Feature Oriented Composition Of B-Spline Surfaces

by

Cristin Barghiel

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1995

Abstract

Detailed features are added to composite spline surfaces in a multi-layered fashion by means of an efficient displacement scheme. The feature orientation is arbitrary, and the underlying domains may be partially overlapping and non-linearly transformed. By mapping only the control vertices, defined as displacement vectors in a diffuse coordinate system, we can manipulate the pasted elements in real-time, independently or as a whole.

The procedure, known as *pasting*, is more than a refinement instrument. It makes possible the rendering of a surface composition selectively, and thus expedites the rendering process. It can be used to model surfaces by interactively changing displacements, control vertices, or the domain layout. Base surfaces may serve as sliding paths for pasted clusters whose shape and motion are determined by the underlying topography, resulting in an organic motion effect. Pasting also applies to higher-dimension entities, which may incorporate non-geometric components, such as color, opacity or brightness.

Acknowledgements

I would like to thank Dr. Richard Bartels, who has guided my footsteps with wisdom and patience, and has encouraged me to explore the area of geometric design. I would also wish to thank my readers, Dr. Charlie Colbourn and Dr. Steve Mann, for their time and constructive criticism. I cherish the friendship of the CGL family, the many exciting chats and brain-storming sessions during the late night hours, and most notably the valuable advice given to me by wizards such as Alex Nicolaou, Robert Kroeger, and Frank Henigman.

Financial support for the research presented in this thesis was provided by post-graduate scholarships from the Natural Sciences and Engineering Research Council of Canada, and the Province of Ontario's Information Technology Research Centre.

I dedicate this work to my parents, whose infinite love and support have given me the strength to go forth, and to Dr. Dan Somnea, the first to show me the magic world of computer graphics.

Trademarks

Silicon Graphics is a registered trademark of Silicon Graphics Incorporated.

PasteMaker and the Splines Library are under the copyright of the Computer Graphics Laboratory at the University of Waterloo.

All other products mentioned in this thesis, such as Math.h++ and Tools.h++, are trademarks of their respective companies. Our use of general descriptive names, trade names, and trademarks, even if not precisely identified, is not an indication that such names may be used freely in all circumstances.

Contents

List of Figures

Chapter 1

Introduction

Free-form surfaces have become popular tools in the hands of today's industrial designers and computer animators. Back in the early 1960's, non-parametric functions of one or more arguments marked the first attempt at representing smooth shapes by means of a computerized model; by virtue of their definition, however, these functions were not adequate for fine surface control and adjustment, and had little to offer in terms of initial parameter specification.

The parametric curve and surface functions introduced independently by de Casteljau [dC63] and Bézier [B66] quickly became a standard design tool with valuable properties, including *affine invariance*, the *convex hull* property, and the *variation diminishing* property [Far93]. Key points on the convex hull of the surface, known as *control vertices*, now allowed the modeler to specify the shape of a surface patch to finer detail and with increased accuracy. To edit a patch, one would change the position of one or more control vertices, thus affecting the aspect of the whole geometry. A separate stream of research led by Schoenberg and de Boor [dB78] in the late 1950's – early 1960's and popularized by Riesenfeld in the 1970's, brought about

the B-spline, a parametric representation more general than Bézier's approach and with important new properties such as better *continuity control.*

1.1 Motivation And Goals

B-Splines are easy to maintain and economical with respect to the amount of data that defines them. Large B-spline surfaces are typically used to represent sophisticated models; often, surfaces must coexist in a model that undergoes change, and certain aspect criteria need to be preserved. For example, surfaces defining a flexible head profile must remain smooth throughout any motion or stretching. Such requirements are hard to meet when the component surfaces are assembled via alignment or blending operations. A more suitable alternative is to use *simple B-spline tensor product surfaces* that consist of internally aligned and guaranteed continuous pieces called *patches.*

A remaining problem, though, is the possibly high number of control vertices needed to represent such a surface. If the model is intended for further adjustment rather than static viewing, speed and space-saving considerations suggest that it is desirable to represent a surface with fewer control vertices, while keeping the shape information intact. If, however, extensive detailing is necessary, fewer control vertices may be insufficient. There are two ways to increase the number of control vertices: elevate the degree of the surface, or refine it by inserting domain points, called *knots.* Boehm's algorithm [Boe80] and the Oslo algorithm [BBB87] are among the more popular refinement methods. In either case, however, the main complaint is that the algorithm produces more control vertices than desired, and increases the computation needed to evaluate the surface.

An alternative solution, aimed at maintaining a low number of control vertices,

was presented by Forsey and Bartels [FB88]: detail surfaces, called *overlays* or *features*, are applied onto a *base* surface in a hierarchical fashion to create a composite surface of increased complexity. Each overlay is a normal displacement from a conveniently chosen reference point. The approach focuses on uniform B-splines, assumes a strictly nested (tree-like) layering of the overlays, and requires that all domains be parametrically aligned.

In this thesis we generalize Forsey's method, elaborate on a displacement method proposed by Bartels [BF91], and investigate ways of applying it to interactive, real-time modeling, as well as to real-time animation. To avoid computing the displacement of every point along the feature surface, we define an *approximate* displacement mechanism that involves only the control vertices, and we illustrate that, by recursively refining the feature, the approximate displacement approaches the ideal one at the limit. One employs this efficient, yet inexact approximate displacement method during an interactive previewing session, then applies the true displacement for the final result.

We remove the nesting requirement for the surface regions, allowing them to be transformed freely – independently, as clusters, or as a whole – and to overlap so as to form a Directed Acyclic Graph (DAG) rather than just an N-ary Tree. There is to be no restriction on the transformations applied to the regions (as long as an inverse transformation is defined), or on the normal or tangential displacement. The study focuses on non-uniform B-splines, which call for a more general displacement scheme, and on the feasibility of the approximation technique (called *pasting*) for previewing, free-form design, and animation.

Under this approach, the pasting procedure becomes more than a refinement tool; it provides the possibility of rendering a surface composition selectively, thus expediting the rendering process. It can be used to model compositions by interac-

tively changing displacements, control vertices of one or more surfaces, and region layouts. Base surfaces may serve as *sliding* paths for pasted clusters whose motion and shape are determined by the underlying topography, generating an *organic motion* effect. Pasting also applies to higher dimension entities that incorporate non-geometric components, such as color, opacity, and brightness.

1.2 Overview

The second chapter introduces the terminology and notation, presents an overview of B-splines and displacement mappings, and describes the pasting method from a mathematical standpoint.

Chapter 3 discusses the hierarchical aspects of the pasting procedure. The discussion is geared towards an interactive, dynamic approach to modeling. It suggests an efficient clipping algorithm for two-dimensional B-spline surface domains, based on barycentric coordinates. Finally, it introduces the pasting data structures and a set of pasting-related operations.

Chapter 4 lists a number of applications to free-form modeling and animation. Then it describes PasteMaker, a C++ application that implements the elements presented in the previous chapters. A section is dedicated to operations either derived from the pasting procedure or closely related to it. The chapter illustrates PasteMaker's architecture by means of the Model-View-Controller (MVC) paradigm, and closes with comments on the editor's user interface.

Chapter 5 summarizes the findings, explores more exotic areas of applicability, and probes into future research.

Appendix A is a complete listing of the C++ header files representing the core

of the paste library (SPaste) and PasteMaker's engine. The software was written at the Computer Graphics Lab, as a component of the Splines Library [VB92].

Appendix B illustrates the pasting concepts in a set of color plates. The images were created with PasteMaker, the editor presented in Chapter 4.

Chapter 2

Feature-Oriented Pasting: The Basic Approach

After establishing the notation used throughout the discussion, the chapter introduces the concept of a B-spline surface, defines the pasting procedure, and evaluates the performance of the pasting operation as a modeling and animation instrument.

The pasting philosophy is independent of the dimension of the space in which it is formulated. To facilitate its understanding, however, we focus our discussion on the three-dimensional euclidean space, R^3. We denote points by upper case Roman characters, and represent them as triples of coordinates: P $= (x, y, z)$, where $x, y, z \in R$. Greek or Italic fonts are used to represent regular scalars such as x, y, z in lower case, and surface domains (D) in upper case. Vectors are given in lower case bold, and are expressed as scaled unit components in each spatial direction: $\mathbf{v} = (\alpha\mathbf{i}, \beta\mathbf{j}, \gamma\mathbf{k})$, with $\alpha, \beta, \gamma \in R$. Finally, we denote transformations by upper case bold characters, as in $\mathbf{T}$ or $\mathbf{F}$.

2.1 An Overview of B-Spline Surfaces

In the broadest sense, a three-dimensional spline surface is defined in terms of a two-dimensional domain D and a collection of three-dimensional control vertices knows as CVs. Depending on the type of CV, one can identify two classes of spline surfaces: *point splines* and *vector splines.*

We base our presentation on a widely used type of spline surfaces, the *B-spline tensor product.* The shape of the tensor product is determined by a rectangular mesh of *control vertices* and a pair of *basis functions*, $B(u)$ and $B(v)$ over domain D, defined in each parametric direction – u and respectively v. Bézier bases and non-uniform rational bases ($NURBS$) are among the most frequently used [BBB87]; both types represent feasible incarnations of the pastable surfaces introduced next.

2.1.1 Point and Vector B-Splines

A *point B-spline surface* $\mathrm{S}(u,v)$ is a linear combination of control points $\mathrm{P}_{i,j}$[1] and *basis functions* $B_{i,k}(u)$ and $B_{j,\ell}(v)$:

$$\mathrm{S}(u,v) = \sum_i \sum_j \mathrm{P}_{i,j} B_{i,k}(u) B_{j,\ell}(v) \tag{2.1}$$

where k and ℓ are the orders of the first and second basis function respectively.

