REPRESENTATION OF AND MODELING WITH ARBITRARY DISCONTINUITY CURVES IN SCULPTURED SURFACES

by

Marc S. Ellens

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

The University of Utah

August 1997

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Marc S. Ellens

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Elaine Cohen

Richard F. Riesenfeld

Jamie Painter

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Marc S. Ellens in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Elaine Cohen
Chair, Supervisory Committee

Approved for the Major Department

Tom Henderson
Chair/Dean

Approved for the Graduate Council

Ann W. Hart
Dean of The Graduate School

ABSTRACT

Representing data after it has undergone a fundamental topological change, such as cracking, ripping, or folding, or after the introduction of arbitrary feature curves, as happens during the creation of darts, corners, or fractures, continues to be a significant challenge. Ideally, the representation is modified without having to reformulate the representation entirely. If the original model is composed of faceted polyhedra, it is possible to do this. However, many models today are being represented by smooth parametric tensor product surfaces such as B-splines, which do not easily support arbitrary discontinuities. During the design process, when discontinuities are introduced, such models are often tesselated into triangles, which would henceforth be the model's representation. In this case, the resulting model is often not useful for further design. This thesis introduces an extension of the B-spline surface representation, called the *torn B-spline surface*. The torn B-spline representation provides flexibility not previously found in similar parametric surfaces by incorporating *tear* curves, *crease* curves, and other arbitrary $C^{(-1)}$ feature curves into the representation itself. Simulation events or other design processes which result in discontinuities in the representation do not necessitate a change in representation, and it is possible to use B-spline design methods on the resulting torn surface model. This makes design with discontinuities more viable. The representation and associated algorithms used to support it are introduced, as well as some higher-order design operators which take advantage of this representation and some example applications.

To Julie, Karys, and Serena.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

This work was made possible by the advice and support of a great number of people. Thanks go to: my wife and family for putting up with the long hours and lack of attention (I am going to be home more now); Elaine Cohen and the rest of my committee, for excellent technical advice; my coworkers, for putting up with my constant jabber; my fellow graduate students, for not letting me give up and providing someone to compete with ("It's not my fault, really!"); Beth Cobb, for her clear-thinking advice and friendship; my friends at Mountain Springs, for pushing me to finish this without knowing what it was I was doing—and loving me anyway; my parents, for being my parents; and God, for being my God. May I never have to do this again.

This work was supported in part by DARPA (N00014-92-J-4113) and the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinions, findings, conclusions, or recommendations expressed in this document are mine and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

In the field of computer-aided geometric design (CAGD), use of an appropriate representation is the key to effectively conveying a structure's geometry. There are many tradeoffs among the representations, based on factors such as whether or not the data accurately represents the original model, which will be referred to as *fidelity*, and the number of common operations under which the set of representable models is closed which measures the *completeness* of the representation. Other factors to be weighed include ease of use, size, speed, and flexibility. The flexibility of a representation is measured in terms of the number of its supporting operations in the same way a mathematician may evaluate an algebra in terms of the number of common operators that can be applied. Consider the tradeoffs between polygonal representations, various parametric tensor product surface representations, and constructive solid geometric (CSG) representations. In the ideal world, all shapes would be represented exactly and the particular representation would not be an issue. However, the real world usually has more complexity than we are able to represent, so we approximate. As a rule, the more complexity a model has, the more it is approximated. The triangle or otherwise faceted representation is flexible and closed under most operations at the expense of larger size and decreased fidelity. The tensor product B-spline representation is compact and easy to manage at the expense of being slower and more incapable of representing the results of some operations. A CSG representation is even more compact and easier to understand and use while being even less flexible. However, technological advances in computer speed and memory have reduced the impact of the size and speed requirements of a representation. The crucial factors have become fidelity, completeness, flexibility, and ease of use.

The driving force behind this research is the need to provide a representation for which operations that introduce discontinuities into tensor parametric surfaces are closed. Operations in which discontinuities are introduced as a result of a simulation or other automatic process are particularly difficult since the designer often has little control over the outcome and the results are not easily representable by current parametric tensor product surface representations. Consider the following examples where parametric tensor product surfaces are likely to be used.

The first example is the much studied area of thin plate deformations[74]. Thin plate deformations are used for modeling the behavior of everything from cloth[81] to steel. Thin plate dynamics can be modeled with simple springs and dashpots in a linear constraint/feedback system or in a complex nonlinear finite element optimization system. In either case the physical characteristics of the plate may cause the plate to tear or fracture under stress, causing a change that is probably not supported by the model's representation. The simulation results may be translated into a different representation which may be capable of representing the discontinuity but fails to retain the smoothness information within the rest of the model. Sometimes the model is reconstructed with explicit constraints holding together the new edges with the old smoothness information. All cases result in additional work for the designer, particularly if the results will be used in the context of further design operations or analysis. Ultimately, information about the model is lost during this process.

Another related example is *stamping*[27, 3, 36]. In this process, a thin malleable material is forced under pressure to assume a particular shape by compressing the material between two forms. The simulation of this process is extremely difficult since there is high pressure and heat, both of which may alter the state and the dynamics of the original material. Fractures, tears, and creases are common and cause difficulty in both the simulation and the representation of the model itself.

In the field of geology we see another example of physical simulation resulting in unrepresentable complex shapes. Here the discontinuities are three-dimensional as earthquakes and other natural forces cause rock layers to fracture and slide past one another creating complex systems of slips and folds. An initial representation may be parametric tensor product surfaces that form the boundaries of the rock layers, stacked on each other to make a *nonmanifold* three-dimensional model. Once the rock layers separate, indication of the three-dimensional nature of the crack is virtually impossible with current parametric tensor product representations.

In the medical field, physicians can simulate procedures such as plastic and reconstructive surgery[62] allowing them to make better planning decisions. Current representations in this area are polygonal although actual skin is rarely faceted. The ability to support arbitrary continuity features in parametric boundary representation models could be well used by this field.

Finally, a designer using a CAD system may want to include continuity features within a surface, such as creases or tears. These features may drive a particular functional aspect, such as aerodynamics, or an aesthetic aspect of the design.

Currently, triangles are the most common representation in these cases, because they are flexible and easy to use and they support the arbitrary topologies and continuity features which may result. However, design with parametric surfaces, and in particular, tensor product B-splines, is becoming more prevalent. Current design techniques need to provide adequate support for these surfaces and the by-products of their design processes. Ideally, a design operation that introduces discontinuities should result in models that are members of the original representational set. Currently this is not the case.

In these cases, flexibility and fidelity appear to be the two most critical factors. To produce an accurate model, the initial representation must be accurate and all prior operations must retain that information as well as reliably incorporate new information during the process. Of particular concern are the continuity features contained in the representation. Flexibility is the key to obtaining an accurate initial representation, whereas fidelity is the key to retaining the accuracy through modifications. The faceted representations usually consist of many small facets that together approximate the continuity information present in the large model.

Usually continuity information is maintained by an equation which measures the energy present between adjacent facets and so determines a smoothness measure of the model. Continuity features in the larger model are usually identified along the boundaries between facets and the energy equation is suitably modified to reflect the change in continuity. Higher-order tensor product surfaces provide a more accurate initial model but are unable to represent all the continuity features which may be introduced. A *feature curve* cannot be introduced into a tensor product surface without significantly altering the representation. This, in many cases, prevents the use of some available design techniques since this type of smoothness information cannot be represented by a single tensor product surface.

The *torn B-spline surface* representation, initially presented in [29], is designed to bridge the gap between the faceted representations and the higher-order parametric representations. It provides the geometric flexibility of the faceted representations while providing the fidelity and size of the parametric representations. The key elements of the torn B-spline representation are the arbitrary $C^{(-1)}$ feature curves known as *tear curves*. The basis for this representation, the representation itself, and several of the more common evaluation routines applicable to this class of surfaces will be introduced, as well as some higher-order design operators which demonstrate the flexibility of the representation within a design system. Finally this representation will be applied to the problems in the examples introduced, demonstrating the effectiveness of the representation.

CHAPTER 2

BACKGROUND

The foremost consideration when assessing a representational need is determining the best representation class for the job in terms of flexibility and fidelity.

2.1 Representation Classes

Within a representation class, a given representation has the power to represent a particular set of models. The representation usually consists of a basic element, such as a triangle or surface, and its set of representable models can be classified by whether or not collections of these basic elements are used. In addition, representations vary in how closely they can approximate a given object, providing further classification.

A given representation also supports design operations, called *methods*, which are particular to that representation. For example, refinement is usually identified with parametric surfaces, in particular, those which have basis functions, such as tensor product B-splines. Other representations within this same class attempt to provide similar operations. The resulting models of these representation-linked operations generally stay within the same representation. Thus the set of representable models is closed with respect to these operations.

In addition to being used with their methods, representations can be used in a variety of design processes (often called operations as well) which can be modified to support a variety of representations, even those in other classes. For example, the tensor product B-spline and triangular-faceted representations can both be used in a physically based modeling process. The results of this set of operations may fall outside of the original representation's set of representable models, sometimes even outside the set of representable models for the class. Therefore, the representation or the representational class may not be closed with respect to these operations.

During the course of a design session, a designer constructs a model, modifies the model, possibly simulates some aspect of the model's function using the model itself, or makes modifications to the model through some automatic process. Then he may make changes to the model in response to feedback or examination of the resulting model's structure; which starts the process over again. If, at some point, the representation changes as a result of a particular operation, the feedback loop is altered. Design operations, in particular, methods, previously used for construction and modification may no longer be available and the designer may be incapable of making the necessary modifications without starting over. This is particularly true when the resulting model contains characteristics necessary for the final model.

Changing the representation class of the model during the modeling process can cause significant problems and force designers into using more cumbersome representations, dealing with less accuracy, choosing less intuitive design procedures or ultimately settling for less than what is required by the design specifications.

2.2 To Facet or Not to Facet...?

Within a complex design system, the choice of an internal representation affects both the interaction that the designer has with the system and the final outcome of the design process. If the choice were simple, there would not be much difference among design systems and the opinions that gave rise to them. As it is, design systems range from very low order, such as points with adjacency information (a.k.a. facets), to very high order, such as implicit surfaces[2]. Since many real-life objects have smooth, sculptured shapes, the challenge with the low-order faceted representation is to make the models look and behave like higher-order models without the size of the model becoming prohibitive[71, 46, 47]. The challenge with higher-order representations is to make them easy to use in a practical design system.

One of the primary reasons faceted representations are used within a design system is that they are well understood and well supported. Higher-order representations can be difficult to construct, manipulate, and simulate and are only recently being found in the larger commercial design systems. In contrast, the finite element method, the most popular simulation technique, caters to faceted representations (in two-dimensional simulations) since the meshes consist of interconnected data points (although higher-order physical relationships between mesh points are often used). Many design systems use tessellated models for this reason alone.

However low-order representations have disadvantages. Along with the size, which causes these representations to be difficult to manage and manipulate, another disadvantage is appearance. Faceted models have angular silhouettes over curved portions of their surfaces and can be subject to unwanted mach banding at adjacent edges[6]. In addition, higher-order representations are being actively investigated and are increasingly being used in commercial design systems. The computation time often consumed with higher-order representations is being countered with faster computers, making these representations a viable alternative to the traditional approach of using faceted models in a design system. Despite these disadvantages, facets are still popular and are implemented in systems which support a large range of design capabilities.

2.3 Tensor Product B-splines

Since tensor product B-splines have become commonplace in major design systems, it is important to understand the capabilities of the representation and have a clear idea of the extensions that can and need to be made to support the desired design operations. This thesis introduces the torn tensor product B-spline, a tensor product B-spline surface representation for which the set of representable models is closed under most design and simulation operations which may introduce discontinuities. This closure problem is difficult because the tensor product B-spline surface representation does not support discontinuities of arbitrary geometry. Since tensor product B-spline surfaces are being used more extensively in design processes, alternate representations which satisfy this closure requirement are more urgently needed. The problem is further characterized by looking closer at the examples mentioned earlier and the situations in which the need for representations closed under these operations arises and causes difficulty.

2.4 Physically Based Modeling

Several of these examples fall in the category of physically based modeling. Unfortunately, physically based modeling is a very general term which can be applied to just about any design or simulation environment which uses physical characteristics to help define the model. Even the use of lighting models such as radiant illumination (radiosity) could be considered physically based modeling because the resulting image is a product of a (albeit simplified) physical simulation. More commonly, however, physically based modeling refers to the use of physical characteristics to determine a model's shape, position and/or orientation in space. Material characteristics like mass, density, moments of inertia, elasticity, and plasticity combined with physical behaviors in context such as gravity, collision detection, and connectivity are examples of the physical characteristics considered in these design operations. A subclass of these problems deal with the physical characteristics of a single model. Multiple models require additional linkage and kinematic information. This thesis primarily addresses the single model case. The model may be a solid model whose basic elements have volume, or a boundary representation whose basic elements are surfaces and connected by constraints at the edges. Models discussed in this thesis are composed of parametric surfaces. Specifically, this thesis addresses manifold and nonmanifold boundary representations.

2.4.1 Thin Plate

The simplest and most well-understood dynamic simulation is that of the thin plate under tension[74, 39]. The thin plate problem is kept simple by assuming that the thickness of the plate does not contribute anything significant to the formulation of the problem. This allows a two-dimensional simulation of the plate which substantially reduces the complexity and the solution time. For the most part, standard thin plate dynamics is uninteresting since most simulations use a variation of the standard dynamics formulation. However, the use of continuous surface representations for thin plates can be difficult if the dynamics can support discontinuities such as tears or fractures. Although they may seem avoidable, these situations may occur in any instance in which plasticity is used. A case could be made that a model of dynamics without the capacity for fracture is not an accurate model, since it more accurately reflects real life.

Plasticity in materials engineering is most often represented by a stress-strain graph. The stress is the amount of force applied to a material, and the strain is the amount of deformation caused by the force. Although simplified greatly, the graph in Figure 2.1 is useful for reference[81]. Normally, a material's stress-strain graph is nonlinear and changes according to the history of the stress on the material. The elastic limit point (E) is the point up to which removal of the forces will cause the material to return to its original rest state. Stress beyond the elastic limit will cause the material to permanently deform. The material breaks when stressed beyond the breaking point (B).

Most parametric surface representations break down when the material reaches the breaking point. Either the discontinuity is ignored or a secondary representation is used, such as a triangular or other faceted tessellation or the visual representation

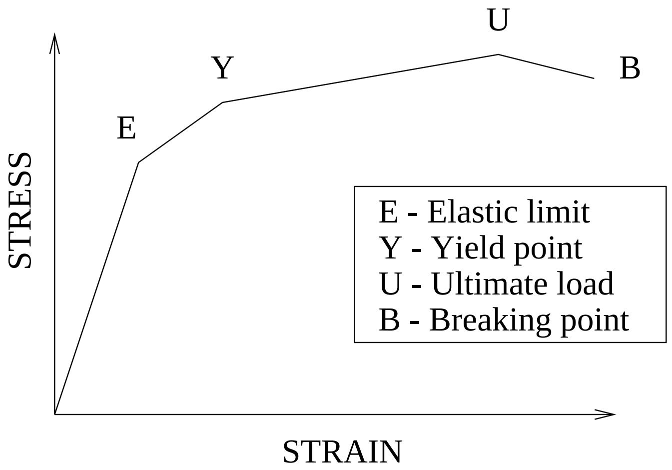


Figure 2.1. Stress-strain graph.

of the physical elements (e.g., plastic springs between control points[81]). In either case, the set of representable models is no longer closed under this simulation operation and the resulting representation may not retain smoothness information which is critical to the model.

2.4.2 Stamping

A more complex situation is the dynamics of stamping. In this manufacturing process, a thin sheet of material called a *blank* is cut to a specified shape and loaded into a press. The material is then deformed by forcing the material into a die by means of a punch. Although thin-plate dynamics play a large roll in the simulation of this process, a better simulation requires the use of material thickness and viscosity. The four general categories of critical problems in sheet metal forming are fracture, wrinkling or buckling, undesired sheet deformation, and springback[43, 36]. In some cases, however, these "problems" are not really problems but desired features. In the case of lance or emboss punching, (see Figures 2.2 and 2.3), the resulting fracture and the accompanying buckle are part of the design[27]. These simulation results need to be represented as accurately as possible. Although simulation of stamping processes are traditionally done using FEM, designing with higher-order parametric surfaces is more common, and the need for a consistent representation throughout the design process is becoming more evident.

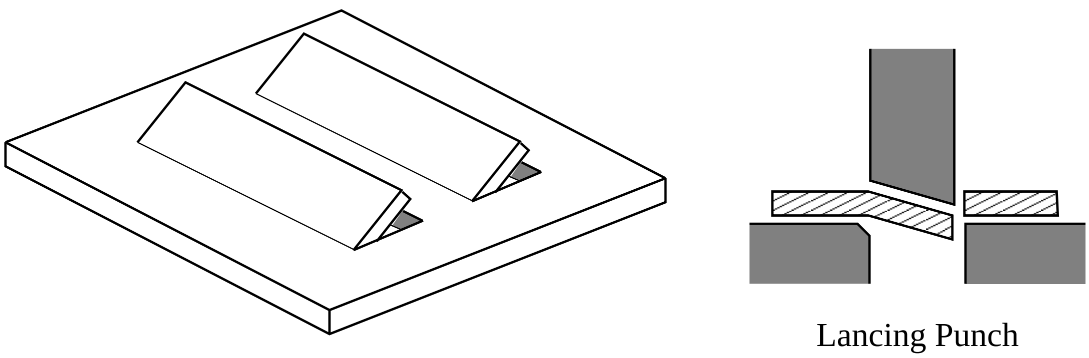
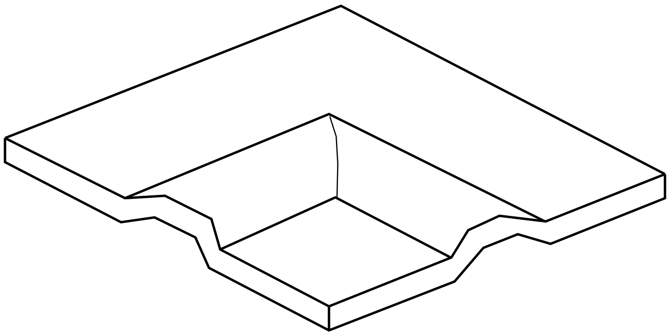


Figure 2.2. Lance forms.



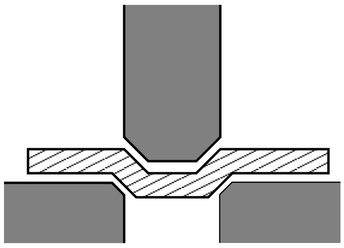


Figure 2.3. Embossing.

2.4.3 Geology

The study of fractures in rock layers is extremely complex and the representations used in the simulation of these interactions are inadequate at best. A good example of the need for discontinuities within a representation comes from the interaction of layers when a lower layer fractures and an upper layer deforms, or folds over the fracture (see Figure 2.4). This is known as *drape-folding*[89]. The different compositions of the layers may even cause the layers to separate resulting in a pocket between the layers. Other fractures may extend through the layers requiring that the representation track the fracture propagation through the different materials. The process of shearing also can create fractures and creases in the layers of material which are deformed[66]. Currently most simulation of geological phenomena is done with faceted representations using a constraint system solved by the finite element method. This is because of the frequent use of the finite element method in engineering disciplines and the current difficulty of representing fractures and other discontinuities in higher-order representations.

2.4.4 Medicine

Recently, Peiper has created some interest in surgical simulation, specifically plastic surgery[62, and references therein]. He primarily uses a triangular finite element mesh to represent the surface tissue during a finite element simulation. The ability to represent discontinuities in higher-order surfaces could have a serious

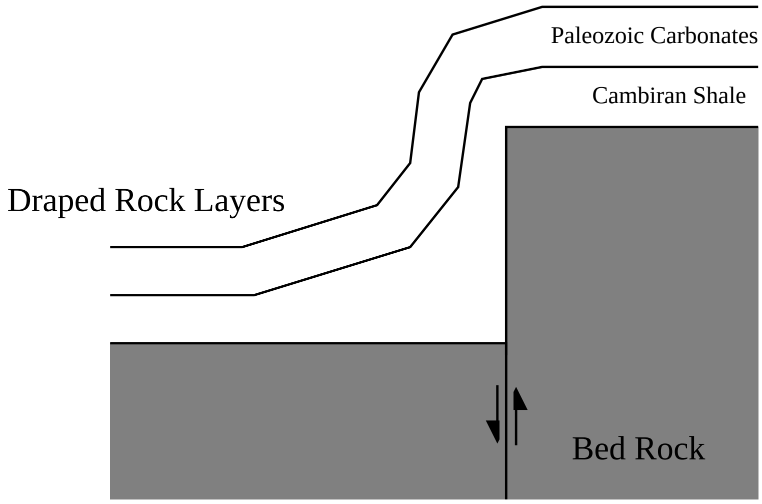


Figure 2.4. Drape folding.

impact on the planning and performance of various plastic surgery techniques. Currently most facial models are represented by polygons[84, 87]. However, parametric surfaces, and tensor product B-splines in particular, would be ideal candidates for representing skin, especially facial tissue, since smooth faces are more visually realistic for simulation. Although smoothness is generally not required for traditional animation[84], the addition of discontinuities to these smooth representations would provide a more realistic result for simulation of plastic and reconstructive surgery.

2.5 Design Techniques

The design processes in the above examples all employ similar techniques which are reviewed in this section. The most common techniques are methods for optimization of functions within the context of solving a system of constraints.

2.5.1 Constraints

There are a variety of ways to solve constraints depending on the characteristics of the constraint functions. For example, systems of linear constraints can be solved by singular value decomposition or QR factorization[72]. Most systems of constraint equations used for interactive physically based modeling are linear approximations to nonlinear systems[21, 35, 34] due to the speed and ease of solving these types of equations. If the process of solving the constraint is itself interesting, as it often is in the case of animation, there are other methods which can be used to iteratively obtain a solution in which the intermediate values are interesting. These constraint solving methods fall into three categories: 1) optimization, 2) dynamic constraints, and 3) reaction constraints[65].

2.5.2 Optimization

The most common of the optimization methods are penalty methods and Lagrangian constraints[64]. Other methods include finite differencing, simulate annealing, augmented Lagrangian constraints, and a host of other derivations of these methods[67]. It is these methods paired with a finite element mesh which comprise the finite element method (FEM)[10]. One difficulty of general optimization techniques is that they may require the computation of the gradient of the function being minimized or maximized and often these derivatives do not exist in closed form. Another difficulty is that most iterative optimization techniques are susceptible to getting caught in local minima or maxima, ultimately failing to find the globally optimal solution. In addition, intermediate values may not have any physical meaning, which may be a criteria if the optimization is being carried out in the context of an animation.

2.5.3 Dynamic Constraints

Dynamic constraints[7, 49, 4, 25] are systems of constraints which are solved by applying critically damped forces which are computed by inverse dynamics. The forces are thought of as occurring though time and therefore result in interesting intermediate solutions. Unfortunately, these techniques are often difficult to use because of their nonlinearity and large number of variables. Dynamic constraints are most often used in systems involving elasticity where the equations can be simplified[81, 88, 32](see Section 2.5.5).

2.5.4 Reaction Constraints

Another area of force-based constraint solving are reaction constraints. Additional forces are added to the system at the appropriate points in order to prevent the constraints from being violated. Typical uses of reaction constraints include path following and interpenetration prevention[5, 57]. The principal advantage to reaction constraints are that they are simple to compute. However, they often do not bear up well when applied to more complex problems[65].

2.5.5 Model Deformation

The animation industry has fueled interest in the theory of model deformation, with most of the theory developed from a physical basis. Elasticity and plasticity derived from the thin-plate model have been used in many instances to provide realistic motion and deformation. The standard methods minimize a variational derivation of an energy functional (similar to the thin-plate energy functional) over the model as a whole. The majority of the earlier work uses triangles to represent the final model[50, 77, 80, 76, 75, 78, 37, 38, 91, 20, 86, 15, 85]. It is only recently that these methods have been applied to splines in the works of Bloor and Wilson[13, 14], Welch and Witkin[90], Moreton and Sèquin[58], Celniker and Welch[21] and Terzopoulos and Qin[79, 68]. A unique method for deforming models based on vibration modes was developed by Pentland[63]. This "modal" dynamics method is only applicable to models described by closed-form functions.

2.6 Finite Elements

The finite element method is easily the most widely used simulation technique today[10, 48], and finite elements are used so often in physical simulation this technique is addressed separately.

The finite element method is essentially the optimization of a system of linear or nonlinear constraints which are approximations to systems of partial differential equations or integral equations. Boundary value problems provide additional constraints on the boundaries which prevent degenerate solutions. The reasons for the finite element method's popularity stem from its flexibility and its extensive resource and support base. The method is well studied and many ready-to-use implementations are publicly available. However, there are some significant drawbacks to this popular technique. An FEM solution requires a well-placed mesh, and although there has been much work in this area (e.g., [55]), it continues to be a problem. In addition, because the systems of equations are generally nonlinear, the method is slow. Only in very simple cases in which the systems are extremely well behaved, sparse, or linear can solutions be obtained in interactive speeds[21, 38]. Finally, the finite element mesh elements are usually represented by facets because the differential equations are easier to form. Therefore the results of the simulation are often not acceptable within the framework of higher-order parametric surface design systems. There have been proposed integrations of higher-order parametric surfaces into the finite element world (e.g., [94]), but none of these solutions can support the fracture and other discontinuities which can easily result from the finite element method and are more easily represented by facets.

To summarize, the applications which would benefit by incorporating discontinuities within the representation are varied despite the fact that the design techniques for generating these situations are standard. However, the problems associated with representing discontinuities are not easily solvable by the prevailing methods. Tradeoffs are required to obtain adequate results depending on the application. In Chapter 3, capabilities of existing representations and previous attempts to address these issues are discussed.

CHAPTER 3

PREVIOUS WORK

Discontinuities introduced during the design process pose difficulties when using higher-order representations. Although the representation can be changed to accommodate the new structure, the design process is disrupted and the resulting representation may not support design operations similar to those available for the original representation. The approaches to solving this representational closure problem include the following:

1. Use as an original representation *a lower-order, generally faceted, representation* which is known to be closed under operations which introduce discontinuities.

2. Use as an original representation *a higher-order, usually parametric, representation* with enough degrees of freedom to represent simple cases. Degeneracies may be introduced to support various continuity features. Representation reverts to faceted representation once a certain complexity is reached.

3. Use as an original representation *a modification or extension of a higher-order representation* which is closed under a number of operations which introduce discontinuities.

The following section reviews the representation classes and how they attempt or could attempt to address the representation closure problem.

3.1 Faceted Representations

Faceted representations have the advantage that they are easy to understand and to implement. The primary disadvantage of the faceted representation is that the collection of facets only approximates smoothness to within some tolerance. If the tolerance is dramatically reduced, the size of the data structure may expand prohibitively. Manipulation of models represented by facets requires some knowledge of the characteristic smoothness of the surface at a higher level than individual facets. Ultimately, this is no different than the problem of representing discontinuities in a higher-order model. On the other hand, the topological flexibility for faceted representations is limited only by the size of the model. If a faceted representation is used for a physically based simulation, the representation will not need to dramatically change even if discontinuities are introduced[73]. Despite these advantages, higher-order surfaces are being used consistently in design systems and useful representations which are closed under these design processes are needed.

3.2 Parametric Surfaces

Parametric surfaces come in many flavors, the most popular of which are multivariate splines. The tensor product B-spline representation is a special case of multivariate splines and one of the more widely used parametric surface representations

3.2.1 Multivariate Tensor Product Splines

Each tensor product B-spline surface blend function is actually just the product of two univarite B-spline blend functions which govern the blending of control points to describe the surface. One of advantages of the tensor product B-spline representation is the fact that it is simple and that algorithms such as refinement and order manipulation are well known. Unfortunately, the support of tensor product surfaces is parametrically rectangular; therefore, any degenerate knot configuration which may contribute to a discontinuity is present along the entire parametric isoline. Most other multivariate splines fall into the category of alternate higher-order representations[22]. Unusual multivariate splines (such as Box splines[59]) do not necessarily have rectangular parametric domains, and so any degenerate knot sequence which creates a discontinuity although isoparametric in nature does not necessarily lie in a particular direction across the surface.

However, these other representations have yet to make a significant impact in design for several possible reasons. First, they are more difficult to understand so are less attractive for use in commercial products. Second, standard components present in other parametric surfaces such as simple basis functions and order relationships are not generally present for multivariate splines. This makes analysis for simulation and interactive design processes very difficult. In addition, the complexity and irregularity of the blending functions make this general class of representations slow. Finally, arbitrary discontinuity within the surface is generally not supported.

The tensor product torn B-spline surface representation presented in this thesis may also be applicable to general multivariate splines. This type of extension is left for future work.

3.2.2 Subdivision Surfaces

Subdivision surfaces are another class of surfaces which demonstrates potential for being able to represent surfaces of arbitrary topological type and embedded discontinuities. These surfaces are constructive surfaces based on a parameter and a control net corresponding to the connectivity of the base-level surfaces. The most common subdivision type are triangles[51, 46, 47, 45] although both quadrilaterals[28] and biquadratic and bicubic tensor product B-splines[19] have been used for the base-level surface type. The topological flexibility of these surfaces is striking, yet several things stand in their way to becoming the surface of choice in a design system. First, simulation and interactive manipulation of these surfaces are difficult because the surface is constructive. Specifically, it is not clear how discontinuities can be added to the model after the model is constructed simply because there may be no clear way to translate the information back to the control mesh. Current implementations deal only with surface reconstruction[46, 47, 45], not interactive manipulation. In addition, differential properties of these surfaces can be difficult to obtain since the mathematical formulation is constructive.

3.3 Alternate Parametric Surface Representations

Most of the development in alternate representations of parametric surfaces has used the general class of splines as the base. The development is spurred by implementations of Béziers, B-splines or NURBS in in popular commercial design systems (e.g., AutoCAD, SoftImage, EZFeatureMill, SurfCAM, 3DStudio). Changing the spline representation has been a popular technique, among the results are γ-splines or $G^{(2)}$-splines, β-splines[30], ν-splines or tension splines[61], G-splines[44], Box splines (a type of multivariate spline)[59], Hayes splines[41], and X-splines[11]. Only the last two have specifically addressed the representation of discontinuities within individual surfaces, but each at the expense of making it difficult to support the more common spline methods of refinement and subdivision. The most interesting, yet least useful, representation is Hayes splines[41]. Hayes splines provide a functional definition of the knot vector with respect to the opposing parametric value. Hayes splines can represent partial discontinuities in a parametric direction, although they need not be isoparametric. One of the obvious difficulties with this representation is the complexity imposed by this additional level of indirection. Hayes splines are hard to describe and substantially harder to use. Recently, X-splines[11] were introduced as a combination of B-splines and Catmull-Rom splines[30]. They appear to be easier to use than Hayes splines, although still providing partial discontinuity across a single spline surface, but the discontinuities are still isoparametric. In addition, the surface is not a true tensor product because of a normalizing factor, so common spline methods such as refinement and subdivision have different meanings.

The alternate representations are generally very complex and too difficult to control for widespread application. In addition, the number of operations for which the set of representable models are closed is quite small.

3.4 Trimmed B-spline Surfaces

Another modification often used with parametric tensor product surfaces like B-splines is a *trimmed* representation. A trimmed surface is a surface whose original domain has been restricted to a set of closed subregions of the original domain. Early techniques developed by Thomas[82] and Carlson[16, 17] approximated the trimmed surface within the restricted domain with a set of polygons. Sarraga[70] suggested using a collection of rational tensor product surfaces to approximate the trimmed surface. However, in each of these techniques the original surface representation is different from the trimmed surface representation, making modifications within the same framework difficult. Representing the trimmed region by an unevaluated two dimensional CSG tree was suggested by Casale[18] but evaluation of this surface can be slow and tedious. Representing the boundaries of the trimmed region by algebraic curves was suggested by Farouki[31], but this technique does not scale well to general tensor product B-spline surfaces. Another technique was developed at the University of Utah by McCollough[56]and uses a parametric curve evaluated in the domain of the surface to represent the boundary of the trimmed region (see Figure 3.1). Trimmed B-spline surfaces are often used in solid model boundary representations in which the boundary surfaces are nonrectangular. The computation of intersections of higher-order parametric surfaces such as B-splines is generally not tractable, and the resulting intersection curves are usually not representable by simple parametric curves embedded in the surfaces (see Figure 3.2). In McCollough's representation, the actual intersection is approximated by a piecewise linear intersection curve along with the corresponding parametric locations of the individual points within each of the surfaces (see Figure 3.3). One major disadvantage to this representation is that a large amount of data is needed to represent the boundaries of the trimmed region when an adjacency is involved.

At first it may seem that the trimmed B-spline representation would support general discontinuities, but there are serious drawbacks to using this representation. First, even though complex topologies can be represented, regions which appear independent by visual cues may not really be independent in the underlying

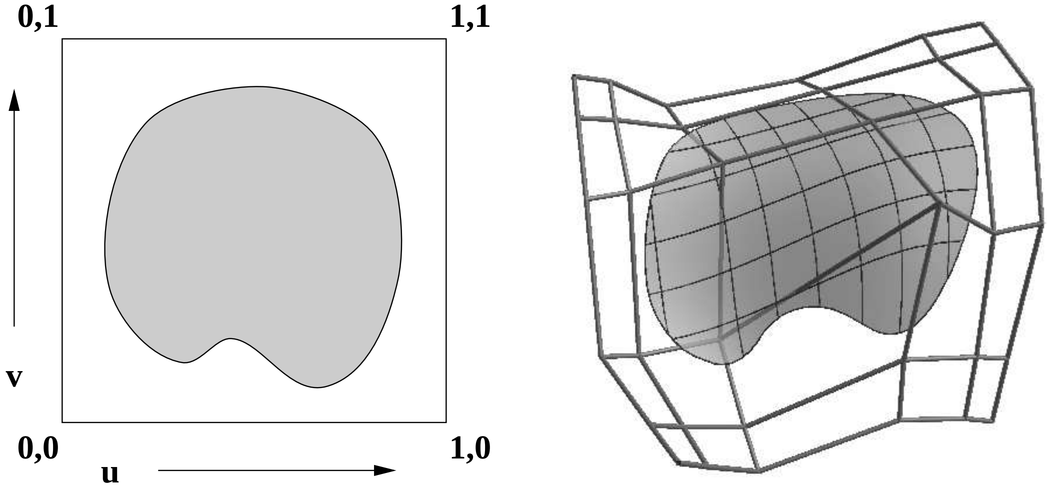


Figure 3.1. Example trimmed surface with parametric curve.

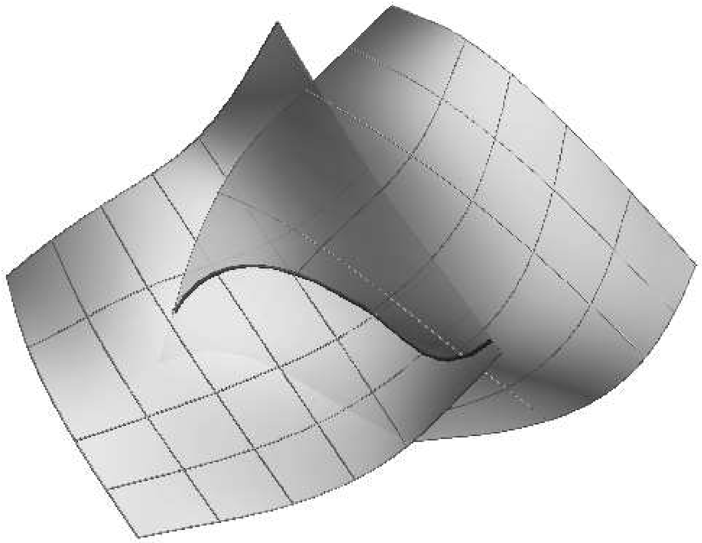


Figure 3.2. Piecewise linear intersection between two surfaces.

