation which best approximates the behavior of *f* at *p*. The matrix of this linear transformation with respect to the usual basis is just the Jacobian matrix of *f* at *p*. The derivative map is sometimes called a *push-forward*, as it works in the same direction as *f*. Similarly we can transfer 1-forms between surfaces. Let $\phi$ be a 1-form on *N*. We define a 1-form $f^*\phi$ on *M*, called the *pull-back of* $\phi$ *under f*, by

$$(f^*\phi)(v_p) = \phi(f_*(v_p)).$$

In words, the pull-back of a 1-form is evaluated on a tangent vector by first pushing forward the vector and then applying the 1-form. Let $\omega$ be a 2-form on *N*. We define a 2-form $f^*\omega$ on *M*, called the *pull-back* of $\omega$ under $f$, by

$$(f^*\omega)(u_p, v_p) = \omega(f_*(u_p)(f_*(v_p))).$$

As with 1-forms, the pull-back of 2-form is evaluated on a pair of tangent vectors by first pushing forward the vectors and then evaluating the 2-form. For consistency, we define the pullback of a 0-form *g* on *N* by

$$f^*g = g \circ f.$$

Pull-backs go in the opposite direction of the function, that is, *n*-forms on *N* are taken to *n*-forms on *M*. Pull-backs interact with wedge products according to the formula $f^*(\omega \wedge \phi) = f^*\omega \wedge f^*\phi$ for *n*-forms $\omega$ and $\phi$ of any degree.

Frame Fields. A *frame field* on $\Re^n$ is a set of mutually orthogonal unit vectors at each point of $\Re^n$. A frame field is defined to be continuous or differentiable if its component vector fields are.

Any smooth surface in $\Re^3$ induces (at least locally) a pair of differentiable unit normal vector fields. The surface is called *orientable* if there is a global differentiable unit normal vector field. Though not all surfaces are orientable (e.g., the Möbius strip), we will assume the surfaces we encounter here are. Any surface that is the range of a single patch is clearly orientable. A normal vector field can be expanded to a frame field by choosing two other unit vector fields orthogonal to each other and the normal vector field. Such a frame field is said to be *adapted* to the surface. By sliding a small distance along the normal vectors, this adapted frame field can be extended to a 3-dimensional neighborhood of the surface. An adapted frame field for a surface is defined to be continuous or differentiable if its extension to a 3-dimensional neighborhood is.

The Structure Equations. Elie Cartan developed the very powerful idea of *moving frames* for studying surfaces by attaching adapted frame fields at each point and considering how such frame fields twist and turn with movements in the surface. The key idea is to express the derivatives of an adapted frame field in terms of the frame field itself. In this way the coefficients of the derivative matrices give direct information about the geometry of the surface.

Let *c* be a patch, i.e., a function from the unit square to 3-dimensional Euclidean space, and let $e = (e_1, e_2, e_3)$ be an adapted frame field for this patch, with $e_3$ the outward normal vector. A frame field is called *right-handed* if $e_1 \times e_2 = e_3$ and *left-handed* if $e_1 \times e_2 = -e_3$. We can compute the differentials of vector fields $e_1, e_2, e_3$ yielding $de_1, de_2, de_3$. $de_i$ is thus a vector of 3 1-forms, and *de* can be thought of as a $3 \times 3$ matrix of 1-forms. Cartan's idea was to express these forms in terms of the frame *e* itself. Hence we define

$$\omega_{ij} = de_i \cdot e_j \text{ for } i, j = 1..3 .$$

These are called the *structure forms*. In matrix notation,

$$\omega = dee^t .$$

What are the properties of these coefficients $\omega_{ij}$, and what do they reveal about the geometry of the surface? If we apply, for example, $\omega_{13}$ to a tangent vector $v_p$ that is tangent to the surface, we obtain