If we replace the control points by *control vectors* representing offsets from some origin, we obtain the definition of a *vector (or displacement) B-spline surface*, $\mathrm{s}(u,v)$:

[1] Commonly referred to as CVs. Here, we define both point *and* vector splines, and must distinguish between the two types of control entities; therefore, we will call them CVs only when their type is irrelevant.

$$\mathbf{s}(u,v) = \sum_i \sum_j \mathbf{p}_{i,j} B_{i,k}(u) B_{j,\ell}(v) \tag{2.2}$$

where $\mathbf{p}_{i,j}$ are the control vectors. In surface modeling, vector splines present the advantage of being independent of the coordinate frame origin. They also allow great flexibility in determining a very reasonable displacement scheme [FB88, BF91], and are well suited to handling Forsey's overlays [For90] more generally and conveniently.

It is possible to write a *point spline*, defined in a frame whose origin is $\mathrm{O} \in R^3$, in terms of a *vector spline*. To represent the vector spline relative to O, we add the origin to both sides of Equation 2.2, which becomes:

$$\begin{aligned} \mathrm{O} + \mathbf{s}(u,v) &= \mathrm{O} + \sum_i \sum_j \mathbf{p}_{i,j} B_{i,k}(u) B_{j,\ell}(v) \Leftrightarrow \\ \mathrm{O} + \mathbf{s}(u,v) &= \sum_i \sum_j (\mathrm{O} + \mathbf{p}_{i,j}) B_{i,k}(u) B_{j,\ell}(v) \end{aligned} \tag{2.3}$$

Since $\mathrm{S}(u,v)$ is rooted at origin O, each of its control points $\mathrm{P}_{i,j}$ can be expressed relative to O and $\mathbf{p}_{i,j}$: $\mathrm{P}_{i,j} = \mathrm{O} + \mathbf{p}_{i,j}$. Then, by Equation 2.1, the above equation yields:

$$\mathrm{O} + \mathbf{s}(u,v) = \mathrm{S}(u,v) \tag{2.4}$$

Equation 2.4 shows that a *point spline* can be written as a point-vector sum of the frame origin and the *displacement spline*. This is the observation that underlies the system of *offsets* used in [FB88, BF91]

2.1.2 The Diffuse Coordinate System

A more interesting displacement scheme, which also underlines the pasting mechanism, involves a *scattered* or *diffuse* coordinate space (DCS), which associates a

"suitably chosen" frame with each control vector. The frame selection process will be defined later. For now, assume that frame $\mathbb{F}_{i,j} \in DCS$ has been assigned an origin $\mathrm{O}_{i,j}$ and an offset $\mathbf{p}_{i,j}$. Then, the *point spline* (Equation 2.1) can be written as:

$$\mathrm{S}(u,v) = \sum_i \sum_j (\mathrm{O}_{i,j} + \mathbf{p}_{i,j}) B_{i,k}(u) B_{j,\ell}(v) \tag{2.5}$$

The choice of origin $\mathrm{O}_{i,j}$ and displacement $\mathbf{p}_{i,j}$ is paramount for the quality of the pasting operation, and determines the behavior of pasted and unpasted surfaces when either the origin or the displacement handle has changed. Before describing the DCS construction, we must briefly dwell on the underlying domain of the B-spline surface.

2.1.3 Greville Displacement B-Splines

The two-dimensional domain of a piecewise B-spline surface is determined by the values of its *knots* [BBB87] in each parametric direction. Consider the general case of a non-uniform B-spline (NUB) surface, which has two basis functions $B_k(u)$ and $B_\ell(v)$ defined as in Equation 2.1 over domain D. Let μ and ν be the number of knots in each direction, and $\{u_0, \ldots, u_{\mu-1}\}$, $\{v_0, \ldots, v_{\nu-1}\}$ the respective nondecreasing knot sequences. Then, the complete surface domain is

$$D = \{(u_i, v_j) | 0 \le i < \mu, 0 \le j < \nu\} \tag{2.6}$$

If the surface has $m \times n$ control vertices ($m < \mu$ and $n < \nu$), the domain over which it is *valid* or *well defined* is D', $D' \subset D$:

$$D' = \{(u,v) | u_{k-1} \le u \le u_{m+1}, v_{\ell-1} \le v \le v_{n+1}\} \tag{2.7}$$

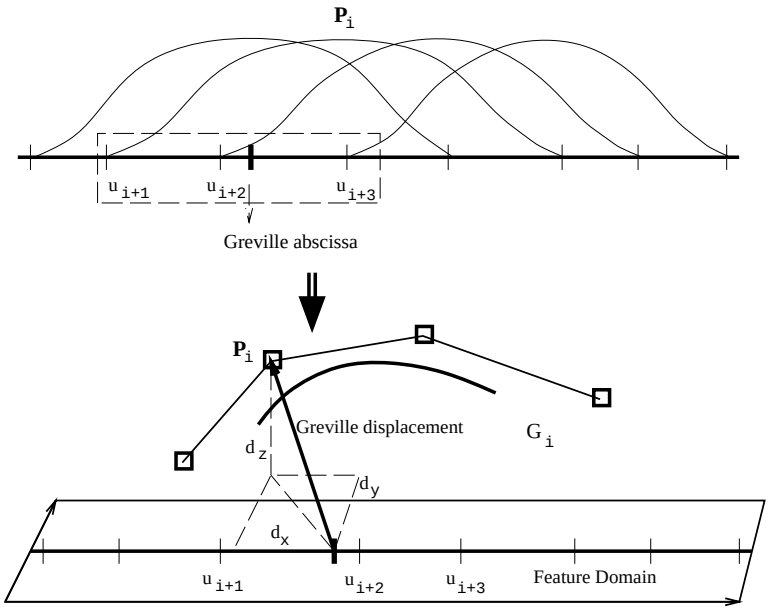

Figure 2.1: Greville abscissa and the displacement scheme.

We illustrate the choice of origin $O_{i,j}$ in Figure 2.1, leaving the variable v aside and concentrating on the variable u for simplicity. A NUB-spline basis is defined such that each control vertex $P_{i,j}$ influences a number of knots equal to the degree of the basis. These knots belong to D', the domain over which the spline is well defined. In Figure 2.1a, for example, vertex $P_{i,j}$ influences domain points u_{i+1}, u_{i+2}, and u_{i+3} (and similarly for v). The domain point over which a control vertex has *maximal influence* is called a *Greville abscissa* [Far93], and is defined as an average of 'degree' d knots:

$$\gamma_i = \frac{1}{d}(u_{i+1} + \ldots u_{i+d}) \tag{2.8}$$

with i ranging from 0 to the number of control vertices minus 1. (and similarly for v) The image of the Greville abscissa on the surface indicates the point likely to be most affected when the corresponding CV moves. The above equation applies with little change to the simpler case of the Bezier basis, whose breakpoints can be

interpreted as knots with multiplicity d.

In summary, the *Greville point* $\gamma_{i,j}$ associated with control vertex $\mathrm{P}_{i,j}$ is defined as a pair of Greville abscissae, one for each parametric direction:

$$\gamma_{i,j} = (\gamma_i, \gamma_j)$$

with γ_i and γj computed as in Equation 2.8.

The inherently two-dimensional Greville points can be embedded into a three-dimensional space by adding an arbitrary coordinate to each point. Let the free coordinate be 0. Then, the three-dimensional Greville point is

$$\Gamma_{i,j} = (\gamma_{i,j}, 0)$$

The array of Greville points forms a discrete *Greville domain*, which shares the same space as the CVs. The latter can then be rewritten as CV offsets in corresponding frames, with $\Gamma_{i,j}$ as the origin and $\mathrm{P}_{i,j} - \Gamma_{i,j}$ as the displacement:

$$\mathrm{P}_{i,j} = \Gamma_{i,j} + (\mathrm{P}_{i,j} - \Gamma_{i,j})$$

Let $\mathbf{d}_{i,j} = \mathrm{P}_{i,j} - \Gamma_{i,j}$ be the *Greville displacement*. The pair $\{\Gamma_{i,j}, \mathbf{d}_{i,j}\}$ forms a local coordinate frame $\mathbb{F}_{i,j}$ for control vertex $\mathrm{P}_{i,j}$:

$$\mathbb{F}_{i,j} = \{\Gamma_{i,j}, \mathbf{d}_{i,j}\} \tag{2.9}$$

Then by Equation 2.5 the diffuse representation of a point B-spline in terms of its Greville points is:

$$\mathrm{S}(u,v) = \sum_i \sum_j (\Gamma_{i,j} + \mathbf{d}_{i,j}) B_{i,k}(u) B_{j,\ell}(v) \tag{2.10}$$

Equation 2.10 is illustrated in Figure 2.1b, which shows the domain embedded in 3-space, and one Greville displacement drawn from the (i,j)th Greville point to the corresponding CV; d_x, d_y, and d_z are the x, y, and respectively z components of the Greville displacement vector. By inverting the sign of the z component it is possible to mirror the displacement relative to the XY plane.

Thus, a point B-spline can be written as a collection of local displacements from the Greville abscissae and represented vectorially as a *Greville Displacement B-spline* by removing the Γ origins from Equation 2.10:

$$\mathbf{s}(u,v) = \sum_i \sum_j \mathbf{d}_{i,j} B_{i,k}(u) B_{j,\ell}(v)$$

Algorithm 2.1 summarizes the process described so far:

Algorithm 2.1 Building a Greville Displacement B-Spline

given point B-spline $\mathrm{S}(u,v) = \sum_i \sum_j \mathrm{P}_{i,j} B_{i,k}(u) B_{j,\ell}(v)$*:*
  for each control point $\mathrm{P}_{i,j}$ *do*
    compute 2D Greville abscissa $\gamma_{i,j}$*;*
    embed $\gamma_{i,j}$ *into 3-space to obtain* $\Gamma_{i,j}$*;*
    compute Greville displacement $\mathbf{d}_{i,j} = \mathrm{P}_{i,j} - \Gamma_{i,j}$
  end
  assemble vector B-spline $\mathbf{s}(u,v) = \sum_i \sum_j \mathbf{d}_{i,j} B_{i,k}(u) B_{j,\ell}(v)$
end

The significance of the above Greville surface representation derives from the *maximal influence* property of the Greville abscissae: by establishing a vectorial

relationship between control points and Greville points, one defines a matrix of *maximal support* for the given point spline, one most likely to preserve the shape and proportions of the surface under arbitrary domain transformation.[2]

2.2 The General Mapping Procedure

Chapter 1 has already mentioned that the main drawback of classic refinement methods is the often unnecessary addition of control vertices leading to "heavy" splines. An alternative is to use a displacement mapping to add detail to a base surface. Although conceptually sound, this solution is not feasible for interactive applications due to the expense involved in displacing every evaluation point in the surface hierarchy. David Forsey's approach [For90] simplified the algorithm by approximating the complete displacement mapping with a mapping of control points only. The procedure is applied recursively for every overlay (or feature) in the hierarchy, and requires only that all overlay domains be strictly nested and aligned.