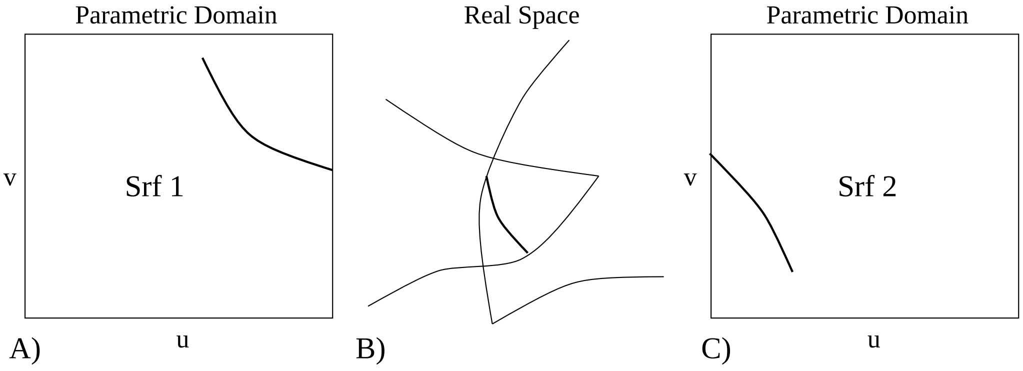


Figure 3.3. Three-way data structure for a trimming curve. A) Parametric curve in surface 1. B) Euclidean intersection curve. C) Parametric curve in surface 2.

representation.

Consider a U-shaped region cut from a uniform bicubic B-spline. Then the underlying surface,

$$S(u,v) = \sum_{i,j=0}^{4,6} P_{i,j} B^k_{i,\tau^{\mathbf{u}}}(u) B^k_{j,\tau^{\mathbf{v}}}(v) \tag{3.1}$$

where the order, k, is 4 (cubic in each parametric direction), the knot vector $\tau_u = \{0,0,0,0,1,1,1,1\}$ and the knot vector $\tau_v = \{0,0,0,0,1,2,3,3,3,3\}$. The knot vectors indicate that in the u direction, there is only one interval, $[0,1]$, but in the v direction, there are three intervals, $[0,1]$, $[1,2]$ and $[2,3]$. These intervals correspond to the piecewise polynomial patches (see Figure 3.4). A single patch of a bicubic B-spline has nonzero basis functions, (i.e., $B^k_{i,\tau^{\mathbf{u}}}(u)$ and $B^k_{j,\tau^{\mathbf{v}}}(v)$) for 16 control points. Unfortunately, a single patch represents the base of locality of a B-spline surface; every point within the patch is dependent on all 16 control points (patch boundaries excepted). Modifying any of the 16 control points corresponding to a particular patch will modify the shape of the entire patch.

Suppose the surface is trimmed so that only a U-shaped region remains (see Figure 3.5). If such a region were cut from a piece of paper, the two ends would be independently flexible. Intuitively, the same would be expected of the region cut from the B-spline surface. Notice that both ends of the U contain sections

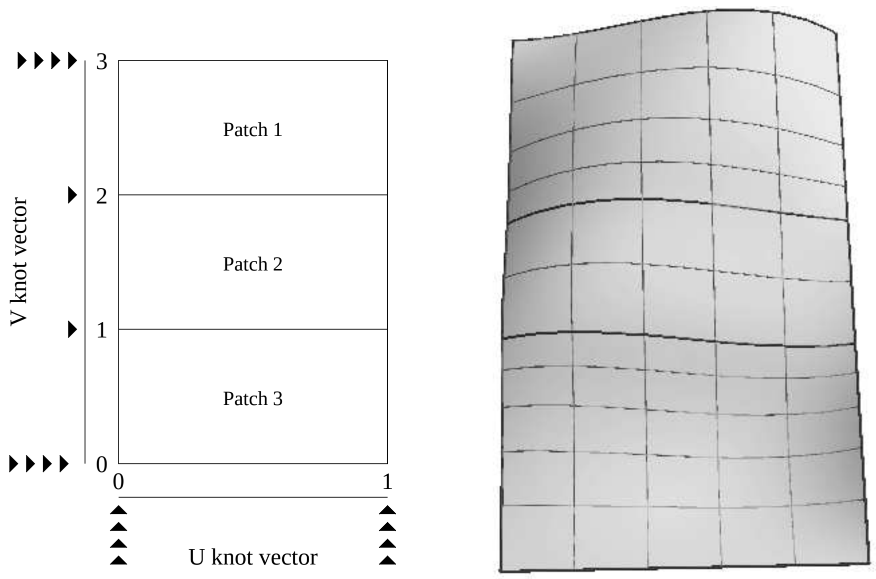


Figure 3.4. Initial surface with knot vectors and patches outlined.

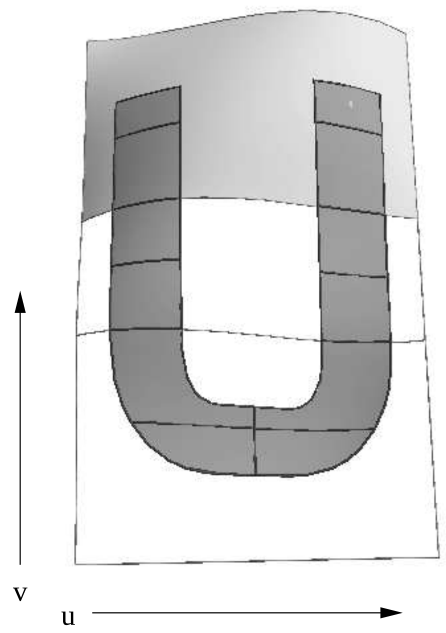


Figure 3.5. Ends of "U" are in the same patch.

from the same patch. Because of this, they are totally dependent on the same control points. If sufficient flexibility were added to the surface by introducing more knots and control points so that the individual patches in the two ends are totally independent of each other (i.e., the spans of the patches do not intersect), then this interdependence could be avoided. Unfortunately, reducing the distance between the two ends (see Figure 3.6(A)) results in a situation in which only subdividing the surface into two surfaces is sufficient to maintain the independence of the ends but subdivision reduces the continuity in the section of the U that crosses the subdivision boundary (Figure 3.6(B)) and is therefore not an acceptable solution.

This interdependence causes another difficulty. Regions are independent only if a sufficient amount of the surface is removed between the two regions. During a design process which introduces discontinuities, the parametric domain often represents a section of a physical model. Removing or trimming away a section of the domain represents removal of material which is often not acceptable.

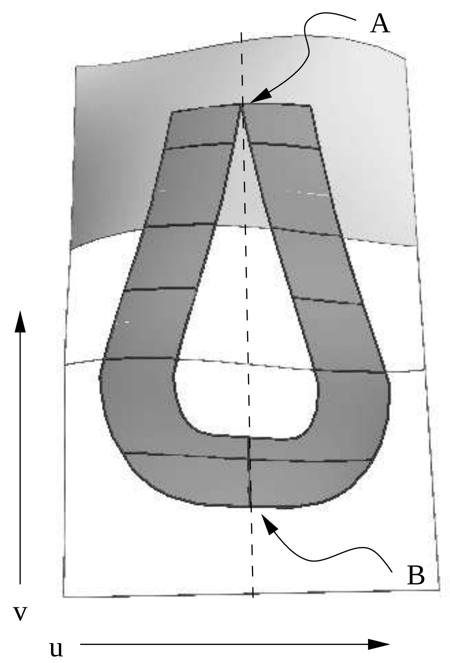


Figure 3.6. Dependence problems in trimmed surfaces.

3.5 Constraints

Regardless of the approach used to represent discontinuities, constraint fulfillment is often used to keep the model together[50, 8, 65, 92, 93, 25, 79]. In a typical design environment, the designer supplies adjacency or tangent requirements and then must convert these requirements into a set of constraint equations that can be numerically or analytically satisfied within the design environment.

The primary difficulties with this approach are the complexity of the constraint functions, the lack of a single coherent parametric space (if the discontinuities were introduced after the model was originally built)[33, 90], and the possibility of not being able to find a solution to the user-provided system of constraints due to the nonlinearity of the equations, over-constraining the system, or getting stuck in local minima. The advantages of this approach include its flexibility, its enormous popularity in current simulation methods (such as FEM[9]), and its large supporting base of research. The torn B-spline representation supports $C^{(-1)}$ continuity but creases require $C^{(0)}$ continuity. Unfortunately, exact solutions to this $C^{(0)}$ continuity requirement are often intractable or nonexistent. Therefore linear constraints are used to obtain an approximate solution in a reasonable amount of time. The particular approach used to construct the linear constraint equations is derived from the approach described by Fowler[35, 34].

3.6 Patching

Combining multisided smooth patches and triangular patches [19, 42, 53, 69, 52, 40, and others] into interesting models has been a long standing approach. The topological flexibility offered by this approach is enormous, but most of these techniques produce only uniformly smooth models. Recently, interest in models which have creases, tears, and other continuity features has increased and new techniques have been developed with support such features[74, 45, 83]. It can safely be said that each of these new techniques also uses a multipatch scheme, in which a significantly large set of basic building blocks are combined with constraints or some other smoothing method to produce a smaller number or smoother set of patches to produce the desired continuity features in the larger model. A recent example [45] begins with a set of triangles and through optimization and feature recognition produces a piecewise smooth representation with the continuity features intact. An earlier example, in computer vision reconstructs surfaces with discontinuities[74]. In contrast to the multipatch approach, the torn B-spline introduces continuity features within a smooth surface.

3.6.1 Topology

The torn tensor product B-spline surface dramatically expands the representational capabilities of a single tensor product B-spline surface. Several common topological terms will be used in this thesis and are defined below[60].

Definition 3.1 *If U is an open set containing x, then U is said to be a* neighborhood *of x.*

Definition 3.2 *A* separation *of a space X is a pair U,V of disjoint nonempty open subsets of X whose union is X.*

In particular, its opposite, *connectivity*, is critical in determining the necessary continuity requirements in the surfaces surrounding tears.

Definition 3.3 *An* m-manifold *is a space X, such that each point x of X has a neighborhood that can be mapped 1-to-1 and onto an open subset of $\mathbb{R}^m$.*

Most boundary representations of solid models are 2-manifold. Tears in a smooth boundary representation make the boundary representation nonmanifold.

Although most data representations theoretically permit nonmanifold topology when combining more than one basic building block, it is rarely used since most real objects have manifold boundary representations. With the increased use of visualization and simulation in the physical sciences and mathematics, the use of nonmanifold topologies has also increased. Bloomenthal and Ferguson[12] use triangles to represent their topology in a recent treatment of nonmanifold topology for implicit surfaces.

The torn B-spline surface described in this thesis allows discontinuities to be introduced into the surface, causing a model to become nonmanifold. Treatment of this situation is given in Section 7.3.

The representations and modeling techniques described in this chapter are the state-of-the-art for representing discontinuities in models. Yet despite their capabilities, these representations and techniques are difficult to describe and use and often fail to meet the flexibility requirements of real-world situations which are becoming increasingly more common. Clearly a flexible representation is capable of representing arbitrarily complex discontinuities without information loss in other areas of the model and is easy to understand without adding undo complexity to the representation. In the following chapter, the torn tensor product B-spline surface representation is presented. The torn B-spline surface provides the flexibility without information loss that is necessary for embedding discontinuities in the inherently smooth surface representation.

CHAPTER 4

TORN B-SPLINES

The *torn B-spline* representation is built upon the well-known B-spline representation. It uses some key techniques from the trimmed B-spline representation to provide some of the most useful functionality of the B-spline class, such as evaluation and display. In addition, modeling techniques such as designing with *feature curves* and *constraints* are integral parts of this representation. The previous research on incorporating discontinuities within models was reviewed in Section 3.3. Section 4.2 contains the technical foundation for torn B-splines, including definitions for the B-spline representation and other core definitions. Section 4.3 introduces the torn tensor product B-spline. In the following section, several types of discontinuities are presented to lay the foundation for the rest of the chapter. It is these types of discontinuities that the torn B-spline surface is able to represent.

4.1 Continuity Features

The examples in the previous sections present several situations in which discontinuities arise within the context of the modeling process. These discontinuities characterize the model or differentiate the model from other models. Therefore they are called *continuity features*.

What is a *continuity feature*? By word analysis, a *feature*, according to Webster, is "a prominent or conspicuous part or characteristic"[26, p. 487, definition 1]. *Continuity* in this sense, refers to the smoothness of the model. The definition a continuity feature is as follows.

Definition 4.1 *A* continuity feature *is a characteristic change in the continuity of a model.*

Continuity features are often the main focus of an outline drawing of a model since they characterize the shape of the model. The vertices and edges of polygons, curved boundaries of a sculpted surface, and folds in material are all continuity features. In a design system, the continuity features are represented by vertices, edges, or curves, all defined within the context of the higher-order modeling constructs such as surfaces or solids. In most situations, these features are "well behaved" in that they form boundaries of the higher-order elements used to construct the model. Occasionally, these features are not a natural part of the boundary (i.e., they are not part of a closed loop which would form a boundary in a surface model) and so are not easy to compute or use. The designer must then switch representations or perform some additional work in order to adequately represent this type of feature.

The continuity features introduced or discussed in this thesis are defined below.

Definition 4.2 *A* tear *is a smooth parametric curve in the parametric domain of a surface along which the surface is geometrically discontinuous (see Figure 4.1(A)).*

Definition 4.3 *A* crease*[45] is a smooth parametric curve in the parametric domain of a surface along which that surface has $G^{(0)}$ continuity but not $G^{(1)}$ continuity (see Figure 4.1(B)).*

Although these features are defined in terms of geometric continuity, it is assumed that the parametric surface is standard and has no other constraints unless noted so that these features can be discussed in terms of parametric continuity.

Definition 4.4 *A* critical point *is the endpoint of a tear in the interior of a surface, whether or not it is part of a crease. An endpoint on a surface boundary is not critical unless the boundary is constrained to be adjacent to a boundary of another surface in the neighborhood of that point (see Figure 4.1(C)).*

Definition 4.5 *A* dart*[45] is a crease whose endpoint lies in the interior of a continuous surface (see Figure 4.1(D)).*

If the interior endpoint of the dart is also the endpoint of a tear, then that endpoint is a *critical point*. In particular, the continuity characteristics change at the end of a dart.

The same smoothness characteristics are required for all points on the surface except for the points on the tear and the critical points. A critical point is the only point on a tear whose connected neighborhood contains both sides of the tear. These critical points play an important role in the torn B-spline data structure.

Definition 4.6 *A* corner*[45] is the point at which two or more creases join. Usually a corner describes a $C^{(1)}$ discontinuity between the parametric representations of the creases within the surface (see Figure 4.1(E)).*

This term can also be used for the intersection of two or more tears, although the resulting geometry is quite different. A more general definition of a corner could allow a corner to exist in the middle of a single crease, but given the assumption that a single curve is $C^{(1)}$ with respect to parametric space of the surface, corners require two or more curves.

Definition 4.7 *A* fracture *is a pair of parametric surfaces embedded in a solid which represents a discontinuity within the solid (see Figure 4.1(F)).*

Practically speaking, any parametric surface slice which intersects the fracture will have a tear in the surface along the intersection. In the engineering world, a fracture is one of the most complex continuity features used. This thesis introduces a method for representing the parametric slices of the solid, and issues involved in representing the solid itself will be discussed.

4.2 Technical Background

First the basic definition of a tensor product B-spline surface and some additional terms which are frequently used will be given.

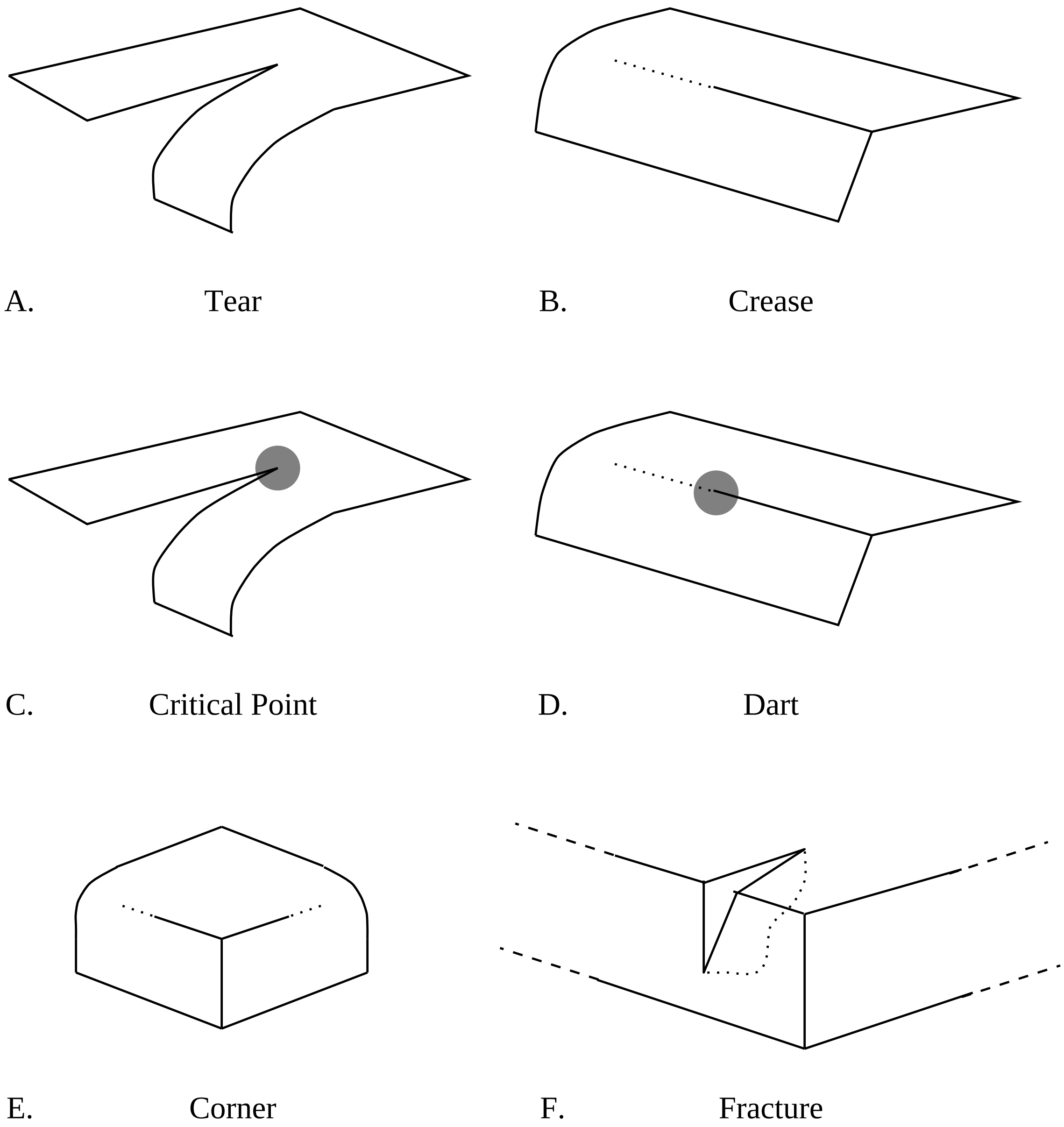


Figure 4.1. Continuity features.

4.2.1 Tensor Product B-spline

Definition 4.8 *A nonuniform tensor product B-spline surface,*

$$Q(u,v) = \sum_{i,j=0}^{m,n} P_{ij} B_{i,\tau^{\mathbf{u}},k_u}(u) B_{j,\tau^{\mathbf{v}},k_v}(v), \tag{4.1}$$

is defined by the set of coefficients, $\{P_{ij}\}$, the knot vectors, $\tau^{\mathbf{u}} = \{\tau_i^u\}, \tau^{\mathbf{v}} = \{\tau_i^v\}$, and the B-spline basis functions, $\{B_{i,\tau^{\mathbf{u}},k_u}(u),\ B_{j,\tau^{\mathbf{v}},k_v}(v)$ where $\{B_{i,\tau^{\mathbf{u}},k_u}(u)$ $(B_{j,\tau^{\mathbf{v}},k_v}(v))$ is the i^{th} (j^{th}) B-spline basis function of order $k_u (k_v)$ over the knot vector $\tau^{\mathbf{u}} (\tau^{\mathbf{v}})$, respectively.

To simplify notation where there is no confusion, the knot vector and the order will be inferred from the parametric variable that is used, so $B_{i,\tau^{\mathbf{u}},k_u}(u) = B_i(u)$ and $B_{j,\tau^{\mathbf{v}},k_v}(v) = B_j(v)$. In addition, the product of the basis functions will be abbreviated as $B_{ij}(u,v) = B_i(u)B_j(v)$. To distinguish the torn B-spline definition from the above definition, a surface from Definition 4.8 will be referred to as the *standard* tensor product B-spline surface.

4.2.2 Span

Definition 4.9 *The* span, *$\mathcal{S}(u,v)$, is the set of (ordered pairs of) subscripts whose corresponding basis functions are nonzero at (u,v), i.e.,*

$$\mathcal{S}(u,v) = \{(i,j) | B_{ij}(u,v) \neq 0\}. \tag{4.2}$$

Definition 4.10 *Let $q(t) = (u(t), v(t))$ be a parametric curve, $t \in [t_{min}, t_{max}]$, in the parameter space of a surface Q. The* span *of a curve, $\mathcal{R}(q)$, is defined as the union of the set of (ordered pairs of) subscripts whose corresponding basis functions are nonzero for some $(u(t), v(t))$ on the curve, q, i.e.,*

$$\mathcal{R}(q) = \bigcup_{t \in [t_{min}, t_{max}]} \mathcal{S}(q(t)). \tag{4.3}$$

The span will be used to determine which control points are crucial for determining the set of parametric values along a continuity feature.

4.2.3 Patch

Definition 4.11 *The* patch *of a span, $\mathcal{G}(\mathcal{S})$, is the closure of the set of (u, v) all of which have the same span.*

In general, if $\mathcal{S}_1$ and $\mathcal{S}_2$ are two spans of either points or curves, and $\mathcal{S}_1 \subset \mathcal{S}_2$, then $\mathcal{G}(\mathcal{S}_1) \subset \mathcal{G}(\mathcal{S}_2)$.

The knot lines (or interior knot values) of the B-spline definition delineate the patches of the surface. For the torn B-spline representation, the continuity characteristics may not change within a patch.

4.3 Torn B-splines

Consider the process of introducing discontinuities into a model given the tools that are currently available. There are two natural ways to think of this process. First, basic modeling elements can be put together in such a manner that the discontinuity lies along the boundary between elements. Where the model needs to be smooth, constraints can be used to enforce some degree of smoothness. This is a "bottom-up" approach. The second way is to arrange the parameters of the basic modeling element so that a discontinuity is formed within the modeling element itself. For example, a tensor product B-spline can have multiple knots, each knot lowering the degree of the surface by 1. This is the "top-down" approach.

The "bottom-up" approach is easy to understand. However, since the surfaces may be parameterized differently, this may become a highly nonlinear constraint problem, one that may not be solvable. In addition, individual surfaces behave independently, requiring highly specialized code to make modifications to the region as a whole. Finally, if discontinuities were added to a smooth surface, the original surface's structure provides a wealth of smoothness information away from the discontinuity that is thrown away when multiple surfaces are constructed. The "top-down" approach is more desirable for these reasons. Unfortunately, current representations are limited in their flexibility for representing discontinuities; therefore the "top-down" approach is limited to certain representations in certain situations, the merits of each were discussed earlier.

The bottom-up approach to accommodating discontinuities within a single tensor product B-spline representation results in a set of trimmed B-spline surfaces, each with its own set of coefficients (control mesh) and a set of constraints which *tie* the trimmed regions back together along the smooth parts, but the constraints along the smooth parts are just composed of matching coefficients since both trimmed regions came from the same surface. The torn B-spline representation is based on this concept as applied to the "top-down" approach. An exact solution for smoothness away from the discontinuities is guaranteed implicitly and the parameterization matches across the discontinuities by default by focusing on supporting the discontinuity within the structure instead of continuity between two different structures. In addition, the surface is one complete unit with full knowledge of all its domain and is capable of supporting operations on the surface as a whole.

The development of the torn B-spline representation from the "top-down" approach is best understood by first considering the two-dimensional (or 2D) case of a torn B-spline curve. Figure 4.2 shows a torn B-spline curve. To tear a B-spline curve, q, we introduce a *tear point* at parametric location, $\hat{t}$. Note that $\mathcal{R}([t_{min}, \hat{t}]) \bigcap \mathcal{R}([\hat{t}, t_{max}]) = \mathcal{S}(\hat{t})$. Let η be a classification function which separates the curve into two distinct regions, $[t_{min}, \hat{t}]$ and $[\hat{t}, t_{max}]$. (Although $\hat{t}$ does not actually exist in both regions, we assume the limiting case.) $\mathcal{S}(\hat{t})$ then occurs in the spans of both regions. If q is torn at $\hat{t}$, then there must be two distinct locations for $q(\hat{t})$. In order to make these two locations independent of each other, their spans must be independent; hence, the torn representation for each region must have distinct control points for each i in $\mathcal{S}(\hat{t})$. These additional points are called the *overlap polygon.* In Figure 4.2, row a contains the original control polygon of a curve. Row b contains the set of control points, Q3-Q6 and P7, for $\mathcal{R}([\hat{t}, t_{max}])$, and row c contains the set of control points, P0-P6, for $\mathcal{R}([t_{min}, \hat{t}])$. The function, η, then determines which curve region a parametric location is contained in. A point on the curve, $q(t)$, is evaluated by using the appropriate control points from the original control polygon and the overlap polygon as appropriate for the region of the curve which contains $q(t)$ as determined by η.

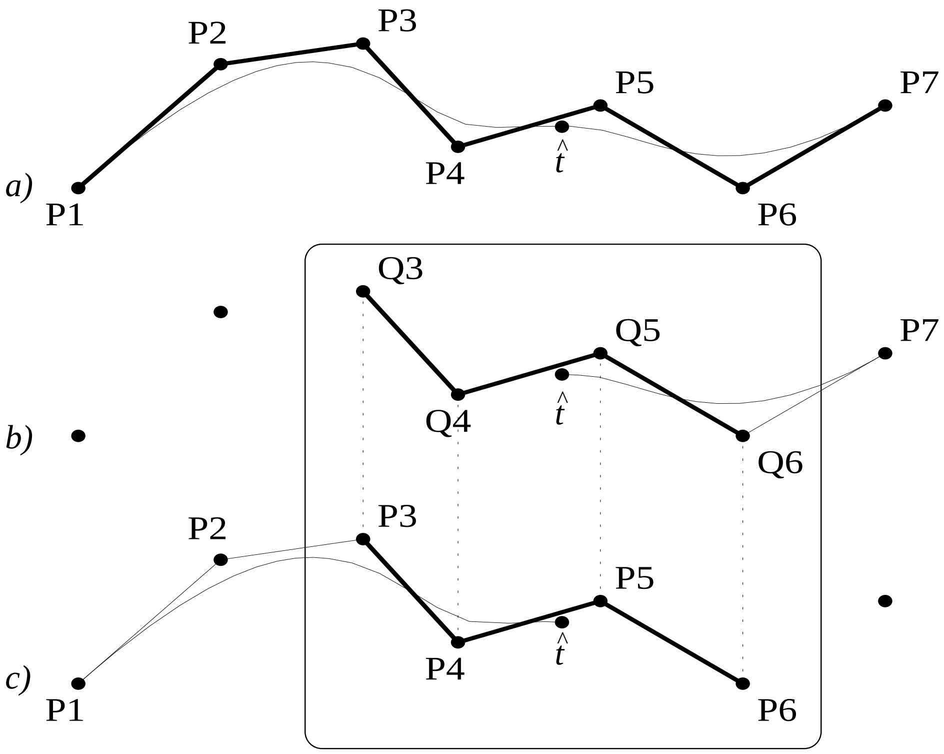


Figure 4.2. Torn B-spline curve.

In the curve case, the two curve regions can be made independent by subdividing the curve at the tear point. Since subdivision is a well-established technique, the torn B-spline curve is mostly academic. The torn B-spline surface is much more interesting. Informally, a torn B-spline surface is comprised of an underlying tensor product B-spline surface and one or more curves of discontinuity or *tear curves*. Consider first, the case where a single tear curve, $q(t)$, separates a surface into two distinct regions. Again the requirement is that the surface regions on either side of the tear curve be independent of each other. The two regions are again classified by the function, η. In order for the regions to be independent of each other, the span of the tear in one region must be independent of the span of the tear in the other region, which is analogous to the torn B-spline curve case. That is, a point on the tear in one region must use a completely different set of coefficients for the span of that point. This implies that $\mathcal{R}(q)$ in one region is independent of $\mathcal{R}(q)$ in the other region. The additional coefficients required to make the spans independent are stored in the *overlap mesh*. Figure 4.3(A) shows an isoline drawing of the torn B-spline surface. Figure 4.3(B) shows the patches which contain the tear curve. Figure 4.3(C) shows the points in the span of the tear curve, $\mathcal{R}(q)$, which are included in the overlap mesh. Once again, each region, separated by η, is associated with a particular set of control points selected from the original mesh and the overlap mesh. Although the maximal number of degrees of freedom (DOFs) associated with the discontinuity is fixed by the configuration of tear curves in the surface and the separation of regions (see Section 4.7), the methods used to determine the distribution of these points are implementation dependent (see Section 4.10.7).

Unfortunately, tear curves which fully separate the domain are not the most common case, since full separation is often representable by other methods (i.e., subdivision or trimmed surfaces). More typically, one or more of the endpoints of the tear curve is in the middle of the surface; often in the middle of a patch as well. These partial tears are related to Thomas' *cut curves*[82]. When the tears separate the surface, it is clear how to determine the classification function, η. When the

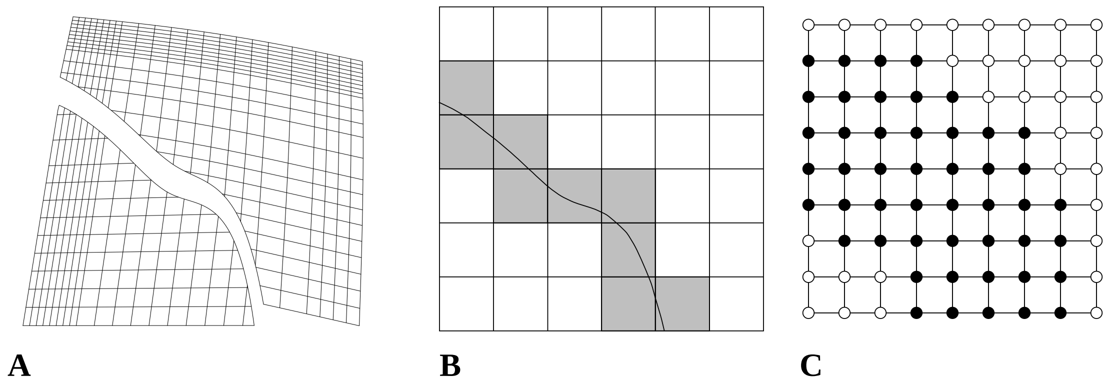


Figure 4.3. Torn B-spline surface with complete tear.

tears do not separate the surface, the distinction is no longer clear. Fortunately, the original smoothness of the surface is required away from the discontinuities; thus only one surface location is associated with a given parametric location. Regardless of how the regions are classified, the evaluation of a given point must be well defined in the context of the torn surface. A method for making this classification well defined is presented in Sections 4.7 and following.

If a tear curve has an endpoint in the middle of the surface, the neighborhood of this critical point has an assumed continuity requirement; the surface must have the same continuity as the underlying B-spline surface except possibly at the tear. If the tear has an endpoint in the middle of a patch, the entire patch must remain connected. Recall that all points in the interior of the patch have the same span; therefore they are all dependent on the same coefficients. This dependence because the cross-patch continuity is being maintained on all patch boundaries and the span is essentially the smallest piece of the surface which is able to support the cross-patch continuity described by the order and knot vector of the underlying tensor product B-spline surface. This characteristic of the patch means that if two points are connected in the same patch, then there is only one configuration of coefficients given the boundary continuity requirements for that patch. Conversely, if the spans of two parametric locations are different in at least one coefficient, then the two points are not connected within the patch. Therefore, either the patch is separated from one patch boundary to another, or any two points on the patch are connected and the patch retains its original continuity. If the endpoint of the tear lies on a patch boundary, then this connectivity of the patch is not a problem, since the discontinuity crosses the entire patch. However, if the endpoint lies in the middle of a patch, there are several options available, the choice of which ultimately determines the degree to which the patch is kept smooth without modifying the tear itself.

1. Require that the entire patch containing the endpoint remains connected (continuous).

2. Allow the patch to become separated beyond the original tear curve description.

3. Introduce additional flexibility which creates a patch boundary at that point.

If maintaining smoothness where there is no discontinuity is more important than having full discontinuity along the entire length of the tear curve, then the first option is preferable to the second. This choice also prevents artifacts of the classification scheme from being present in the surface since the additional portion of the patch that is discontinuous from the second option is dependent on the classification of the regions. The third option was investigated, and initial experimentation indicated that it was a viable option also. Full integration and analysis of the effects of such a change are left for future work. An example of partial tear is given in Figure 4.4. Figure 4.4(A) is an isoline drawing of the surface. Figure 4.4(B) shows the patches of the surface that the tear extends through the hatched patch is the patch whose span is removed to satisfy the first option given above. Figure 4.4(C) shows the resulting control points that are in the overlap mesh.

Since the initial overlap mesh computation is derived directly from the span of the tear curve, the span of the tear's mid-surface endpoint is included in the overlap mesh. However, regardless if the tear ends on the boundary of a patch or in the interior of a patch, if any coefficient in the span of the endpoint is in the overlap mesh,

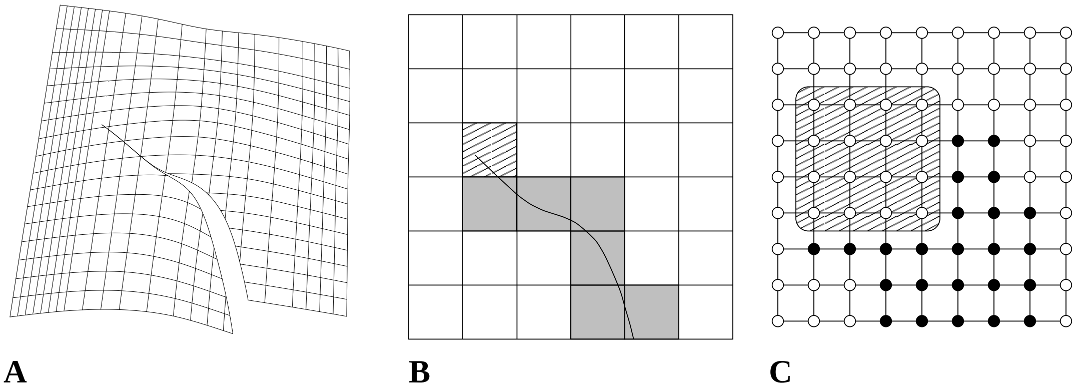


Figure 4.4. Torn B-spline surface with partial tear. A. Isolines from the surface. B. Parametric domain with the tear's span highlighted. C. Control points of the surface, highlighting the control points used in the overlap mesh of the tear.

the discontinuity extends across the patch. So the span of the endpoint is removed from the overlap mesh to maintain the continuity of the surface at the endpoint. In Figure 4.5(a), shaded regions of patches correspond to spans that have been added; the outlined regions (circled) correspond to spans which are subsequently removed. In Figure 4.5(b), the shaded points of the corresponding control mesh are added, and, likewise, the outlined points are removed. The remaining points in the darkly shaded regions make up the overlap mesh. Additional considerations for adding and removing points from the overlap mesh are discussed in Section 4.7.

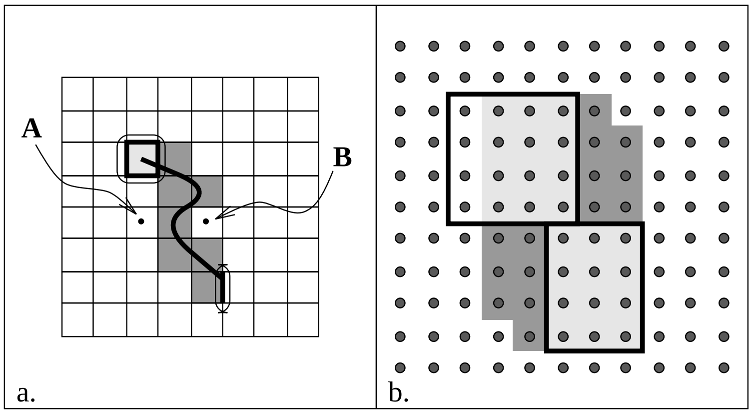


Figure 4.5. Subpatch diagram with tear.