$$\omega_{13}(v_p) = de_1(v_p) \cdot e_3 .$$

$de_1(v_p)$ gives the change in the $e_1$ frame vector for motions along the tangent vector $v_p$. Thus $\omega_{13}(v_p)$ gives the component of the change in the $e_1$ frame vector in the direction of the $e_3$ frame vector. In other words, $\omega_{13}$ gives the rate at which the $e_1$ frame vector is turning towards the $e_3$ frame vector. The same reasoning applies to all the $\omega_{ij}$, so for any $v_p$, $\omega_{ij}(v_p)$ is the rate of turn of the *i*-th frame vector towards the *j*-th frame vector for

movement in the *v* direction at point *p* in the surface. It can be shown that $\omega_{ii} = 0$ and furthermore that $\omega$ is a skew-symmetric matrix, i.e., it is the negative of its transpose: $\omega^t = -\omega$. Intuitively this means that the rate of turn of the *i*-th frame vector in its own direction is zero (which has to be true for any unit vector) and that the turn of the *i*-th frame vector toward the *j*-th frame vector is the negative of the turn of the *j*-th frame vector toward the *i*-th frame vector. The proof is simple: just compute the differential of the dot product of two frame vectors. Since $e_i \cdot e_j$ is constant,

$$0 = d(e_i \cdot e_j) = de_i \cdot e_j + e_i \cdot de_j = \omega_{ij} + \omega_{ji}.$$

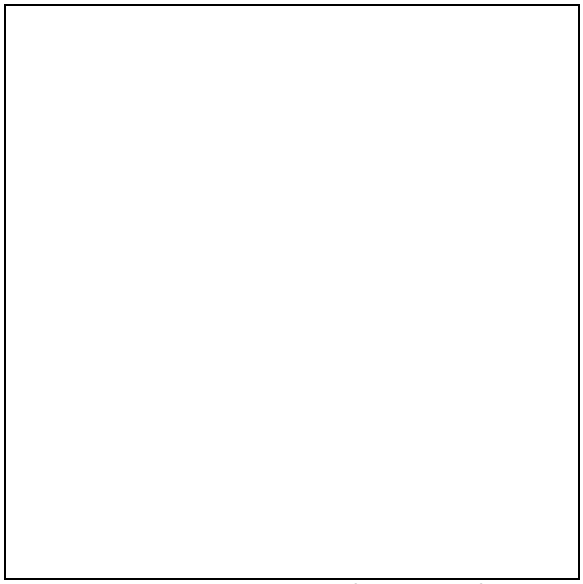
If we multiply both sides of $\omega = dee^t$ by the matrix *e*, we obtain the *structure equations* in matrix form

$$de = \omega e.$$

The Metric. The *metric*, also called the *first fundamental form*, is basically a device for measuring the lengths (squared) of tangent vectors. It is not an *n*-form. It is customarily denoted by the symbol *I* (Roman numeral *one*.) The metric repeatedly appears in the study of the geometry of surfaces.

Definition A-8. The *metric I* on $\Re^n$ is the function from the set of all tangent vectors $v_p$ of $\Re^n$ such that $I(v_p) = \|v\|^2$. The metric on a surface patch in $\Re^3$ is defined by restricting the metric on $\Re^3$ to tangent vectors of the patch.

The metric can be expressed in terms of 1-forms, which should not be too surprising, since 1-forms are also real-valued functions of tangent vectors. Let *x*, *y*, and *z* be the coordinate functions for $\Re^3$, i.e., if $p = (p_1, p_2, p_3)$ is a point in $\Re^3$ then $x(p) = p_1$, $y(p) = p_2$, and $z(p) = p_3$. Consider the 1-forms *dx*, *dy*, and *dz*. We saw earlier that the differentials of the coordinate functions simply pick out the corresponding component of the vector part of tangent vectors. Therefore we have



$$I(v_p) = \|v\|^2 = v_1^2 + v_2^2 + v_3^2 = dx(v_p)^2 + dy(v_p)^2 + dz(v_p)^2$$

or in short

$$I = dx^2 + dy^2 + dz^2 = dX \cdot dX,$$

where $X$ denotes the point $(x, y, z)$. Matters become more interesting when we consider the metric on a surface patch in $\Re^3$.