The pasting approach described here generalizes Forsey's algorithm and enhances the modeling potential of the resulting composition. It involves at least two surfaces: a *base surface* on which to paste S_B, defined over domain D_B, and a *feature surface* S_F defined over domain D_F. Assume the feature point to be mapped is $\mathrm{S}_F(r,s)$, determined by point $(r,s) \in D_F$, and its mapped image on the pasted surface is $\widetilde{\mathrm{S}}_F(r,s)$, as shown in Figure 2.2.

The process of pasting S_F onto S_B starts with mapping D_F into D_B by an *invertible* transformation $\mathbf{T}$. Let the mapped domain point be $(u,v) \in D_B$, with

[2] Invertible transformations are particularly relevant to the pasting process.

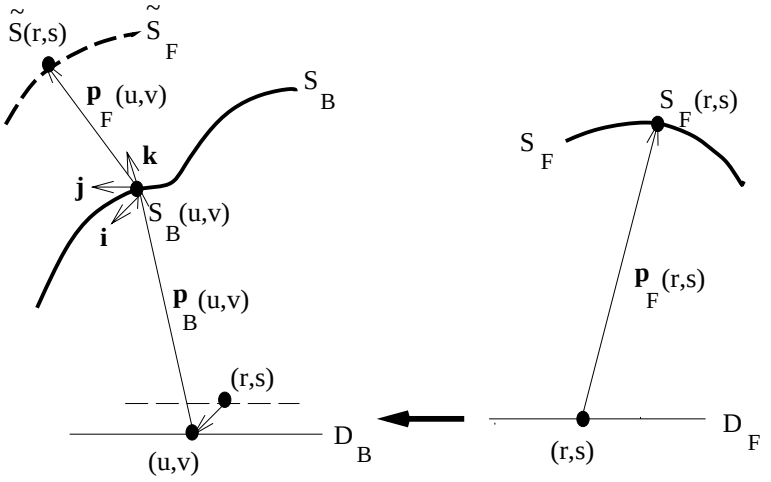

Figure 2.2: The one-point mapping procedure.

pre-image $(r,s) = \mathbf{T}^{-1}(u,v) \in D_F$, and let $\mathrm{S}_B(u,v)$ be the point on base surface S at (u,v). If $\mathbf{s}_F$ is the vector representation of surface S_F (see Equation 2.2) and $\mathbf{d}_F$ its displacement matrix, the displaced feature point $\widetilde{\mathrm{S}}_F(r,s) = \mathrm{S}_F(\mathbf{T}^{-1}(u,v))$ results from applying the $\mathbf{d}_F(\mathbf{T}^{-1}(u,v))$ displacement to its mapped origin $\mathrm{S}_B(u,v)$.

The quality of the result is highly dependent on the type of displacement chosen for $\mathbf{d}_F$: either *global* displacement relative to a unique origin O, as in Equation 2.3, or *local* displacement of each point $\mathrm{P}_{i,j}$ relative to origin $\mathrm{O}_{i,j}$, as in Equation 2.5. A global displacement is unlikely to be sufficiently sensitive to the topography of the base surface and to generate a uniformly distorted pasting. However, if the displacement is anchored at multiple origins within the feature domain, as the second option suggests, a more satisfactory mapping is achieved.

Ideally, each local coordinate frame should be representative of the shape of the surface and its behavior under topographic stress. A good candidate for a diffuse coordinate system with this property is the Greville displacement spline introduced earlier in this chapter (Equation 2.9), which assigns the Greville points to frame

origins, and computes the displacements by subtracting each Greville point from its corresponding control point. This choice is valid as long as one is interested in mapping only the control points.

To apply the feature displacement onto the mapped origin $\mathrm{S}_B(u,v)$, one must set up a local coordinate frame $\mathbb{F}$ at $\mathrm{S}_B(u,v)$, where $\mathbb{F}_B = \{\mathrm{S}_B(u,v), \mathbf{v}(\mathbf{i},\mathbf{j},\mathbf{k})\}$. The yet undetermined vector $\mathbf{v}$ must reflect the curvature of the base surface at the mapped origin $\mathrm{S}_B(u,v)$, and convey the direction of $\mathbf{d}_F$'s vector as mapped by transformation $\mathbf{T}$. The tangent-plane vector components $\mathbf{i}$ and $\mathbf{j}$ are given by the partial derivatives of base surface S_B at $\mathrm{S}_B(u,v)$, and

$$\mathbf{k} = \begin{cases} \mathbf{i} \times \mathbf{j} & \text{for positive displacement} \\ \mathbf{j} \times \mathbf{i} & \text{for negative displacement} \end{cases}$$

If the pre-image domain points are $r(u,v)$ and $s(u,v)$, the partial derivatives are:

$$\begin{cases} \frac{\partial \mathrm{S}_B}{\partial r} = \frac{\partial \mathrm{S}_B}{\partial u}\frac{\partial u}{\partial r} + \frac{\partial \mathrm{S}_B}{\partial v}\frac{\partial v}{\partial r} \\ \frac{\partial \mathrm{S}_B}{\partial s} = \frac{\partial \mathrm{S}_B}{\partial u}\frac{\partial u}{\partial s} + \frac{\partial \mathrm{S}_B}{\partial s}\frac{\partial v}{\partial s} \end{cases} \tag{2.11}$$

Efficiency considerations suggest that Equation 2.11 be replaced with a good yet inexpensive approximation, such as the *difference quotient*.

$$\frac{\partial \mathrm{S}_B}{\partial r} \approx \frac{\mathrm{S}_B(u(r+\epsilon,s), v(r+\epsilon,s)) - \mathrm{S}_B(u(r,s), v(r,s))}{\epsilon}$$

Difference quotients yield satisfactory results for a very small ϵ value that is inversely proportional to the distance between two successive domain points.

Finally, the pasted image $\widetilde{\mathrm{S}}(r,s)$ results from embedding the feature displacement $\mathbf{d}(r,s)$ into the local coordinate frame $\mathbb{F}$. The operation is a point-vector addition performed in the given frame context:

$$\widetilde{\mathrm{S}}_F(r(u,v), s(u,v)) = \mathrm{S}_B(u,v) + \mathbf{d}(r,s)$$

2.3 Pasting Greville Displacement B-Splines

The previous section has made no assumptions about the number or type of feature points mapped onto the base surface. The same procedure applies to one or many points, and its result depends mostly on the displacement scheme chosen for the feature surface. One instance of the algorithm is used in the *pasting operation*, which maps only the control mesh of the feature, and generates a new B-spline surface defined in terms of the mapped CVs (Plate 1). The surface representing the displacement is the Greville vector B-spline (Equation 2.9). Pasting a feature S_F onto a base S_B reduces to mapping each of the feature Greville displacements onto the base surface, and rebuilding a point B-spline surface from the mapped control points $\tilde{P}_{i,j}$.

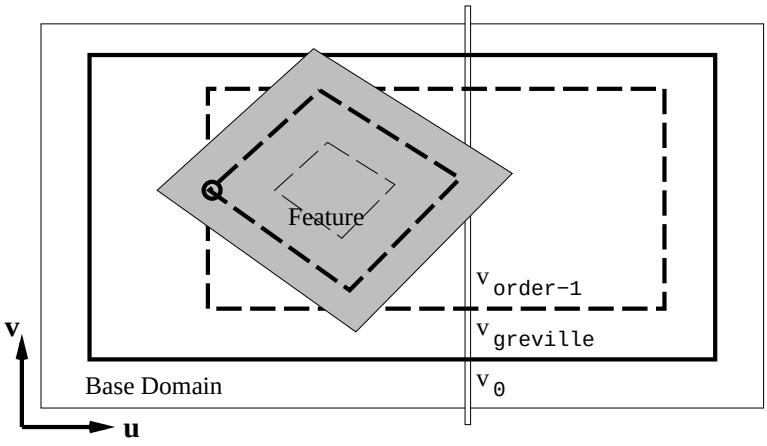

Figure 2.3: Domain classification.

Recall that mapping a feature CV first calls for an evaluation of the base surface at a point whose image in the feature domain is the Greville abscissa. To guarantee a successful evaluation, the mapped feature Greville must lie inside the complete base domain D (Equation 2.6) and fit inside the part of the base domain over which

the base surface is defined (Equation 2.7). Thus, the required range of a feature Greville is:

$$\begin{cases} \frac{u_1+\ldots+u_{k-1}}{k-1} \geq u_{k-1} & \text{and} \quad \frac{u_{m+1}+\ldots+u_{m+k-1}}{k-1} \leq u_{m+1} \\ \frac{v_1+\ldots+v_{\ell-1}}{\ell-1} \geq v_{\ell-1} & \text{and} \quad \frac{v_{n+1}+\ldots+v_{n+\ell-1}}{\ell-1} \leq v_{n+1} \end{cases}$$

The complete base domain and the valid domain coincide for Bezier bases, but are not identical for NUB or NURB bases. Figure 2.3 shows the domain relationships for NUB bases and the containment conditions a feature domain must satisfy to ensure successful mapping. The figure shows a hashed feature domain lying atop the white base domain, with its dotted Greville domain barely inside the base's dotted valid domain.