4.4 Torn B-spline Definition

Definition 4.12 *The* torn B-spline surface *representation has several parts:*

1. *an underlying tensor product B-spline surface definition, with the requisite order, knot vectors, and control mesh;*
2. *a set of tear curves, $\{\gamma_\kappa\}_{\kappa=1}^{T}$, defined within the parameter space of the underlying B-spline surface, along which the surface is discontinuous;*
3. *a set of control meshes, $\{O_{ij}^{(\kappa)}\}_{\kappa=1}^{T}$, called* overlap meshes, *which contain the additional coefficients,*
4. *a masking function, μ, which identifies for each region, c, the composition of control points from the original control mesh, P_{ij}, and the overlap meshes, $O_{ij}^{(\kappa)}$,*
5. *a piecewise mapping, η_c, whose value is 1 if a given parametric point, (u, v), is contained in the parametric region, c, and 0 otherwise. This function may be ambiguous if the point lies on a tear curve or its extension.*

Then we define the torn B-spline *surface as*

$$T(u, v) = \sum_{c=0}^{T} \eta_c(u, v) \sum_{i,j=0}^{m,n} P_{ij}^{(c)} B_{i,j}(u, v) \tag{4.4}$$

where

$$P_{ij}^{(c)} = \begin{cases} P_{ij} & \text{if } \mu_c(i, j) = 0 \\ O_{ij}^{(\kappa)} & \text{if } \mu_c(i, j) = \kappa \text{ otherwise.} \end{cases} \tag{4.5}$$

For utility, let $\hat{\eta}(u, v) = c \iff \eta_c(u, v) = 1$.

Let us examine each of these parts in turn.

4.5 Underlying B-spline Surface

The underlying tensor product B-spline surface is the basis for the torn B-spline surface. The order and knot vector of the torn B-spline surface are inherited from it, and its control mesh serves as the base control mesh of the torn B-spline surface.

4.6 Tear Curves, γ_κ

As discussed in earlier sections, the tear curves which are part of the torn B-spline representation can have arbitrary parametric geometry and extend completely or partially across the surface. They may abut, but not cross, each other or themselves. The type of curve is limited only by the implementation. There is no limit on the number of tear curves in a torn B-spline surface.

4.7 Overlap Mesh, O

The additional DOFs allowed by the introduction of tears can be computed by examining the span of the tear.

Definition 4.13 *The* overlap mesh*, $O_{ij}^{(\kappa)}$, is the mesh of additional coefficients associated with the tear, γ_κ. By definition, $O^{(0)} \equiv P$.*

There are several requirements which make the classification of points a well-defined process. The first requirement is that the surface must retain its original degree of smoothness except along tear curves. Before the discontinuity is introduced, the knot vector describes the parametric continuity of the surface. In particular, for a tensor product B-spline surface of order 4 in the u direction, the surface will be $C^{(3)}$ in regions between knots and $C^{(3-m)}$ at knots of multiplicity m with respect to the u parameter. After the introduction of the tear discontinuity, the same degree of parametric continuity is maintained everywhere on the surface, except at the discontinuity. The second requirement is that the surface is discontinuous along the tear curve except where, to be so, would violate the first requirement. Finally, all additional degrees of freedom contained in the overlap mesh of each tear must be present in the new representation and must be independent. In a quick jump ahead to Section 4.10.6, when more than one tear is in a surface, the classification of the regions and the automatic distribution of the new control points may make two independent regions dependent, dropping conflicting degrees of freedom from the picture. This case eliminates certain choices for region classification and control point distribution.

4.7.1 Maximal Independence

When a tear curve is added to a B-spline surface, the tear curve makes the two regions on either side of the tear independent of each other. Depending on the separation of the domain into parametric regions, the number of additional degrees of freedom may be different. It can be proven that the additional coefficients present in the overlap mesh are both necessary and sufficient to make two points on opposite sides of the tear independent. However, in some cases, this independence is not the only independence required. In particular, if two points on the same side of the tear are to be independent of each other, the proof given below does not apply. In this case, the implementation of the torn B-spline data structure determines whether or not this independence is made available. In any case, for any given implementation and configuration of tear curves, there is a *maximal number of degrees of freedom* which can be utilized. This number is obtained by summing up the degrees of freedom in the original surface plus all the degrees of freedom in the overlap meshes. This concept, although it is implementation dependent, will be used throughout the remainder of this thesis.

However, it is possible to determine the globally maximal number of degrees of freedom for a given tear configuration regardless of implementation. A very simple heuristic can be used to determine the maximal number of degrees of freedom for a given tear. The algorithm essentially counts the number of times per coefficient that the curve doubles back on itself. In short, the curve is divided into *sections* where all of the points in the same section have the same span. We proceed along the curve, piece by piece, using the span of the piece as a "window" of sorts, marking the coefficients with a *tag*. After all coefficients have been marked for a given section of the curve, the tags of the coefficients in the window are all nonzero and negative; the tags for the coefficients outside of the window are zero if they have not been seen and greater than zero if they have been seen. The ordinal value of the tag indicates the number of times the window has "revisited" the coefficient. The detailed algorithm is presented below.

1. Identify sections of the tear curve according to spans, where all the points in a

section have the same span. (Only connected points can be in a given section, and points on patch boundaries make their own sections).

2. Initialize an array of integers to 0, one for each coefficient in the control mesh. Call this array the *tag array* and each item the *tag* of the coefficient.

3. Begin at one end of the tear and proceed through all of the sections consecutively, doing one of the following for each coefficient in the span of the section.

   (a) If the coefficient's tag is 0, set it to -1.

   (b) If the coefficient's tag is less than 0, do nothing.

   (c) If the coefficient's tag is greater than 0, increment it and negate it (i.e. 2 becomes -3).

   For each coefficient not in the span of the section, if the coefficient's tag is less than 0, negate it. Proceed to the next section.

4. After all sections have been processed, for each coefficient, if the coefficient's tag is greater than 0, negate it.

5. For each coefficient in the span of each endpoint (unless the endpoint lies on a boundary such as another tear, a trimming curve, or the boundary of the parametric domain), if the coefficient's tag is greater than 0, decrement it.

The resulting tag for each coefficient indicates the number of additional unique degrees of freedom required for maximal independence of the tear curve at that coefficient. Although the algorithm is simple, it may not be clear how best to use this information, since, at this point, it is a heuristic. However, it may affect how the tear curve should be split and may ultimately determine the best implementation. These and other related questions are left for future work.

4.7.2 Sufficiency of $O_{ij}^{(\kappa)}$

The span of the curve is used to compute the additional coefficients (DOFs) needed in the overlap mesh. Do these additional DOFs provide enough flexibility to cause a given tear curve to be discontinuous along its length? This question can be answered by carefully examining the construction of each O_{ij}. The necessity of each of the additional DOFs in the overlap meshes is demonstrated by the fact that in order for the parametric regions on either side of a parametric curve embedded in a given B-spline surface to be independent, the spans of the curve in each region must be independent of each other. To impose the requirement that the neighborhood around the ends of the tear curves have the same continuity characteristics as the original surface, the spans of the endpoints must be totally dependent on the same coefficients.

Are more coefficients necessary to describe the surface on either side of the tear? To show that these are the only coefficients needed, we first show that if a point lies between two other points then the intersection of the spans of the two other points is a subset of the span of the first point.

Theorem 4.1 *Let points A, B, and C on a B-spline surface of order k_u and k_v in the u and v directions, respectively, be described by parametric locations, (u_A, v_A), (u_B, v_B), and (u_C, v_C), respectively. Let C be* between *A and B if $min(u_A, u_B) \leq u_C \leq max(u_A, u_B)$ and $min(v_A, v_B) \leq v_C \leq max(v_A, v_B)$. Recall that $\mathcal{S}(p)$, is the span of the point, p. Then if $P_{\bar{i},\bar{j}} \in \mathcal{S}(A) \cap \mathcal{S}(B)$ then $P_{\bar{i},\bar{j}} \in \mathcal{S}(C)$.*

Proof. Assume, for now, that A, B, and C do not lie on knot lines. Let

$$\begin{aligned}
\mathcal{S}(A) &= P_{[i_A,\ldots,i_A+k_u-1],[j_A,\ldots,j_A+k_v-1]} \\
\mathcal{S}(B) &= P_{[i_B,\ldots,i_B+k_u-1],[j_B,\ldots,j_B+k_v-1]} \\
\mathcal{S}(C) &= P_{[i_C,\ldots,i_C+k_u-1],[j_C,\ldots,j_C+k_v-1]}.
\end{aligned}$$

Consider first the argument for the row index i; the argument for j follows analogously. If $P_{\bar{i},\bar{j}} \in \mathcal{S}(A) \cap \mathcal{S}(B)$, then

$$max(i_A, i_B) \leq \bar{i} \leq min(i_A, i_B) + k_u - 1.$$

Since C is between A and B,

$$min(i_A, i_B) \leq i_C \leq max(i_A, i_B) \tag{4.6}$$

$$min(i_A, i_B) + k_u - 1 \leq i_C + k_u - 1 \leq max(i_A, i_B) + k_u - 1. \tag{4.7}$$

Therefore,

$$i_C \leq \bar{i} \leq i_C + k_u - 1$$

which with the analogous argument for $\bar{j}$, implies that

$$P_{\bar{i},\bar{j}} \in \mathcal{S}(C) = P_{[i_C,\ldots,i_C+k_u-1],[j_C,\ldots,j_C+k_v-1]}.$$

As stated above, it is assumed that the points do not lie on knot lines. This implies that the number of indices for a given direction that are in the span of a point is equal to the order in that direction. If A or B lies on knot lines (at the minimum in either direction), the spans become smaller, but because the restriction becomes tighter, the argument still holds. However, if C lies on a knot line in a given direction, the span of C is reduced by the number of knots at that parametric location. This implies that Equation 4.7 does not follow directly from Equation 4.6 without additional argument.

Suppose then that u_C lies on a knot with multiplicity m. Then the number of indices in the span of C in the u direction is $k_u - 1 - m$ and the maximum index for i in the span of C is $i_C + k_u - 1 - m$. We need to show that $min(i_A, i_B) + k_u - 1 \leq i_C + k_u - 1 - m$ (the right-hand side of Equation 4.7 holds if $m \geq 0$). By the definition of the B-spline basis functions, the number of knots between the parametric locations of two points determines the difference between the beginning index values of the spans of the two points. This means that

$$min(i_A, i_B) + m \leq i_C.$$

Then,

$$min(i_A, i_B) \leq i_C - m,$$

and so

$$min(i_A, i_B) + k_u - 1 \leq i_C + k_u - 1 - m,$$

which is what we needed to show.

The argument is similar if $max(i_A, i_B)$ or $max(j_A, j_B)$ lies on a knot line. ∎

Definition 4.14 *For A and B on a B-spline surface, σ, A and B are* independent *of each other, if either $\mathcal{S}(A) \cap \mathcal{S}(B) = \emptyset$ or if $\mathcal{S}(A) \cap \mathcal{S}(B) = \mathbf{T}$ and $\hat{\eta}(A) = \rho_A$ and $\hat{\eta}(B) = \rho_B$ then for all $(i,j) \in T$, $\mu_{\rho_A}(i,j) \neq \mu_{\rho_B}(i,j)$.*

If the surface is not torn, then there is only one mesh, and the two points are independent of each other if the intersection of their spans is empty. If a surface is torn, then the spans can overlap, as long as each index pair in the intersection corresponds to a control point in a different mesh. Now we show that, in fact, nothing is missing from O^κ.

Definition 4.15 *A and B are* on opposite sides *of the tear γ_κ if the straight line (in parametric space) between A and B intersects the tear an odd number of times (where intersection with a tangent of the tear that is not an inflection counts as two intersections, and an intersection at a tear endpoint is one intersection).*

Even if two points are not on opposite sides of a tear by this definition, a complete classification, where every point is on one side of the tear or the other, is sometimes useful. The containment function, μ_c, is used to provide this kind of complete classification in Definition 4.12 of the torn B-spline surface.

Theorem 4.2 *If A and B are on opposite sides of the tear, γ_κ, and $\mathcal{S}(A)$ and $\mathcal{S}(B)$ are not part of a constraint region, then $\mathcal{S}(A)$ and $\mathcal{S}(B)$ are independent of each other.*

Proof. Assume A and B are independent of each other and on opposite sides of the tear γ_κ, and assume that $\mathcal{S}(A)$ and $\mathcal{S}(B)$ are not part of a constraint region. Let C be a point on γ_κ such that C lies on the line between A and B. Let $\hat{\eta}(A) = \rho_A$ and $\hat{\eta}(B) = \rho_B$. Suppose then that there exists an index pair, (i,j), such that $\mu_{\rho_A}(i,j) = \mu_{\rho_B}(i,j)$ and therefore $(i,j) \in \mathcal{S}(A) \cap \mathcal{S}(B)$. This implies that $(i,j) \notin O^\kappa$. Then by the Theorem 4.1, $(i,j) \in \mathcal{S}(C)$. However, by definition, all $(i,j) \in \mathcal{S}(\gamma_\kappa)$ are in O^κ unless they are part of a constraint region. If $(i,j) \in O^\kappa$ then $\mu_{\rho_A}(i,j) \neq \mu_{\rho_B}(i,j)$ since the A and B are in regions on opposite sides of the tear, and this is a contradiction. ∎

As indicated by the assumptions, Theorem 4.2 does not apply to the case when individual control points are removed from the overlap mesh in order to maintain smoothness in the regions surrounding the tear. Then there are fewer DOFs and points on opposite sides of the tear are, by design, not completely independent of each other. Results pertaining to constraint regions will be presented in Section 4.10.3.

These theorems show that the defined overlap meshes are both necessary and sufficient to provide the maximum flexibility around the discontinuity.

Unfortunately, the assumptions also state that the points A and B had to be on the opposite sides of a tear. If A and B are on the same side of a given tear, or more precisely, $\eta(A) = \eta(B)$, then they are dependent on each other as though the tear did not exist, since they are guaranteed to use the same set of control points by η. This implies that even if a tear should pass between the two points and double back before terminating, theoretically requiring independence between the points, this independence is not guaranteed. See Section 4.10.1 for a discussion of how to split tears so that η is defined appropriately.

4.8 Masking Function, μ

The masking function is the embodiment of the constraints within the data structure. As such, this function is dependent on the overall structure of the tears within the torn B-spline surface. Two aspects of the particular implementation discussed here need to be clear before proceeding. The first is that the tears are connected to the boundaries of the parametric region (or another tear) by adding invisible parametric line segments, called *extensions*, to the endpoints of the tears. These extensions partition the surface into disjoint parametric regions indicated by c. The containment function, η_c, is a classification function and embodies these disjoint regions; it is discussed further in Section 4.9. The second is the concept that the disjoint regions and the tears are related to one another by an ordering, called the *signature ordering*.

Definition 4.16 *A* valid signature ordering *for a torn B-spline surface is a (possibly partial) ordering of parametric regions and tears of a torn B-spline surface such that*

1. *every tear and every disjoint region are in the graph of the ordering,*
2. *every tear directly precedes one and only one region in the graph,*
3. *no region is preceded by more than one tear,*
4. *tears are not directly preceded by tears, and*
5. *regions are not directly preceded by regions.*

By this definition, a region may precede any number of tears (including none), but a region's direct precedence of more than one tear is discouraged for simplicity and ease of implementation.

Definition 4.17 *The* signature *for a region of a torn B-spline surface is defined by the portion of the signature ordering of the torn B-spline that precedes and includes the region.*

With these definitions, every distinct parametric region, c, has a unique signature. The implementation and other issues surrounding the signature ordering for a surface is discussed in Section 4.10.6.

Definition 4.18 *The* masking function, $\mu_c(i,j) : \mathbb{Z}^2 \rightarrow \{0,\ldots,T\}$, *determines if a particular coefficient,* $O_{ij}^{(\kappa)}$, *is used in parametric region,* c. *Let* $(u_0, v_0) = \gamma_\kappa(t_{min})$, $(u_1, v_1) = \gamma_\kappa(t_{max})$ *and* $\mathcal{D} = \bigcup\{\mathcal{S}(u_i, v_i) | i = 0, 1; (u_i, v_i)$ does not lie on another discontinuity (such as a surface boundary or another tear)$\}$. *Then* $\mu_c(i,j) = \kappa$ *if*

1. *there exists a point,* (u,v), *on* γ_κ, *such that* $(i,j) \in \mathcal{S}(u,v) - \mathcal{D}$,
2. $O_{ij}^{(\kappa)}$ *exists and* κ *is the last tear in the signature of* c, *and*
3. $O_{ij}^{(\kappa)}$ *has not been replaced by another DOF by being a member of the span of an extension (more details are in Section 4.10.7).*

There are special considerations for tears which intersect.

4.8.1 Intersecting Tears

A tear may intersect another tear in one of two ways: 1) end to end or 2) side to end. Other cases are not allowed by the definition of a tear curve. Where the endpoint of a tear lies is critical to determining the continuity of the surface near that point. If the endpoint of a tear lies on the edge of a discontinuity (either the boundary of the surface, or another tear), then the span of the endpoint may be used as part of the overlap mesh in its entirety because no continuity needs to be maintained at that point. The two discontinuities (in particular, the tears) are said to have a *parent-child* relationship at the point of intersection (see Figure 4.6). Since no modifications need to be made when processing the tear that the second tear abuts to, it is considered the *parent*. The abutting tear is considered the *child* since the computation of the additional coefficients depends on the intersection.

The end to end case is very similar in that a parent-child relationship is also determined. In this case, however, the particular relationship is not necessarily clear. In most cases, it makes no difference which tear is the parent and which is the child. The only difference seen by experimentation is that the determination of

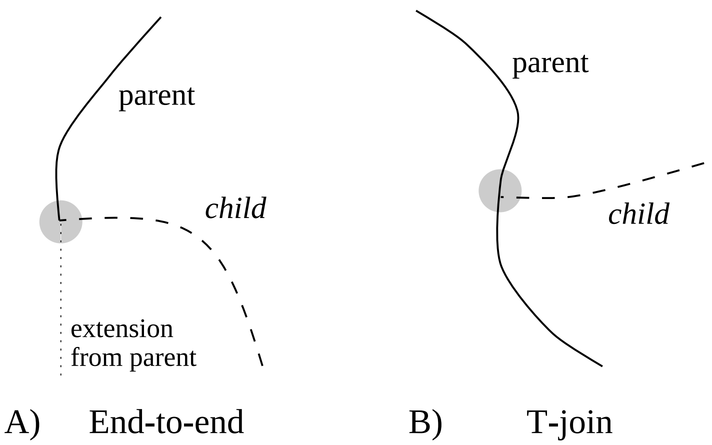


Figure 4.6. Example of parent-child relationships for two types of intersections.

the coefficients may be more complex for a particular relationship in some cases. The exact nature of this complexity relates to the composition of the control meshes for some regions. In particular, experiments have shown that fewer exceptions to the general composition rules are encountered when the parent tear precedes the child tear (see Section 4.10.6 for a discussion of precedence rules and ordering). The parent-child relationship pertains only to a given intersection. In particular, a set of tears may have a circular relationship when considering all endpoint intersections of all tears (see Figure 4.7).

These conditions provide the inherent constraints of the representation by enforcing continuity within patches containing the endpoints of the tear curves and providing additional DOFs to guarantee the independence of patches which are separated by the tear. For the definition and discussion of signatures, see Section 4.10.6. For a discussion of prevention and propagation, see Section 4.10.7.

4.9 Containment Function, η

Suppose that two points, A and B, are on opposite sides of a tear as shown in Figure 4.8. Then $\mathcal{S}(A) = \{(i,j) | i = 3,4,5,6; j = 2,3,4,5\}$ and $\mathcal{S}(B) = \{(i,j) | i = 3,4,5,6; j = 4,5,6,7\}$. Then $\mathcal{S}(A) \bigcap \mathcal{S}(B) = \{(i,j) | i = 3,4,5,6; j = 4,5\}$. If these

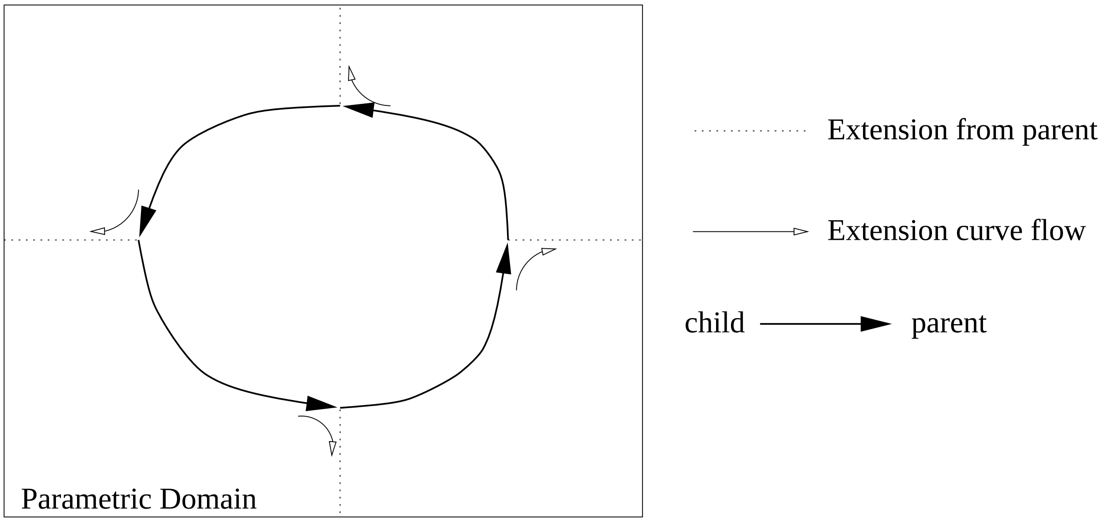


Figure 4.7. Example of circular parent-child relationships.

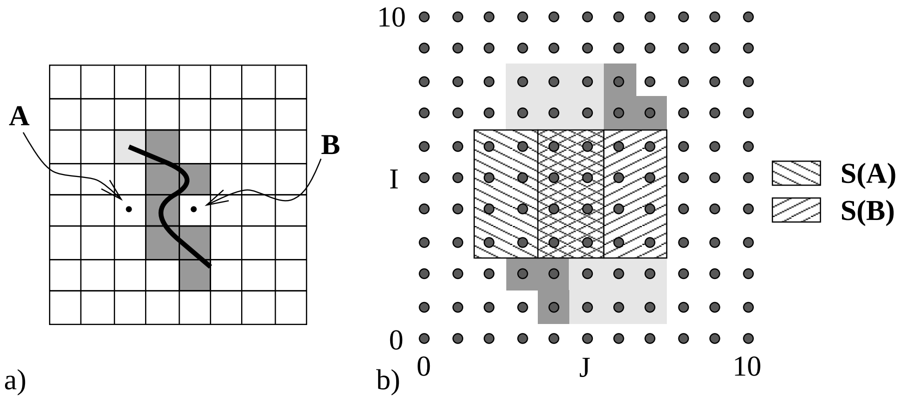


Figure 4.8. Illustration of overlapping spans for two given points. a) Parametric domain with points A and B on opposite sides. b) Control mesh indicating spans.

two points are to be independent, it is necessary to partition, or separate, the surface into regions which are evaluated with different sets of control points. In particular, control points with subscript pairs in the intersection of the spans must have a unique control point identity for each side. The containment function, η, classifies each parametric location with respect to each of these separated parametric regions. The containment function, η, can be constructed, for example, via a set of oriented boundary loops which are determined by extending tear curves to boundaries (see Figure 4.9). The parametric locations on the boundaries between parametric regions are defined in more than one parametric region so the classification must have more information. If the parametric location is on a tear, an *is_left* flag is used to disambiguate the results of this function. If the parametric location lies on an extension of the tear curve, an arbitrary choice can be made since the parametric regions must be continuous along the extension. If the parametric location is on a tear, this ambiguity is expected. Otherwise, the representation enforces smoothness in that region, so the choice is arbitrary.

Definition 4.19 *The* containment function $\eta_c(u,v) : \mathbb{R}^2 \to \{0,1\}$ *is the characteristic function of the restricted domain of the parametric region, c.*

$$\eta_c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in domain(c) \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

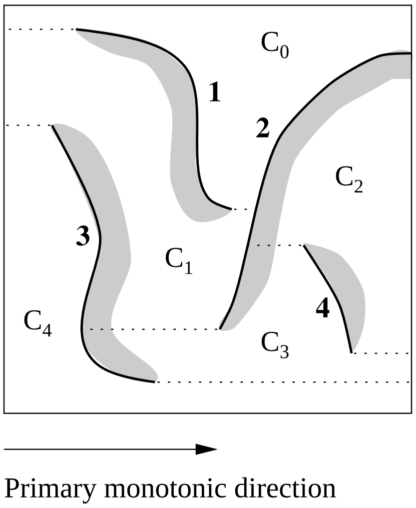


Figure 4.9. Tears in a surface with dashed extension curves.

$\sum_{c=0}^{T} \eta_c(u, v)$ represents the number of distinct parametric regions which contain a particular parametric location. In the interior, $\sum_{c=0}^{T} \eta_c(u, v) = 1$. Along interior boundaries, that is, tears and their extensions, $\sum_{c=0}^{T} \eta_c(u, v) = 2$. At points of intersection of the interior boundaries, the sum can be higher.

4.10 Implementation

In Section 4.7 it was shown that the additional degrees of freedom created by the introduction of a tear into the torn B-spline surface are both necessary and sufficient to provide the maximum flexibility allowed by the discontinuity. Determining the mask values μ for each parametric region is probably the most challenging task in implementing the torn B-splines. Intuitively, each additional coefficient created by the discontinuities in the surface must be used by a parametric region to differentiate it from its adjacent parametric regions across the discontinuity. This is accomplished by assigning to each parametric region a unique combination of additional DOFs from the tears in the structure.

One procedure which determines the parametric regions and their unique combinations of DOFs can be outlined as follows:

1. Split tears into monotonic segments.

2. Compute $O^{(\kappa)}$.

3. Determine necessary extension directions and extend curves.

4. Separate the domain into parametric regions, c_ι, i.e., define η_c.

5. Order parametric regions and tears.

6. Construct $\mu_c(i,j)$.

4.10.1 Splitting Tears

Tear curves are split into monotonic sections with respect to the parametric domain for two primary reasons. Foremost, the monotonicity provides a framework within which parametric regions and tears can be ordered without cycles. A natural ordering can then be derived from adjacency and parametric value information. This is important for determining a unique set of control points for each parametric region. The second reason can be most clearly seen by considering some examples. In Figure 4.10, the tear curve enters and leaves the same patch twice (known as "doubling back"). Intuitively, this should cause the patch to be split into three independent patches (I, II, and III in Figure 4.10). However, since the two outside patches are on the same side of the tear, they will belong to the same parametric region and therefore use the same set of control points. That is, the two outside sections, I and III, will not be independent as expected. A related example is in Figure 4.11. In this figure, the tear curve "spirals" back into the same patch again, causing even more confusion. In Figure 4.10 it is at least clear which region each section of the patch belongs to, even though they are dependent. In Figure 4.11, the center section of the patch in question belongs to the regions on both sides of the curve. Unless the two parts of the tear curve which pass through the patch are differentiated, the description of the surface is ambiguous. In both of these examples, it is not necessary for the curves to pass through the same patches to cause these problems. They only need to pass through patches which are dependent on the same coefficients. It is easy to see that in most cases, if a curve spirals or

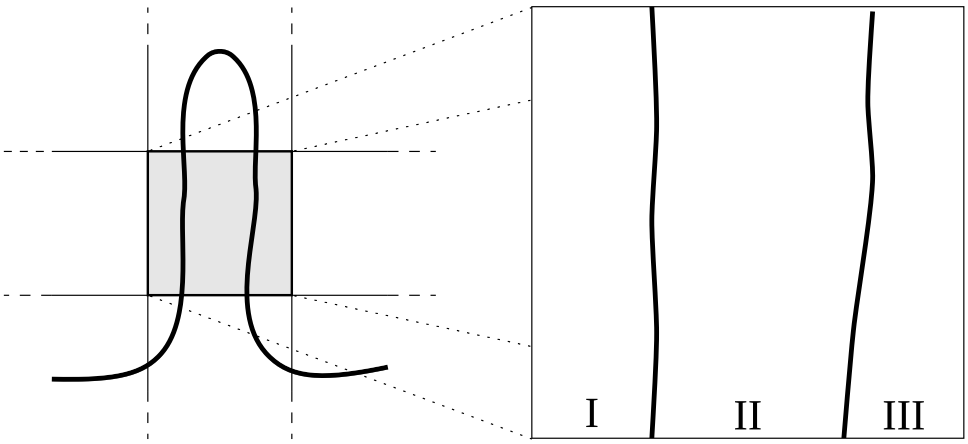


Figure 4.10. Example of a curve that "doubles back."

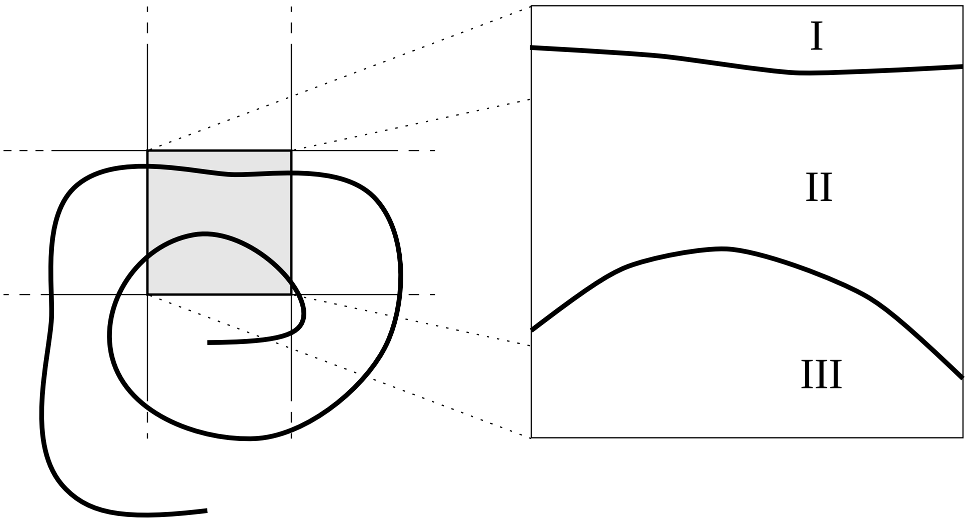


Figure 4.11. Example of a curve that spirals.

doubles back, that this type of problem is likely to occur.

Spirals are eliminated completely by splitting new tear curves into monotonic sections in any given primary direction since to spiral the curve must have a minimum or maximum relative to any given direction. However, isolating monotonic sections in a spiral may result in a given curve section doubling back (see I, II, and III, in Figure 4.12). Unfortunately, the doubling back case is more difficult to correct. Splitting the curve into monotonic sections with respect to both primary parametric directions will solve this problem, but splitting curves this way in all cases is not necessary. It would be the most effective if the problem cases could be identified and split only if necessary. Since identifying these problems (especially the doubling back case) is nontrivial, they will be left for future work.

Splitting a tear results in two independent curves whose endpoints intersect. If endpoints intersect by accident, it is usually difficult to detect, but in this case the exact nature of the intersection is known. The parent-child relationship (introduced in Section 4.8.1) can be assigned arbitrarily by ordering the curve segments

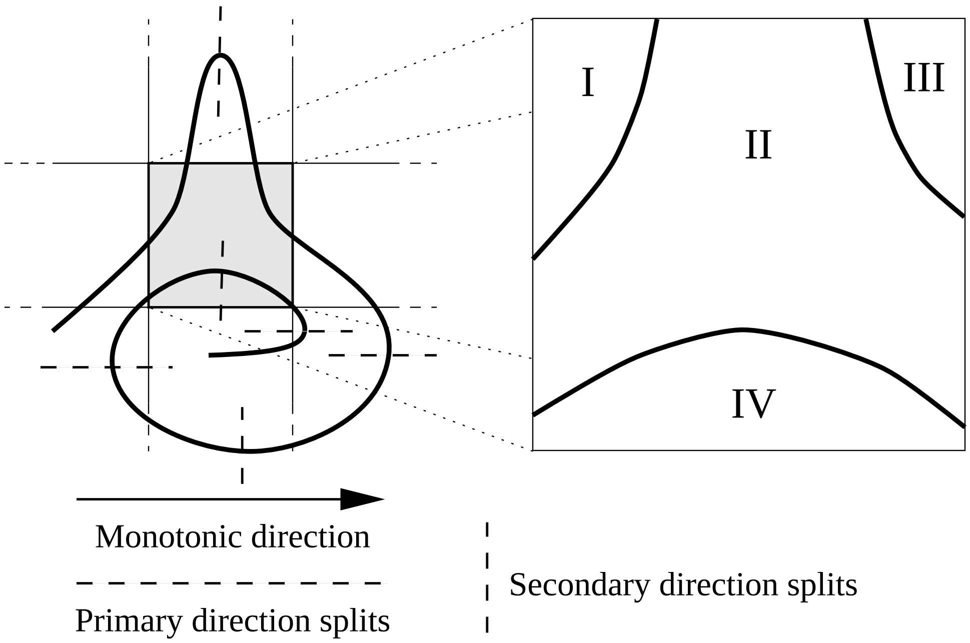


Figure 4.12. Example of a curve that spirals and “doubles back” and the monotonic splits that may be required.

according to beginning parametric value. The parent's endpoint in question is extended in the monotonic direction whereas the child's endpoint is not extended (see Section 4.10.3).

4.10.2 Computation of $O^{(\kappa)}$

Computation of $O^{(\kappa)}$ proceeds directly from the maximal independence algorithm presented in section 4.7.1.

4.10.3 Determination of Extension Directions

The tears are extended from their endpoints in the parametric domain out to another boundary, either an actual boundary of the surface, or another tear or extension. The span of the extension is considered a continuity constraint region, since an extension bounds two or more surfaces whose control points in the span of the extension must be the same. Figure 4.9 shows an example of a surface with a set of tears and their extensions. The regions in Figure 4.9 labeled C_i are the parametric regions separated by the extended tears. The wise choice of the extension direction is essential to a maximally independent surface.

The extension directions are crucial because they influence the ordering of the parametric regions and tears and affect the ease of maintaining the boundary conditions within the structure. Since the extensions are added by the structure, they need to be as unobtrusive as possible. In light of the discussions in the previous section on the importance of monotonicity, it would seem that the best extension would maintains the monotonicity of the tear in the primary direction. Particularly, the span of the extension must avoid intersecting the overlap mesh of the tear. Since the span of the extension is shared by regions on both sides of the tear, automatically maintaining continuity across the extension would be impossible. This is now developed formally.

Definition 4.20 *An extension, ϵ, of γ_κ is* valid *if $(i,j) \notin \mathcal{S}(\epsilon)$ for all $(i,j) \in O^\kappa$.*

To see why a valid extension is required, suppose that span of the extension curve contains a pair (i,j) that is also contained in the set of control points of O^κ.

Choose a point, x, on the extension curve such that $(i, j) \in \mathcal{S}(x)$ (see Figure 4.13). Because of the continuity of B-splines, there exists two points, $\hat{x}_1$ and $\hat{x}_2$, in the neighborhood of x such that $\hat{x}_1$ is on one side of the extension curve and $\hat{x}_2$ is on the other side. Given that the region on one side of the extension does not use O^κ and the region on the other side does, then these two points are not dependent on the same control points and the regions are not the same along the extension curve at x even though they are required to be.

Even with this criteria, there are generally several options, two of the most obvious are given here.

1. Extend the tear by continuing the tear in the tangent direction (in parametric space) of the tear at the endpoint. This approach has the advantage of being easy to understand conceptually, and it maintains the continuity of the tear at the endpoint.