Let $c:[0,1]^2 \to \Re^3$ be a surface patch. Our goal is to find a formula for the metric on this surface. From the previous paragraph we have $I = dc \cdot dc$. Let $u$ and $v$ be the coordinate functions on $[0,1]^2$. We have

$$dc = c_u du + c_v dv,$$

where $c_u$ and $c_v$ are tangent vector fields on the surface obtained by taking partial derivatives with respect to $u$ and $v$. Let $E = c_u \cdot c_u$, $F = c_u \cdot c_v$, and $G = c_v \cdot c_v$. Then

$$I = dc \cdot dc = (c_u du + c_v dv) \cdot (c_u du + c_v dv)$$
$$= c_u \cdot c_u du^2 + 2c_u \cdot c_v du dv + c_v \cdot c_v dv^2$$
$$= E du^2 + 2F du dv + G dv^2$$

in terms of the differentials *du* and *dv*. Strictly speaking, this is actually the pull-back of *I* by the function *c*. *I* is an example of a quadratic form, that is, a quadratic polynomial of 1-forms. We can write $I$ in matrix notation in the following way:

$$I = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} E & F \\ F & G \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}$$

Isometries. Roughly speaking, two surfaces are isometric if one can be deformed into the other without stretching or tearing. For example a flat sheet and a cylinder are isometric, but a cylinder and a sphere are not. The precise definition of isometries is as follows.

Definition A-9. Let *M* and *N* be surfaces in $\Re^3$. Let *f* be a smooth mapping from *M* to *N*. *f* is an isometry if its derivative map $f_*$ preserves the lengths of tangent vectors.

Letting $I_M$ denote the metric of *M* and $I_N$ denote the metric of *N*, this definition is equivalent to $f^* I_N = I_M$. That is, the pullback via *f* preserves the metric. Two surfaces are said to be *isometric* if there exists an isometry between them. Surface properties which are preserved by isometries are said to be *intrinsic*, and other surface properties are said to be *extrinsic*.

To help understand the distinction between intrinsic and extrinsic properties, let us imagine a race of two-dimensional beings living on a surface, with no awareness of height. What could such beings discover about the geometry of their world? One thing they could do is measure distances, and hence they could determine the metric of their world. Thus they would be able to tell whether they lived on a flat plane or on a sphere, since these have different metrics. In fact, they could completely determine the intrinsic geometry of their world but none of the extrinsic geometry.

The normal vector field of a surface is an extrinsic property. To see this, think of bending a flat sheet of paper. Paper has the property of resisting changes to its metric. Some experimentation will convince the reader that a sheet of paper can be bent into a great variety of shapes. However the sheet of paper cannot assume all shapes. For instance it is impossible to wrap even a small piece of a sphere with a flat sheet. The class of shapes that a piece of paper can be bent into is precisely the set of surfaces isometric to a flat sheet.

Integration and the Area Element. In the bending algorithm developed in this dissertation the penalty functions are defined as integrals of certain functions over a surface. We denote the integral of a function $f$ over a surface $S$ by $\int_S f dA$.

Our goal in this section is to define such an integral and to develop a formula for evaluating it. The term $dA$ is the area element and is an example of a 2-form. The notation might lead one to think that $dA$ is the differential of something, but this is not necessarily the case. However the notation is too firmly established to change. The proper object to integrate over a surface turns out to be a 2-form rather than an ordinary function. By our rule for scalar multiplication, $fdA$ is a 2-form.

Definition A-10. Let $c:[0,1]^2 \to \Re^3$ be a patch, and let $e = (e_1, e_2, e_3)$ be an adapted frame field for this patch, with $e_3$ the normal vector. Further assume that the frame field is right-handed, i.e. $e_1 \times e_2 = e_3$. We define the *area element dA* on $S = c([0,1]^2)$ to be the 2-form such that at each point, $dA(e_1, e_2) = 1$.