If all above conditions are satisfied, pasting proceeds as follows:

Algorithm 2.2 The Basic Pasting Procedure

given the feature's Greville point matrix Γ *and displacement matrix* $\mathbf{d}$

for each $\Gamma_{i,j} = (\gamma_i, \gamma_j, 0)$ *do*

let (u,v) = *image of point* $(\gamma_i, \gamma_j) \in D_F$ *in domain* D_B

evaluate S_B *at (u,v) to obtain base image* $\mathrm{S}_B(u,v)$

build local frame $\mathbb{F}_B = \{\mathrm{S}_B(u,v), \mathbf{v}(\mathbf{i},\mathbf{j},\mathbf{k})\}$

let $\tilde{\mathrm{P}}_{i,j} = \mathrm{S}_B(u,v) + \mathbf{d}_{i,j}$ *in the context of* $\mathbb{F}_B$

end

end

The algorithm is illustrated in Figure 2.4, which simplifies the example by varying parameter u and keeping v constant. The dotted line links the base domain point (u,v), to its image $\mathrm{S}_B(u,v)$ on the base surface. Point (u,v) represents the

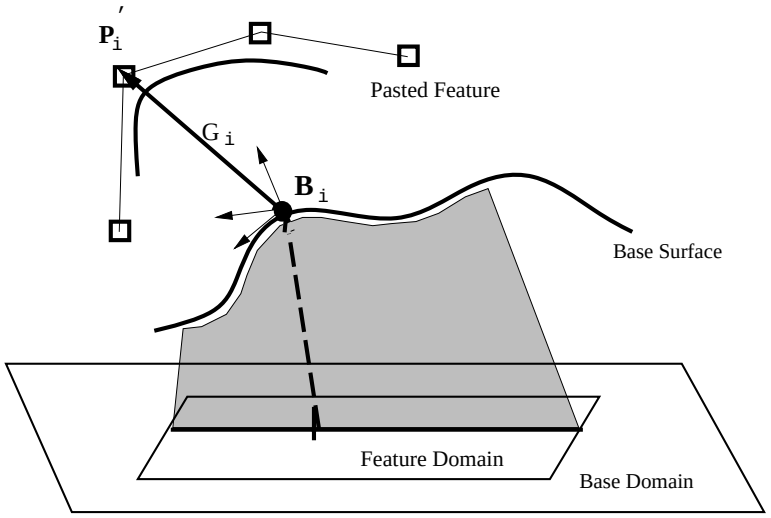

Figure 2.4: Domain and surface mapping.

image of feature Greville $\Gamma_{i,j}$ on the base domain. The pasted control point $\tilde{\mathrm{P}}_{i,j}$ results from applying the Greville displacement $\mathbf{d}_{i,j}$ to its mapped origin $\mathrm{S}_B(u,v)$ in the local frame determined by $\mathrm{S}_B(u,v)$ and $\mathbf{v}$.

If all CVs are mapped in a similar fashion, the end-result is a *free standing* surface that reflects the topography of the underlying base while still preserving some of its original contour traits. If a base surface point changes, and the point has a domain image that overlaps one of the feature's domain pre-images, the mapped feature will also change shape. Should more features be pasted recursively on one or more bases, then every point of the composite will lie in some hierarchy of domains, and the changes operated at a base level will propagate through the hierarchy to affect the features above, in the order in which they were pasted. This will be made clearer in Algorithm 3.5.

As long as the dynamics of the pasted composition does not affect the domain dependencies between bases and features, the procedure described in Algorithm 2.2

can be simplified by eliminating the remapping of the feature Greville points:

Algorithm 2.3 The CV Pasting Procedure

given the feature's mapped Greville point matrix $\widetilde{\Gamma}$
and Greville displacement matrix $\mathbf{d}$
for each $\widetilde{\Gamma}_{i,j} = (u, v, 0)$ *do*
evaluate S_B *at (u,v) to obtain base image* $\mathrm{S}_B(u, v)$
build local frame $\mathbb{F}_B = \{\mathrm{S}_B(u, v), \mathbf{v}(\mathbf{i}, \mathbf{j}, \mathbf{k})\}$
let $\widetilde{\mathrm{P}}_{i,j} = \mathrm{S}_B(u, v) + \mathbf{d}_{i,j}$ *in the context of* $\mathbb{F}_B$
end
end

2.4 Algorithm Evaluation

Interdependencies among feature and base surface domains can be traced throughout the pasted hierarchy, and can be used to detect the surface that ultimately represents the shape of the compound at a given point. An obvious example is the evaluation of a pasted composition at a point specified in the base domain, as shown in Figure 2.5. The spikes that pierce the overlay domains indicate what domains are encountered along the search path, and what domain point the respective evaluation would occur at.

The figure also points out a possible problem that may arise when the evaluation point is very close to the edge of an overlapping domain. The dotted circle, for example, denotes a point probably shared by base domain D_1 and feature domain D_2. If the two surfaces are not C^0 continuous (ie. do not touch) at the common domain

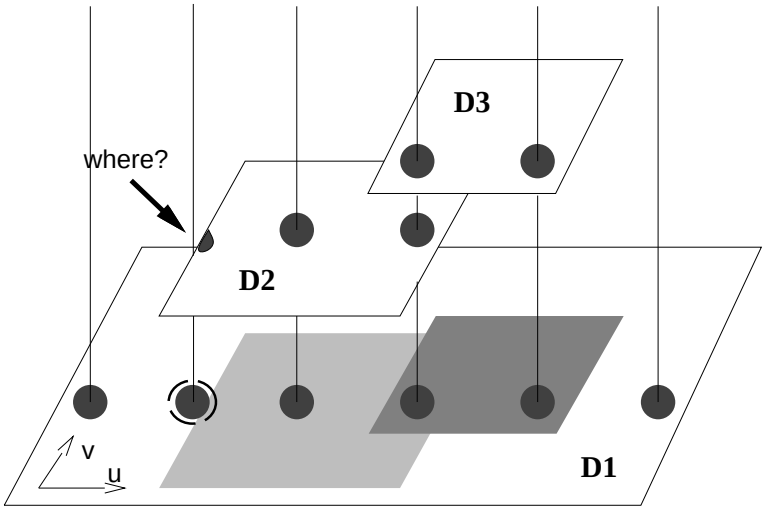

Figure 2.5: Grid evaluation on hierarchical domains.

point, an arbitrary decision on which surface to evaluate might yield unpleasant or undesired visual results. Irrespective of the choice of surface, the discontinuity cannot be hidden; it can only be made less obvious. Such cases are infrequent, though, since the most common goal is to "grow" a seamless detail from a bare base surface. To minimize the gap between feature and base along the junction points, the Greville displacements along the feature's edges should have a zero z component. This requirement is easily accomplished with *end-point interpolating* tensor product surfaces, whose only condition on the edge points is to be equal in z. Since the two-dimensional Greville points are embedded in the 3D space at an arbitrary z value, one can conveniently use the minimum of all z values in the control mesh for this purpose.

It should be emphasized that the mapped surfaces are only approximations of the true displacement surface. Therefore, even if invisible to the naked eye, seams might still exist. The accuracy of the approximation depends on five factors: the feature and base domain alignment, the level of the feature and base refinement,

the curvature of the base surface, the displacement scheme, and the precision of the difference quotient evaluation (Equation 2.2). Regardless of the value of these parameters, the boundary of the pasted feature is merely a close approximation of the surface it rests on, and will carry a small gap factor as long as its domain is not perfectly aligned with the underlying domain. If the two domains are equiparametric, irregularities may still occur, especially if the feature surface is unsufficiently refined. Plates 3 and 4 illustrate this case: in Plate 4, the red feature surface pierces the blue surface on which it is pasted, but no longer does so after refinement (Plate 3).

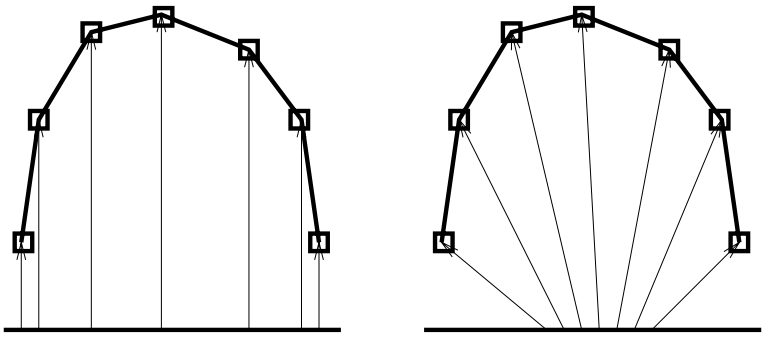

Figure 2.6: Examples of displacement schemes.

The displacement scheme is another determining factor of the pasting quality. We have already decided that representing the feature surface in terms of its Greville displacements is likely to yield satisfactory results. Depending on the size of the domain and the placement of the control points, the resulting displacements may be more or less aligned, as shown in Figure 2.6. The picture on the left depicts a scheme in which all x and y displacements are zero; the figure on the right demonstrates a more de-aligned scheme. Neither representation is perfect, and neither will perform equally well or badly under a non-planar base surface topography. However, experiments have shown that displacements with a zero tangential com-

ponent are overall more intuitive and better shape-preserving representations than others. An example of a more extreme case is illustrated in Plate 6, in which the green surface is pasted onto the purple one, but intersects it rather violently due to its negative displacement vectors.

These and other properties of the pasting operation can be turned into useful features in a well designed application. The multiple uses of this procedure and the details of hierarchical pasting are presented in the chapters to come.

Chapter 3

Changing the Hierarchy

The previous chapter has presented the pasting procedure as a mapping of features onto a *known* set of bases. Rarely, however, is the structure of the composition rigid or known a priori; surface modeling requires repeated morphological changes and stresses the need for flexible data structures. This chapter discusses the hierarchical aspects of the pasting mechanism, geared towards an interactive approach to modeling. First, the chapter presents a two-dimensional clipping algorithm based on barycentric coordinates, suited to the shape of the surface domains. Then, it describes the pasting data structures and their basic operations.

3.1 Modeling Through Interactive Pasting

Pasted compositions come alive through changes applied to their control vertices and domains. As far as pasting is concerned, control vertex editing is synonymous with surface remapping on unchanged base domains, as described in Chapter 2. Domain transformations, on the other hand, modify the composition layout and

require a complete domain *and* surface remapping.