2. Extend the tear by an isoparametrically straight segment in the primary monotonic direction, away from the tear in the direction closest to the tangent at the endpoint. The advantages of this approach are that the signature ordering tends to remain the same independent of the order in which tears

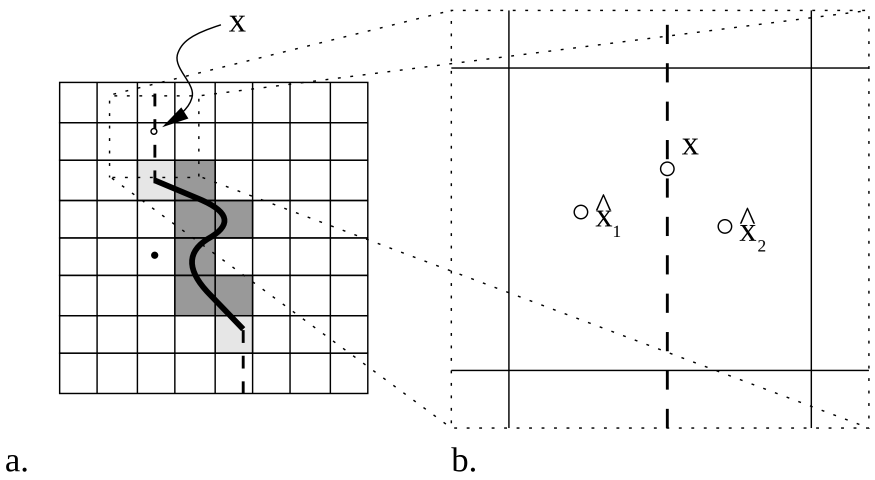


Figure 4.13. Points in the neighborhood of x on either side of the extension.

are extended, and it is generally easier to deal with computationally.

For computational purposes, the second option was chosen for implementation in this thesis. For tears that have been split into monotonic sections, the tangent of the tear at the split point is perpendicular to the monotonic direction, so extensions are made in the direction opposite the normal of the tear relative to parametric space. In the case of a parametrically straight tear, where the normal is not well-defined, the choice is arbitrary. If convenient, the tear may be extended so that the orientation of the extended tear relative to the primary monotonic direction will match that of the other tears.

Another special case occurs when two extensions "run into each other." This happens when the extension directions are opposites of each other and the end-points have the same position perpendicular to the monotonic direction (as in Figure 4.14(A)). The are several options in this case.

1. Choose one curve and extend it slightly perpendicular to the monotonic direction and then proceed as before (as in Figure 4.14(B)).

2. Choose another direction for one of the tears (as in Figure 4.14(C)).

The first option is also probably the best option. This option has the advantage of keeping the ordering characteristics provided by the monotonicity of the tears. The second option has the advantage that a certain set of directions can be preset so that the monotonicity can be maintained and the various choices can be permuted until a solution is found.

Since these extensions are critical to the separation of regions and ultimately to the composition of coefficients for those regions, a solution must be guaranteed if an option is to be viable. A viable solution has the following characteristics:

1. all extensions of tear curves satisfy the requirements of avoiding the overlap mesh of their corresponding tears,

2. no extension may terminate at the endpoint of another tear.

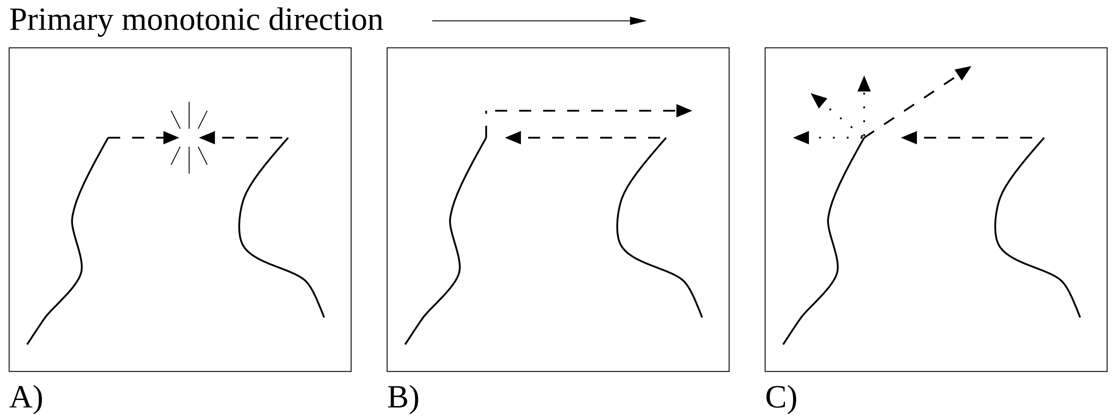


Figure 4.14. Extensions that "run into each other." (A) The problem. (B) Solution 1: Right angle sidestep. (C) Solution 2: Alternate directions.

It is easy to see that the first option given above will always provide a solution. Since there are a finite number of possible extension curves (two for each tear curve) and an infinite number of extension distances, there will always be a distance to extend the curve before turning in the monotonic direction so as to avoid running into any other extension.

The viability of second option is a bit more difficult to show. In order for two extensions to "run into each other," they must be aligned exactly and moving in opposite directions. Further, no other tear endpoint lies on this "line of sight" between the two endpoints (if so, the two would not run into each other). Suppose we label these two extensions as a *conflicting pair*. Let $n(< 1)$ be the number of preset extension directions (given as offsets to the tangent of the curve at the endpoint). Suppose, then, that there exists at least one conflicting pair and that for both extensions, all other possible directions would result in a conflicting pair. Taking it one step further, suppose that the one of the extensions switches directions in response to the conflict. In order to continue having a problem, the new conflicting pair must have potential conflicts in all directions. In the limit, there must be $(2T)^n$ extensions causing conflict, where T is the number of tears, since each extension has a conflicting extension in each direction. But if $n > 1$, this is impossible since there are only $2T$ extensions. Therefore, as long as there are 2 or more possible extension directions, the second option will provide a solution.

Unfortunately, the second option may require finding a solution among $(2T)^n$ possibilities, so heuristics should be used to govern the selection process. Fortunately, experimentation has shown that although the possibility of a conflict occurring is high, especially in artificial situations, a simple set of rules will result in a solution in $O(T)$ time. Following is a brief list of one rule set that is effective.

4.10.4 Initial Extension Rules

Given a curve, determine the initial guess of the extension direction for the extension at the beginning of the curve by the following rules. For the sake of terminology, the tangent of the curve at its beginning points in the direction of curve. The inverse of this tangent points away from the curve. However, the tangent of the curve at the end of the curve points away from the curve and the inverse tangent points toward the curve. All evaluations are relative to the parametric space of the surface.

1. Determine which of the primary monotonic direction or its inverse is closest to the inverse tangent direction at the beginning of the tear. Use this direction if the tangent at the beginning of the tear is not perpendicular to the primary direction.

2. If the tangent at the beginning of the tear is perpendicular to the primary monotonic direction, determine the direction closest to the inverse of the tangent at the end of the tear. Use this direction if the tangent at the end of the tear is not perpendicular to the primary direction.

3. If both ends of the tear have tangents perpendicular to the primary monotonic direction, use the extension direction closest to the vector from the end of the tear to the beginning of the tear, provided this vector is not perpendicular to the primary monotonic direction.

4. If all of the above fail, use the secondary monotonic direction closest to the inverse of the tangent at the beginning of the curve. This is guaranteed (except for rare degenerate cases) to be nonzero if all of the above fail.

Once an initial guess is made, any enumerative technique can be used to find a solution for which there are no conflicts.

The removal of conflicting pairs is necessary in order to produce exactly $T+1$ separate parametric regions. If T tears are in the surface, there is the potential for $T+1$ unique coefficients for a particular (i, j) pair in the torn surface. There must be a unique parametric region for each of these potential coefficients in order to provide the maximum flexibility for each of the tear curves. If a conflicting pair persists, the number of separate parametric regions is reduced by 1.

After the directions have been determined, the curves are extended and information regarding intersections is compiled for each tear.

4.10.5 Parametric Regions, η_c

The current implementation uses trimming loops to separate the domain into parametric regions although other techniques such as a quadtrees may be just as applicable. Information about where the extended tear curves intersect the boundaries and other tears is used to compile a set of boundary loops. One notable side effect of this process is that the particular curve segments of the tear curves which make up the boundary for a given parametric region can be cached. This information is used to determine the signature ordering of the parametric regions and tears (see Section 4.10.6).

4.10.6 Ordering Parametric Regions and Tears

Finally, the piecewise functions, $\mu_c(i, j)$, determine the composition of coefficients from the overlap meshes and original mesh in each of the parametric regions. We refer to the ordering which determines these compositions as a *signature* ordering (defined in Section 4.8). The signature of each parametric region must be unique. The signature ordering is an alternating, possibly partial, precedence relation, beginning with a single root parametric region. The ordering then alternates after this: tear, region, tear, region, etc. Every tear and parametric region are required to be part of this precedence relation. From this relation, the signature ordering for each of the parametric regions is the portion of the precedence graph which precedes the given parametric region (defined formally in Section 4.8). The precedence graph is denoted by an ordered set. For example, suppose a precedence relation for a given torn B-spline surface is $c_I < \gamma_A < c_{II} < \gamma_B < c_{III}$, where c_i are the parametric regions and γ_i are the tear curves. The root of the graph is c_I and its signature ordering is denoted by the empty set, $\{\}$, since nothing precedes it. The signature ordering for c_{II} is denoted $\{c_I\gamma_A\}$. In a similar fashion, the signature ordering for c_{III} is denoted $\{c_I\gamma_A c_{II}\gamma_B\}$. Although every tear and region must be included in the signature for the surface, at certain points the relation is only partially specified. In this case the notation used is illustrated in the following example: if $c_I < \gamma_A < c_{III}$ and $c_{II} < \gamma_B < c_{III}$ are the partial orderings that are known for the surface, then $\{[\{c_I\gamma_A\}, \{c_{II}\gamma_B\}]c_{III}\}$ denotes the precedence relation for the surface. The rules of precedence are given in Section 4.10.6.2 below.

A valid signature ordering does not require that an ordered relationship exists between all combinations of tears and regions. It is possible for the relation to have branches (see cases 2a and 2b, below), in which case a precedence relationship is not defined between members of the branches. However, branching is allowed only in some cases (for examples of inappropriate branching, see cases 1a and 1b, below). It is usually desirable for the precedence relation to be totally ordered, even if the signature ordering is valid. In the case of branches, either a specific method is used to create a total ordering or an arbitrary precedence is assigned for convenience. There are two primary reasons to create a total ordering. First, in some cases it is necessary to prevent DOFs from being dropped. Second, if it is not necessary, then assigning an arbitrary order will not affect the outcome. The reasoning behind this is that if a particular ordering affected the outcome, where the outcome is the interdependence between two parametric regions, then either a precedence relationship can be determined, or it is required (as in the branching case, above). The essential requirement is that the signatures be different: this is what is required to provide the maximum independence between regions. As long as the signatures follow the rules below, any set of unique signatures will result in the same amount of freedom. This can be easily shown by observing that if the signatures are unique, then the difference in composition between adjacent surfaces is the addition of the DOFs associated with the tear which forms the boundary. Therefore, completing the ordering is a good idea regardless of the particular situation. Determining an order with the fewest complications is best accomplished by reorienting the tears so they all travel in approximately the same direction. Then the ordering can easily be determined. For the purposes of this discussion, tears refer to the actual tear combined with its extensions.

4.10.6.1 Reorientation of Extended Tears

Once we have determined the necessary extensions for each of the tear curves, the extended tear curves can be reoriented so that all of the tears are oriented in the same direction within the domain of the surface. If v is the primary monotonic direction, then the extended curves, except for the few rare cases, can be oriented so that the beginning of the curve has the smallest v value. In the rare case in which the tear curve was extended in the secondary monotonic direction, the reorientation is relative to the secondary monotonic direction (in this case, u). After reorientation, the curve's left side is the region to the left of the curve when looking from the beginning to the end.

4.10.6.2 The Precedence Relation

The precedence relation is determined by the following rules.

Rule 1: A parametric region, c_I, precedes a tear, γ_A, $c_I < \gamma_A$, if any part of γ_A's left side lies on an edge of c_I.

Rule 2: A tear, γ_A, preceeds a parametric region, c_I, $\gamma_A < c_I$, if γ_A's right side lies on an edge of c_I.

Rule 3: (Transitivity) If $c_I < \gamma_A$ and $\gamma_A < c_{II}$, then $c_I < \gamma_A < c_{II}$.

Rule 4: No tear may be in immediate predecessor of more than one parametric region.

These rules create a partial ordering (possibly with branches) of alternating parametric regions and tears within the surface. Additional rules used to create a total ordering are given below. When all parametric regions have a unique signature ordering, the signatures are considered to be *valid*.

There are several tear configurations which result in branching of the signature ordering. To provide a valid signature ordering, additional relational pairs may be necessary. The cases in question are:

1a. Two tears intersect the left parametric boundary where the lower tear is oriented toward the boundary and the upper tear is oriented away from the boundary (Figure 4.15).

1b. Two tears intersect the right side of the same tear within the surface where the lower tear is oriented toward the boundary and the upper tear is oriented away from the boundary (Figure 4.15).

2a. Two tears intersect the right parametric boundary where the lower tear is oriented away from the boundary and the upper tear is oriented toward the boundary (Figure 4.16).

2b. Two tears intersect the left side of the same tear within the surface where the lower tear is oriented away from the boundary and the upper tear is oriented toward the boundary (Figure 4.16).

Cases 1a and 1b are similar with the parametric boundary, b, and the larger tear, γ_C, playing the same role. In both cases, the parametric regions, c_I and c_{III}, are not ordered with respect to each other. From the rules given previously, $c_I < \gamma_A$, $\gamma_A < c_{II}$, $c_{III} < \gamma_B$ and $\gamma_B < c_{II}$, and in Figure 4.15(1a), $c_{IV} < \gamma_C$, $\gamma_C < c_I$, $\gamma_C < c_{II}$ and $\gamma_C < c_{III}$ in addition (see Figure 4.15(1b)). The signatures with this partial ordering are

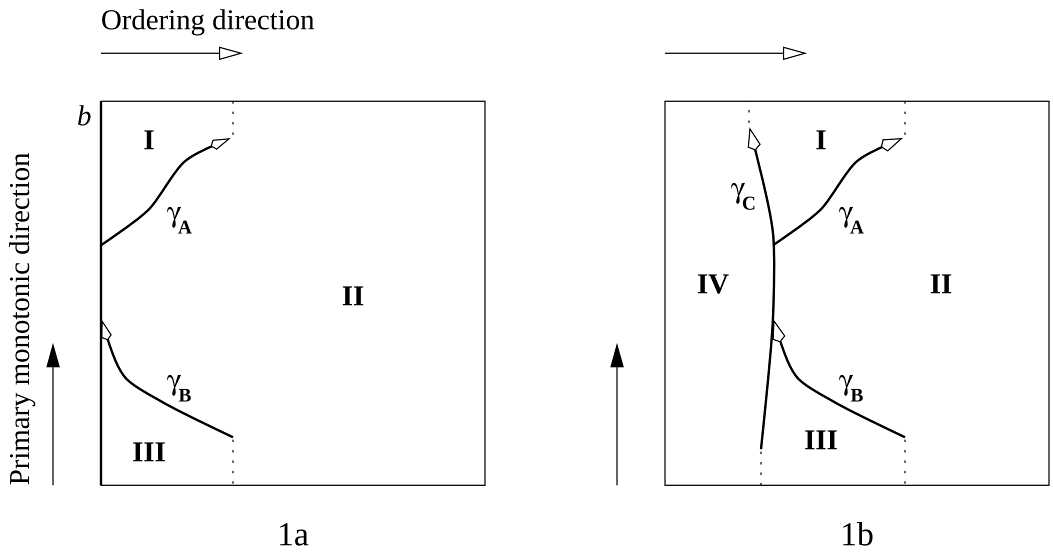


Figure 4.15. Common boundary signature conflicts.

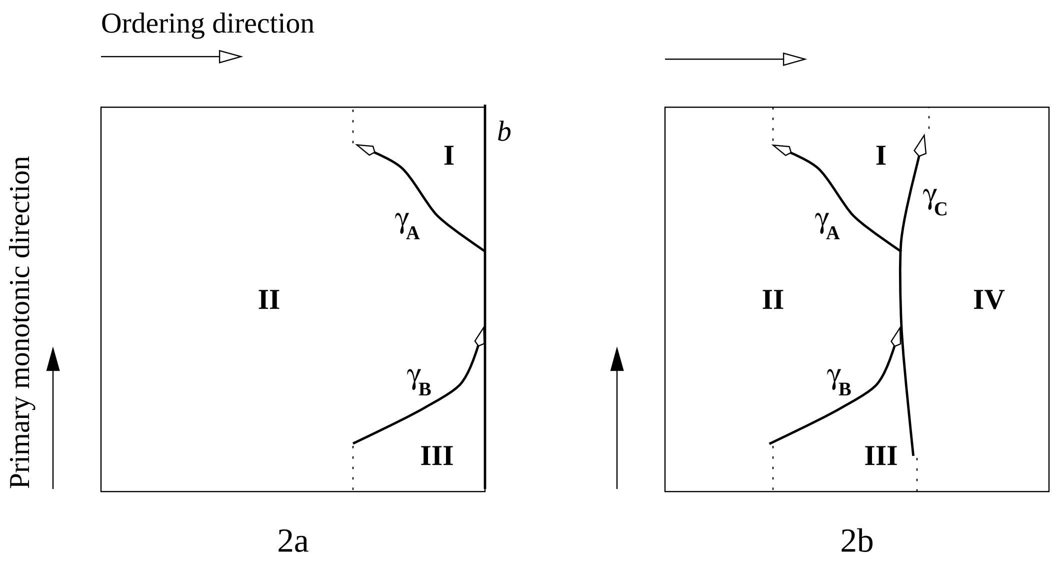


Figure 4.16. Branching of signatures.

$$\begin{array}{rcl} sig(c_I) & = & \{\} \\ sig(c_{II}) & = & \{[\{c_I\gamma_A\},\{c_{II}\gamma_B\}]\} \\ sig(c_{III}) & = & \{\} \end{array} \qquad \begin{array}{rcl} sig(c_I) & = & \{c_{IV}\gamma_C\} \\ sig(c_{II}) & = & \{c_{IV}\gamma_C[\{c_I\gamma_A\},\{c_{II}\gamma_B\}]\} \\ sig(c_{III}) & = & \{c_{IV}\gamma_C\} \\ sig(c_{IV}) & = & \{\} \end{array}$$

where the first column is for case 1a and the second column is for case 1b. Since c_I and c_{III} have the same signature (in both 1a and 1b), this ordering is not a valid signature ordering for either case. Therefore additional relational pairs are needed to produce a valid signature ordering.

The following rule is given to correct this problem.

Rule 5: Suppose a tear, γ_i, immediately precedes more than one parametric region, say c_I and c_{II}, or these two parametric regions are not preceded by any tear. Let γ_j be the tear immediately following c_I. Create the relation, $\gamma_j < c_{II}$.

The creation of this additional relation can be described through the conceptual movement of the common boundary away from the other intersecting tears (be it a surface boundary or a tear), creating what could be called a *phantom region*. If the common boundary is isoparametric, this would then cause a conflict as described in Section 4.10.3 since the extensions would now "run into each other." Since the tears actually intersect a common boundary, choosing any alternative extension directions would resolve the conflict. These new extensions produce an ordering between one of the tears and one of the parametric regions. For example, if the boundary were moved in Figure 4.15 and the tear extension conflicts resolved using option 2 of Section 4.10.3, the configuration in Figure 4.17 results. The additional relation $\gamma_A < c_{III}$ combined with the already established relations is sufficient to determine the valid signature ordering,

$$[c_{IV} < \gamma_C <]\, c_I < \gamma_A < c_{III} < \gamma_B < c_{II}, \tag{4.9}$$

for the surface as a whole (portion enclosed in [] is for case 1b), with signatures:

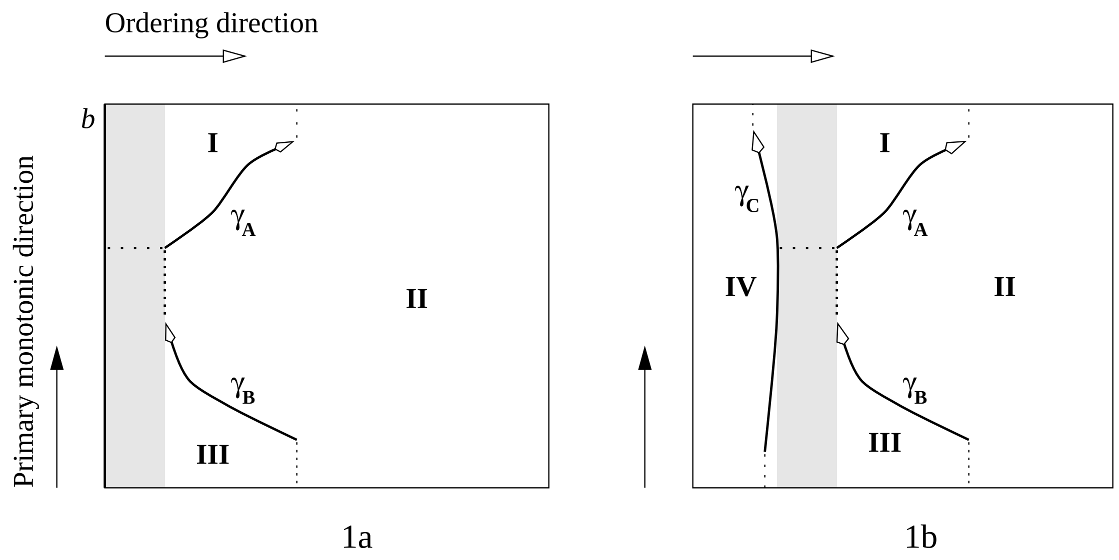


Figure 4.17. Resolution of common boundary signature conflicts by introducing phantom regions.

$$\begin{aligned} sig(c_I) &= \{\} \\ sig(c_{II}) &= \{c_I\gamma_A c_{III}\gamma_B\} \\ sig(c_{III}) &= \{c_I\gamma_A\} \end{aligned} \qquad \begin{aligned} sig(c_I) &= \{c_{IV}\gamma_C\} \\ sig(c_{II}) &= \{c_{IV}\gamma_C c_I\gamma_A c_{III}\gamma_B\} \\ sig(c_{III}) &= \{c_{IV}\gamma_C c_I\gamma_A\} \\ sig(c_{IV}) &= \{\}. \end{aligned}$$

Cases 2a and 2b in Figure 4.16 are similar with the parametric boundary, b, and the larger tear, γ_C, playing the same role. Once again, the parametric regions, c_I and c_{III}, are not ordered with respect to each other. In these cases, $c_{II} < \gamma_A$, $c_{II} < \gamma_B$, $\gamma_A < c_I$ and $\gamma_B < c_{III}$, and the additional relations, $c_I < \gamma_C$, $c_{II} < \gamma_C$, $c_{III} < \gamma_C$ and $\gamma_C < c_{IV}$. Here cases 1a and 1b are different from cases 2a and 2b. The current ordering produces the following signatures:

$$\begin{aligned} sig(c_I) &= \{c_{II}\gamma_A\} \\ sig(c_{II}) &= \{\} \\ sig(c_{III}) &= \{c_{II}\gamma_B\} \end{aligned} \qquad \begin{aligned} sig(c_I) &= \{c_{II}\gamma_A\} \\ sig(c_{II}) &= \{\} \\ sig(c_{III}) &= \{c_{II}\gamma_B\} \\ sig(c_{IV}) &= \{c_{II}\gamma_A[\{c_I\gamma_B\}, \{c_{III}\gamma_C\}]\} \end{aligned}$$

where the first column is for case 2a and the second column is for case 2b. As it is, the signatures for each of the parametric regions are unique (regardless of which branch is taken in the ordering assigned to c_{IV}) and thus make up a valid signature ordering. However, since $sig(c_{IV})$ is not well defined, an arbitrary ordering should be chosen to totally specify the relation. As indicated earlier, total ordering can be derived for all signatures to simplify the situation. A total ordering is obtained by arbitrarily asserting that $c_I < \gamma_B$. The following new signatures would result if the precedence relation were augmented as described.

$$sig(c_{III}) = \{c_{II}\gamma_A c_I \gamma_B\} \qquad \begin{aligned} sig(c_{III}) &= \{c_{II}\gamma_A c_I \gamma_B\} \\ sig(c_{IV}) &= \{c_{II}\gamma_A c_I \gamma_B c_{III}\gamma_C\} \end{aligned}$$

After a valid signature for each parametric regions has been identified, the composition of each of the control meshes for each of the parametric regions via the function μ is determined.

4.10.7 Determination of $\mu_c(i,j)$

The signatures of each parametric region provide most of the information needed to construct μ. Since for any parametric region, the coefficient for a given a subscript pair (i,j) may only be obtained from single source, the function $\mu_c(i,j)$ maps each coefficient, (i,j), of a particular parametric region, c, to a given tear, κ. To begin, all coefficients are from the original mesh, P_{ij}, so all map to tear 0 (an index for which there is no tear). The $\mu_c(i,j)$ are constructed in the order in which c appears in the signature ordering of the surface. For notational convenience, assume that $c_\iota < \gamma_\sigma < c_{\iota+1} < \gamma_{\sigma+1} < \ldots$. Then

$$\mu_{c_{\iota+1}}(i,j) = \begin{cases} \sigma & \text{if } O^\sigma_{ij} \text{ exists} \\ \mu_{c_\iota}(i,j) & \text{otherwise.} \end{cases} \tag{4.10}$$

This means that for a given parametric region, $c_{\iota+1}$, its control mesh is constructed by compositing the additional coefficients in the overlap mesh of tear, γ_σ, on top of the current composition of the previous parametric region, c_ι. However, recall that the coefficients which describe the parametric regions must be identical along any extensions of the tear in order to maintain continuity away from the tear. This requires that $\mu_{c_1}(i,j) = \mu_{c_2}(i,j)$ where $\{P_{ij}\}$ is in the span of an extension and c_1 and c_2 are the parametric regions on either side of the extension. In some cases, conflicts occur which require some additional attention in order to determine the overlap mesh correctly for a given coefficient and parametric region in the span of an extension. These conflicts are resolved by the procedures below which are performed in the order presented.

4.10.7.1 Update Prevention

In rare cases, the span of an extension between c_1 and c_2, where $c_1 < c_2$ according to the signature order, is updated by coefficients from O^κ. If $c_1 < \gamma_\kappa < c_2$ and c_1 is adjacent to γ_κ then coefficients from O^κ will be part of the control meshes for the parametric regions on both sides of γ_κ effectively removing the additional DOFs created by the discontinuity (see Figure 4.18). In this case, O^κ is prevented from using its coefficients in any part of the extension's span. This is not a problem since the span's DOFs are already represented in a parametric region elsewhere. This additional computation is performed by asserting that O^κ does not exist within the context of the definition of Equation 4.10. The modifications are then propagated through to successive parametric regions.

4.10.7.2 Update Propogation

A more frequent case occurs when the span of an extension intersects the overlap mesh of another tear or when one tear has an endpoint on another tear. A solution is to copy the values of μ from one parametric region to the other within the span of the extension. Since this behavior is integrated into the signature concept, the solution to this problem, known as *forward propagation*, is contained in the definition of μ and so is part of Equation 4.10. The requirement that the spans of extensions be identical in both adjacent surfaces, $c_1 < c_2$, is met because if an overlap O_κ updates c_1 and the span is within the update, then c_2 is also updated by O_κ (see Figure 4.19). This assumes, of course, that the update prevention case of Section 4.10.7.1 does not apply.

The second case, whose solution is known as *backward propagation*, sometimes

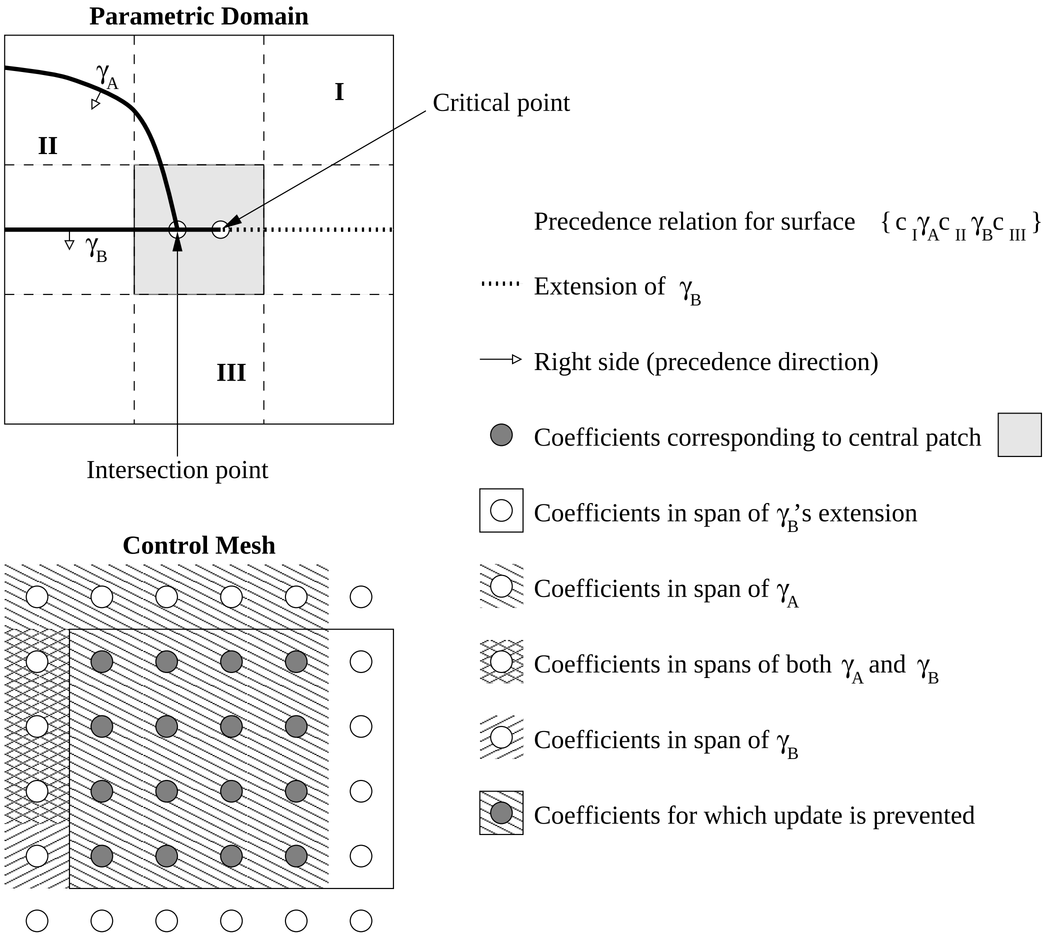


Figure 4.18. Situation requiring update prevention.

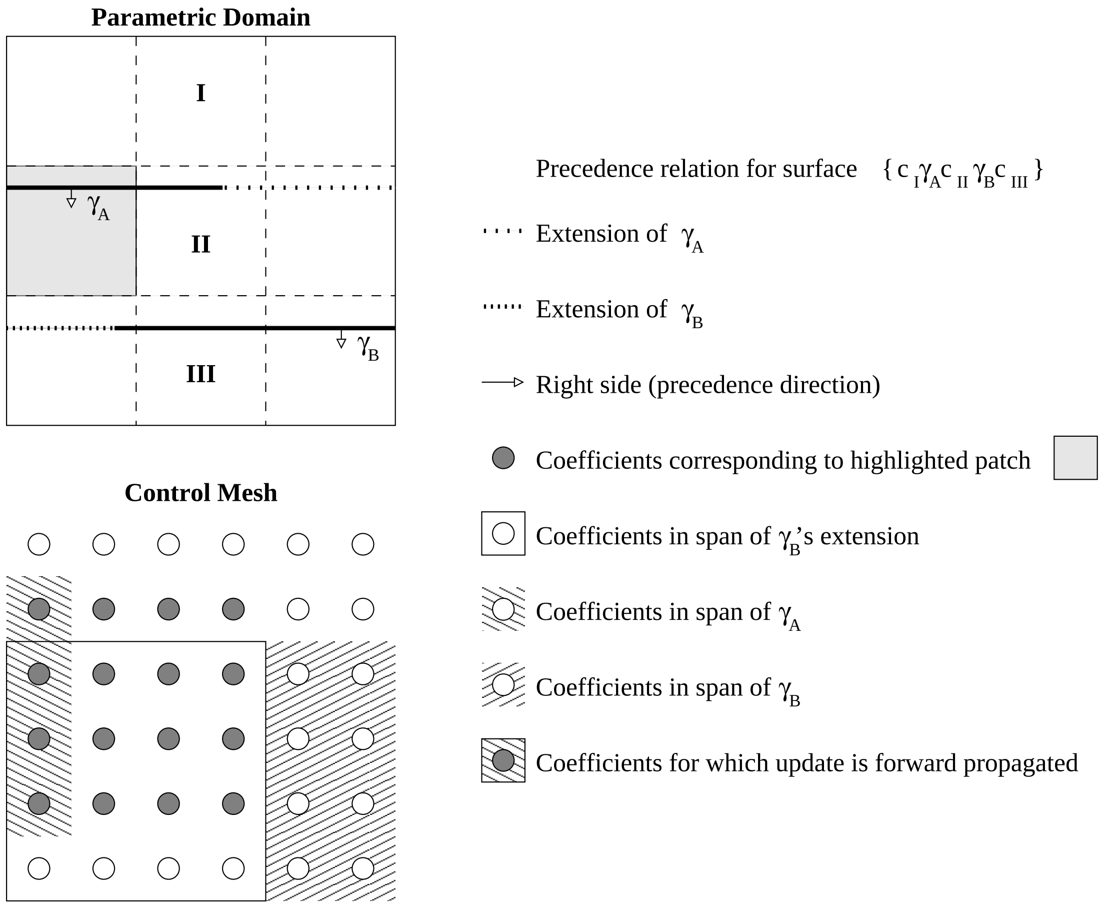


Figure 4.19. Situation requiring forward propagation.

occurs when a tear touches another tear "by accident" but more frequently occurs when a tear is split. Recall the "parent-child" relationship defined in Section 4.10.1. Recall also that when a tear touches another tear in the general case, the child tear has an endpoint on the parent tear. The root of the problem is that the span of the endpoint of a tear that touches a boundary (be it another tear, or not) is not removed from the span of the tear when determining the overlap mesh. The child tear that updates a parametric region adjacent to the extension that connects the parent tear to a boundary will require the use of DOFs from the overlap mesh of the child tear in *all* of its adjacent cached surfaces, even if the inheritance appears backward (see Figure 4.20). The inheritance would be reversed if one of the adjacent cached surfaces precedes the child tear. Therefore, any DOFs in the intersection of the extension's span and the child tear's span must be from the child tear's overlap mesh in all adjacent surfaces. In addition, the DOFs in this intersection must be propagated back to any extensions adjacent to these parametric regions whose spans are also intersect this set (see Figure 4.21).

Back propagation is performed for each child tear endpoint according to the tear order within the signature ordering from Section 4.10.6, after the initial values from the signature for μ are determined.