It can be shown that $dA$ is independent of the particular frame field used to define it. Recall that a 2-form measures the area of the parallelepiped spanned by a pair of vectors. This definition says that $dA$ is the (unique) 2-form that assigns unit area to the square spanned by the frame vectors $e_1$ and $e_2$.

We now ready to define the integral of an arbitrary 2-form on a surface patch and to develop machinery for evaluating such integrals.

Definition A-11. Let $\phi$ be a 2-form on $\Re^2$. Let $f$ be the real-valued function on $\Re^2$ such that $\phi = f dA$, where $dA$ is the area element for $\Re^2$. The *integral of $\phi$ over the unit square* $[0,1]^2$ is defined by

$$\int_{[0,1]^2} \phi = \int_{[0,1]^2} f dA = \int_0^1 \int_0^1 f(u,v) du dv.$$

Let $c:[0,1]^2 \to \Re^3$ be a patch, and let $\phi$ be a 2-form on $S = c([0,1]^2)$. The *integral of $f$ over $S$* is defined as

$$\int_S \phi = \int_{[0,1]^2} c^* \phi .$$

Strictly speaking we integrate only in $\Re^2$. We pull back forms to $\Re^2$ and convert to ordinary iterated integrals of functions. Definition A-11 appears to depend on the choice of the particular patch $c$, but it can be shown that the value of the integral is independent of this choice; i.e., if $\hat{c}$ is another patch such that $S = \hat{c}([0,1]^2)$, then $\int_{[0,1]^2} \hat{c}^* \phi = \int_{[0,1]^2} c^* \phi$. In fact, the main reason we integrate 2-forms on surfaces is to achieve this independence of parameterization. To develop the machinery for computing surface integrals, we must investigate how to compute pull-backs of 2-forms.

To set the stage for this computation, let $c:[0,1]^2 \to \Re^3$ be a patch, let $\phi = f dA$ be a 2-form on $S = c([0,1]^2)$, and let $e = (e_1, e_2, e_3)$ be a right-handed, adapted frame field on $S$. Our goal is to find a formula for the pull-backs of $dA$ and $fdA$ via $c$. Let $u$ and $v$ be the usual coordinate system on the domain $[0,1]^2$. First note that on $[0,1]^2$ the area element is $du \wedge dv$. To see this, compute

$$\begin{aligned} du \wedge dv((1,0)_p, (0,1)_p) &= du((1,0)_p) dv((0,1)_p) - du((0,1)_p) dv((1,0)_p) \\ &= 1 \cdot 1 - 0 \cdot 0 = 1 \end{aligned}.$$

We will show that $c^*(dA) = \sqrt{EG-F^2}\,du \wedge dv = \sqrt{\det(I)}\,du \wedge dv$, where $E$, $F$, and $G$ are the components of the first fundamental form $I$, i.e., $E = c_u \cdot c_u$, $F = c_u \cdot c_v$, and $G = c_v \cdot c_v$. Expressing $c_u$ and $c_v$ in terms of the frame field $e$, we have

$$c_u = c_u \cdot e_1 e_1 + c_u \cdot e_2 e_2$$
$$c_v = c_v \cdot e_1 e_1 + c_v \cdot e_2 e_2.$$

There are no $e_3$ components, since $c_u$ and $c_v$ are tangent to the surface, hence orthogonal to $e_3$. Now applying Theorem A-1, we compute

$$c^*(dA)\big((1,0)_p,(0,1)_p\big) = dA\big(c_*(1,0)_p, c_*(0,1)_p\big) = dA(c_u, c_v)$$
$$= dA(c_u \cdot e_1 e_1 + c_u \cdot e_2 e_2, c_v \cdot e_1 e_1 + c_v \cdot e_2 e_2)$$
$$= \det\begin{bmatrix} c_u \cdot e_1 & c_u \cdot e_2 \\ c_v \cdot e_1 & c_v \cdot e_2 \end{bmatrix} dA(e_1, e_2) = \det\begin{bmatrix} c_u \cdot e_1 & c_u \cdot e_2 \\ c_v \cdot e_1 & c_v \cdot e_2 \end{bmatrix}.$$