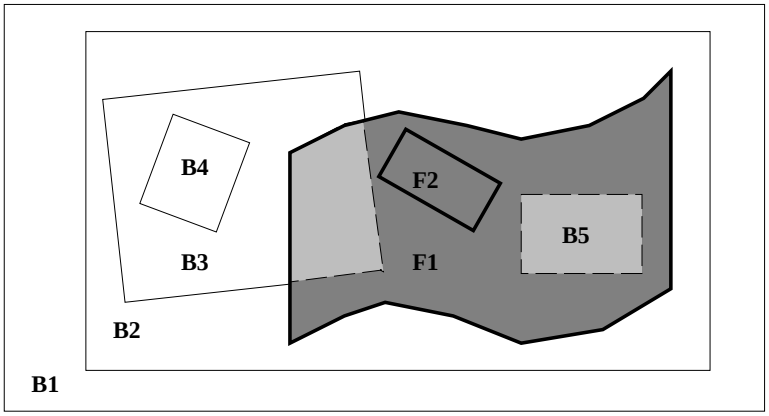

Figure 3.1: Overlapping domains: the polygonal view.

The structure of a pasted hierarchy changes due to domain transformations, new pastings, and pasting removals (or unpastings), all of which are known as *structural modifications*. These operations involve two entities, each consisting of any number of pasted surfaces that form a *base composition* or a *feature composition* respectively. In Figure 3.1, for example, the base composition is $\mathcal{B} = \{B1, B2, B3, B4, B5\}$; similarly, the feature composition is $\mathcal{F} = \{F1, F2\}$. The components of each set are listed in *pasted order*[1]. Domain $B1$ is the *root* of the composition, since it is not pasted onto any domain and is sufficiently large to accommodate all its pasted details.

In this example, the feature domains and the base domains are strictly nested before the pasting of $\mathcal{F}$ onto $\mathcal{B}$. The domain dependencies established thereafter are illustrated in Figure 3.2, where the nodes of the graph represent domains, while edges indicate portions of domains that overlap. Although both $B5$ and $F2$ are subdomains of $F1$, they lie on opposite sides: $B5$, which is a base component, lies

[1]The order in which each was pasted onto its respective base domain(s)

under $F1$, while feature $F2$ is pasted *on* $F1$.

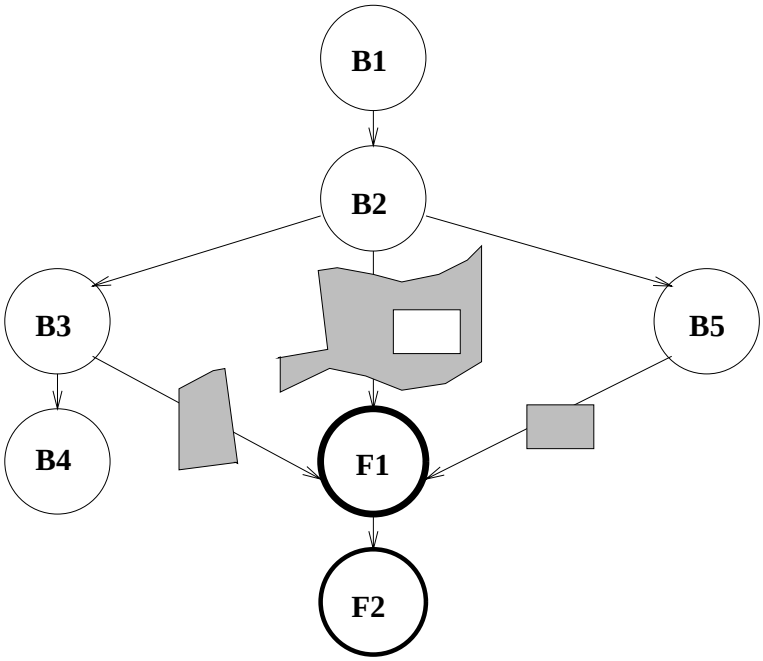

Figure 3.2: Overlapping domains: the structural view.

When the structure of a composition changes, two steps precede the surface mapping operation:

1. recomputation of the set of bases;
2. reassociation of each Greville point on the feature domain with the directly underlying base.

Step 1 determines the **structural base coverage (SBC)**, defined as the set of base domains the root feature touches directly. In Figure 3.1, $SBC_{\mathcal{F}} = \{B2, B3, B5\}$, where $F1$ is the root. A different view of $SBC_{\mathcal{F}}$ is given in Figure 3.2, which represents the structural base coverage as a set of edges incident to

$F1$. The SBC reflects domain dependencies, but provides no information on the size or shape of the exact overlap areas.

The **polygonal base coverage(PBC)** is a prerequisite for Step 2. Given the root domain F_{root} of a feature composition and its structural base coverage $SBC_{\mathcal{F}} = \{B_1, \ldots, B_n\}$, the **polygonal feature residual (PFR)** is defined as a set of polygonal shapes:

$$PFR_{\mathcal{F}} = \{R_1, \ldots, R_m\},\ R_{1\ldots m} \neq 0,\ m \leq n \tag{3.1}$$

where *residuals* $R_{1\ldots m}$ are defined as:

$$R_1 = F_{root} - B_n \tag{3.2}$$

$$R_i = R_{i-1} - B_{n-i+1},\ 2 \leq i \leq m \tag{3.3}$$

Then the polygonal base coverage is the set of polygonal fragments as follows:

$$PBC_{\mathcal{F}} = \{D_1, \ldots, D_m\},\ D_{1\ldots m} \neq 0,\ m \leq n \tag{3.4}$$

where

$$D_1 = B_n \cap F_{root} \tag{3.5}$$

$$D_i = B_{n-i+1} \cap R_{i-1},\ 2 \leq i \leq m \tag{3.6}$$

In Figure 3.2, the incident edges of node $F1$ show the $PBC_{\mathcal{F}}$ polygons computed in the order $B5, B3, B2$. At each step, the residual is intersected with the base, then updated through a *difference* operation. An efficient solution for computing the base coverage is presented in the next section.

3.2 Determining The Base Coverage

The polygonal base coverage definition suggests a straightforward algorithm to compute the set of bases and the feature fragments that intersect them. The algorithm requires two rather expensive geometric operations between planar polygons – *differencing* and *clipping* – which help determine the SBC and the PBC respectively. The method presented below finds the SBC without clipping, and focuses on finding only the *structural* base coverage[2], whose components completely determine the graph dependencies.

Let F_{root} be the root feature of composition F, and $\mathcal{B} = \{B_1, \ldots, B_n\}$ the domains of the base composition, stored in pasted order. We denote the residual by R. Then:

Algorithm 3.1 Finding the Structural Base Coverage

let $R = F - B_n$
if $R = 0$ *add* B_n *to* $SBC_{\mathcal{F}}$, *and exit*
for each B_i, $n - 1 \geq i \geq 2$ *do*
    let $R_{temp} = R - B_i$
    $R_{temp} \neq R$ *then*
        add B_i *to* $SBC_{\mathcal{F}}$
        if $R_{temp} = 0$ *exit*
        let $R = R_{temp}$
    end
end

[2]For the complete algorithm, see the PBC definition (Equation 3.1– 3.6)

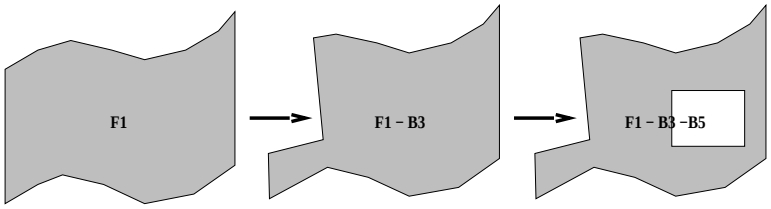

Figure 3.3: The steps of the *Difference* operation.

Figure 3.3 shows how the algorithm is applied to feature $F1$ (from the earlier Figure 3.1 example). The steps highlight the result of the successive differences between F and $\{B3, B5\}$.

The algorithm is linear in the number of base domains, and can be optimized for efficiency with *bounding-box* tests for intersection. The expensive operation is that of computing the difference between two non self-intersecting polygons which can be concave, hole-filled, and fragmented as illustrated in Figure 3.3.

3.2.1 Domain Clipping

Polygonal difference, intersection, and union are boolean operations frequently used to determine visible surfaces, produce high-quality surface details, or distribute scene objects to appropriate processors in parallel ray tracing systems [FvDFJ90]. Generally referred to as *clipping* operations (by the name of the intersection operation), they have spawned two classes of algorithm design based on the precision of the computations:

Image Precision methods operate at the resolution of the display, and determine the visibility of each pixel. A typical example is the z-buffer algorithm [Cat74], which does not require any intersection calculation, but uses a large amount of storage and records only the surface fragments with the closest depth value. Another

algorithm, created by Warnock [War69], subdivides areas into four equal squares recursively, according to the amount of local detail. Warnock's method does not completely eliminate the line intersection computations, and has a poor worst-case behavior.

Object Precision algorithms are display independent, and have a precision equal to that of the objects they operate on. They produce accurate results and use structures easy to traverse after the initial setup overhead, but require extensive intersection computations. The depth-sort algorithm, Binary Space-Partitioning Trees (BSP), and Weiler-Atherton's algorithm [NNT72, FAG83, WA77] are among the most efficient. Another category of object-precision algorithms, called *scan-line* methods, apply scan conversion to determine the intersection, the union, or the difference of arbitrary polygons. A recent improvement belongs to Vatti [Vat92], whose *scan beam* algorithm replaces the unit increments with segment increments equal to the vertical distance between two successive line intersections.

Regardless of the strengths of the algorithm, clipping arbitrary polygons is a complex task. The current solutions can be fast when limited to specific polygonal types (such as rectangles, or strictly convex shapes); otherwise, as is the case with most object precision algorithms, they have expensive overheads. Of all the methods listed above, the BSP trees and the scan-beam algorithm are best suited to computing the base coverage of pasted compositions. Both approaches operate on complex polygons in a $2\frac{1}{2}$-dimensional space, and both yield accurate clippings. Moreover, they construct a spatial graph that lends itself to subsequent boolean computations. However, the overhead of building the graph restricts the effectiveness of these algorithms to static models. The interactive approach promoted by the pasting procedure, involving frequent domain shuffles, requires a fast clipping algorithm with no overhead or precision penalty. While successive clipping of a fea-

ture domain against its bases produces complex polygonal fragments, the original shape of the polygon is a rectangle. The algorithm presented next takes advantage of the simplicity of the initial domain geometry. Even if affinely transformed, a rectangle can be partitioned easily into two or more triangles, as illustrated in Figure 3.4.