4.11 $C^{(0)}$ Feature Curves - Creases

$C^{(0)}$ feature lines, or *creases*, are tears with added constraints that maintain transfinite interpolation of each side of the tear along the curve. *Mutual* curve-curve constraints are introduced in order to maintain this connectivity. More precisely, if $c_L(s)$ is the tear curve in the surface on the left side of the crease, and $c_R(s)$ is the tear curve on the right side, then $c_L(s) \equiv c_R(s)$. Note that in general, there are not enough DOFs to produce $C^{(0)}$ continuity along a proposed arbitrary crease curve. A requirement of $C^{(1)}$ or $G^{(1)}$ smootheness along a crease frequently is not possible to meet. However, several conditions contribute to a reasonable solution;

1. relatively small curvature of creases with respect to parameter space,

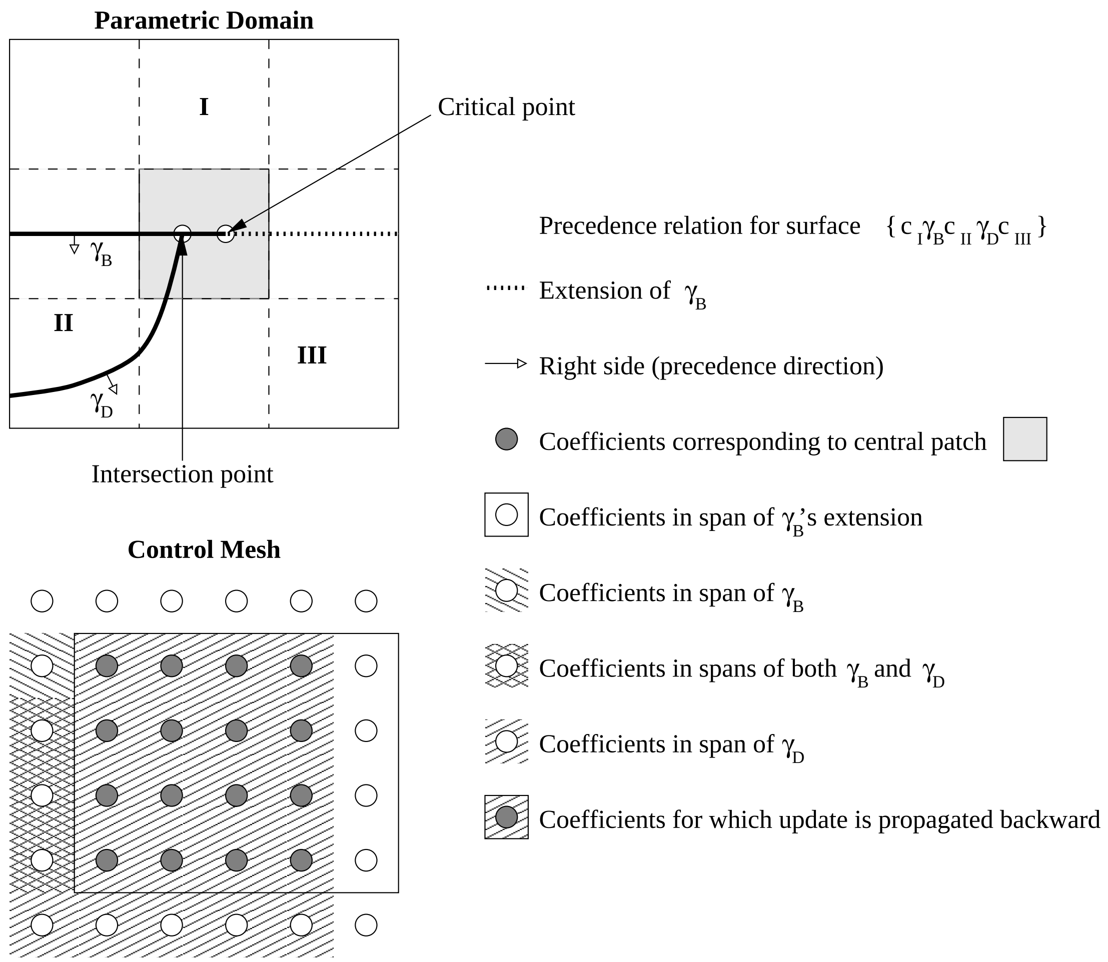


Figure 4.20. Situation requiring backward propagation.

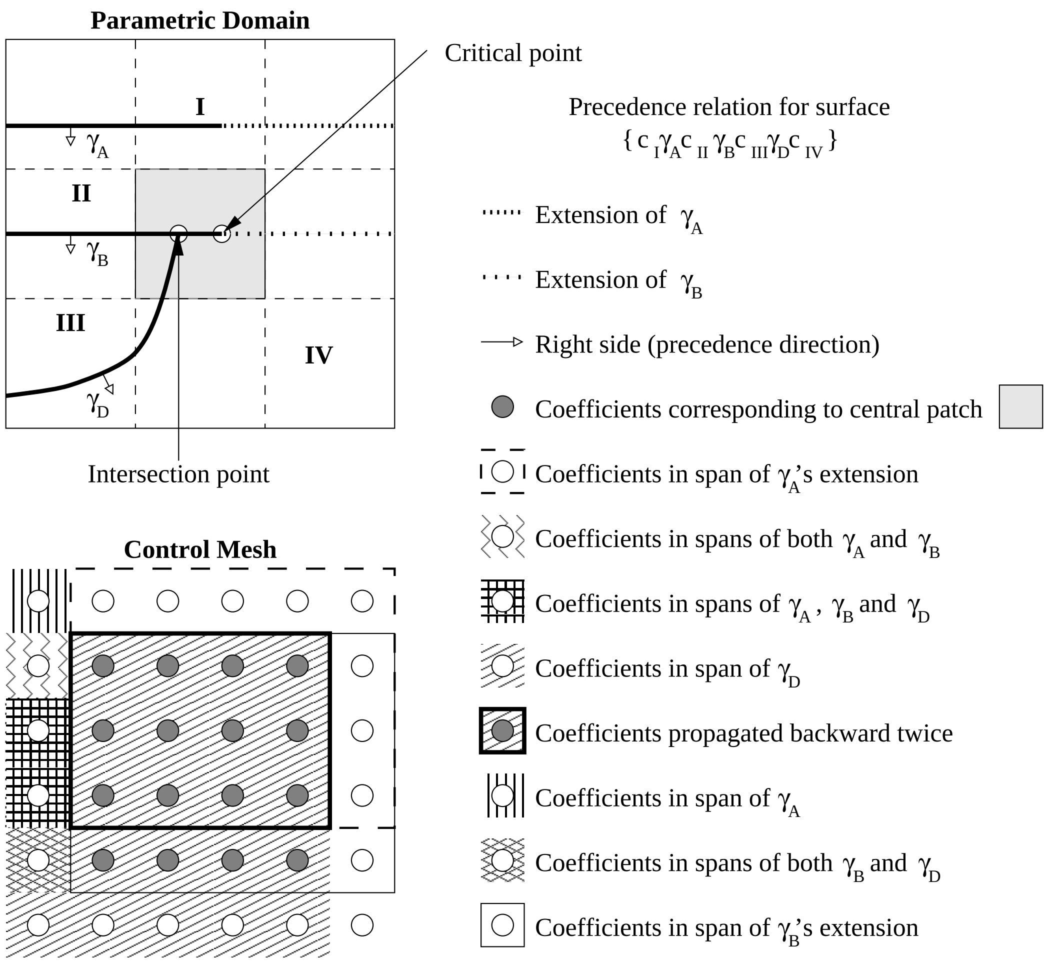


Figure 4.21. Backward propagation through multiple extensions.

2. identical parameterization on each side of the crease curve,

3. existence of a solution (configuration with the original continuity across the crease curve), and

4. the assumption that the constraints can only be approximated because of the nature of the problem.

Minimization of the following equation can be used to help maintain the connectivity after the crease is introduced.

$$\epsilon = \int |c_L(s) - c_R(s)|^2 \, ds. \tag{4.11}$$

Since satisfying this constraint is generally approximated by a linear system and combined with other constraints, a solution in which $\epsilon = 0$ is rarely found, despite the fact that the original configuration satisfies the above constraint exactly.

The introduction of creases into a surface can take place in a variety of situations. First, in the design phase, a crease may be introduced as a feature line. In this phase, the crease is often made isoparametric and the designer may utilize other methods for representing the crease, such as multiple knots, duplicate control points, or multiple surface patches.

During a simulation, creases may be introduced as byproducts of some physical process. Creases in this case are rarely isoparametric so the torn B-spline data structure is ideal. In addition, the crease constraint formulation given above is suitable for linear constraint systems and so can be integrated very easily into most simulation systems. In addition, the fact that at one time the crease did not exist, contributes to the existence of a solution in subsequent time steps.

Finally, in the case of model reconstruction, creases may be introduced as part of a solution to a system of interpolation constraints. The crease may be introduced when it is clear (but is it ever really clear?) that the flexibility of the surface is insufficient to satisfy the constraints on the surface. Much work in vision and model reconstruction has gone into adequately detecting these cases and determining the nature of these continuity features. Once a crease is discovered, there is usually considerable flexibility in deciding exactly where the crease is in parametric space. This presents both opportunity and difficulty. Since placing a curve in parametric space is a highly underconstrained problem, analysis and heuristics provide only a starting guess at a possible solution. The good thing is that an acceptable solution very likely exists because of the large solution space. The difficulty is finding it. Factors to consider are as follows: 1) how much flexibility really exists, or how dense are the data points in the vicinity of the curve; 2) how good is the initial solution relative to the desired tolerance, perhaps endless modifications are not necessary; and 3) how much error is due to incorrect crease placement and how much error could be more appropriately attributed to lack of refinement. To obtain a solution, the suggested crease's constraint is added to the system of constraints already being used by the interpolation.

4.12 Constraints

As already indicated, torn tensor product B-splines are ideally suited for use in linear constraint systems in the same manner as standard tensor product B-splines. Unfortunately, the crease constraint is nonlinear unless it is isoparametric. However, linear approximations to these nonlinear problems have become widely used in constraint systems[90] for several reasons. First, the solutions are reasonable given the flexibility of the surfaces. Designers are looking more for smoothness than for dead-on accuracy. Computing the most accurate solution may not produce the most pleasing design. Second, linear solutions are so much faster than nonlinear constraints that using linear constraints is not prohibitive in the design process. Finally, systems of linear constraints are easy to formulate and require considerably less input from the designer. Information such as the differential of the Gaussian curvature of the surface with respect to each degree of freedom, which is hard to describe and time consuming to compute, is usually not necessary for a linear constraint system (although some components may be the same).

Constraints could be used in conjunction with tears to introduce creases with higher-order continuity, although, at that point they cease to be creases, but it is not clear whether these continuity features are necessary.

4.13 Adding Flexibility Through Refinement

Maintaining $C^{(-1)}$ continuity along the entire feature curve requires additional degrees of freedom. One method of creating enough DOFs to provide this flexibility is to refine the original surface at the ends of the tear so that all tears start and end at subpatch boundaries, then recompute new overlap meshes and define new masking functions. If the end of the tear changes parametric position, as might occur in a physical modeling simulation, the surface would need to be rerefined at the new point. The challenge then, is to maintain as much of the original description in all of the modeling operations as possible. We define the tear-refined B-spline surface as follows:

Definition 4.21 *Suppose that $\hat{\tau}_u$ and $\hat{\tau}_v$ are the new knot vectors for the refinement which include the tear's end points. Let $\alpha_i(\hat{\imath})$ and $\alpha_j(\hat{\jmath})$ be the coefficients for the knot insertion (as defined by the Oslo Algorithm[24]). If $\mu_c(i,j) = 0$, let $O_{ij}^{(\kappa)} \equiv P_{ij}$. We define a new masking function, $\hat{\mu}$, for the refined surface. Then the* tear-refined B-spline surface *is defined by*

$$T(u,v) = \sum_{c=0}^{T} \eta_c(u,v) \sum_{\hat{\imath},\hat{\jmath}=0}^{\hat{m},\hat{n}} R_{\hat{\imath}\hat{\jmath}}^{(c)} B_{\hat{\imath},\hat{\tau}_u}(u) B_{\hat{\jmath},\hat{\tau}_v}(v) \tag{4.12}$$

where

$$R_{\hat{\imath}\hat{\jmath}}^{(c)} = \begin{cases} \sum_{i,j=0}^{m,n} P_{ij}\alpha_i(\hat{\imath})\alpha_j(\hat{\jmath}) & \text{if } \mu_c(i,j) = 0 \\ \sum_{i,j=0}^{m,n} O_{ij}^{(\mu_c(i,j))}\alpha_i(\hat{\imath})\alpha_j(\hat{\jmath}) & \text{otherwise.} \end{cases} \tag{4.13}$$

Modification of surface position can then be computed in terms of the original coefficients.

CHAPTER 5

ANALYSIS

In this section, an analysis of the class of surfaces represented by torn B-splines, using the techniques presented in Section 4.10 to select overlap meshes, masking functions and containment functions and the spatial and computation requirements for using the torn B-spline surfaces, is presented.

5.1 Class of Representable Models

The general class of surfaces represented by the torn B-spline representation is trimmed tensor product B-splines with arbitrary curves of $C^{(-1)}$ continuity which extend from patch boundary to patch boundary. Although the representation supports arbitrary curves of $C^{(-1)}$ continuity whose endpoints are anywhere, the requirement that the patch is smooth everywhere surrounding the endpoints of the tear results in the entire patch being smooth. The alternative is to allow the two patches to have $C^{(-1)}$ continuity in the surface beyond the tear curve.

The smoothness requirement is easily shown by observing that a cubic B-spline has only 16 DOFs. Matching first and second partial derivatives across extension curves to ensure smoothness across patches requires 12 DOFs. However, since four additional DOFs are needed to determine a $C^{(2)}$ curve, the two patches on either side of the tear are fully constrained. One possible method of introducing additional degrees of freedom is through refinement, and this technique is discussed in Section 5.1.

There are other practical restrictions on the geometry of the tears which do not affect the class of representable surfaces but which should be mentioned. The first is that a tear curve may not intersect either itself or another curve. As in the real world, tears which intersect become separate tears at the intersection point, as though they had crossed a boundary of the surface. A self-intersection isolates the surface regions into two independent trimmed surfaces.

Another less obvious restriction is that tears which spiral must be split into multiple tears. This restriction is treated in detail in Section 4.10.1.

5.2 Complexity

Although space and computational requirements have essential minima, neither requirement can be optimized independently of the other. In the following two sections, the space and computation requirements for regular B-spline surfaces, trimmed B-spline surfaces, and torn B-spline surfaces are discussed in relation to each other.

5.2.1 Space

The space required for the regular tensor product B-spline representation is nmd real values for an $n \times m$ grid of $\mathbb{R}^d$ points plus $n + k_u + m + k_v$ reals for the knot vectors and 2 integers for k_u and k_v. The addition of trimming curves to create the trimmed B-spline surface adds only the space for the arbitrary curve representation. In the worst case, for monotonic tears, the torn B-spline data structure requires the original set of nmd reals plus one additional set of nmd reals for each tear in addition to the storage requirements for each tear itself. The identification of the overlap mesh for each tear, which requires nm booleans, and the masking functions for each parametric region, which each require nm integers, are both optional but are time consuming to compute. All other values can be computed as needed without significant penalty. The usual $n+k_u+m+k_v$ reals are required for the knot vectors. See Figure 5.1 for a comparison table. Considering that the B-spline representation is fairly compact, the increase in data requirements is relatively insignificant since T is usually a small constant (i.e., less than 10).

5.2.2 Computation

Analysis of computational requirements for the torn B-spline surface is at best a difficult task. There are a wide variety of tradeoffs available between buffer space

B-spline Type	**Reals**[†]	**Ints**	**Booleans**[†]	**Additional**
Regular	$nmd + n + k_u + m + k_v$	2		
Trimmed	$nmd + n + k_u + m + k_v$	2	curve space	
Torn	$nmd(T+1) + n + k_u + m + k_v$	2	$2nm(T+1)$	tear curve space

[†] n = rows; m = columns; T = number of tears; d = depth of $\mathbb{R}$ space; k_u = row order; k_v = column order

Figure 5.1. Space requirement table

and computation time. Initialization of the representation is the most computationally intensive but needs only to be done once, and then it can be incrementally updated at different stages depending on the amount of buffering and the extent of the modification. Determining the parametric regions is the most intensive subtask. For the trimming loop implementation, this requires arbitrary curve-curve intersection and linking of curves into loops. Other tasks include span computation which involves curve-line intersections and the masking function calculation.

Relative to regular B-spline surface evaluation, torn B-spline surface evaluation is slow. However, torn B-spline surface evaluation is roughly equivalent to trimmed B-spline surface evaluation, especially when performing multiple evaluations when whether the point is contained is not known. Since the repeated cost of multiple evaluations is determining containment, usually by a clipping algorithm, evaluation of a trimmed surface with two trimming loops is no different than evaluation of a torn B-spline with a single tear.

CHAPTER 6

STANDARD METHODS

As regular tensor product B-splines and other spline representations have become more popular and the body of research on them has increased, a certain set of standard methods, or operations, which have been very useful for modeling, have become associated with these parametric representations. These methods include

- Evaluation: used for evaluating the geometric properties of the representation.
- Refinement: used for adding flexibility to a surface.
- Subdivision: used for tessellation of surfaces–among other things.
- Degree Raising: used for changing the smoothness of the surface.
- Knot Removal: used for approximating surfaces.
- Display: used for both isoline and shaded display in advanced graphical modeling systems.

Presented below are the methods as modified for use with the torn B-spline representation. Note that when there are no tears in the torn B-spline surface, the representation is equivalent to the regular B-spline surface representation and further, each of the methods presented below is equivalent to the corresponding method for regular B-spline surfaces. The portions of the algorithms presented in the figures that are added or modified for the torn B-spline surface are boldface.

6.1 Evaluation

The torn B-spline representation derived from the regular tensor product B-spline representation, so evaluation routines at the lowest level are identical. Figure 6.1 provides a comparison of pseudo code for both representations. In the case of a torn B-spline surface, the steps required to evaluate a point at a parametric location, (u, v), are explained further as follows.

1. Determine the parametric region to which the point belongs by determining a c such that $\eta_c(u, v) \neq 0$.

2. Construct a surface from the general orders and knot vectors of the surface and the composite mesh of parametric region.

3. Evaluate $S(u, v) = \sum_{i,j=0}^{m,n} P_{ij}^{(c)} B_{i,j}(u, v)$, where c is the mesh of the parametric region chosen in step 1, using any standard B-spline evaluation method.

The evaluation of an isoline corresponding to a parametric value, u, such as would be used for a line drawing of the surface, can produce either a single torn B-spline curve or a set of regular B-spline curves. For display, a set of individual curves may be the preferred result, whereas for most modeling situations, a torn B-spline curve may be preferred. The steps in Figure 6.2 are analogous to the point case. Since the isoline extends across the entire surface, evaluation is performed with respect to each parametric region.

1. Determine if the isoline passes through the current region. If not, continue with the next region.

2. Create a surface from the parametric region.

Point evaluation: torn B-spline surface

```
1 compute the region containing the parametric location
2 compute a temporary surface for that region
3 evaluate the parametric location in the temporary surface
```

Figure 6.1. Point evaluation. Bold sections are new with torn B-splines.

Isoline evaluation: torn B-spline surface

```
for each parametric region,
   if the region has a valid interval then
   {
        compute a temporary surface for the region
        extract a control polygon for the isoline
        if returning a torn curve then
        {
             if a torn curve hasn't been defined then
                  make a new curve from the control polygon
             else
                  add a region to the torn curve
        }
        else
        {
             make a new curve from the control polygon
             extract a curve segment described by the interval
             add it to the list of returned segments
        }
   }
```

Figure 6.2. Isoline evaluation. Bold sections are new with torn B-splines.

3. Refine the surface so that $order - 1$ knots are at the parametric value, u. This will interpolate the control points of the isoline at u.

4. Extract the control polygon at the index that corresponds to the interpolated control points.

5. Extract the curve section depending on the desired return type.

   To return a torn B-spline curve,

   (a) If no regions have been defined yet, create a regular B-spline curve from the control polygon (the valid region may be needed if the surface is trimmed as well as torn). Otherwise, construct $\eta_r(u)$ for the torn curve by using the valid region, *reg_int*. Here, r represents the separation of the parametric regions of the B-spline curve, equivalent to the parametric regions, c, of the torn B-spline surface.

   To return a set of regular B-spline curves,

   (a) Construct the isoline from the refined control points.

   (b) For each region, extract and return the corresponding segment of the full isoline.

   (c) Append the curve segment to the return list.

Evaluation of a point on a torn B-spline curve is analogous to evaluating a point on a torn B-spline surface so will not be discussed separately.

6.2 Refinement

Refinement takes place in two stages, 1) refinement of the individual parametric regions, then 2) recomputation of the overlap meshes, containment functions, and masking functions. Pseudo-code is listed in Figure 6.3 and the steps are detailed below.

Refinement: torn B-spline surface

```
for each parametric region,
   refine the control mesh for the region
for each tear,
   match tears with their new overlap meshes
recompute the auxiliary functions and their values
```

Figure 6.3. Refinement.

1. Refine each of the parametric regions, each region inheriting the knot vectors and orders of the surface as a whole.

2. Recompute which points belong in the overlap meshes based on the refined control point structure.

3. Recompute η and μ functions based on the refined control point structure.

Refinement of a regular B-spline surface does not change the geometry of the surface, only the flexibility of the surface. The same is true for refinement of a torn B-spline surface.

6.3 Subdivision

Subdivision is initially similar to refinement, but because the parametric domain of the surface changes, more drastic restructuring is needed. The steps for subdivision are given in Figure 6.4 and are explained below.

In the given pseudo-code, it is assumed that only one region of the surface is returned. If more than one region is desired, this process can be repeated for each subdivided section. The mapping functions in steps 4 and 5 allow any or all of the subdivided surface sections to be returned. Recall that the subdivided surface sections are complete surfaces and so will be referred to as such.

1. Subdivide each parametric region, storing the new control meshes for later use.

Subdivision: torn B-spline surface

```
for each parametric region in region_list,
   compile a list of subdivided regions
for each tear in tear_list,
   compile a list of tears clipped to the subdivision region
recompute auxiliary functions for the new lists of tears and regions
create a map between old and new parametric regions
create a map between old and new tears
transfer the overlap information from the old
                      to the new surface using the maps
```

Figure 6.4. Subdivision algorithm.

2. Clip each tear to the boundaries of the subdivided region.

3. Recompute the structure for the new surface. The new regions and new tears are provided in this recomputation function.

4. Create a set of maps, $\psi^{\iota} : c \rightarrow c^{\iota}$, between the old parametric regions and the new parametric regions for each of the returned surfaces, ι. This is a potentially many-to-one mapping.

   The maps are constructed by first extracting one point contained in each of the parametric regions of the old surface (see Figure 6.5). If possible, these points are chosen to be within the domain of the subdivided region as well. Then for each point, the corresponding parametric region in the new subdivided surface is determined. Because this is a many-to-one mapping, $\mu_c(i,j)$ may equal κ for more than one parametric region, c, in the new surface. However, because of the continuity requirements of the original surface, it is guaranteed that the μ functions are nonzero for the same values of κ. If this were not the case, then DOFs would be lost in the translation.

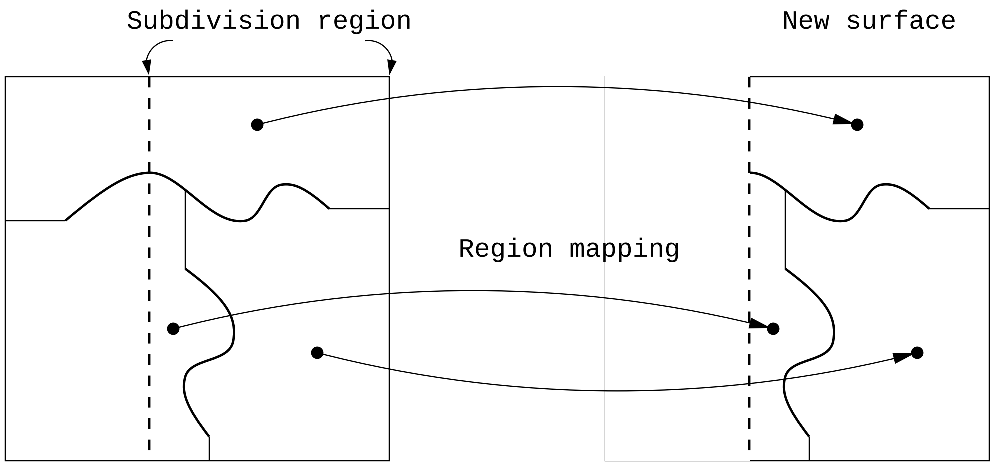


Figure 6.5. Subdivision mapping of parametric regions.

5. Create a mapping between the old tears and the new tears for each of the returned surfaces.

6. Transfer the values for the overlap meshes from the new region list as determined by the mappings just created.

The mapping in step 5 is defined by $\zeta : \kappa \to \kappa$, from tears in the old surface to tears in the new surface (see Figure 6.6). If a tear, κ, does not exist in the new surface, ι, $\zeta^{\iota}(\kappa) = 0^{\iota}$. Let the base control points in the new surface ι, formally, $P_{ij}^{(\iota)}$, be referenced equivalently by $O_{ij}^{(0^{\iota})}$. (Recall that κ indexes start at 1.)

In the transfer function, the new mesh points are reconstructed by the following formulas. For each new surface, ι,

$$P_{ij}^{(\psi^{\iota}(c))} = P_{ij}^{(c)} \text{ for each } c, \tag{6.1}$$

where $P_{ij}^{\nu^{\iota}}$ are the sets of composite control points for each of the parametric regions, ν, in the new surfaces. $P_{ij}^{(c)}$ are the control points for the parametric region, c, in

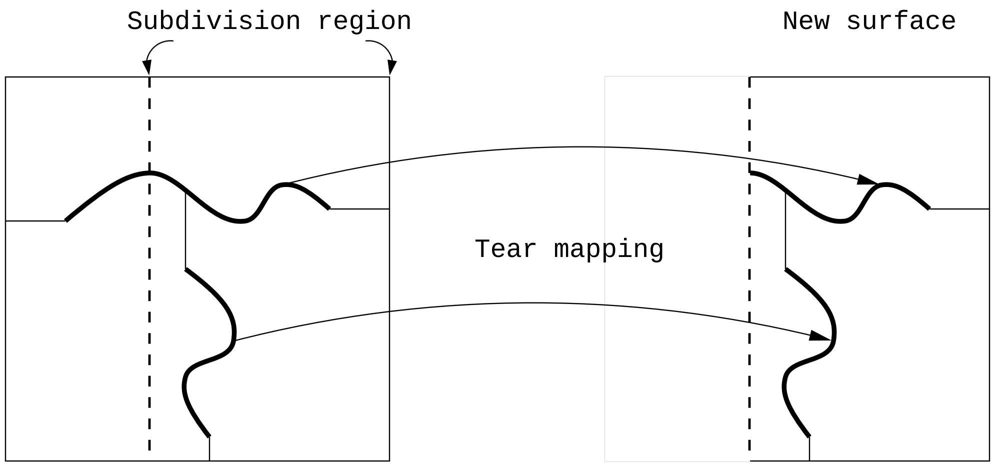


Figure 6.6. Subdivision mapping of tears.

the old surface. Then for each of the old tears, κ, and each of the new surfaces, ι, let

$$\nu_c^\iota = \mu_{\psi^\iota(c)}(i,j). \tag{6.2}$$

Then,

$$O_{ij}^{(\nu_c^\iota)} = P_{ij}^{(\psi^\iota(c))} \text{ for each } c \text{ and } ij \tag{6.3}$$

where μ is defined with respect to the new surfaces and $O_{ij}^{\nu_c^\iota}$ are the sets of control points for each of overlap meshes in the new surfaces.

A slightly modified version of this algorithm will also be used in Section 7.1.1 for splitting up surfaces when tears become complete.

6.4 Degree Raising

Frequently, additional flexibility is desired while modifying a tensor product B-spline or other parametric surface. Refinement is one possible method of increasing the flexibility, but refinement does not change the order of the surface. If the surface is initially order 2 in each direction, the surface is made up of piecewise bilinear patches. Refinement of this surface will result in a piecewise bilinear surface. To increase the smoothness capabilities of a surface, the order of the surface needs to be increased. When raising the degree of a B-spline surface, enough knots are added to the existing knot vector to support the higher order while maintaining the previous continuity characteristics at the original knots. The new control points are computed as a linear combination of the original control points[6, 30].

The degree-raising algorithm for torn B-splines is very similar to the refinement algorithm. Pseudo-code for the torn B-spline degree-raising algorithm is given in Figure 6.7. Since a degree-raised surface and its original surface are geometrically identical and the new surface has more flexibility than the original surface, there are clearly enough degrees of freedom in the new torn surface to accurately represent the original surface. Once again, each parametric region is degree-raised independently; then the tears and the characteristic functions are recomputed. Informally, since the new surface is geometrically equivalent to the original surface, the continuity requirements between parametric regions are not compromised during the process.

6.5 Knot Removal

A surface after knot removal is, at best, an approximation to the original surface. A knot removal algorithm which can approximate a B-spline surface up to a given

Degree Raising: torn B-spline surface

```
for each parametric region,
   degree raise the control mesh for the region
for each tear,
   match tears with their new overlap meshes
recompute the auxiliary functions and their values
```

Figure 6.7. Degree raising.

tolerance has been suggested by Lyche and Mörken[54], and this algorithm can be extended to torn surfaces. The knot removal algorithm for torn B-spline surfaces is given in Figure 6.8.

1. Perform the following two steps for each parametric region:
   (a) Decompose region into scalar surfaces in each dimension,
   (b) Reduce knots in each of the scalar surfaces,
2. Construct a common knot vector set from the reduced scalar surfaces,
3. Refine all scalar surfaces to match common knot vectors,
4. Compose the reduced surfaces into one surface with the common knot vectors.

The primary difference between regular B-spline surfaces and torn B-spline surfaces for knot removal is the decomposition of the surface into scalar surfaces. One possible method for extending this algorithm for torn surfaces is to decompose each of the parametric regions into their respective scalar surfaces so that reduction can be performed on each. A common knot vector can then be constructed from all

Knot Removal: torn B-spline surface

```
for each parametric region,
{
   decompose regions into scalar surfaces
   reduce the knots in the row direction
   reduce the knots in the column direction
}
construct a common knot vector in each direction
refine the scalar surfaces to the common knot vectors
compose the scalar surfaces into a new surface
recompute the auxiliary functions and their values
```

Figure 6.8. Knot removal for regular and torn B-spline surfaces.

of the reduced knot vectors. The parametric regions are then composed individually, and the control points of the original surface and the overlap meshes are determined in a similar manner to the restructuring phase of refinement discussed earlier. With regular B-splines, reducing knots in several scalar surfaces can produce knots that are close but unequal to each other, causing more knots to be added than necessary. Since torn B-splines have significantly more surfaces and contain regions which should not be considered, this problem can be severe. A better algorithm would reduce knots in the surface as a whole, not in each parametric region independently.

Although construction of a common knot vector set is guaranteed to preserve the tolerance when reuniting the surfaces, the nature of torn B-splines suggests that this tolerance may not be maintained in the restructuring phase due to significant dependence on placement of knots near the endpoints of a tear. The extent to which this placement actually affects the final geometry's deviation from the original surface is a subject for further research.

6.6 Display

Two of the more common methods of display for regular B-spline surfaces are isoline and shaded-surface display. Since these methods are widely used for B-spline surfaces and there may be other display techniques which would be applicable, the issues surrounding display techniques for torn B-splines will be considered in general. For most display techniques, straightforward application of the refinement or subdivision algorithms discussed above is sufficient. However, there are shortcuts available because of the structure of torn B-splines.

6.6.1 Effective Use of Cached Surfaces

The torn B-spline surface contains parametric regions which are unique with respect to their composition of control points. Although these regions are easily computable when necessary, it may be practical to construct a regular B-spline surface for each of the regions, including all of its parameters, based on the composition of control points in the region and the general information in the torn B-spline surface. These surfaces can then be cached to facilitate evaluation of the torn B-spline surface. Further, the boundaries of the parametric regions can be instantiated as trimming loops on these cached surfaces.

Another shortcut can be taken in isoline display. To show isolines of a torn B-spline surface, isovalues for the isolines in each parametric direction are determined and the surface is refined independently in each direction so that the control points for these isolines are interpolated in each direction by the refined meshes. However, for torn B-splines, once the isovalues are computed, the individually trimmed parametric regions can be displayed without determining the individual regions for each of the isolines as indicated in the isoline evaluation algorithm in Figure 6.2.

In addition, a display algorithm will generally display the entire surface unless it is trimmed. When a surface is trimmed, however, the display time usually increases dramatically. Figure 6.9(A) illustrates a typical situation requiring isoline evaluation. The performance hit for a trimmed torn B-spline is even worse considering that the isolines are first trimmed to the boundaries of the parametric regions (Figure 6.9(B)) and then to the trimming loops of the surface (Figure 6.9(C)). The result is shown in Figure 6.9(D)). If the 2D intersection of the domains represented by the boundaries of the parametric regions and the trimming loops of the surface are pre-computed (Figure 6.9(E)), then a single clip to the intersected region (Figure 6.9(F)) is performed instead of two clips. Other than the one time intersection of the two regions, the display performance of a trimmed torn B-spline matches that of the nontrimmed torn B-spline surface.

6.6.2 Black Holes

The shaded surface display also benefits from caching the parametric regions as surfaces. One particular issue of a shaded surface display are *black holes*. These holes appear when two surfaces meet at an edge, but the display algorithm does not recognize the adjacency and enforce it. When converting to polygons, the piecewise linear edges of the two surfaces may not match and part of the (usually black) background may be seen between the surfaces. Since the individual cached surfaces can be displayed for a torn B-spline surface, the boundaries of the cached

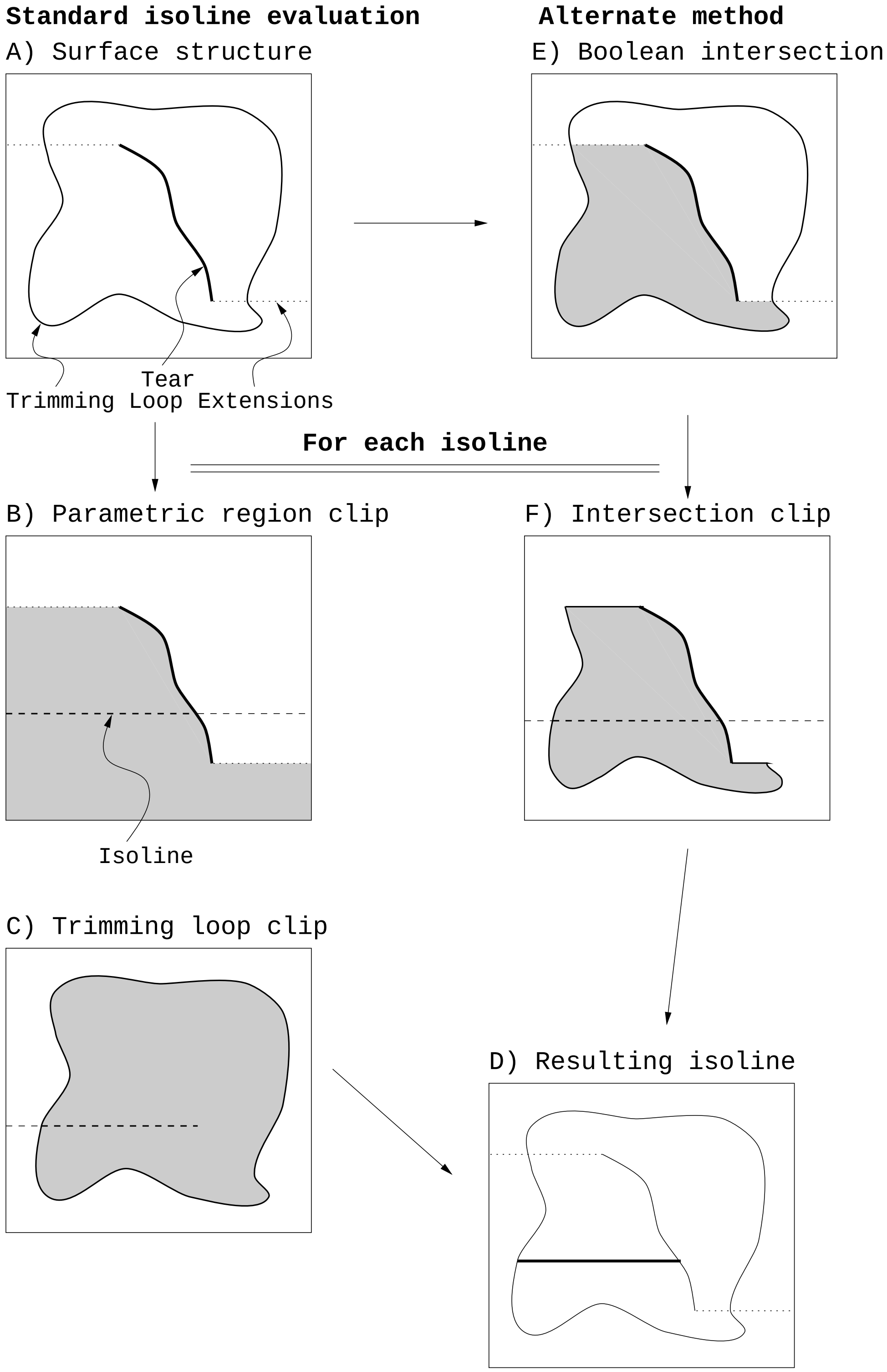


Figure 6.9. Illustration of alternate routes for isoline evaluation.

surfaces along the extensions of tears need to be included in some sort of adjacency maintenance procedure to prevent black holes along the boundary. A procedure of this sort subdivides the surface on either side of the boundary in exactly the same locations so that the resulting polygons on both sides match up vertex by vertex.