Let $M = \begin{bmatrix} c_u \cdot e_1 & c_u \cdot e_2 \\ c_v \cdot e_1 & c_v \cdot e_2 \end{bmatrix}$. We need to show that $\det(M) = \sqrt{EG-F^2}$. We have

$$MM^t = \begin{bmatrix} c_u \cdot e_1 c_u \cdot e_1 + c_u \cdot e_2 c_u \cdot e_2 & c_u \cdot e_1 c_v \cdot e_1 + c_u \cdot e_2 c_v \cdot e_2 \\ c_v \cdot e_1 c_u \cdot e_1 + c_v \cdot e_2 c_u \cdot e_2 & c_v \cdot e_1 c_v \cdot e_1 + c_v \cdot e_2 c_v \cdot e_2 \end{bmatrix} = \begin{bmatrix} c_u \cdot c_u & c_u \cdot c_v \\ c_v \cdot c_u & c_v \cdot c_v \end{bmatrix} = I$$

which implies $\det(M)^2 = \det(I)$ and thus $\det(M) = \sqrt{\det(I)} = \sqrt{EG-F^2}$.

To conclude, we compute $c^*(fdA) = f \circ c\, c^*(dA) = f \circ c \sqrt{EG-F^2}\,du \wedge dv$. Our formula for the surface integral is therefore

$$\int_S f dA = \int_{[0,1]^2} c^*(fdA) = \int_{[0,1]^2} f \circ c \sqrt{EG-F^2}\,du \wedge dv$$
$$= \int_0^1 \int_0^1 f(c(u,v)) \sqrt{E(u,v)G(u,v) - F(u,v)^2}\,du dv.$$

Curvature and the Second Fundamental Form. The curvature of curves and surfaces is essentially a measure of how tangent vectors at a point turn in the normal direction as the point moves along the respective tangent vectors. For curves this measure at any point is a single number, but for surfaces it is not, since there are multiple directions in which one can move. The curvature information about a surface is captured by a quadratic form known as the *second fundamental form*, defined later.

Definition A-12. Let $\alpha:[a,b]\to\Re^2$ be a plane curve with $\|\alpha'(t)\|=1$, i.e., a *unit speed* curve. The *curvature of* $\alpha$ *at t,* denoted by $\kappa(t)$, is defined as $\kappa(t)=\alpha''(t)\cdot J(\alpha'(t))$, where $J(x,y)=(-y,x)$. The function *J* represents a counter-clockwise 90 degree rotation in $\Re^2$.

That is, the curvature is the component of the acceleration vector in the normal direction. In geometric terms, the curvature at a point on a plane curve is the inverse of the radius of the osculating circle at that point. There is an inherent ambiguity in the sign of the curvature, since there are two possible choices for the normal direction. The ambiguity is resolved in Definition A-12 by the customary rule that left-turning curves have positive curvature and right-turning curves have negative curvature. Note that the sign of the curvature of a plane curve is changed by parameterizing the curve in the opposite direction.

To get at the curvature of surfaces, we examine the curves formed by cutting the surface with a plane that includes the surface normal vector. Such a curve is called a *normal section curve*. Because of the sign ambiguity mentioned in the previous paragraph, we cannot directly use the curvature of such curves to define the curvature of a surface. However, oriented surfaces are equipped with a well-defined outward normal, which we can make use of to produce an unambiguous definition of surface curvature.