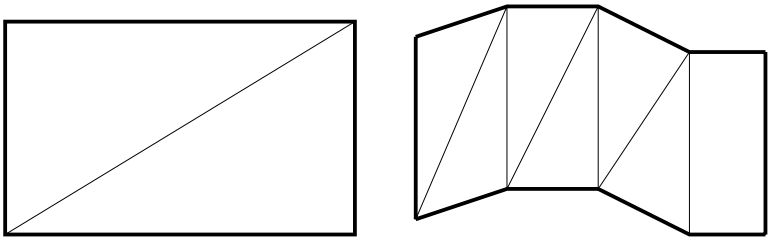

Figure 3.4: Domain discretization and triangulation.

Once represented as a set of adjacent triangles, the polygon delegates the clipping task to its components. Although a boolean operation between p polygons of n triangles each, has complexity $\mathcal{O}(pn)$, the method is feasible because triangles are simple geometric primitives offering a low-cost, object-precision clipping advantage.

The difference operation between two triangulated polygons is a new set of triangles whose cumulative contour is not relevant to the polygonal base coverage, and therefore need not be computed. The difference algorithm produces a set of disjoint and possibly adjacent triangles, as follows:

Algorithm 3.2 Computing the Difference of Two Polygons, $P1 - P2$

for each triangle $T \in P1$ *do*

let residue $R = T$

for each triangle $Q \in P2$ *and while* $R \neq 0$ *do*

let $R = R - Q$ *(possibly following a bounding-box test)*

end

if $R \neq 0$ *add* R *to difference*

end

The problem of finding the geometric difference between two domains reduces to computing the difference between two triangles that intersect at arbitrary angles. A solution based on barycentric coordinates is discussed briefly in the next section.

3.2.2 Triangle Geometry

Consider a triangle with vertices A, B, C and a coplanar point P, as in Figure 3.5. P can be represented in terms of the triangle as a *barycentric combination* of the three vertices:

$$\mathrm{P} = \alpha_{\mathrm{A}}\mathrm{A} + \alpha_{\mathrm{B}}\mathrm{B} + \alpha_{\mathrm{C}}\mathrm{C} \tag{3.7}$$

where α_{A}, α_{B}, and α_{C} are the *barycentric coordinates* of P with respect to A, B, C, and satisfy the condition:

$$\alpha_{\mathrm{A}} + \alpha_{\mathrm{B}} + \alpha_{\mathrm{C}} = 1,\ \alpha_{\mathrm{A}}, \alpha_{\mathrm{B}}, \alpha_{\mathrm{C}} \in R \tag{3.8}$$

Equations 3.7 and 3.8 form a linear system of equations in α_{A}, α_{B}, and α_{C}, with solutions:

$$\alpha_{\mathrm{A}} = \frac{\begin{vmatrix} \mathrm{P}_x & \mathrm{B}_x & \mathrm{C}_x \\ \mathrm{P}_y & \mathrm{B}_y & \mathrm{C}_y \\ 1 & 1 & 1 \end{vmatrix}}{\frac{1}{2}\begin{vmatrix} \mathrm{A}_x & \mathrm{B}_x & \mathrm{C}_x \\ \mathrm{A}_y & \mathrm{B}_y & \mathrm{C}_y \\ 1 & 1 & 1 \end{vmatrix}},\ \alpha_{\mathrm{B}} = \frac{\begin{vmatrix} \mathrm{A}_x & \mathrm{P}_x & \mathrm{C}_x \\ \mathrm{A}_y & \mathrm{P}_y & \mathrm{C}_y \\ 1 & 1 & 1 \end{vmatrix}}{\frac{1}{2}\begin{vmatrix} \mathrm{A}_x & \mathrm{B}_x & \mathrm{C}_x \\ \mathrm{A}_y & \mathrm{B}_y & \mathrm{C}_y \\ 1 & 1 & 1 \end{vmatrix}},\ \alpha_{\mathrm{C}} = \frac{\begin{vmatrix} \mathrm{A}_x & \mathrm{B}_x & \mathrm{P}_x \\ \mathrm{A}_y & \mathrm{B}_y & \mathrm{P}_y \\ 1 & 1 & 1 \end{vmatrix}}{\frac{1}{2}\begin{vmatrix} \mathrm{A}_x & \mathrm{B}_x & \mathrm{C}_x \\ \mathrm{A}_y & \mathrm{B}_y & \mathrm{C}_y \\ 1 & 1 & 1 \end{vmatrix}}$$

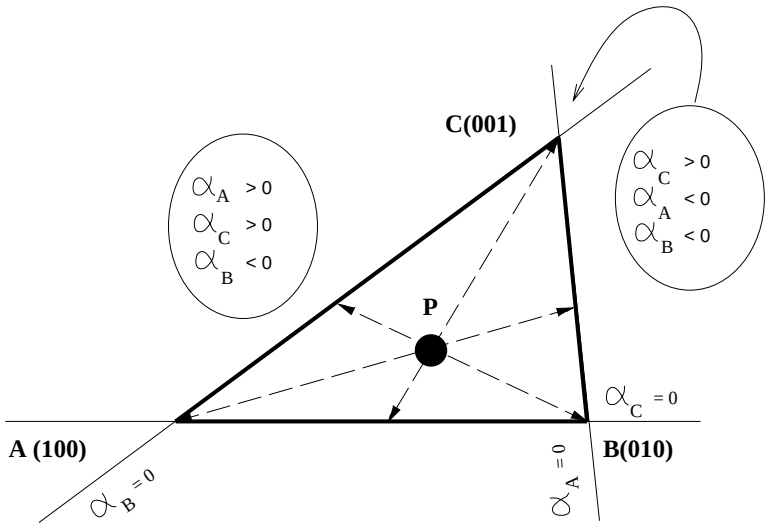

Figure 3.5: Triangle representation in barycentric coordinates.

At each vertex the corresponding α is 1, while the other two α's are 0 (Figure 3.5). Along the axes defined by the triangle edges, the barycentric coordinate of the facing vertex is 0, indicating that the vertex does not contribute to the position of any point on that axis. On each side of the axis, α is positive in the half-plane containing the vertex (towards the *inside* of the triangle), and negative in the *outer* side. A point P is inside the triangle if $\alpha_{A,B,C} > 0$.

As Figure 3.5 illustrates, the alpha values partition the barycentric space into two areas:

- the region *within the edges of the triangle*, where $0 \leq \alpha_{A,B,C} \leq 1$;
- the region *outside the triangle*, where at least one barycentric coordinate lies outside the [0,1] interval. A further subdivision of this region is possible:
  - the *front-face* space, delimited by one "defining" edge and the outward extension of the other two edges. The subspace is said to *face* the vertex

opposite the defining edge. If one endpoint of an arbitrary line segment lies in the *front-face* space, the segment either does not intersect the triangle, or if it does, it must first intersect the defining edge. In this space, two α's are always positive.

- the *back-face* space, delimited by the outward extension of two "defining" edges. The subspace lies at the *back* of the vertex where the defining edges intersect. If one endpoint of an arbitrary line segment lies in the *back-face* space, the segment either does not intersect the triangle, or if it does, it first intersects one of the two defining edges. In this space, two α's are always negative.

Figure 3.5 shows the signs of the barycentric coordinates of points lying in the *front-face* of vertex B, and in the *back-face* of vertex C respectively.

It is possible to construct a function that maps α values to barycentric regions, and stores the information in a 6-bit word[3]:

$\alpha_A > 0$	$\alpha_A = 0$	$\alpha_B > 0$	$\alpha_B = 0$	$\alpha_C > 0$	$\alpha_C = 0$

A case-driven algorithm can be developed on this basis, taking into account the alpha regions as well as the configuration of the two triangles. Figure 3.6 gives an example of every configuration – one, two, three vertices, or no vertex inside the base triangle – and shows the result of the difference operation as shaded polygons. Similar principles apply to the union and intersection of two triangles.

To compute the contours generated by a boolean operation, the algorithm must find the points where the triangles intersect. Given a pair of coplanar segments |AB| and |UV|, the point of intersection K results from a system of parametric line equations with unknowns t and τ:

[3]Three bits would not permit the detection of all three α placements: $\alpha > 0$, $\alpha = 0$, and $\alpha < 0$.

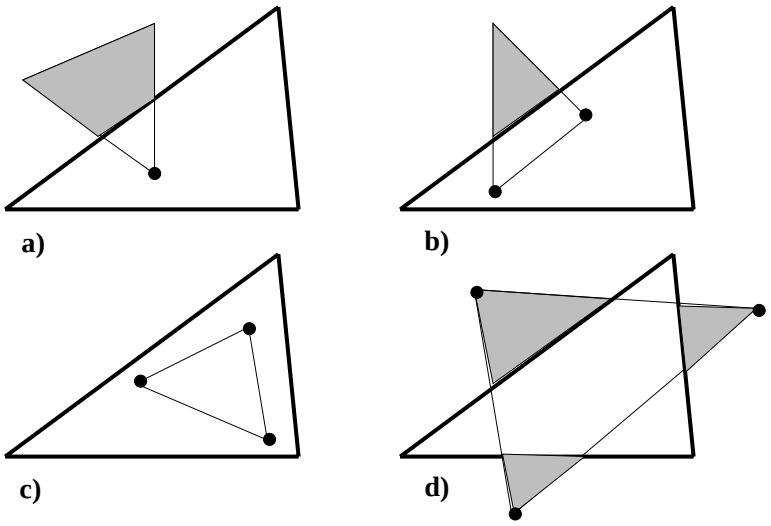

Figure 3.6: The four main clipping cases.

$$\begin{cases} \mathrm{K}(t) = (1-t)\mathrm{A} + t\mathrm{B} \\ \mathrm{K}(\tau) = (1-\tau)\mathrm{U} + \tau\mathrm{V} \end{cases} \tag{3.9}$$

Point K is a valid intersection if $\begin{cases} 0 \leq t \leq 1 \\ 0 \leq \tau \leq 1 \end{cases}$.