Unfortunately, if the boundaries of the parametric region are represented by trimming loops, the display of torn B-spline surfaces is slower than regular B-spline surfaces. However, the representational flexibility gained by the introduction of arbitrary discontinuities offsets this disadvantage.

CHAPTER 7

MODELING WITH DISCONTINUITIES

In a comprehensive modeling system, many unusual shapes arise from seemingly ordinary modeling operations. These irregular shapes can be the result of explicit or implicit, low-order or high-order operations. An explicit (or direct) operation is an operation in which the designer has control over the resulting shape subject to geometric requirements. An implicit (or indirect) operation is an operation in which the designer does not have control over the resulting shape but rather has control over certain properties of the resulting shape. A low-order operation requires the designer to manipulate the parameters of the representation directly. A high-order operation allows the designer to manipulate other parameters which have a predefined effect on the model's representation. For example, a low-order, explicit operation would be manipulating the control points of a B-spline surface. A high-order explicit operation would be a bending operator for the same B-spline surface. Instead of control points, the designer manipulates a radius, a center point, and start and end positions[23]. A low-order implicit operation might be a physically based operation where the user specifies boundary conditions for a B-spline surface, but the remaining degrees of freedom are computed according to the physics of the simulation. An example of a high-order implicit operation might be physically based fairing operator[90, 21]. The fairing function allows the user to specify the curvature requirements of the surface subject to the physical simulation laws. The final outcome of the surface is not necessarily known by the user, and direct specification of individual surface parameters may not be permitted.

Discontinuities within a model can arise in many different situations and as a result of all different types of operators. Several operators which can be used to create irregular shapes from models while maintaining a consistent closed boundary representation are introduced in Chapter 8. First, however, we will look at the interaction of tears with a conventional closed (manifold) boundary representation and some of the more basic issues involved in creating complex B-spline models.

7.1 Tears and Trims

Most models constructed from B-splines are not created solely from the B-spline surface with the canonical four parametric boundaries. To require this would be to limit the interesting models which can be created. In fact, most modeling systems use some method of trimming tensor product B-spline surfaces when they are part of a boundary representation of a model. This type of trimming is easy to support for torn B-spline surfaces as well.

First, the algorithm which constructs the overlap mesh must recognize the trimming boundaries as the boundaries of the surface. The primary issue, in this case, is whether or not a critical point of a tear is still a critical point. Since the domain is restricted, what previously was a critical point of a tear may now lie on or outside of a boundary, making it noncritical. A secondary issue is whether or not a tear extends from one boundary of a trimmed region to another boundary of the same region. The tear would separate the domain of the surface, even if the tear is completely in the interior of the parametric domain of the surface. Generally, trimmed boundaries are not expanded once trimmed. Boolean operations may further limit the surface, but expansion takes place by the addition of other surfaces. Because of this, a tear which separates the trimmed region (Figure 7.1(A)) can be extended to the boundaries of the trimmed surface (Figure 7.1(B)) and treated as though it was a complete tear (see Section 7.1.1). If the tear is not made complete as suggested, it is possible that the two regions may share control points and could therefore be dependent.

Next, the parametric regions are trimmed against the surface's trimming curves in order to perform evaluation and display as explained in Section 6.6. If trimming loops are used to represent the parametric regions, then a 2D Boolean intersection

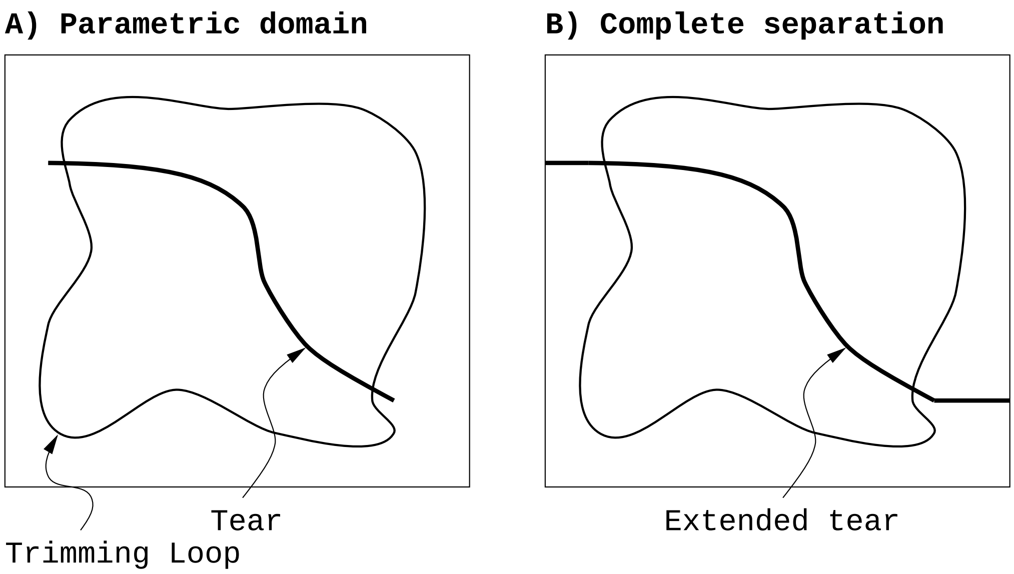


Figure 7.1. Extending a tear to completely separate the parametric domain.

is performed on each pair of loops. The primary concern for this procedure is the robustness of the 2D Boolean operation which is performed using these curves. Numerical problems in the intersection can result if the trimming curves of the original surface coincide with the boundaries of the parametric regions. In most cases, however, *a priori* knowledge can provide the necessary information to adequately determine the intersected regions.

7.1.1 Complete Tears

Not all tears begin and end in isolation. Frequently, tears originate at boundaries and continue to other boundaries. On occasion, tears may make a complete loop effectively isolating a portion of the domain of the surface. Fortunately, this case can be handled without difficulty in nearly the same manner as any other case. Enough DOFs are added to provide a completely independent section of the surface through normal tear processing. However, at times, a single torn surface needs to be separated into two individually trimmed surfaces when a complete tear is made. One primary advantage of this is the simplification of the data structures. A disadvantage is that the connection between the parametric regions of the original surface is lost.

The conversion of a complete tear to two trimmed surfaces is intuitively straightforward. First, the portions of the tears which are part of the complete loop are identified and the remaining portions of the tears are identified as either inside or outside the complete loop. Then a surface is created for both of the separated regions with the complete loop providing the trimming information. The remaining tears are then added to the surface which contains the tear in its nontrimmed region. At this point, the auxiliary functions μ and η are redefined for both of the surfaces.

Following is a more detailed procedure for performing this conversion with a discussion of some of the factors that should be considered.

1. Identify the newly closed loop. It is assumed without loss of generality that only one closed loop exists. If the parametric regions corresponding to the η functions are not represented by trimming loops, then they are converted to trimming loops at this time—one for each region.

2. *Optional*: Concatenate tears that make up the new loop if possible. See Section 7.2 for details on this procedure.

3. Mark the sections of the tears which make up the loop, separating the inside and outside. During initial construction of the parametric regions, the sectioning of the tears may be performed. If this information is computed and cached, the sections do not need to be recomputed here. The appropriate sections are merely marked in this case.

4. Construct two individual surfaces, OutsideSrf and InsideSrf as follows:

   (a) Knot vectors, orders, and end conditions are copied to new surfaces without tear information.

   (b) Construct the trimming loop for each new surface by performing a 2D Boolean operation between the newly formed loop and the trimming loops from the original surface as follows:

$$LoopOut = OldLoop - NewLoop$$

and

$$LoopIn = OldLoop \cap NewLoop.$$

(c) Add the sections of the original surface's tears contained in loopin to the InsideSrf surface. Add all but one of the tears which make up NewLoop to OutsideSrf along with any portion of any tear which is contained in OutsideSrf.

(d) Recompute μ and η functions for the new surfaces.

(e) Define two mappings from the parametric regions in the original surface to parametric regions in each of the new surfaces as detailed in Section 6.3.

(f) Define two mappings from tears in the original surface to tears in each of the new surfaces as detailed in Section 6.3.

(g) Update the overlap meshes using the coefficient mapping functions from the previous two steps. This is also detailed in Section 6.3.

Several of the above steps require additional explanation. In step 4c, the number of tears is reduced by exactly one. Intuitively, it may seem that all the tears which make up the boundary of the closed loop could be eliminated since the surface is being separated, but upon further inspection, it is clear that this naive approach leads to the ordinary trimmed surface dependency that is to be avoided. Consider a simple example: two tears are added to a surface, each having the same endpoints (see Figure 7.2). Assume that they are not the same tear. In the original torn surface, the point A in the surface above tear I is independent of the point in the surface below tear II, despite the fact that the two curves may pass through the same patches along their entire length. If this surface is split into two surfaces—the interior and exterior of the loop—and if the exterior surface is implemented as single trimmed surface, the points A and B will be dependent.

A single parametric region can be removed because the interior surface (ignoring its contained tears) behaved in the old surface like a single parametric region. Since the surface is being separated, one of the tears which helped make the surface

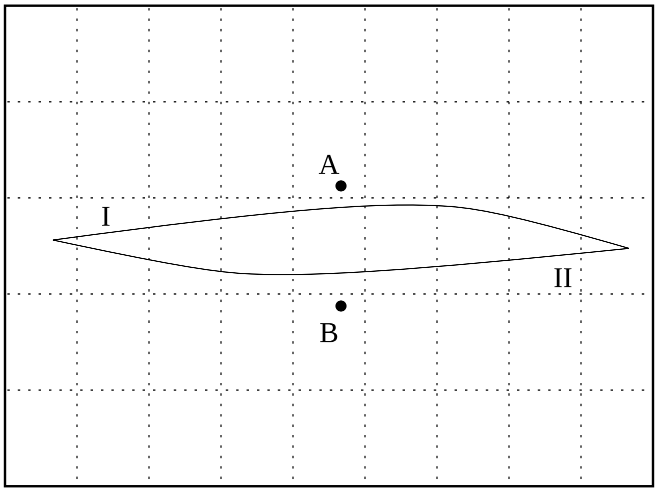


Figure 7.2. Illustration of tear dependence during conversion of complete tears. Dotted lines delineate subpatches.

unique can be removed. The choice of which tear to remove is arbitrary. However, heuristics indicate that the tear with the most DOFs is the best option. The potential drawback is that OutsideSrf will end up with DOFs which do not affect the final surface shape.

The last several steps of the separation algorithm are taken from the subdivision case and are discussed in detail in Section 6.3.

7.2 Tear Reduction and Equivalence

At times it may be desirable to reduce the number of tears that are maintained in the surface or change the configuration of tears within the surface. Tear reduction is desirable whenever two tears meet end to end, as in the case of complete loop formation. It may also be possible to isolate a tear within the complete loop, so a configuration change may be appropriate. The general rule for tear reduction (by concatenation) is the inverse of tear splitting (see Section 4.10.1). If the concatenation of the two tears results in a monotonic section, or, in other words, the splitting algorithm would not resplit the tear, then the tears may be concatenated. The primary advantage of tear reduction is the simplification of the surface structure.

Reconfiguration of tears is generally less critical and may be difficult to assess and perform automatically. One primary area of usage is in surface reconstruction where the tear configuration and surface structure are not *a priori* knowledge. Determination of the tear configuration is then a very important and necessary task.

Consider the following example, illustrated in Figure 7.3. Suppose that discontinuities occur in the surface such that they extend away from a given point in more than two distinct paths. The logical choice is to create tears in tangent continuous segments. The configuration shown in Figure 7.3(A) would result in three distinct tears emanating from the intersection point. The configuration in (B) would result in two tears with the second tear's end point intersecting the first at the intersection point. The configuration in (C) would be similar; only a choice needs to be made between two possible distinct configurations. However, in any of these cases, the choice is either two tears in two ways or three tears meeting at one point. The choices differ only in aesthetics and structure; the flexibility of the surfaces are equivalent. The tears are equivalent for a very simple reason. Separating a tear into two sections is an equivalence operation if the tears are in monotonic sections. This is easily seen by the following observations. First, by assigning the parent-child relationship between split sections, we see the span of the split point is only in the overlap mesh of the child tear. Since the tear is monotonic, it does not double back, so the two sections do not have any control points in common except for those in

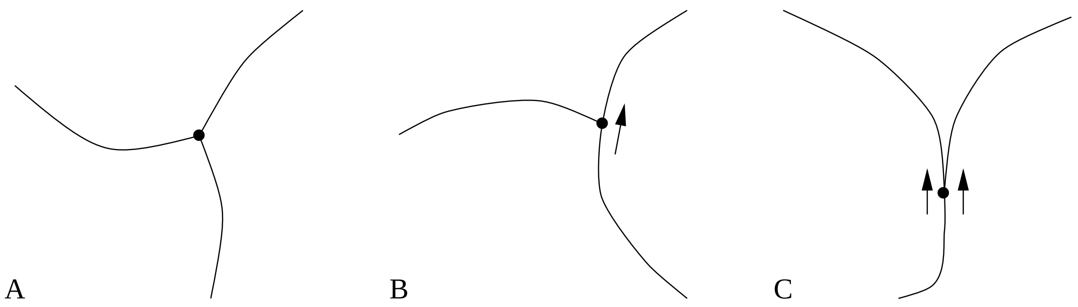


Figure 7.3. Tear configuration. (A) No tangent directions. (B) One tangent direction. (C) More than one tangent direction.

the span of the endpoint. Therefore, splitting a monotonic tear does not change the flexibility of the surface and so is an equivalence operation. From this it is obvious that any configuration choice is equivalent to any other providing they are monotonic sections.

If the tears are equivalent, then why are all choices not equal? The difference, besides aesthetics, is in underlying structure. The computational efficiency of many algorithms, such as those used for display and constraint maintenance on the surface, are dependent on the number of tears in the surface. In the situation of a closed loop, isolation of a tear on one side of the loop means that it can be removed from the opposite surface, thereby reducing the number of tears in the surface. This is the primary advantage of tear reduction as well. Consider the example in Figure 7.4. The current tear configuration (A) has three individual curves which have formed a complete loop. A simple approach to splitting the surface would result in the outer surface having three tears, I and IIa (B) and the inner having one tear IIb (C). Reconfiguration of tears I and II can isolate tear IIb in the interior of the loop (C). Splitting the surface in this configuration results in a single tear for the outer surface and a single tear for the inner surface (D) which cuts the complexity by roughly 25% (from four cached surfaces to three).

Determination of equivalencies is deceivingly complex. Visually, the best solution is usually clear. However, the difficulties lie in the expense of testing each of the possible equivalencies if equivalence is to be determined automatically. It is easy to determine if an adequate number of DOFs exist in the two sets of overlap meshes, but testing each individual configuration is a combinatorial nightmare. It is true that κ is usually small and that certain configurations can be ruled out (those resulting from splits cannot be reduced further), but an exponential algorithm is a recipe for problems. Future work in this area may likely turn up a more efficient method for determining these equivalencies. At this point, user intervention may be the best alternative.

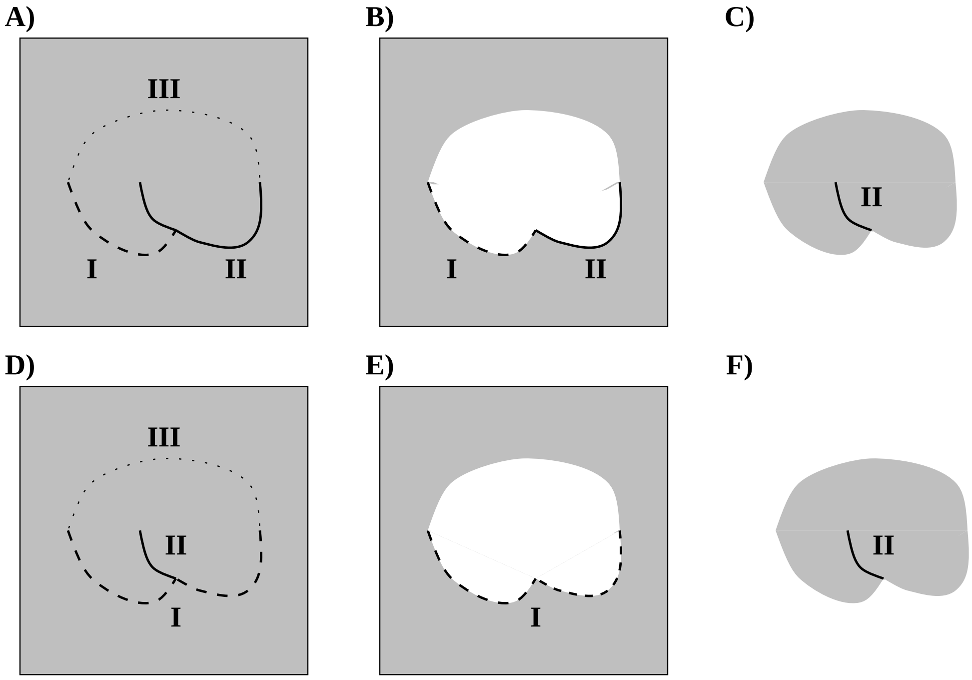


Figure 7.4. Tear reduction. (A) Initial configuration. (B) and (C) Unaltered split. (D) Reconfigured tears. (E) and (F) Reconfigured split.

7.3 Tears, Creases, and Manifolds

Since most boundary representations of solid objects are manifolds, the occurrence of a nonmanifold representation usually means that either

1. it is a simplification of a manifold,
2. it is part of a larger representation,
3. it belongs to a 3D layered or similar representation, or
4. it is an invalid representation.

In the following sections, examples of situations which result in each of these possibilities are discussed.

7.3.1 Tears in Thin Plates

A nonmanifold representation may be a simplification of a manifold. This is particularly true in the case of a thin plate representation where a single 2D surface or set of 2D surfaces in $\mathbb{R}^3$ represents a 3D solid modeled by extrusion. Because the extrusion distance is small, physical effects caused by the material's thickness are assumed to be negligible.

The addition of a tear in this case causes little concern. An extruded torn surface is a manifold as well and poses few special problems. Determination of the extrusion direction for a given location on the surface is only slightly more difficult for a torn B-spline surface than a regular tensor product B-spline surface. During a process such as stamping, a thin plate extrusion direction may need to be modified so that it varies across the surface. The additional complexity may cause difficulty in areas where the tear edges are in close proximity to each other. Extrusion in the direction of the surface normal on each side of the tear could cause the 3D model to self-intersect. In particular, if a crease is introduced into the surface and the extrusion is made into the concave portion of the crease, the surface is guaranteed to self-intersect. However, this self-intersection problem is present for both representations. Fortunately (or unfortunately) this problem is present in other fields, most notably that of computer-aided manufacturing (CAM), where offsetting surfaces produces toolpaths for computer numerically-controlled (CNC) machining and so continues to be a topic of active research. Solutions presented in CAM can be readily adapted to this problem. Since discontinuities are an area of great interest in physical simulation, many solutions to these types of problems exist and the most appropriate action depends entirely on the situation.

Occasionally, the volume of the extruded solid may need to be preserved. Stretching a surface would cause the width of the extrusion to be reduced where the surface is stretched. If the surface is very plastic so that it stretches considerably before "breaking," the width of the surface near the failure point may be nearly zero. On the other hand, if the material is rigid, the width may not change much before the surface breaks. When the surface breaks, a tear or other continuity feature may be introduced. A related difficulty may occur if the material is thick enough to exhibit slightly different behavior between the top boundary of the material and the bottom boundary. In this case, properly registering the continuity features between the top boundary and the bottom boundary or introducing some more complex tear geometry will result in acceptable solutions. To register curves in this case and others to follow, a matching parameterization is determined. For parametric curves, this defaults to a reparameterization of one of the curves. If the tears are not represented by parametric curves, then the registration is more difficult. It may be appropriate to convert the tears to parametric curves in order to facilitate registration.

Figure 7.5 shows a thin plate whose top boundary is torn to a greater extent than its bottom boundary. The solution in this case is to create two tears on the top boundary surface and one tear on the bottom boundary surface. The tear nearest the edge in the top surface is registered with the bottom tear. The interior tear in the top surface contains a custom ravine one of whose endpoints matches the interior endpoint of the tear in the bottom surface.

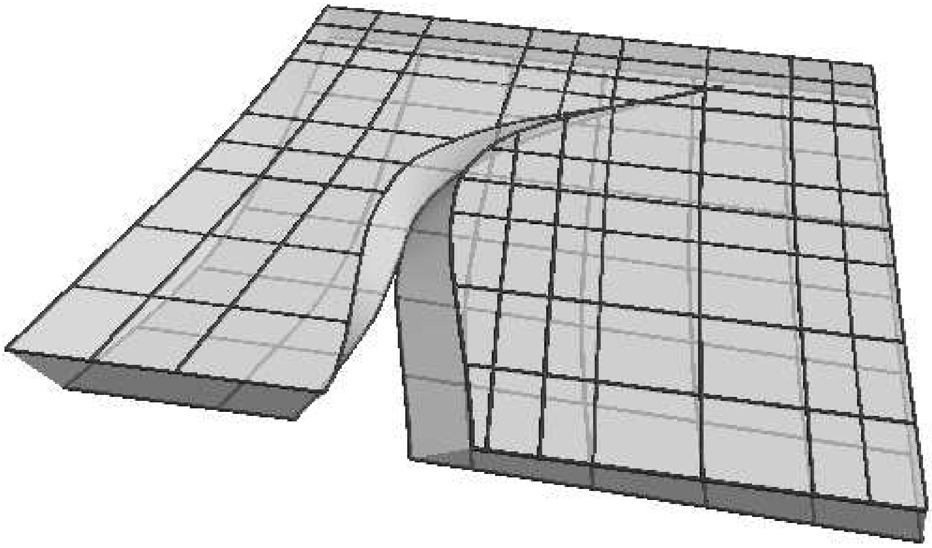


Figure 7.5. Custom thin plate with mismatched tears.

7.3.2 Multiple Surfaces and Tears

Many sculptured models are composed of multiple surfaces, "stitched" together into a boundary representation which complete encloses the 3D space occupied by the object. Often during a simulation or other modeling process, individual surfaces are singled out and modified by themselves because using the entire model would be cumbersome and computationally prohibitive. It is possible that within this context, a single surface may acquire continuity features as a result of the modeling process. Unfortunately, these features usually cause the model to become nonmanifold, that is, no longer encompassing a volume completely. Given that this situation may arise, the following techniques may be useful to the designer.

1. Filling in the cracks.

   A tear creates a hole or separation in the surface that is technically not part of the boundary. In the larger context of a manifold boundary representation, this is a problem. Unless a crease is introduced to reattach the edges of the tear, the edges of the tear must be joined with some other part of the model. A very natural response is to patch the hole with another new surface or surfaces. There are many options in this case, but two possible operations introduced in Chapter 8, the shear (or ruled) fill and the ravine fill are good candidates for this task. Briefly, the shear fill creates a simple ruled surface between the edges, and the ravine fill creates two ruled surfaces between an auxilary curve and each edge. A user-defined curve can be provided for the auxilary curve, and this curve can be used to maintain consistency across boundaries (see Boundary restructuring, below).

2. Boundary restructuring.

   Since continuity features may reach the edges of the parametric domain of a single surface, boundaries of adjacent surfaces may need to be restructured to accommodate the new topology of the torn surface. The new surface's edges may need to be reregistered with the older surfaces' edges. This may not be too difficult, but it is usually critical to maintaining a valid boundary representation.

3. Constraining the tear domain.

   "An ounce of prevention is worth a pound of cure." Placing additional constraints at the boundaries of a simulated surface could prevent undesired behavior from occurring in the first place. For instance, if a discontinuity is approaching a boundary, constraining the boundary to remain adjacent to its neighboring surface can prevent unwanted restructuring of the model's boundary adjacencies, even if the tear reaches the boundary.

4. Maintaining consistency across boundaries.

   Since most continuity features which arise during simulation arise at boundaries, constraining the domain of the tear may not be physically realistic. A more plausible scenario has the continuity feature originating at an edge where two surfaces are adjacent and propagating in both directions from the failure point. This could be considered a single continuity feature and may needed to be treated as such. There are several approaches to this problem. First, the two tears can be propagated independently of each other. The surfaces used to fill the cracks can be adjusted so that they meet at a common boundary since they already meet where the two original surfaces are adjacent. If the surfaces are filled with a linear shearing operation, the filling surfaces are adjacent by definition. If the surfaces are filled with the ravine operation, a custom curve with an adjacency point can be used. In this case, the first section of the curve is used by one surface, and the latter section of the curve is used by the second surface.

A second option would be to maintain a higher-level tear which is kept at the object representation level. In most boundary representations, surfaces are gathered into a single data structure (sometimes called a *shell*) which is a full representation of the model. The tears exist at the surface level and have no interaction with other surfaces except when the shell requires adjacency. Because the overall data structure would need to be modified in this case, the first solution is recommended.

7.3.3 Higher-Dimensional Complex Models

Solid models which have a boundary representation lack the internal structure which may be necessary for some physical simulations. Typically, this occurs when the model becomes large or has internal structure that is not evident from the exterior. A good example is a model of the human body. In this case, the skin does not reflect the interior structure of the body. Physical simulations of deformation and, more recently, electrochemical interactions require a representation of the interior of the body being studied. Traditional finite element meshes for solid models contain interior nodes which, in turn, reflect the structure of the interior of the object. Boundary representations of these models contain multiple layers embedded within each other, each representing the boundary between two distinct solids. It is understood that these interior boundaries represent both the exterior of one region and the interior boundary of another. In this manner, the manifold behavior of the real world is simplified by a nonmanifold representation.

This technique, called *3D layering*, can be interpreted in several ways. The first interpretation given above, considers each enclosed (or disjoint) region to be a manifold described by its enclosing boundaries. Properties of the interior are usually simplified to be uniform or some combination of the boundary properties. Another interpretation is derived from the thin plate concept extended to composite materials. In this scheme, boundaries of thin volumes are *composited* on each other, separated by some extrusion distance. By arbitrary designation, the uppermost boundaries reflect their characteristics through the extruded volume to the next boundary surface. Physical simulations of such structures rely on the thin plate characteristics of each level while modeling the interaction between the surfaces as demanded by the characteristics of the adhesive. In this case, the interior structure of the model is greatly simplified while providing the rich interactions of the surfaces.

The difficulties which arise from the introduction of discontinuities are similar in each case. To maintain a valid layered structure, the boundaries must completely enclose a region. Although the solutions in Section 7.3.2 apply in these cases, the physically realistic solution may not be clear. For example, a rip in a layer of composite material may be interpreted in different ways. It could represent a void in the interior of the material which would need its own boundary description. It could also represent a partial separation of the layer, known as a *pocket*, which would best be represented by filling in the region, creating a new boundary between the two adjacent composite layers. In the medical field where interiors are fluid, it could represent a breech of a boundary between two interiors allowing the characteristics of each volume and the boundary between them to change.

A more complex example can be seen in geology. In studying layered rock formations, scientists have noted that certain layers are more fluid than others and exhibit different physical characteristics. In a concept called *drape folding*, softer layers are draped over distinct discontinuities which are present in lower, more rigid layers. As the bedrock shifts over a slip zone, a sharp discontinuity appears, but the more fluid layers of rock above it are bent and crushed, ultimately filling the void which would ordinarily be left behind. In this case, the boundaries which surround the bedrock are torn and the tears are extruded to the boundary below, effectively partitioning the volume between the boundaries. The upper-level surfaces are not torn, yet are registered appropriately with the surfaces below them. The volume between these surfaces is fluid and does not cause an automatic propagation of the discontinuity that a rigid volume would. The fluid layer is then composed of the draped upper surface and the split lower surface with a portion of its boundary that is exposed along the extrusion. Care must be taken in this case to make the extruded surfaces on each side of the separation to be coincident.

Other problems may also arise when the layers physically slip over each other. The initial surface may need to be reregistered. Voids may occur between layers, requiring the addition of another surface. These problems are made more difficult by the presence of discontinuities, but the solutions in most cases are extensions of the above suggestions.

It is clear that modeling with discontinuities is by no means easy or straightforward given the possible difficulties that may arise. In the next chapter, several modeling operators are presented which provide basic methods for creating and modeling with torn B-spline surfaces.

CHAPTER 8

OPERATIONS

Modeling operators provide high-level handles on complex shape construction, making it relatively easy to create models without necessarily knowing a lot about the underlying representation. Some modeling operators are presented below which are designed to ease the construction of models with discontinuities. These operators are not only applicable in initial design but are useful in simulation as well. They will be discussed in order of increasing complexity. Most operators apply to single surfaces; their generalization to complex models will be discussed as necessary. The operators are described in the context of a modeling language called Shape Construction Language (SCL) developed by the Alpha1 research group at the University of Utah[1].

8.1 Cut Operator

The introduction of a tear into a model, by means of a *Cut* operator, is the simplest operation involving discontinuities. The operands are a single surface and a curve embedded in the surface's parametric space. The resulting model is a torn surface (see Figure 8.1).

```
tornsrf = cut( srf, crv );
```

The basic functionality of this operator is provided by the torn surface data structure itself. The addition of a single tear to a surface is a fairly simple task. However, there is some preprocessing which makes construction of a valid torn surface easier as the surfaces become more complex. First, the new tear curve, `crv`, is intersected with all existing tears in `srf`. Since it is not legal to cross tears, the curve is split at the intersection points and each new section becomes a distinct tear.

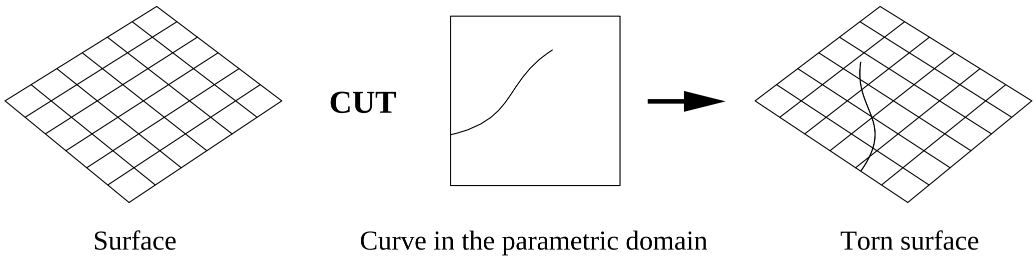


Figure 8.1. The *Cut* operator.

Next, each of these new tears is tested for monotonicity given the primary direction and is split if necessary. In this stage, information about the coincident end points is saved on the tear to ease the μ and η function constructions in later stages. Finally, the tears are added to the surface and determination of the supporting structures proceeds. The resulting surface can then be further modified by another shape manipulation technique.

This operator is extensible to shells as well in the following form:

```
tornshell = cut( shell, srf, crv );
```

where `srf` is a pointer to the surface within the shell to which the tear is added. Surface adjacency information stored in the shell needs to be updated after the surface is torn only if the trimming loops for the cached surfaces are used for other techniques such as boolean operations with other models or elimination of black holes in a shaded rendering. In most cases this is not necessary since the actual parametric boundary of the surface has not changed.

The *Cut* operator only modifies the structure of the surface or shell; it does not actually modify its shape. However, additional shape operators will clearly show the effects of the structure change.

8.2 Shear Operator

A slight modification of the *Cut* operator, the *Shear* operator will create a simple ruled surface between the two edges of a given tear in a surface (see Figure 8.2).

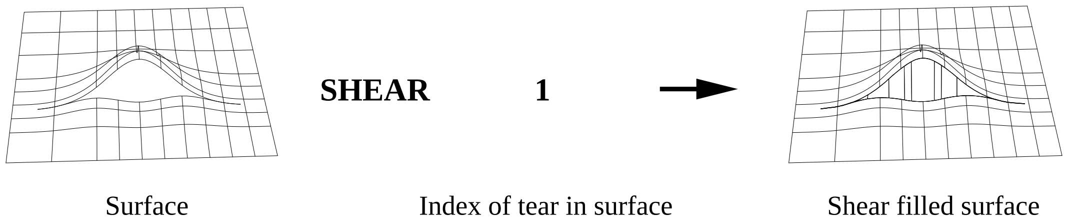


Figure 8.2. The *Shear* operator.

If the torn surface represents topographical information, the region containing a "shear" drop-off between two elevations can be made manifold with this operator.

```
shearsrf = shearfill( srf, tearidx );
```

Tears can be specified by either a pointer to the actual tear, or the index of the tear in the surface. The resulting surface contains a list of *fill surfaces*, or surfaces which fill the gap between two discontinuous regions, associated with each tear as appropriate. Both the *Cut* operator and the *Ravine* operator following use this concept of fill surfaces. The fill surfaces created for the *Shear* operator are ruled surfaces between the two edges of the tear. Since a particular side of a tear may be partitioned to fall into different parametric region, more than one ruled surface may result. If one side of the tear is discontinuous (by joining with another tear) the ruled surfaces may not be connected over the entire length of the curve. A different solution may be in order since this situation is not well defined with ruled surfaces. The determination of the actual curves used for the sides of the ruled surface is probably the most difficult task. The regions of the surface on opposite sides of the tear may have drastically different behaviors. In the current implementation, the following procedure is used to determine the curves.

1. The curves on each side of the tear are evaluated within the surface to obtain a piecewise linear curve to within some tolerance of the surface. This results in curves with a very large number of control points.

2. To facilitate further approximation, the degree of the curves are raised to support smoothness.

3. The curves are then reduced to within yet another tolerance using the Lyche/Mörken knot removal technique[54]. (Tangent discontinuities in the parametric tear curves may be lost if both the degree is raised and the curves are smoothed.)

4. The knot vectors of the curves are registered to the same interval, [0,1] and are merged into a single knot vector.

5. The merged knot vector is used to make the two curves compatible by refining them to the same level.

Since portions of the tear may be part of the boundary in different parametric regions, the curve is partitioned according to distinct pairs of parametric regions on opposite sides. For example, if a tear's left side bordered three different regions, [0, 0.4], [0.4, 0.6] and [0.6, 1], and the right side of the tear bordered two different parametric regions, [0, 0.5] and [0.5, 1], then a ruled surface would be made between the tears for each of the distinct pairs of regions, [0, 0.4], [0.4, 0.5], [0.5, 0.6] and [0.6, 1] (see illustration in Figure 8.3).

The disadvantages of this procedure are evident. First, two stages of approximation may cause the difference between the edges of the ruled surfaces and the actual tear edges to exceed acceptable tolerances. Of course, this is highly dependent on the actual shape of the surface and the complexity of the curve. The parameters that work in one case may not work in another, even if the parametric curve remains the same. Second, the knot removal may not result in a curve with a similar parameterization scheme. This affects the registration phase, not in the actual registration, which is straightforward, but in the resulting pairing of the curves. A particular parametric location on an edge may no longer be paired with the same parametric location on the other side of the curve. A better approach for this procedure would be to develop a technique for extracting a higher-order

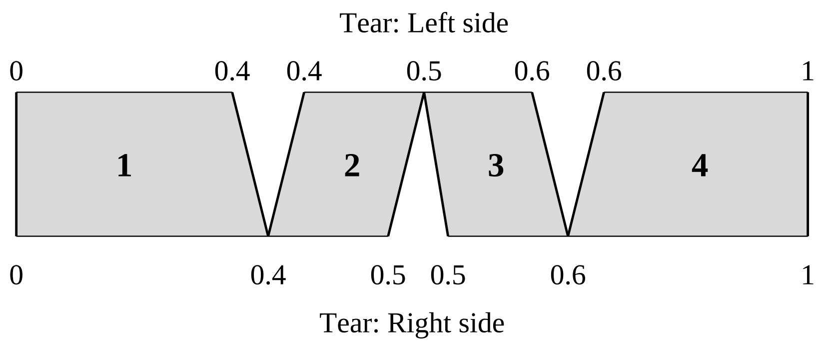


Figure 8.3. Individual ruled surfaces produced by *Shear* operator in a surface with multiple tears.

approximation to the curve embedded in the surface while maintaining roughly the same parameterization as the original tear. This way only a single approximation is used, and the registration problem is minimized. Fortunately, in actual experiments with this operator, manipulation of the tolerances has provided adequate results for such tasks as rendering the surface.