Definition A-13. Let $c:[0,1]^2\to\Re^3$ be a surface patch, and let *n* denote the outward normal vector field of *c*. Let $w_p$ be a unit tangent vector and $n_p$ be the outward normal vector at a point *p* on the surface patch. Let $\alpha:(-1,1)\to\Re^3$ be a unit speed

parameterization of the normal section curve formed by the intersection of the surface patch and a plane through $p$ that includes the normal vector $n_p$, such that $\alpha(0) = p$ and $\alpha'(0) = w_p$. The *normal curvature at p in direction w* is defined by

$$k(w_p) = \alpha''(0) \cdot n_p.$$

The minimum and maximum values of the normal curvature at a point are called the *principal curvatures*, usually denoted by $\kappa_1$ and $\kappa_2$, and the tangent vectors at which these are attained are called the *principal directions*. It can be shown that the principal directions are orthogonal where defined. Principal directions are not defined at *umbilic points*, at which $\kappa_1 = \kappa_2$.

The second fundamental form is a quadratic form that captures all the curvature information of a surface.

Definition A-14. Let $c: [0,1]^2 \to \Re^3$ be a surface patch, and let $n$ denote the outward normal vector field of $c$. The *second fundamental form of* $c$, denoted by $II$ (Roman numeral *two*), is the mapping from tangent vectors of $c$ to the reals defined by $II = -dc \cdot dn$.

The formula for normal curvature can be derived by applying the second fundamental form to unit speed normal section curves, which demonstrates that *II* captures all the curvature information of a surface. As in Definition A-13, let $\alpha: (-1,1) \to \Re^3$ be a unit speed parameterization of the normal section curve formed by the intersection of the surface patch $c$ and a plane through $p$ that includes the normal vector $n_p$, such that $\alpha(0) = p$ and $\alpha'(0) = w_p$. Applying *II* to $\alpha'$ yields

$$II(\alpha') = -dc(\alpha') \cdot dn(\alpha') = -\alpha' \cdot n'_\alpha,$$

where $n_\alpha$ denotes $n$ restricted to $\alpha$. But $\alpha'$ is a tangent vector to the surface, so $\alpha' \cdot n = 0$. Taking the derivative of this equation yields $0 = \alpha'' \cdot n + \alpha' \cdot n'$, implying that

$$II(w_p) = II(\alpha'(0)) = -\alpha'(0) \cdot n'_\alpha(0) = \alpha''(0) \cdot n_p = k(w_p).$$

The second fundamental form can be thought of as giving the rate at which a unit tangent vector $w_p$ is turning in the *n* direction as *p* moves in the *w* direction. Equivalently, the second fundamental form can be thought of as giving the rate at which *n* is turning in the $-w$ direction as *p* moves in the *w* direction. The rate at which *n* is turning in the direction perpendicular to *w* at the point *p* has no effect on the normal curvature in the *w* direction (though it certainly affects the normal curvature in the direction perpendicular to *w*). To see this, consider a unit speed normal section curve through *p* with tangent vector $w_p$. The effect of *n* turning in the direction perpendicular to *w* is to twist the curve about its axis, which does not change its curvature.

Let *u* and *v* be the coordinate functions on $[0,1]^2$. The matrix of the second fundamental form in terms of *du* and *dv* is

$$\begin{bmatrix} L & M \\ M & N \end{bmatrix},$$

where $L = c_{uu} \cdot n$, $M = c_{uv} \cdot n$, and $N = c_{vv} \cdot n$. The curvature information can be extracted by premultiplying the matrix of *II* by the inverse of the matrix of the first fundamental form *I*. It can be shown that the eigenvalues of the matrix $[I]^{-1} \cdot [II]$ are the principal curvatures $\kappa_1$ and $\kappa_2$ and that the eigenvectors of this matrix are the principal directions. A remarkable fact is that the second fundamental form and the metric completely determine the geometry of a surface.

Theorem A-2 (The Fundamental Theorem of Surfaces). Let $c_1 : [0,1]^2 \to \Re^3$ and $c_2 : [0,1]^2 \to \Re^3$ be regular surface patches. If both the first and second fundamental forms of $c_1$ and $c_2$ are equal, then $c_1([0,1]^2)$ and $c_2([0,1]^2)$ are congruent; that is, they can be superimposed by a rigid motion.