Algorithm 3.2 performs well on the average case, and can be optimized to bypass non-intersecting triangles and produce better triangulations of the clipped areas (to avoid slivers). As the areas become smaller, fewer triangles overlap, and the method terminates quickly. This approach exploits *area coherence*, a property of shapes that cover significant portions of the underlying area, thus requiring few intersections with the neighboring layers. Quick termination and simple computations recommend the case-driven barycentric algorithm as a good choice for interactive pasting.

3.3 Contained Directed Acyclic Graphs

If the structural base coverage of a feature contains more than one domain, the domain dependencies form a directed graph with interconnected nodes (Figure 3.2). Rather than being strictly nested as in Forsey's approach [For90], domains overlap *arbitrarily.* Theoretically, loops generated by partially overlapping entities do not increase the complexity of the model, as long as the areas of intersection (the polygonal base coverage) are circuit-free; however, the concern for real-time surface composition renders loops impractical, and shifts the focus to structural base coverage. Therefore, the pasting algorithm does not know the extent to which domains overlap, and cannot allow a feature to become its own base, as Figure 3.7 illustrates.

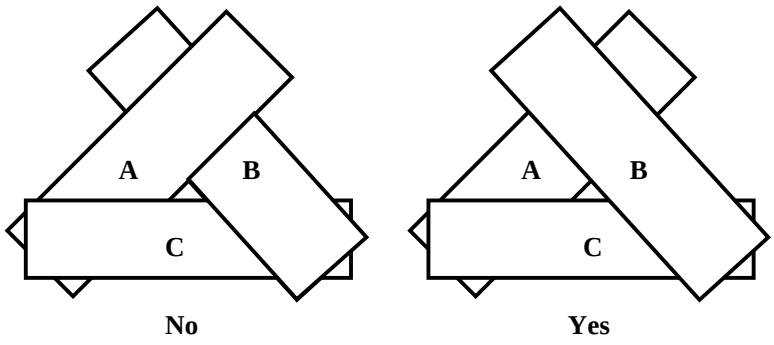

Figure 3.7: Domain overlapping with and without loops.

The first picture shows domains A, B and C combined in a loop. If domain A changes, the mapping process will traverse the graph in the order $A \rightarrow C \rightarrow B \rightarrow A \rightarrow \ldots$ ad infinitum. To avoid an infinite loop, the algorithm requires all domain planes to be strictly layered, as shown in Figure 3.7b. Also, domains are assumed to be non self-intersecting, or if they are, they are not *pasted* onto themselves.

Selective mapping[4] is a complex mechanism that handles loops by storing the *polygonal* base coverage of every feature and eliminating the polygons that intersect. For the sake of simplicity, this section assumes the layout of pasted domains forms a loop-free directed graph, known as Directed Acyclic Graph or DAG[5].

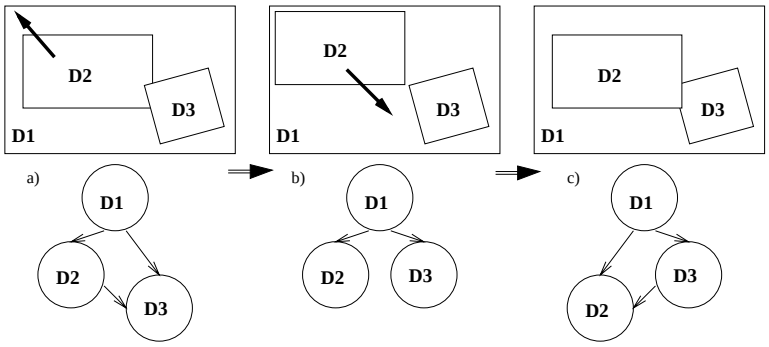

Figure 3.8: Domain shuffling in a DAG.

The solution presented in this section views the model as a system that records only its *current* state. Thus, once a surface slides completely from under its features (Figure 3.8 and Plates 11, 13, and 14), it ceases to determine the shape of the pasted features, and is pasted *on top of* these ex-features if shifted back across the ex-feature domain. This mechanism acts as a safeguard against loop formation and provides an intuitive rule for structural motion.

3.3.1 Definitions

Surface domains are transformed individually or in groups in a manner that preserves the graph acyclicity. This discussion concentrates on groups uniquely determined by one component, known as a *pivotal domain.* A cluster defined by its

[4] See Chapter 5

[5] *Composition DAG* if representing the structure in its entirety.

pivot forms a *pivotal group*, which is intrinsically loop-free and remains so under structural modification.

Three classes of pivotal groupings are common:

- **single grouping** – contains only the pivot itself;
- **exhaustive grouping** – selects the DAG rooted at the pivot;
- **contained grouping** – ignores domains partially overlapping the inclusive domains.

Single and exhaustive groupings are special cases of contained grouping. The latter constructs a **contained DAG**, which is a subset of the composition DAG with the property that all its domains (i.e. nodes) lie completely inside the subset's root domain. In Figures 3.2 and 3.1, for example, $F1$ is an element of the contained DAG whose root is $B2$, since $F1$ and all its descendents – as well as $B3$, $B5$, and $B2$ itself – are inside $B2$.

The contained DAG notion is a component of the **containment paradigm** formally introduced below.

Definition 3.1 *Given a directed acyclic graph G(V,E), nodes* $s, t, x_1 \ldots x_n \in V$, *and a directed node-path* $\mathcal{P}(s,t)$ *from* s *to* t, $\mathcal{P}(s,t) = (s, x_1, \ldots, x_n, t)$, *a* **reverse node-path** *is* $\mathcal{P}^{-}(s,t) = (t, x_n, \ldots, x_1, s)$.

The reverse node-path is an ordered set that results from traversing a trajectory from t to s in the direction opposite to the one following the sense of the edges. Depending on the connectivity of the graph, there may be several directed (or reverse) paths between s and t. In Figure 3.9, which is a rendition of Figures 3.2 and 3.1

with additional visual cues, there are three reverse paths between nodes $B1$ and $F1$: $\mathcal{P}_1^- = (F1, B3, B2, B1)$; $\mathcal{P}_2^- = (F1, B2, B1)$; and $\mathcal{P}_3^- = (F1, B5, B2, B1)$. Notice that nodes $B1$, $B2$, $F1$ appear in all three sets.

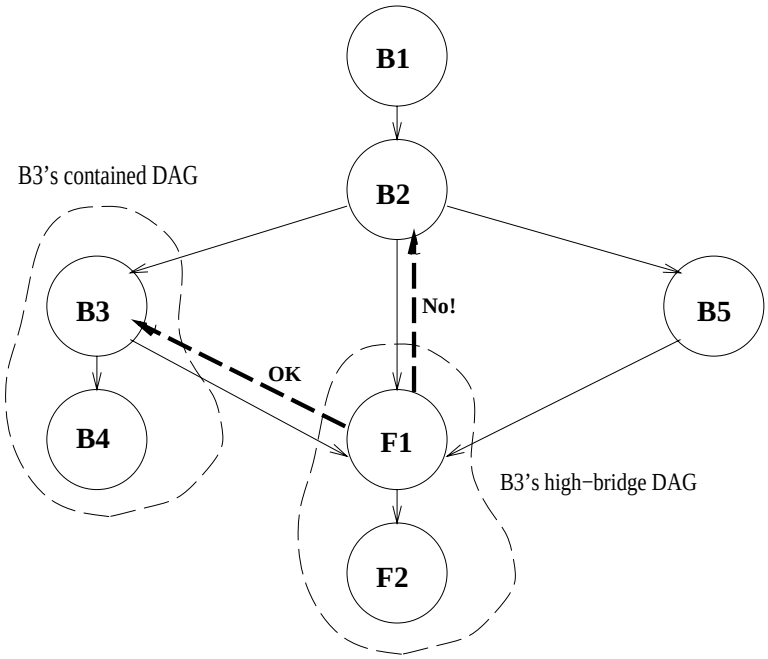

Figure 3.9: Contained DAG and its high-bridges.

Definition 3.2 *Given a directed acyclic graph G(V,E) rooted at $r \in V$, two nodes $s, t \in V$, and a directed node-path $\mathcal{P}(r,t)$, node t is* **contained** *relative to s iff $s \in \bigcap_1^n \mathcal{P}_i^-(r,t)$, where n is the size of the complete set of reverse paths.*

A node t is contained relative to another node s if any reverse path from t to the root of the DAG includes s. For the DAG illustrated in Figure 3.9, node $F1$ is contained relative to $B2$ and to root $B1$. Notice that both $B1$ and $B2$ appeared in the result of previous example as nodes common to all reverse paths from $B1$ to $F1$.

Definition 3.3 *Given a directed acyclic graph G(V,E) rooted at $r \in V$, and a node $s \in V$, the* **contained DAG** *rooted at s is $G_s^{cont}(V', E')$, where $V' \subseteq V$ is the set of all the nodes contained relative to s, and $E' \subseteq E$ is the set of edges whose endpoints are in V'.*

If we use the same example, the contained DAG rooted at $B3$ consists of only two nodes, $B3$ and $B4$, because no other node except $B4$ contains $B3$ in all its reverse paths; indeed, $F1 \notin G_{B3}^{cont}$ because $B3 \notin (F1, B2, B1)$. The only edge of G_{B3}^{cont} is the one that links $B3$ and $B4$. The contained DAG may or may not coincide with the subgraph rooted at the given node. In Figure 3.11b, for instance, G_{D2}^{cont} contains only $D2$ because neither child is spawned exclusively by $D2$.