8.3 Ravine Operator

Another variation of the *Cut* operator, the *Ravine* operator creates a straight line between the endpoints of a tear and creates two ruled surfaces one from each side of the tear extending to this *base curve* (see Figure 8.4). This situation is particularly useful with models which contain multiple surfaces. The *Ravine* operator can be extended to support arbitrary curves of connection between any two end points, even if they lie in separate surfaces.

```
ravinesrf = ravinefill( srf, tearidx );
ravinesrf2 = ravinefillcrv( srf, tearidx, basecrv );
```

The *Ravine* operator differs from the *Cut* operator in that two ruled surfaces are created for each section instead of one. The third curve which forms the boundary of both sets of ruled surfaces is either computed by creating a straight line between the tear's endpoints or is provided by means of a custom base curve. The technique

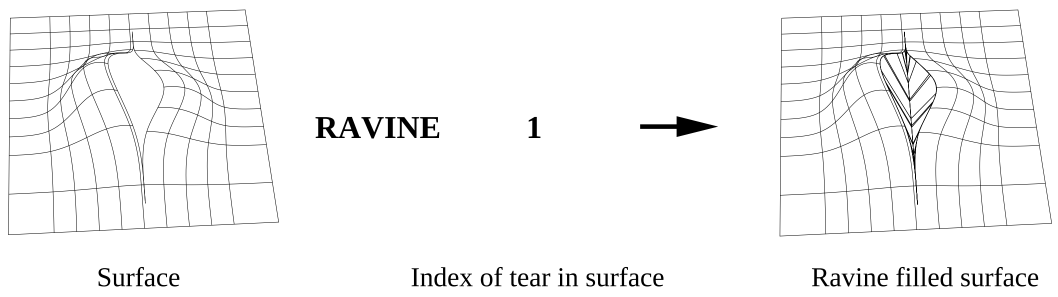


Figure 8.4. The *Ravine* operator.

used in the straight *Ravine* operator to determine the embedded curves is similar to that of the *Shear* operator. However, the registration of the two sides of the surfaces is replaced with registration to a linear segment or single curve (see Figure 8.5). The number of ruled surfaces is determined by the number of parametric regions on each side of the tear, not distinct pairs of regions as is the case for the *Shear* operator.

If an arbitrary base curve is provided, each side of the tear needs to be registered with this base curve. If the tear crosses multiple parametric regions, the base curve needs to be partitioned equivalently. It is not necessary that the base curve have end points that match the end points of the curve. However, this may significantly modify the boundary description of the surface (although not any more than a

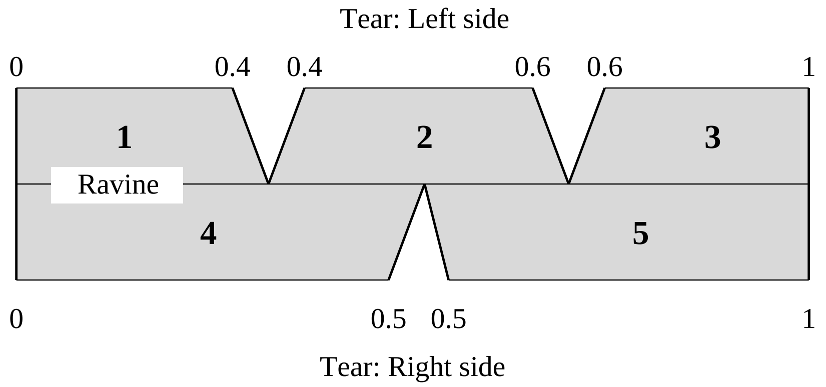


Figure 8.5. Individual ruled surfaces produced by *Ravine* operator in a surface with multiple tears.

nonfilled tear would) if the tear intersects the boundary. Therefore, some care is required when using this operator in the context of a more complex model.

8.4 Thin Plate Constructions

The operators presented below all construct thin plate solid representations from the minimal single surface representations used for thin plate simulation. These constructions accept one or two possibly torn surfaces and result in a manifold solid.

For the purposes of these operators, a thin plate representation requires that the top and bottom surfaces be represented equivalently and the sides are simple, usually ruled, surfaces.

The third thin plate operator, *Custom Thin Plate*, takes two surfaces, a top and a bottom, each of which can have its own unique topology. Ruled surfaces are attached between the two surfaces along all boundaries.

8.4.1 Fixed Width Plate

The *Fixed Width Plate* assumes that the top and bottom surfaces are equivalently torn and the distance between the top and bottom surfaces is uniform (up to representation capabilities). In this constructor, a single surface which represents a thin plate is extruded in a given direction for a given distance (see Figure 8.6). An alternate method called the *Normal Thin Plate* determines the extrusion direction based on the normal as it varies across the surface. This variation, however, may result in an approximation only. The format for such constructors would be

```
thinplate1 = FixedThinPlate( tornsrf, direction, distance );
thinplate2 = NormThinPlate( tornsrf, distance, tolerance );
```

where `tornsrf` is the shape containing surface, `direction` is a 3D vector, and `distance` is the extrusion distance. The resulting `thinplate1` is a collection of surfaces in a shell.

For the *Normal Thin Plate* operator, no direction is given since the normal of the surface is used. However, an optional tolerance is provided since the offset surface is

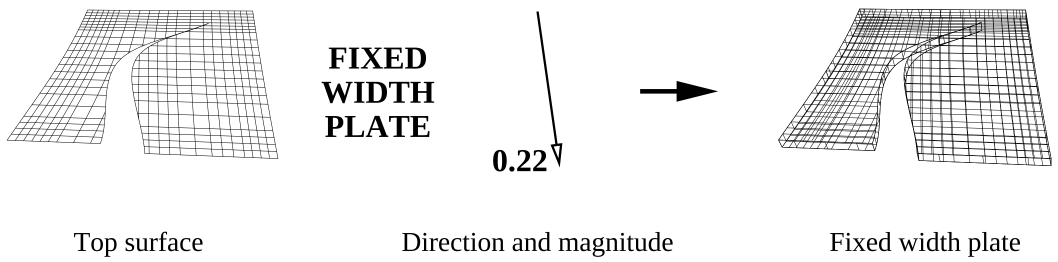


Figure 8.6. The *Fixed Width Plate* operator.

an approximation. In addition, some surface shapes may result in self-intersections, discontinuities and other anomalous regions. Since these operators essentially construct offsets of the given surface, they are subject to all the difficulties that offsetting a regular B-spline surface may have. Once again, the returned object is a shell (or collection of B-spline surfaces with adjacency information).

8.4.2 Variable Width Plate

The *Variable Width Plate* operator also assumes that the top and bottom are cut equivalently, and for this case, the distance between the two surfaces (or surface thickness) may be computed as inversely proportional to the amount that the surface is stretched at that point (see Figure 8.7). The concept, in this case, is that a tear must occur where the surface thickness is near 0. The *Variable Width Plate* operator takes a sculptured surface, `tornsrf`, and a structure which contains the vector valued width information, `widthsrf`. The easiest implementation of `widthsrf` is by another surface which represents either the bottom surface directly, which is equivalent to the *Custom Thin Plate* operator described in the next section, or a surface which represents the difference between the top and bottom surfaces (an example constructor is shown below). This structure could be expanded to contain enough information in order to compute the width of the plate on the fly. Since the top and bottom are equivalently torn, no fill surfaces are necessary between opposite sides of a tear in the same surfaces to maintain the topology of the solid model for this operator.

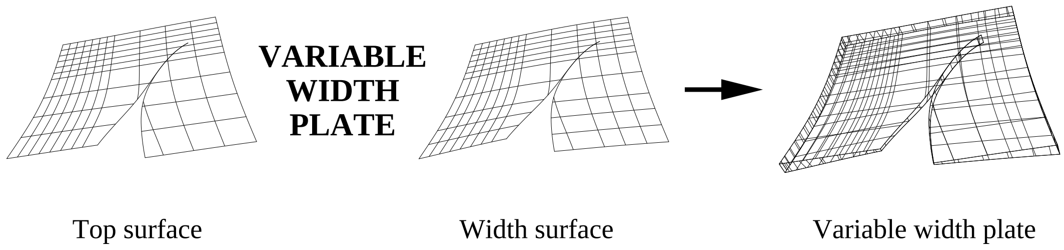


Figure 8.7. The *Variable Width Plate* operator.

```
thinplate3 = VarThinPlate( tornsrf, widthsrf );
```

The primary difficulty with the *Variable Width Plate* is deciding what the width of the plate should be at any given point. As a more complex variation of the *Fixed Width Plate*, a reference surface, which is a representation of the amount the surface has stretched at each parametric location, perhaps in terms of the percentage width of the surface, may be used. Then, when the plate is constructed and new stretch values are computed, the surface can be properly offset by adding these surfaces together.

The width or other stretch values can be determined by any of several different methods. First, if the physical simulation which deforms the surface has a concept of how much a surface is stretched at a given point, values can be obtained from the simulation. For example, when defining linear elastic elements between nodes on the surface, the elastic elements maintain a rest state (or an original length). When compared to the actual length, the ratio of the lengths provides a measure of how much the element has expanded or contracted. This in turn determines the forces transferred between nodes. Related to this is the method of using the linear stretch terms, which are a product the first partials in the primary directions, that are used in most linear optimization techniques for tensor product B-splines[90]. These terms do not have to be part of the simulation. However, if they are, they can be used without incurring any additional computational cost. Otherwise, computing the stretch terms can add to the cost of computation, especially if the thin plate construction is dynamic.

If a sampled set of widths for the offset surface is given, the offset surface can be determined by surface reconstruction techniques. Unfortunately, unless special care is taken to keep the widths along tears at 0, additional fill surfaces will likely be needed between corresponding tear edges in the two surfaces. If an offset width surface is provided, the offset surface can be obtained by adding the two surfaces. If the surfaces have compatible parameterizations, this is straightforward. Otherwise, an additional reparametrization step is needed (which, unfortunately, may result in an approximation).

8.4.3 Custom Thin Plate

Usually a thin plate is represented by a single surface for simulation and other analytical methods. However, occasionally, a second, or bottom, surface may be available. The *Custom Thin Plate* constructor allows the most flexibility of the thin plate constructors, by using this second surface that represents the bottom of the plate (see Figure 8.8).

```
thinplate4 = CustomThinPlate( tornsrftop, tornsrfbot );
```

This added information makes this operator simpler than the others and prevents it from being susceptible to the same types of approximation problems. One particular situation in which this operator may be useful is when multiple surfaces are being used as part of a 3D layered representation. The individual layers can easily be represented by a custom thin plate generated from the surfaces within the structure.

8.5 Layering Constructor

The *Layering Constructor* is designed to support the construction of models by 3D layering. The input to this constructor is a set of surfaces, top to bottom, and information about how the surfaces are put together, called *bindings*.

```
layeredVolume = threeDLayer( tsrf1, bindings1, tsrf2,
```

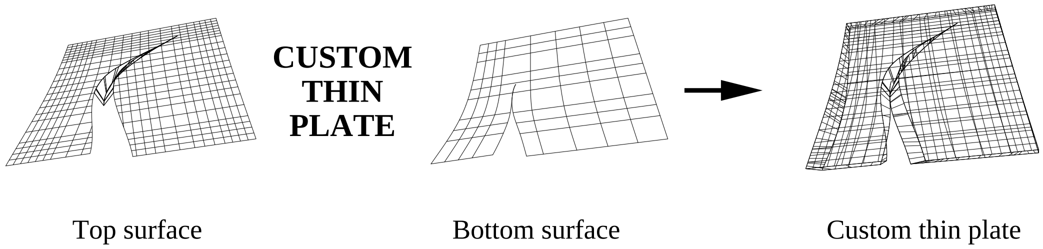


Figure 8.8. The *Custom Thin Plate* operator.

```
                    bindings2, bottomsrf );
```

The bindings are list of structures which contain information about how each tear in the first surface is bound to another tear in the second surface. The binding structure contains an index for a tear in the first surface, an index for the tear in the second surface, and a bind value that describes how use the indices. The binding values can be either *match* or *thru*. A *match* binding means that the tears indicated by the indices are matched. In this case, ruled surfaces are created between the edges of the tears, left side to left side and right side to right side (see Figure 8.9). The tears are assumed to be oriented correctly. A *thru* binding assumes only one of the indices is nonzero, and it means that there is no matching tear in the other surface. In this case, the parametric tear curve from the indicated surface is evaluated in the other surface and the new curve is used to create the bottom of the ravine. Then the *Ravine* operator is used to create ruled surfaces for the indicated tear (see Figure 8.10). It is not necessary that all tears in all surfaces be bound, particularly if they have already been filled.

With many match bindings, a set of ruled surfaces attached end to end would be created. In this case, a single higher-order surface could be constructed from the collection of ruled surface boundary curves on each side of the tears. The difficulty with this method would be the fact that these surfaces would need to be constructed by interpolating the boundary curves which could easily result in approximation or oscillation problems.

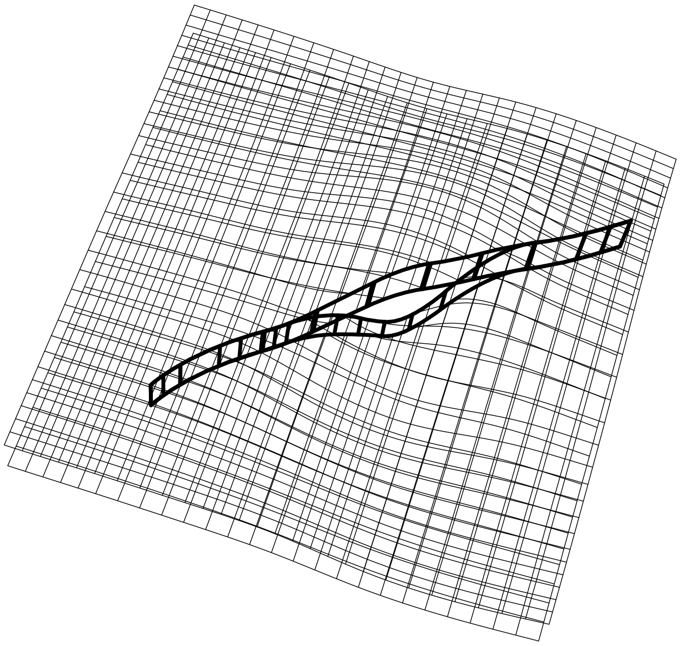


Figure 8.9. Example of a *match* binding, {1, match, 1}.

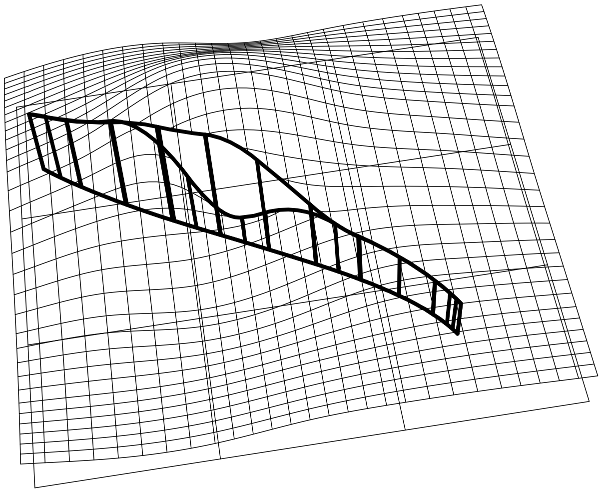


Figure 8.10. Example of a *thru* binding, {1, thru, 0}.

8.6 Fold Operator

The Fold Operator constructs a crease within a surface and folds or bends the surface at the crease.

```
foldedSrf = foldSrf( srf, centerpt,
                     creasedir, extent, bendAngle );
foldedSrf2 = foldSrf( srf, centerpt, crease, bendAngle,
                      {anchorpts},{bendpts} );
```

The operator takes a surface to fold, `srf`, a description of where to fold the surface, a measure of how big the crease is `extent`, and a measure of how much to fold the surface, `bendAngle`. Describing where to fold the surface requires both a center position or focal point, `centerpt`, and a vector, `creasedir`, which indicate the direction of the crease. The bend measure is an angular specification where 0 indicates no bending and 180 indicates a complete fold. Positive bend angles refer to a fold produced in a clockwise direction around the tangent direction of the tear. Negative bend values indicate a bend in the counter-clockwise direction. The size of the crease is given in terms of a percentage of the surface. One hundred percent indicates a crease which extends from edge to edge, and 50% indicates a crease whose longest reach is half of the distance to the edge farthest from the center position. Together, `centerpt`, `creasedir` and `extent` describe the tear which is added to the surface.

The constraint which represents the bend is derived from the newly constructed tear, and is prepared for use in a minimization process. In addition, *anchor* points are sampled from the surface to the left of the tear (relative to the direction of the tear curve in parametric space) to provide a way for that surface to remain stationary. Five anchor points are sampled as follows:

1. The vector orthogonal to the tangent direction (the parametric normal direction) at the center position in the stationary surface is constructed and its point of intersection with the boundary is determined. The first three points

are selected by sampling points at 5% (*close*), 50% (*mid*), and 95% (*far*) of the distance along this normal vector (see Figure 8.11(A)).

2. Based on a percentage parameter, *back*, which behaves like the parameters, *close*, *mid*, and *far*, above, line segments are drawn which connect the *back* point on normal vector to the endpoints of the tear. The points at the percentage of these segments indicated by the *side* apremeter are selected as the fourth and fifth points (see Figure 8.11(B)).

3. Finally a set of bend points is constructed in a process similar to the anchor points only these points are evaluated with the normal in the opposite direction.

The bend points are then rotated by the bend angle counter-clockwise around the surface's tangent direction determined by the focal point and the crease direction. These sets of points are submitted to a minimization procedure along with the crease constraints and a folded surface is returned. (Minimization of torn surfaces is discussed in Section 8.7.) Alternately, a custom crease can be specified, along with a point on the curve (analogous to the center position) at which to perform

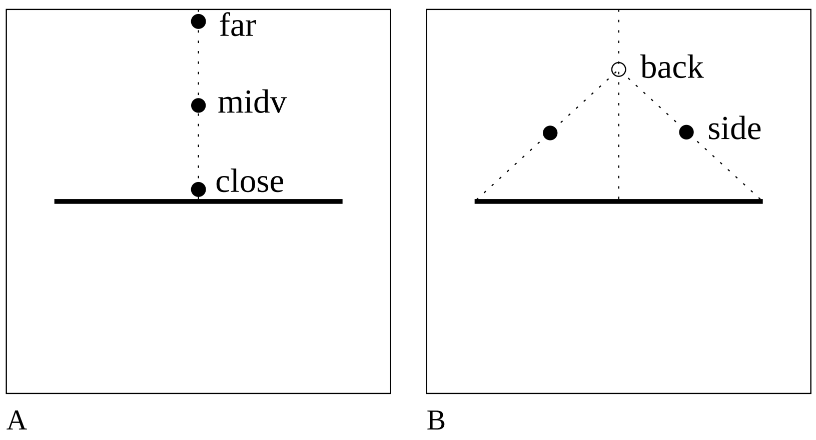


Figure 8.11. Diagram of parameters used to compute anchor points.

the fold. Although this allows nonlinear creases, performing a fold around a custom curve is not generally solvable if the crease is to be maintained to a given tolerance. In addition, the various automatically constructed anchor points which are used in the minimization can actually be any points which embody the type of shape construction desired. Allowing modification of the parameters, $close$, mid, far, $back$, and $side$, can provide fine tuning of the shape of the surface.

Clearly this procedure can be prone to error since the anchor and bend points are not guaranteed to exhibit any kind of properties other than that they are in the surface (unless the surface is trimmed), they are not the same, and they are relatively easy to compute in most cases (trimmed B-spline surfaces can be more difficult). They also do not adequately represent the shape of the surface in most cases. Ideally, the anchor and bend points would be customizable in any version of the operator. In some cases, the tear which is constructed by the given parameters may not provide enough degrees of freedom to be able to adequately represent the folded surface. This may result in a torn crease or an undesirable surface shape. Another improvement would be to use surface normal constraints near the center position rather than transformed bend points. However, care would need to be taken to maintain the surface shape after the fold. Maintaining the surface shape after the fold is, in itself, a subject for further study.

Figures 8.12-8.15 represent the various stages of a paper airplane being folded by using the fold operator. The original shape of the surface was maintained by successively adding position constraints to a system of both position and curve adjacency constraints subject to a curvature minimization solution using the Lagrange Multiplier Method discussed in Section 8.7.3 below.

8.7 Minimization

Constrained minimization techniques are standard in modeling processes where fair surfaces are desired. Most commonly, the equations for finite elements are minimized subject to boundary constraints to obtain a solution for a physical system. Since these techniques can be readily applied to B-splines using linear

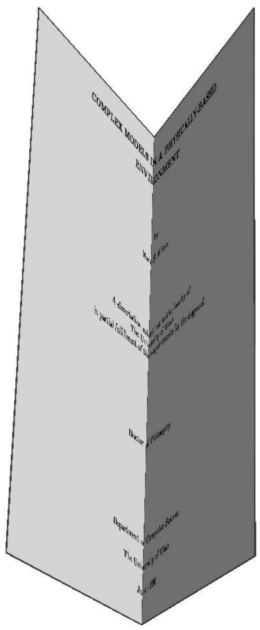


Figure 8.12. Paper airplane: Primary fold.

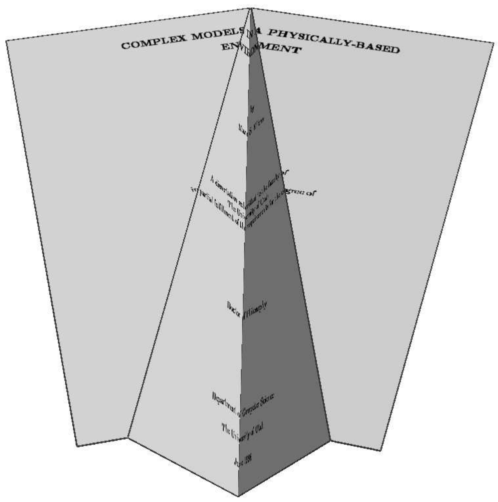


Figure 8.13. Paper airplane: Initial wing folds.

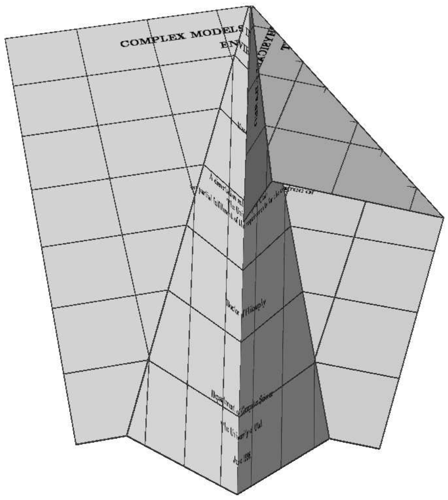


Figure 8.14. Paper airplane: Second wing folds.

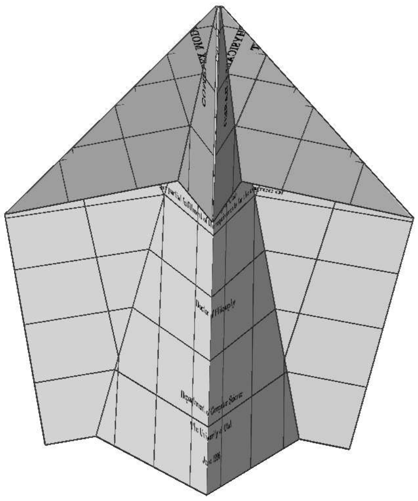


Figure 8.15. Paper airplane: Final wing folds.

systems of equations, they can also be applied to torn B-spline surfaces and shells. The techniques outlined below are a summary of those presented by Welch and Witkin[90] which were adapted for use with torn B-splines in this thesis.

8.7.1 Objective Functions

In order to obtain a fair surface shape, one possible objective function measures the stretching and bending of the surface in terms of differential tangents and curvature:

$$Q(w) = \int_w \|G\|_\alpha^2 + \|B\|_\beta^2 . \tag{8.1}$$

In this equation, G and B are the first and second fundamental forms of the surface, w, which represent the parameterization invariant differential geometry of the surface[30], and α and β are the weights for the matrix norms of the fundamental forms. These two terms control the resistance to stretching and bending, respectively.

When this equation is linearized to obtain the *thin plate under tension* model[77, 20], the following equation results:

$$Q(w) = \int_w \sum_{i,j=1}^{2} \alpha_{ij} D_i w D_j w + \beta_{ij} (D_i D_j w)^2, \tag{8.2}$$

where $D_i w$ is the partial derivative of the surface w at the ith parameter.

Welch and Witkin[90] use a composite B-spline surface representation whose objective function formulation is similar to that of torn B-splines. The torn B-spline surface is composed of parametric regions, each having the same level of refinement, but with a restricted set of control points. Then

$$Q(w) = \int_w \sum_{i,j=1}^{2} \begin{pmatrix} \alpha_{ij} p^T D_i b p^T D_j b \\ + \\ \beta_{ij} (p^T D_i D_j b)^2 \end{pmatrix}, \tag{8.3}$$

where b is product of the basis functions for the surface and p is the composite control point vector which is a concatenation of all control points, both in the original mesh, and in the overlap meshes of the torn B-spline surface.

Minimization requires the determination of the Hessian matrix, which is a matrix of sums of products of first and second partials that provide the transformation of motion and gradients in one space to motion and gradients in another space. To isolate the Hessian matrix from Equation 8.3, p is moved outside the integration resulting in the following equation:

$$\begin{aligned} Q(w) &= p^T \int_w \sum_{i,j=1}^{2} \left(\begin{array}{c} \alpha_{ij} D_i b \otimes D_j b \\ + \\ \beta_{ij}(D_i D_j b \otimes D_i D_j b) \end{array} \right) p & (8.4) \\ &= p^T H p & (8.5) \end{aligned}$$

where $\otimes$ is the outer product defined as follows: given vectors $\mathbf{A} = [\alpha_i]$ and $\mathbf{B} = [\beta_j]$, $\mathbf{A} \otimes \mathbf{B} = \mathbf{C}$ where $\mathbf{C}_{ij} = \alpha_i \cdot \beta_j$.

8.7.2 Linear Constrained Optimization

Not suprisingly, the objective function described in the previous section has a minimum when all degrees of freedom are 0. To obtain a more reasonable solution, constraints must be added to the system. Since B-splines lend themselves very nicely to linear constraint formulation[34, 21, and others], they can be introduced into the linear system with very little effort. Each additional linear constraint becomes a row in the matrix which represents the system of equations to be solved.

There are two classes of linear constraints which are used with torn B-splines as implemented for this thesis. The first are simple geometric constraints, such as points, tangents and twists. For each of these constraints, one row is added to the constraint matrix. The other class of constraints, known as *transfinite* constraints, deal with constraints over higher-dimensional geometric objects such as curves or surface patches. In particular, a curve constraint, such as the one suggested for implementing creases, requires integration of the constraint function over the curve. Curve constraints are generally formulated as constraints of the gradient of the difference between an embedded curve and its desired shape. With this formulation, the number of rows added to the constraint matrix equals the number of control points (or DOFs) in the surface.

Once the linear constraints are determined, there are standard ways of finding the minimum of the system. Two of the most common are the method of Lagrange multipliers and the penalty method. Each of these methods has its own advantages and disadvantages.

The minimization problem can be stated as

$$\begin{gathered} min \left\| \tfrac{1}{2}p^T H p - p^T f \right\|, \\ p \text{ subject to } Ap = b \end{gathered} \tag{8.6}$$

where H is the Hessian of the function to be minimized, f is the gradient optimization term, and $Ap = b$ is the linear constraint system to be solved. The standard approach is to transform this problem into an unconstrained problem by the methods given below such that any solution for $\hat{p}$, when transformed back, would satisfy the constraints, or

$$\underset{\hat{p}}{min} \left\| \frac{1}{2}\hat{p}^T \hat{H} \hat{p} - \hat{p}^T \hat{f} \right\|. \tag{8.7}$$

Since this system is at a minimum when its derivatives are 0, then we can solve

$$\hat{H}\hat{p} = \hat{f}, \tag{8.8}$$

and then transform the resulting $\hat{p}$ back to obtain the solution to the constrained minimization problem.

8.7.3 Lagrange Multiplier Method

The Lagrange multiplier method is one possible method for transforming the constrained system to the unconstrained system. In this method, a new variable, y, is introduced for each linear constraint row in A. It is necessary to minimize $F(p, y)$ with respect to p and y where

$$F(p, y) = \frac{1}{2}p^T H p - p^T f + (Ap - b)^T y. \tag{8.9}$$

The augmented system obtained by differentiating this equation by each of the elements of p and then each of the elements of y is:

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ y \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix}. \tag{8.10}$$

Lagrange multipliers are simple to formulate and provide an exact (or least squares) solution to the constraints while minimizing the Hessian, but this method has several disadvantages. First, an additional variable, y (the Lagrange multiplier), is added for each of the constraint rows, dramatically increasing the number of variables in the system. In addition, the constraint matrix needs to be linearly independent before trying to solve the system. In most cases, a least squares fit to the constraints is desired, which requires $A^T A$ (where A is the original constraint matrix) as the constraint matrix to be included in the system. Typically, these matrix multiplies are computationally expensive. Finally, the augmented system is not positive definite anymore, so care must taken when solving this system.

The Singular Value Decomposition (SVD) method can be used to solve this system of equations since it is designed to deal with problems at singularities. This method was used for all examples in this thesis involving minimization.

8.7.4 Penalty Method

In contrast, the penalty method is significantly more difficult to formulate. The transformed system in this case is

$$min \left\| \begin{pmatrix} \sigma A \\ H \end{pmatrix} p - \begin{pmatrix} \sigma b \\ f \end{pmatrix} \right\|. \tag{8.11}$$

In particular, determining the weights, σ, associated with the particular constraints is somewhat of a black art. In depth knowledge of the relative worth of the constraint functions is an absolute necessity. The constrained minimum is approached by the unconstrained minimum as $\sigma \to \infty$, so allowing for a changing σ during the solution process is essential. If σ is too small, then a large constraint residual must again be subjected to another minimization step. However, this method is inherently more stable since the constraint matrix does not need to be normalized. In addition, since the constraint rows do not need to be independent, incremental updates of the factored matrices (used in some solution techniques) is possible.

CHAPTER 9

APPLICATIONS

The representational power available in the torn tensor product B-spline surface has been discussed and various operators to construct these surfaces have been presented. This chapter describes how the torn B-spline surface can be used for more complex modeling in the application areas discussed at the beginning of this thesis. In addition, various examples in these areas will be presented.

9.1 Surface Reconstruction

The problems encountered when reconstructing models from data are a research area unto themselves. Yet with a few examples it can be seen how torn B-spline surfaces can be used to solve some of the more common problems. It is assumed for these examples that the data are given as three-dimensional (or 3D) points corresponding to a particular parametric location on the surface. Surface reconstruction with other types of data can be addressed with the same approach.

One common difficulty that often arises when reconstructing surfaces is when the data makes an abrupt jump from one elevation to another along a line or curve. These jumps are common when reconstructing models from 3D laser scanner data. Parts of the model that are obscured by other features from a particular direction are primary candidates. Other range data properly registered can be used to fill in the gaps. As with all data in this category, an initial step of determining the nature and structure of the discontinuity is required. Once the discontinuity is discovered, a surface can be constructed from the given data.

To illustrate, consider the sequence of surfaces in Figure 9.1. Figure 9.1(A) is the original surface and the data sampled from that surface. There are 200 points generated by random sampling and 20 evenly placed points around the perimeter to make sure the corners were adequately constrained. For this example, the original surface is order 4, with 6×6 control points in the mesh. Figure 9.1(B) shows the surface reconstructed by interpolation with a normal B-spline surface with 9×9 control points so that the number of DOFs for the interpolated surface is definitely larger than the number of DOFs on the original surface. The number of data points ensures that the system is overconstrained and that a least squares solution is required. The error values, ϵ, given in Figure 9.1, are the sum of the squares of the distances between the data points and their corresponding actual surface positions. It is evident that even though there are significantly more DOFs in the new surface than are in the original surface, the distribution of the DOFs is not effective enough to produce reasonable results. On the other hand, Figure 9.1(C) shows a surface reconstructed with *a priori* knowledge of the discontinuity and application of discontinuity within the interpolated surface. Clearly this new surface is a more reasonable reconstruction.

However, rarely is the discontinuity known before reconstruction. The surface in Figure 9.1(D) has a discontinuity that is an approximation of the original with representative modifications, such as a smaller size and more variation since the discontinuity is likely approximated by measuring gradient differences (with standard edge detection techniques). The full extent of the tear is not likely to be discovered because the data do not vary much near the ends because of the continuity requirements. In addition, the structure of the data may make it difficult to properly register the discontinuity in the parametric domain. This final surface has a larger error, yet still manages to capture the structure of the data significantly better than standard interpolation. The torn B-spline surfaces constructed for interpolation are 6x6 to match the DOFs of the original surface. Even when keeping the number of DOFs approximately the same, the torn B-spline surface's structure is much more desirable.

Another difficulty occurs when the original model has an edge and the data reflects this feature. For this case assume the data are unorganized, that is, no specific parametric locations accompany the points to be interpolated. The most

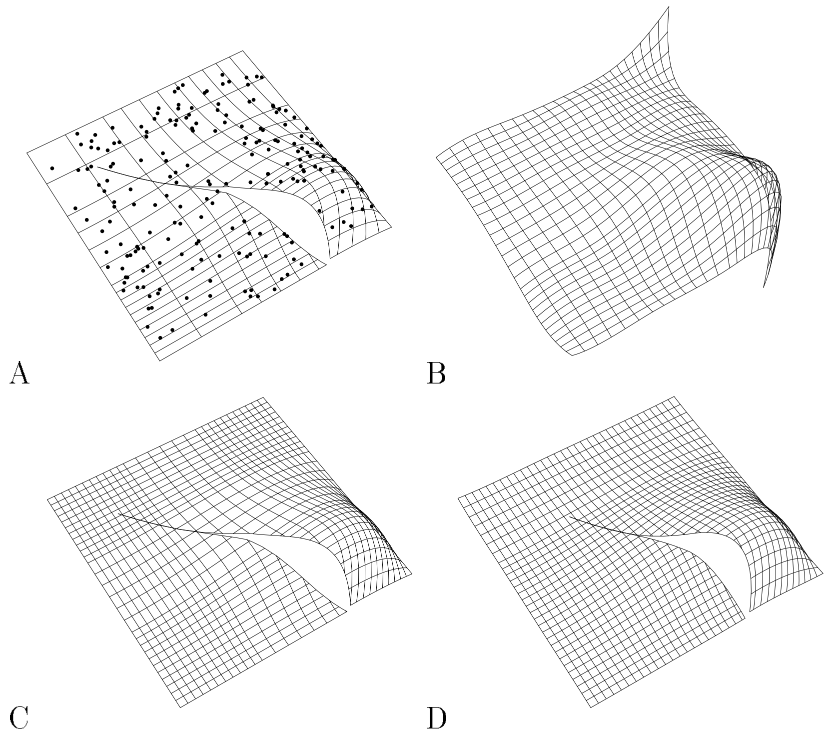


Figure 9.1. Interpolation. (A) Reference surface with data points. (B) Interpolation with smooth surface ($\epsilon = 0.756666$). (C) Interpolation with torn surface same tear($\epsilon = 0.000068$). (D) Interpolation with torn surface, different tear ($\epsilon = 0.056544$).

common method for dealing with this problem is to construct multiple surfaces which meet at the edge. Here the advantages of the torn B-spline surface are less obvious but just as significant. First, not all edges are straight and the curved—nonisoparametric ones can cause problems if not carefully registered. Second, registration of the parametric data with the individual surfaces can be a nontrivial task, even for edges which are straight, if they are not aligned with the isoparametric lines of the data source. If the registration problems can be solved, using multiple surfaces to represent the data is nearly equivalent in complexity to using torn B-spline surfaces. In both cases, additional DOFs and constraints are often used to adequately represent the surface. In the case of torn B-spline surfaces, however, the parameterization of the surfaces near the edge may be more desirable since the parameterizations are guaranteed to match across the edge. Unfortunately, a general crease is not always representable with a single torn surface. In this case multiple surfaces is the best option.