A proof can be found in [ O'Neill 66 ]. Intuitively, this theorem says that the shape of a surface is fully determined by the metric and the curvature up to rigid motions.

References. We have barely scratched the surface of the field of differential geometry. For more detail, here are some good reference books on the subject. *Elementary Differential Geometry,* by O'Neill is a good introductory text. *Differential Forms with Applications to the Physical Sciences*, by Flanders is a book on applications of differential forms written for scientists and engineers. *Solid Shape*, by Koenderink provides an intuitive, visual approach to the subject with many diagrams and verbal explanations. *A Comprehensive Introduction to Differential Geometry* by Spivak is a five volume survey of the field for the mathematically sophisticated.

# Bibliography


[ Alpha_1 90 ] *Alpha_1 User's Manual*, Engineering Geometry Systems, Salt Lake City, Utah, May 1990

[ Barnhill 84 ] R.E. Barnhill and R.F. Riesenfield, ed., *Computer Aided Geometric Design*, Academic Press, 1984

[ Barr 84 ] A.H. Barr, "Global and Local Deformations of Solid Primitives," *Computer Graphics,* vol. 17, no. 3, pp. 21-30, July 1984 (SIGGRAPH 84)

[ Barsky 81 ] B.A. Barsky, *The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures*, Ph.D. Thesis, University of Utah, Salt Lake City, Utah, December 1981

[ Barsky 90 ] B.A. Barsky and T.D. DeRose, "Geometric Continuity of Parametric Curves: Construction of Geometrically Continuous Splines," *IEEE Computer Graphics and Applications*, vol. 10, no. 1, pp. 61-62, January 1990

[ Bulirsch 80] R. Bulirsch and J. Stoer, *Introduction to Numerical Analysis*, Springer-Verlag, 1980

[ Celnicker 91 ] G. Celnicker and D. Gossard, "Deformable Curve and Surface Finite-Elements for Free-Form Shape Design," *Computer Graphics*, vol. 25, no. 4, July 1991 (SIGGRAPH 91)

[ Chadwick 89 ] J.E. Chadwick, D.R. Haumann, R.E. Parent, "Layered Construction for Deformable Animated Characters," *Computer Graphics*, vol. 23, no. 3, July 1989 (SIGGRAPH 89)

[ Chiyokura 88 ] H. Chiyokura, *Solid Modeling with Designbase*, Addison-Wesley, 1988

[ Cobb 84 ] Elisabeth S. Cobb, *Design of Sculpted Surfaces using the B-spline Representation*, Ph.D. Dissertation, Dept. of Computer Science, University of Utah, 1984

[ Coons 67 ] S. Coons, *Surfaces for Computer Aided Design of Space Forms*, Technical Report, MIT, 1967, Project MAC-TR 41.

[ Coquillart 90 ] S. Coquillart, "Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling," *Computer Graphics*, vol. 24, no. 4, August 1990 (SIGGRAPH 90)

[ DeRose 85 ] T.D. DeRose, *Geometric Continuity, a Parameterization Independent Measure of Continuity for Computer Aided Geometric Design*, Ph.D. Dissertation, technical report UCB/CSB 86/255, August 1985

[ DeRose 88 ] T.D. DeRose and B.A.Barsky, "Geometric continuity, shape parameters, and geometric constructions for Catmul-Rom splines," *ACM Transactions on Graphics*, vol. 7, pp. 1-41, 1988

[ Duncan 80 ] J.P. Duncan and G.W. Vickers, "Simplified Method for Interactive Adjustment of Suraces," *Computer Aided Design,* vol. 12, no. 6, pp. 305-308, November 1980

[ Farin 90 ] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1990

[ Flanders 63] H. Flanders, *Differential Forms with Applications to the Physical Sciences*, Academic Press, 1963

[ Forsey 88 ] D.R. Forsey and R.H. Bartels, "Hierarchical B-spline Refinement," *Computer Graphics,* vol. 22, no. 4, August 1988 (SIGGRAPH 88)

[ Gregory 89 ] J.A. Gregory, "Geometric Continuity," pp. 353-371, *Mathematical Methods in Computer Aided Geometric Design*, ed. T. Lyche and L. Schumaker, Academic Press, 1989

[ Hagen 87 ] H. Hagen, and G. Schulze, "Automatic Smoothing with Geometric Surface Patches," *Computer Aided Geometric Design*, vol. 4, pp. 231-236, 1987

[ Hahn 89 ] J.M. Hahn, "Geometric Continuous Patch Complexes," *Computer Aided Geometric Design*, vol. 6, pp. 55-67, 1989

[ Halstead 93 ] M. Halstead, M. Kass, and T. DeRose, "Efficient, Fair Interpolation using Catmull-Clark Surfaces," *Computer Graphics*, Annual Conference Series, August 1993 (SIGGRAPH 93)

[ Himmelblau 72 ] David M. Himmelblau, *Applied Non-Linear Programming*, McGraw Hill, 1972

[ Koenderink 90 ] Jan J. Koenderink, *Solid Shape*, MIT Press, 1990

[ Loomis 90 ] Lynn H. Loomis and Shlomo Sternberg, *Advanced Calculus*, Jones and Bartlett, 1990

[ Loop 90 ] C. Loop and T. DeRose, "Generalized B-spline Surfaces of Arbitrary Topology," *Computer Graphics,* vol. 24, no. 4, August 1990 (SIGGRAPH 90)

[ Lott 88 ] N.J. Lott, and D.L. Pullin, "Methods for Fairing B-Spine Surfaces," *Computer Aided Design*, vol. 20, no. 10, December 1988

[ Moreton 92 ] Henry P. Moreton and Carlo H. Sequin, "Functional Optimization for Fair Surface Design," *Computer Graphics*, vol. 26, no. 2, July 1992 (SIGGRAPH 92)

[ O'Neill 66 ] B. O'Neill, *Elementary Differential Geometry*, Academic Press, 1966

[ Perry 86 ] S.R. Perry, *Free-form Deformations in a Constructive Solid Geometry Modeling System*, Ph.D. Dissertation, Department of Civil Engineering, Brigham Young University, April 1986

[ Sabin 76 ] M.A. Sabin, *The Use of Piecewise Forms for the Numerical Representation of Shape*, Ph.D. Dissertation, Budapest, 1976

[ Sederberg 86 ] T.W. Sederberg and S.R. Perry, "Free-Form Deformation of Solid Geometric Objects," *Computer Graphics*, vol. 20, no. 4, August 1986 (SIGGRAPH 86)

[ Spivak 70 ] Michael Spivak, *A Comprehensive Introduction to Differential Geometry*, Vol. I., Publish or Perish Press, 1970

[ Strang 80 ] Gilbert Strang, *Linear Algebra and Its Applications*, Academic Press, 1980

[ Terzopoulos 87 ] D. Terzopoulos, J. Platt, A. Barr, K. Fleischer, "Elastically Deformable Models," *Computer Graphics*, vol. 21, no. 4, July 1987 (SIGGRAPH 87)

[ Vernon 76 ] M. Vernon, G. Ris, and J.P. Musse, "Continuity of Biparametric Surface Patches," *Computer-Aided Design*, vol. 8, no. 4, October, 1976.

[ Voelcker 93 ] Herbert B. Voelcker and Aristides A. G. Requicha, "Research in Solid Modeling at the University of Rochester: 1972-87," from *Fundamental Developments of Computer-Aided Geometric Modeling*, ed. Les A. Piegl, Academic Press, 1993

[ Warner 71 ] Frank.W. Warner, *Foundations of Differentiable Manifolds and Lie Groups*, Scott, Foresman and Co., 1971

[ Welch 92 ] W. Welch and A. Witkin, "Variational Surface Modeling," *Computer Graphics*, vol. 26, no. 2, July 1992 (SIGGRAPH 92)