9.2 Stamping

One of the more common issues in the stamping process is the formation of wrinkles, folds and tears, whether they are intentional or not. Such aberrations in the structure usually require a change in representation. With torn B-spline surfaces, then need for changing the representation is eliminated since the discontinuities can be incorporated into the surface. The following examples illustrate the intentional formation of such features.

9.2.1 Lance Punching

In normal punching, a punch with beveled edges is inserted into a sheet of material causing a section of the material to be removed in the shape of the punch. In lance punching, a portion of the punch does not go through the material leaving the material attached along a straight-edged crease[27]. The shape of the cut portion is determined by the shape of the tool. Figure 9.2 shows a single surface which has been lance punched twice by two different punches resulting in differently-shaped tabs. This surface contains a single tear around each of the cut

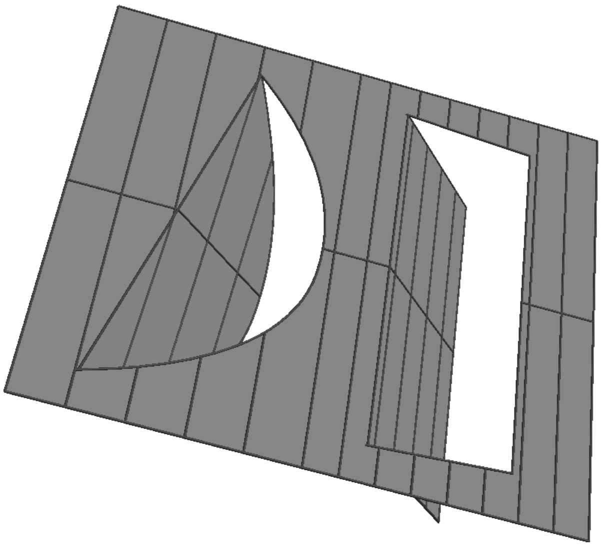


Figure 9.2. Single surface with two lance punches.

portions and a tear representing each of the creases along which a curve adjacency constraint is maintained.

9.2.2 Embossing

Embossing is similar to punching in that a fold or region of high curvature is intentionally created around the perimeter of the punch. Rather than introduce large numbers of DOFs to represent the region of high curvature, a crease is created. Figure 9.3 represents one portion of an embossing that has rounded corners. This embossed surface contains a single tear and a curve adjacency constraint along the entire length of the tear. In a regular tensor product surface, the degree of freedom required to produce similar results would be prohibitive.

9.3 Geology

Drape folding is one of the more interesting geological formations that is not easily represented by ordinary smooth parametric surfaces. This formation is caused by a vertical shift in bedrock position along a fault line. The layers situated on top of the bedrock are often more fluid creating a rock layer which is draped from the higher section to the lower section without actually separating along the fault.

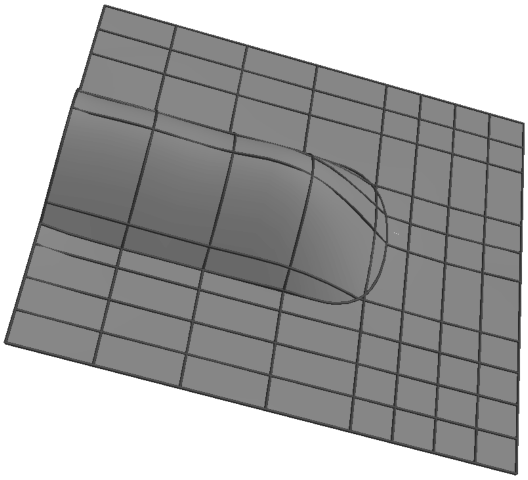


Figure 9.3. Section of an embossed model.

The representation of the boundaries of these layers is easily accomplished with torn tensor product B-spline surfaces combined into the 3D layering representation described in Section 8.5. In ordinary circumstances, the individual layer boundaries would be composites of surfaces and would not easily be integrable into a single comprehensive representation.

Figures 9.4-9.7 illustrate the stratification of these layers as represented by the torn B-spline surfaces. Figure 9.4 shows the top boundary of the geological model with uniform smoothness throughout and a plateau on one corner of the model. Notice the colors of the layers represented in the model which are visible along the edges. This top surface is a regular tensor product B-spline surface.

Figure 9.5 shows the model with the top layer removed. This layer is directly above the bedrock and has a fold in the boundary near the fault line. The fold is represented by a single tear from on edge of the surface to the adjacent edge, with a higher tolerance curve adjacency constraint along the entire edge.

Figure 9.6 shows the bedrock layer. This surface contains a tear identical to that of the surface directly above it, but instead of a constraint, the sheer drop in the bedrock is filled by a ruled surface as discussed in Section 8.2.

Figure 9.7 provides a close-up of the composition of the layers and the effects of

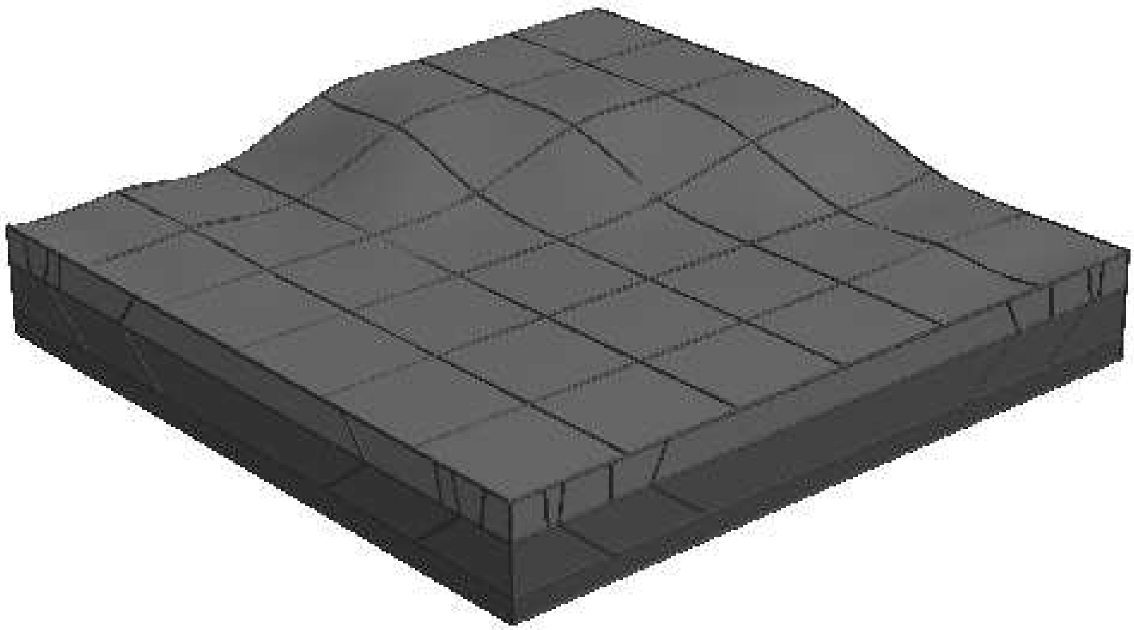


Figure 9.4. Drape fold example, 3D layered model.

the bedrock shift on the upper layers. The edges bounding the 3D layered model are composed of ruled surfaces between the layers as discussed in Section 8.5. In the construction of this example, no interlayer binding was used. The individual layers interacted only at the edges since the bedrock layer's tear is shear filled and the draped layer's tear is a crease, neither of which requires any additional extension.

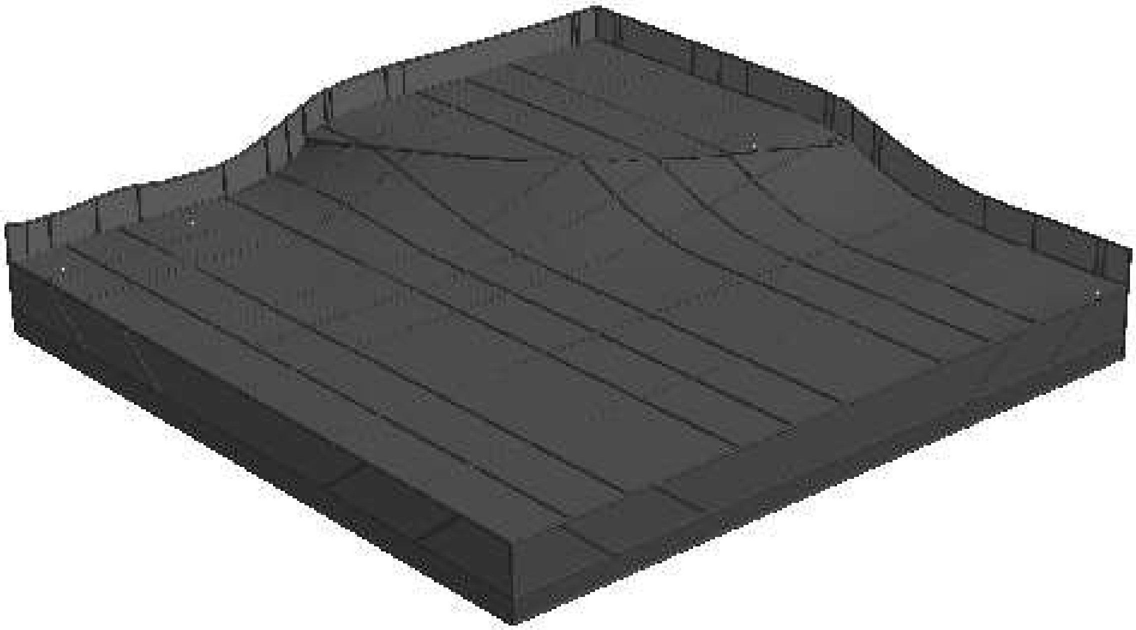


Figure 9.5. Drape fold showing crease in second layer.

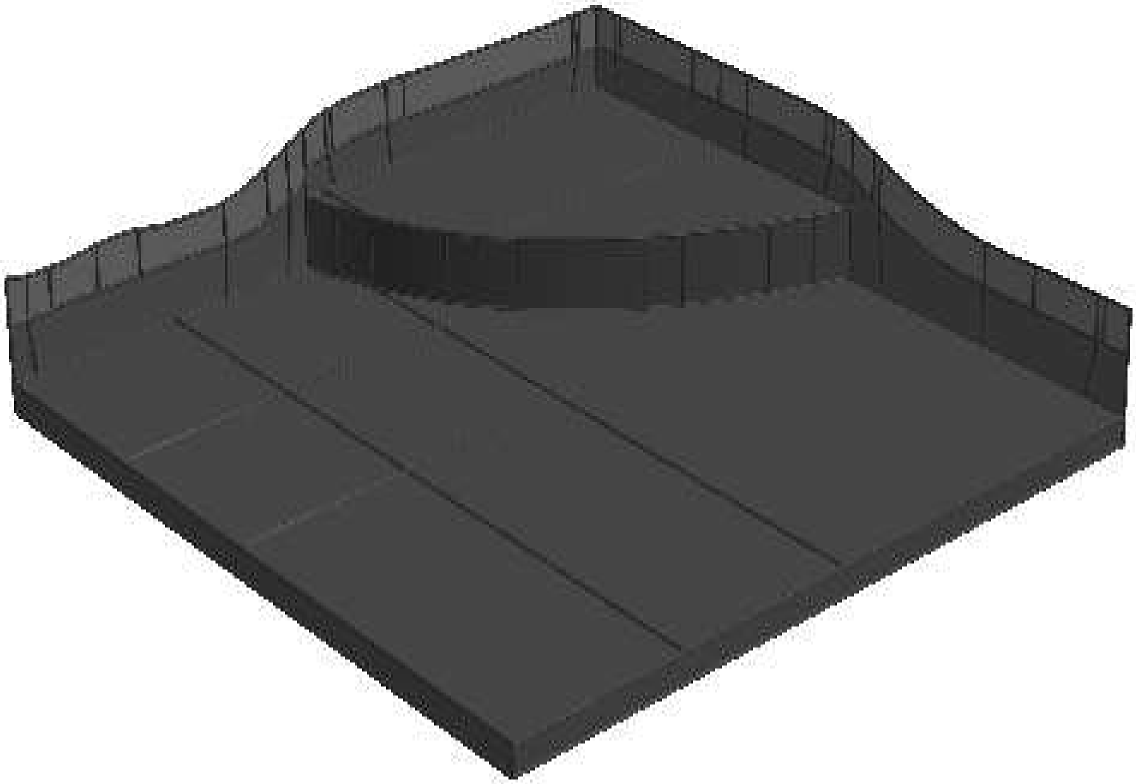


Figure 9.6. Drape fold showing fault in third layer.

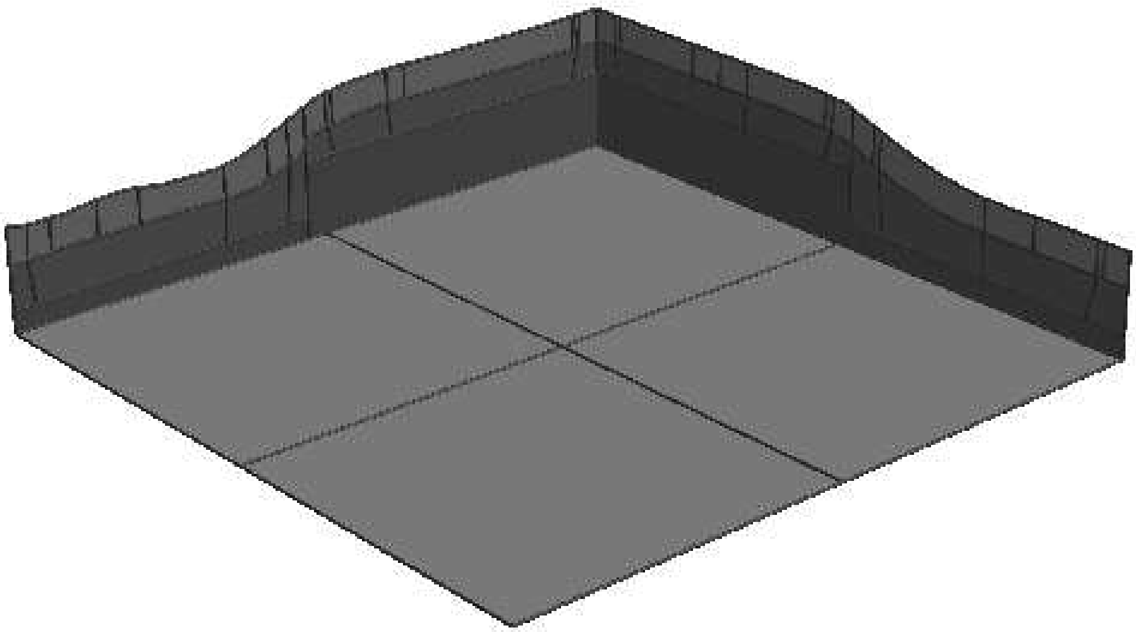


Figure 9.7. Drape fold bottom surface with back sides showing layered relationship.

CHAPTER 10

CONCLUSION

The problems associated with representing discontinuities in the modeling process are difficult. The torn tensor product B-spline representation was introduced to provide an easy and intuitive means for representing some of the more difficult situations. Existing higher-order representations which may be pieced together often have parameterization difficulties and may require extensive continuity constraints, often requiring significant user intervention. In addition, models composed solely of polygons lack the ability to further approximate the smooth surface they represent once discontinuities are introduced. The torn B-spline representation can incorporate arbitrary (specifically non-isoparametric) curves of discontinuity within a single surface representation.

The torn tensor product B-spline representation is derived from the widely used B-spline representation and, as such, supports the more familiar B-spline modeling operations such as refinement and subdivision. When using this representation, a modeling process which includes discontinuities no longer has to terminate or switch representations at the point of their introduction. The torn B-spline representational space is closed under modeling operations which introduce discontinuities. Designers can then continue the modeling process without interruption and incoporate the current structure in further modeling operations.

Examples from geology, sheet metal forming, and surface reconstruction demonstrate this power of the torn B-spline representation. Situations which were previously difficult, if not completely intractable, have straightforward solutions when the torn B-spline representation is used. Basic modeling operations for a command-based modeling environment utilize this representational power to dramatically reduce the amount of user intervention often required when producing these complex models by other methods.

CHAPTER 11

FUTURE WORK

Although the torn tensor product B-spline representation dramatically expands the flexibility available to the designer, there are still improvements that can be made, many of which have already been mentioned.

Foremost of these improvements would be an easier and more powerful crease maintenance mechanism. In the current representation, creases require a curve constraint to be maintained during the modeling process. The best situation would be a representation which was able to handle creases as easily as tears within the torn B-spline representation. However, the adjacency issues are so difficult and complex that the solution would likely be similar to a constraint system. Assuming that constraint maintenance is the most viable solution, it can still be improved. One difficulty of the current system is that the curve position or adjacency constraint is usually an approximation, even if there are adequate degrees of freedom available in the surface. The attractiveness of this approximation is its speed since these systems are usually linear, but there is a significant amount of *a priori* knowledge about the creases, most notably that both edges have the same parameterization, the same amount of flexibility, and that an exact solution does exist, i.e., the original, uncreased surface. A custom constraint maintenance system would be the most effective option in this situation, although the possible inclusion of other constraints needs to be considered.

Although this thesis presents several high-level modeling operators, these are clearly not all of the operators that could be useful. A modeling system which utilizes torn B-splines needs to have all of the representational flexibility at its fingertips otherwise the power will not be adequately harnessed. Additional techniques for constructing torn B-spline shells (collections of surfaces with adjacency information) would be useful for a CSG type of modeling system which uses boundary representations. Additional operators to combine parts of surfaces, to sew surfaces together, and to split surfaces apart could be developed to aid in the design process. The flexibility introduced by the 3D layered representation could be further utilized by operators which allowed for easy manipulation of these surfaces after construction.

The torn tensor product B-spline surface is introduced in this thesis since B-splines are one of the most commonly used smooth-surface representations. B-splines work well for the basis of this representations because the control of the surface local, depending on number of control points and order or the surface. This allows regions of varying continuity without adding new surfaces. Other parametric surface representations which have this characteristic of locality are prime candidates for extension. One potentially interesting class of surfaces are G^2 splines[30] which include γ-splines and β-splines.

Of course, complex representations have the tendency to be slower than simpler ones, but the speed issues encountered in the torn B-spline representation result more from the underlying trimmed surface representation than from the elements of the torn B-spline representation itself. More efficient trimmed B-spline surfaces would mean more efficient torn B-spline surfaces.

REFERENCES

[1] *Alpha 1 User's Manual.* Engineering Geometry Systems, Salt Lake City, 1992.

[2] Bajaj, C. L., and Ihm, I. Smoothing polyhedra using implicit algebraic splines. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 79–88, ACM SIGGRAPH, New York, 1992.

[3] Balun, T., Tang, S. C., Chappuis, L. B., and Wu, J. H. Application of mechanics methods to evaluation of forming and process design. Proceedings of the Sheet Metal and Stamping Symposium, SP–944 (Detroit, Mich., March 1–5, 1993), pages 181–187, SAE, Warrendale, Penn., 1993.

[4] Baraff, D. Analytical methods for dynamic simulation of non-penetrating rigid bodies. Proceedings of SIGGRAPH'89 (Boston, July 31–August 4, 1989), in *Computer Graphics* 23,3 (August 1989), pages 223–232, ACM SIGGRAPH, New York, 1989.

[5] Baraff, D. Curved surfaces and coherence for non-penetrating rigid body simulation. Proceedings of SIGGRAPH'90 (Dallas, August 6–10, 1990), in *Computer Graphics* 24,4 (August 1990), pages 19–28, ACM SIGGRAPH, New York, 1990.

[6] Bartels, R. H., Beatty, J. C., and Barskey, B. A. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers, Inc., Los Altos, Calif., 1987.

[7] Barzel, R. Modeling with dynamic constraints. SIGGRAPH Course Notes #17, July 1987.

[8] Barzel, R., and Barr, A. H. A modeling system based on dynamic constraints. Proceedings of SIGGRAPH'88 (Atlanta, August 1–5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 179–188, ACM SIGGRAPH, New York, 1988.

[9] Bathe, K.-J. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[10] Bathe, K.-J., and Wilson, E. L. *Numerical Methods in Finite Element Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1976.

[11] Blanc, C., and Schlick, C. X-splines: A spline model designed for the end-user. Proceedings of SIGGRAPH'95 (Los Angeles, August 6–11, 1995), pages 377–386, ACM SIGGRAPH, New York, 1995.

[12] Bloomenthal, J., and Ferguson, K. Polygonization of non-mainfold implicit surfaces. Proceedings of SIGGRAPH'95 (Los Angeles, Calif., August 6–11, 1995), pages 309–316, ACM SIGGRAPH, New York, 1995.

[13] Bloor, M. I. G., and Wilson, M. J. Representing PDE surfaces in terms of B-splines. *Computer-Aided Design* 22,6 (July–August 1990), 324–331.

[14] Bloor, M. I. G., and Wilson, M. J. Using partial differential equations fo generate free-form surfaces. *Computer-Aided Design* 22,4 (1990), 202–212.

[15] Carignan, M., Yang, Y., Magnenat Thalmann, N., and Thalmann, D. Dressing animated synthetic actors with complex deformable clothes. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 99–104, ACM SIGGRAPH, New York, 1992.

[16] Carlson, W. E. An algorithm and data structure for 3d object synthesis using surface patch intersections. Proceedings of SIGGRAPH'82 (Boston, July 26–30, 1982), pages 255–263, ACM SIGGRAPH, New York, 1982.

[17] Carlson, W. E. *Techniques for the Generation of Three Dimensional Data for Use in Complex Image Synthesis.* PhD thesis, Ohio State Univ., September 1982.

[18] Casale, M. S. Free-form solid modeling with trimmed surface patches. *IEEE Computer Graphics and Applications* 7,1 (January 1987), 33–43.

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

[20] Celniker, G., and Gossard, D. Deformable curve and surface finite-elements for free-form shape design. Proceedings of SIGGRAPH'91 (Las Vegas, Nev., July 28–August 2, 1991), in *Computer Graphics* 25,4 (August 1991), pages 257–266, ACM SIGGRAPH, New York, 1991.

[21] Celniker, G., and Welch, W. Linear constraints for deformable b-spline surfaces. Proceedings of Symposium on Interactive 3D Graphics (Cambridge, Mass., March 29–April 1, 1992), pages 165–170, ACM SIGGRAPH, New York, 1992.

[22] Chui, C. K. *Multivariate Splines.* Capital City Press, Montpelier, Vt., 1988.

[23] Cobb, E. S. *Design of Sculptured Surfaces using the B-spline Representation.* PhD thesis, Univ. of Utah, June 1984.

[24] Cohen, E., Lyche, T., and Riesenfeld, R. Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing* 14,2 (October 1980), 87–111.

[25] Cohen, M. *Interactive Spacetime Constraints for Linked Figures.* PhD thesis,

Univ. of Utah, June 1992.

[26] Costello, R. B., Ed. *Random House Webster's College Dictionary.* Random House, Inc., New York, 1992.

[27] Dixon, J. R., and Poli, C. *Engineering Design and Design for Manufacturing: A Structure Approach.* Field Stone Publishers, Conway, Mass., 1995.

[28] Doo, D., and Sabin, M. Analysis of the behaviour of recursive division surfaces. *Computer-Aided Design* 10,6 (November 1978), 356–360.

[29] Ellens, M. S., and Cohen, E. An approach to C^{-1} and C^0 feature lines. In *Mathematical Methods for Curves and Surfaces*, T. L. M. Daehlen and L. Schumaker, Eds. Vanderbilt Univ. Press, LaVergne, Tenn., 1995.

[30] Farin, G. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press, Inc., New York, 1988.

[31] Farouki, R. T. Trimmed-surface algorithms for the evaluation and interrogation of solid boundary representations. *IBM Journal of Research and Development* 31,3 (May 1987), 314–334.

[32] Feynman, C. R. Modeling the appearance of cloth. Master's thesis, MIT, May 1986.

[33] Forsey, D. R., and Bartels, R. H. Hierarchical B-spline refinement. Proceedings of SIGGRAPH'88 (Atlanta, August 1–5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 205–212, ACM SIGGRAPH, New York, 1988.

[34] Fowler, B. M. Geometric manipulation of tensor product surfaces. Proceedings of Symposium on Interactive 3D Graphics (Cambridge, Mass., March 29–April 1, 1992), pages 101–108, ACM SIGGRAPH, New York, 1992.

[35] Fowler, B. M., and Bartels, R. H. Constraint based curve manipulation. Submitted for publication., 1992.

[36] Fukutomi, K., Maruyama, M., Kudo, M., and Nagamitsu, T. Verification of finite element method applied to automotive body panels. Proceedings of the Sheet Metal and Stamping Symposium, SP–944 (Detroit, Mich., March 1–5, 1993), pages 173–179, SAE, Warrendale, Penn., 1993.

[37] Gascuel, M.-P., Verroust, A., and Puech, C. A modeling system for complex deformable bodies suited to animation and collision processing. SIGGRAPH Course Notes #28, July 1991.

[38] Gourret, J. P., N. M.-T., and Thalmann, D. Modeling of contact deformation between a synthetic human and its environment. *Computer-Aided Design* 23,7 (September 1991), 514–520.

[39] Grandin, Jr., H. *Fundamentals of the Finite Element Method.* Macmillan Publishing Co., Indianapolis, Ind., 1986.

[40] Grimm, C. M., and Hughes, J. F. Modeling surface of arbitrary topology using manifolds. Proceedings of SIGGRAPH'95 (Los Angeles, August 6–11, 1995), pages 359–368, ACM SIGGRAPH, New York, 1995.

[41] Hayes, J. G. New shapes from bicubic splines. NPL Report NAC 58, National Physical Laboratory, September 1974. Presented at CAD 74: International Conference on Computers in Engineering and Building Design, Imperial College, London.

[42] Herron, G. *Triangular and Multisided Patch Schemes.* PhD thesis, Univ. of Utah, September 1979.

[43] Hishida, Y., and Wagoner, R. W. Experimental analysis of blank holding force control in sheet forming. Proceedings of the Sheet Metal and Stamping Symposium, SP–944 (Detroit, Mich., March 1–5, 1993), pages 93–99, SAE, Warrendale, Penn., 1993.

[44] Hollig, K., and Mogerle, H. G-splines. *Computer Aided Geometric Design* 7,1 (September 1990), 197–207.

[45] Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W. Piecewise smooth surface reconstruction. Proceedings of SIGGRAPH'94 (Orlando, Fla., July 25–29, 1994), pages 295–302, ACM SIGGRAPH, New York, 1994.

[46] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. Surface reconstruction from unorganized points. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 71–78, ACM SIGGRAPH, New York, 1992.

[47] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. Mesh optimization. Proceedings of SIGGRAPH'93 (Anaheim, Calif., August 2–6, 1993), pages 19–26, ACM SIGGRAPH, New York, 1993.

[48] Huebner, K., and Thornton, E. A. *Finite Element Method for Engineers.* 2nd edition. Wiley, New York, 1982.

[49] Isaacs, P. M., and Cohen, M. F. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. Proceedings of SIGGRAPH'87 (Anaheim, Calif., July 27–31, 1987), in *Computer Graphics* 21,4 (July 1987), pages 215–224, ACM SIGGRAPH, New York, 1987.

[50] Light, R. A., and Gossard, D. C. Modification of geometric models through variational geometry. *Computer-Aided Design* 14,4 (July 1982), 209–214.

[51] Loop, C. Smooth subdivision surfaces based on triangles. Master's thesis, Univ. of Utah, August 1987.

[52] Loop, C. Smooth spline surfaces over irregular meshes. Proceedings of SIGGRAPH'94 (Orlando, Fla., July 25–29, 1994), pages 303–310, ACM SIGGRAPH, New York, 1994.

[53] Loop, C., and DeRose, T. Generalized B-spline surfaces of arbitrary topology. Proceedings of SIGGRAPH'90 (Dallas, August 6–10, 1990), in *Computer Graphics* 24,4 (August 1990), pages 347–356, ACM SIGGRAPH, New York, 1990.

[54] Lyche, T., and Morken, V. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design* 4,3 (1987), 217–230.

[55] Malone, J. G., Plunkett, R., and Hodge, Jr., P. G. An elastic-plastic finite element solution for a cracked plate. *Finite Elements in Analysis And Design* 2,4 (April 1986), 389–407.

[56] McCollough, Jr., W. T. Trimmed surfaces. Master's thesis, Univ. of Utah, July 1988.

[57] Mirtich, B., and Canny, J. Impulse-based simulation of rigid bodies. Proceedings of Symposium on Interactive 3D Graphics (Monterey, Calif., April 9, 1995), pages 181–188, ACM SIGGRAPH, New York, 1995.

[58] Moreton, H. P., and Sèquin, C. H. Functional optimization for fair surface design. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 167–176, ACM SIGGRAPH, New York, 1992.

[59] Mueller, T. I. *Geometric Modelling with Multivariate B-splines.* PhD thesis, Univ. of Utah, June 1986.

[60] Munkres, J. R. *Topology.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[61] Nielson, G. M. Some piecewise polynomial alternatives to splines under tension. *Computer Aided Geometric Design* (1974), 209–235.

[62] Peiper, S., Rosen, J., and Zeltzer, D. Interactive graphics for plastic surgery: A task-level analysis and implementation. Proceedings of Symposium on Interactive 3D Graphics (Cambridge, Mass., March 29–April 1, 1992), pages 127–134, ACM SIGGRAPH, New York, 1992.

[63] Pentland, A., and Williams, J. Good vibrations: Modal dynamics for graphics and animation. Proceedings of SIGGRAPH'89 (Boston, July 31–August 4, 1989), in *Computer Graphics* 23,3 (August 1989), pages 215–222, ACM SIGGRAPH, New York, 1989.

[64] Pierre, D. A. *Optimization Theory with Applications.* John Wiley and Sons, Inc., New York, 1969.

[65] Platt, J. C., and Barr, A. H. Constraint methods for flexible models. Proceedings of SIGGRAPH'88 (Atlanta, August 1–5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 179–288, ACM SIGGRAPH, New York, 1988.

[66] Plotnikov, L. M. *Shear Structures in Layered Geological Bodies.* A. A. Balkema Publishers, Brookfield, Vt., 1991.

[67] Press, W. H., Flanner, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge Univ. Press, Cambridge, England, 1986.

[68] Qin, H. *D-NURBS: Dynamic Non-Uniform Rational B-Splines.* PhD thesis, Univ. of Toronto, 1995.

[69] Sarraga, R. F. Computer modeling of surfaces with arbitrary shapes. *IEEE Computer Graphics and Applications* 10,2 (March 1990), 67–77.

[70] Sarraga, R. F., and Waters, W. C. Free-form surfaces in GMSolid: Goals and issues. In *Solid Modeling by Computers*, M. S. Pickett and J. W. Boyse, Eds. Plenum Press, New York, September 1984, pages 187–209. Proceedings of a symposium held September 25-27, 1983, at the General Motors Research Laboratories, Warren, Mich.

[71] Schroeder, W. J., Zarge, J. A., and Lorensen, W. E. Decimation of triangle meshes. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 65–70, ACM SIGGRAPH, New York, 1992.

[72] Strang, G., Ed. *Linear Algebra and Its Applications*, second ed. Academic Press, Inc., New York, 1976.

[73] Szeliski, R., and Tonnesen, D. Surface modeling with oriented particle systems. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 185–194, ACM SIGGRAPH, New York, 1992.

[74] Terzopoulos, D. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 8,4 (July 1986), 413–424.

[75] Terzopoulos, D., and Fleischer, K. Deformable models. *The Visual Computer* 4 (1988), 306–331.

[76] Terzopoulos, D., and Fleischer, K. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. Proceedings of SIGGRAPH'88 (Atlanta, August 1–5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 269–278, ACM SIGGRAPH, New York, 1988.

[77] Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. Elastically deformable models. Proceedings of SIGGRAPH'87 (Anaheim, Calif., July

27–31, 1987), in *Computer Graphics* 21,4 (July 1987), pages 205–214, ACM SIGGRAPH, New York, 1987.

[78] Terzopoulos, D., Platt, J., and Fleischer, K. From goop to glop: Heating and melting deformable models. Tech. rep., Schlumberger Palo Alto Research, Palo Alto, Calif., November 1988.

[79] Terzopoulos, D., and Qin, H. Dynamic NURBS with geometric constraints for interactive sculpting. *ACM Transactions on Graphics* 13,4 (April 1994), 103–136.

[80] Terzopoulos, D., and Witkin, A. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications* 8,6 (November 1988), 41–51.

[81] Thingvold, J. A. Elastic and plastic surfaces for modeling and animation. Master's thesis, Univ. of Utah, March 1990.

[82] Thomas, S. *Modelling Volumes Bounded by B-spline Surfaces.* PhD thesis, Univ. of Utah, June 1984.

[83] Turk, G., and Levoy, M. Zippered polygon meshes from range images. Proceedings of SIGGRAPH'94 (Orlando, Fla., July 25–29, 1994), pages 311–318, ACM SIGGRAPH, New York, 1994.

[84] Vince, J. *3-D Computer Animation.* Addison-Wesley, Reading, Mass., 1992.

[85] Volino, P., Courchesne, M., and Thalmann, M. M. Versatile and efficient techniques for simulating cloth and other deformable objects. Proceedings of SIGGRAPH'95 (Los Angeles, August 6–11, 1995), pages 137–144, ACM SIGGRAPH, New York, 1995.

[86] Wang, Y. F., and Wang, J.-F. Surface reconstruction using deformable models with interior and boundary constraints. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 14,5 (May 1992), 572–579.

[87] Watt, A., and Watt, M. *Advanced Animation and Rendering Techniques.* ACM Press, New York, 1992.

[88] Weil, J. The synthesis of cloth objects. Proceedings of SIGGRAPH'86 (Dallas, August 18–22, 1986), pages 49–54, ACM SIGGRAPH, New York, 1986.

[89] Weinberg, D. M. Some two-dimensional kinematic analyses of the drape-fold concept. In *Laramide Folding Associated with Basement Block Faulting in the Western United States*, V. M. III, Ed. The Geological Society of America, Inc., Boulder, Colo., 1978.

[90] Welch, W., and Witkin, A. Variational surface modeling. Proceedings of SIGGRAPH'92 (Chicago, July 27–31, 1992), pages 157–166, ACM SIG-

GRAPH, New York, 1992.

[91] Witkin, A., Fleischer, K., and Barr, A. Energy constraints on parameterized models. Proceedings of SIGGRAPH'87 (Anaheim, Calif., July 27–31, 1987), in *Computer Graphics* 21,4 (July 1987), pages 225–231, ACM SIGGRAPH, New York, 1987.

[92] Witkin, A., and Kass, M. Spacetime constraints. Proceedings of SIGGRAPH'88 (Atlanta, August 1–5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 159–168, ACM SIGGRAPH, New York, 1988.

[93] Witkin, A., and Welch, W. Fast animation and control of nonrigid structures. Proceedings of SIGGRAPH'90 (Dallas, August 6–10, 1990), in *Computer Graphics* 24,4 (August 1990), pages 243–252, ACM SIGGRAPH, New York, 1990.

[94] Yen, W.-c. J. *On Representation and Discretization of Finite Element Analyses.* PhD thesis, Univ. of Utah, December 1985.