Definition 3.4 *Given a directed acyclic graph G(V,E) rooted at $r \in V$, and a node set $W \subseteq V$, the portion of the node path shared by all $w \in W$ is an ordered set defined as $\mathcal{P}_\Omega(G, W) = \bigcap_i \mathcal{P}_i(r, w)$. The* **minimum container** Ω *of all $w \in W$ is the last node of $\mathcal{P}_\Omega(G, W)$.*

In Figure 3.9, the path shared by $B3$, $F1$, and $B5$ is $\mathcal{P}_\Omega = (B1, B2)$. $B2$ is the minimum container, ie. the node with the shortest path to all the elements in W. If W represents the structural base coverage of a feature, then the minimum container is the topmost domain that fully contains the feature *and* the base coverage domains. Thus, if we were about to add $F1$'s composition to the DAG rooted at $B1$, we would identify $\{B3, B2, B5\}$ as its structural coverage and $B2$ as the *closest* node to all three. Since $B2$ is the closest node to $F1$'s base coverage, it also represents the root of the *smallest* contained DAG that includes $F1$.

$F1$ happens to touch $B2$ directly, which explains why $B2$ is both a component of the structural coverage set *and* the minimum container. In Figure 3.11b, however,

$D4$'s structural coverage is $\{D2, D3\}$, while the minimum container is $D1$. To simplify the notation, we will refer to $\Omega(G, W)$ as Ω_W whenever G is implicit.

Definition 3.5 *Given a directed acyclic graph G(V,E) and a contained DAG denoted by* $G_s^{cont}(V', E')$, *with* $s \in V'$, $V' \subseteq V$ *and* $E' \subseteq E$, *the* **cutset** *of* $G_s^{cont}(V', E')$ *is* $E^{cut} = E_{low}^{cut} \bigcup E_{high}^{cut}$, *where:*

- E_{low}^{cut} *is the* **low cutset** *of the contained DAG, and is equivalent to the set of edges incident to node s.*
- *the* **high cutset**, E_{high}^{cut}, *is the set of edges whose tail node is in* V', *and whose head node is in* $V - V'$.

Definition 3.6 *The* **bridge nodes** *of a contained DAG* $G_s^{cont}(V', E')$ *are the set* $V^{cut} = V_{low}^{cut} \bigcup V_{high}^{cut}$, *where:*

- *known as the* **low-bridge node set**, V_{low}^{cut} *consists of the tail nodes of* E_{low}^{cut}.
- V_{high}^{cut} *represents the* **high-bridge node set**, *and consists of the head nodes of* E_{high}^{cut}.

The high-bridge nodes, or **high-bridges** for short, are the domains that partially overlap the contained DAG. If the contained DAG reduces to one domain as for *single-grouping*, the high-bridges are its immediate children in the composition DAG; *exhaustive-grouping* has an empty high-bridge set. High-bridges can have pasted features of their own, in which case it is possible to determine the high-bridge DAG's (G^{high}) as contained DAG's rooted at high-bridge nodes.

As noted earlier, $F1$ is not in G_{B3}^{cont}; however, it overlaps $B3$ partially and qualifies as a high-bridge node. Domains $B2$ and $B3$ forms a high-bridge DAG with respect to G_{B3}^{cont}.

3.3.2 Algorithms

In an interactive environment, the structure of the composition DAG must adapt to structural changes rapidly. Since the contained DAG, the minimum container, and the high-bridges are sensitive to hierarchical modifications, they need to be recomputed efficiently.

The first algorithm determines the contained DAG $G_x^{cont}(V', E')$ from a directed acyclic graph $G(V, E)$.

Algorithm 3.3 Computing the Contained DAG

for each node s of DAG $G(V, E)$ *rooted at r do*
    let $count_s = in\_degree_s$;
end
call **visit** *(G,* G^{cont}*, x);*
 # The recursive **visit** *routine:*
procedure **visit** *(G,* G^{cont}*, v)*
    add v to G^{cont}*'s node collection* V'
    for each child c of v do
        if $count_c = 1$ *then*
            add c's incoming edges to G^{cont}*'s edge collection* E'
            visit(G, G^{cont}*, c)*
        else
            $count_c = count_c - 1$
        end
    end
end

The algorithm visits the graph nodes in a mixed depth-first and breadth-first manner, adding nodes to the contained DAG if they have been reached from *all* their incoming edges. If the domain layout is completely nested, the composition DAG becomes an N-ary tree, and the algorithm collapses into a depth-first search with redundant *count* tests on the nodes.

To determine the high-bridges, it is possible to modify the above algorithm to compute the difference between the set of potential high-bridges and the nodes confirmed to be contained. Algorithm 3.4 determines the high-bridge set V_{high}^{cut} of the contained DAG rooted at node x. Let r be the root of the DAG.

Algorithm 3.4 Computing the High-Bridge Nodes

for each node s of the DAG $G(V, E)$ *rooted at r do*
    let $count_s = in\_degree_s$;
end
let the high-bridge set $V_{high}^{cut} = \{0\}$
call **findHigh**(*G*, V_{high}^{cut}, *x*);
 # The recursive **findHigh** *routine:*
procedure **findHigh**(*G*, V_{high}^{cut}, *v*)
    for each child c of v do
        if $count_c = 1$ *then*
            if $in\_degree_c > 1$ *remove c from* V_{high}^{cut}
            findHigh(*G*, G^{cont}, *c*)
        else
            add c to V_{high}^{cut}
            $count_c = count_c - 1$
        end

end

end

The removal operation in **findHigh** is particularly expensive on graphs with high in-degree nodes. If the contained DAG is already known, the bridge finding algorithm can be replaced by a linear traversal of the contained nodes, aimed at identifying the *high cutset* of the contained DAG. Then the high-bridges are given by the end-nodes of the high cutset edges. In our implementation we have used this short-cut with significant improvement in performance.

As the composition DAG increases in density, the search required by the two algorithms discussed here becomes more expensive. It is possible, however, to perform the search faster on a tree representation of the composition, one that records the coordinate frame dependencies and offers easy access to the roots of contained DAG's. This data structure is known as a *correction frame tree.*

3.4 Correction Frame Trees

Each surface domain is defined in a two-dimensional coordinate frame of given origin and orientation. The shape, position, and orientation of the domain are expressed in terms of a transformation relative to this frame. If the origin of the frame is $O_D(10,0)$ in world coordinates, a domain displaced by, say, $T(2,7)$ relative to O_D is in fact at $P(12,7)$ in the world frame. In frame $O'_D(5,4)$, the same transformation would yield $P'(17,11)$ in world space. The frame in which the domain exists is called the *parent frame*; the transformation relative to the frame is known as the *to-parent transformation.*

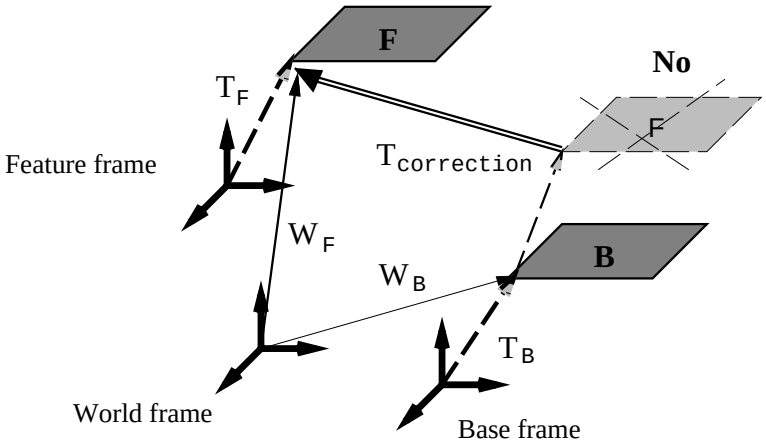

Figure 3.10: Correction frame diagram.

When pasted onto a base domain, the feature domain "attaches" itself to the base by placing its frame in the domain space of the base. The operation replaces the feature's *parent frame* with the *base frame*. But, as the above example demonstrates, if the to-parent transformation is not zero, a straightforward change of frame would cause the feature to "leap" towards another point in the common reference frame. The effect of this unexpected jolt is unintuitive and undesirable in the interactive modeling context. The point is illustrated in Figure 3.10, where the base transformation T_B is composed with the feature transformation T_F to yield the feature placement in base space (light hash). To keep the feature's original position (dark hash) unchanged, it is necessary to compute a **correction transformation** $T_{correction}$ to neutralize the effect of the frame switch.

Both feature and base frames refer to the *world space*, a common frame chosen for convenience, their *to-world transformations* being W_F and W_B respectively. Then, the correction needed to maintain the feature's constant position and orientation relative to the reference frame is the inverse of the base's to-world transfor-

mation:

$$T_{correction} = W_B^{-1} \tag{3.10}$$

A closer analysis of the correction mechanism indicates that $T_{correction}$ must be *stored* when the feature is pasted, but need not be *applied* then, because the feature's to-world transformation (W_F) is still valid. W_F is invalidated when the base's transformation, T_B, changes, in which case W_B is updated to W'_B and the feature's to-world transformation becomes:

$$W_F = T_F T_{correction} W'_B,\ W'_B = f(T_B) \tag{3.11}$$

Equation 3.11 makes the convenient simplifying assumption that the parent and world frames of an unpasted domain D coincide: $T_D \equiv W_D$.

If the base transformation T_B were constant, then by Equations 3.10 and 3.11:

$$W_F = T_F W_B^{-1} W_B = T_F \tag{3.12}$$

Equation 3.12 shows that if the base domain does not change, the pasted feature behaves independently of its parent, as if standing alone. Therefore, when unpasted, the feature must refresh W_F only if the transformations undergone while pasted were credited to its base:

$$W_F = \begin{cases} T_F T_{correction} W_B & \text{if} \quad T_{correction} \neq W_B^{-1} \\ T_F & \text{otherwise} \end{cases}$$

The mechanism introduced so far extends easily to multiple bases, which require that the feature maintain a to-parent transformation for each base frame. The correction and to-parent transformations can be stored in the edges of the composition DAG and updated as the structural layout changes.

The to-world transformations are unique for each domain and therefore stored in the graph nodes. If feature F has n bases, $B_1 \ldots B_n$, it must define n to-parent transformations $T_1 \ldots T_n$ and n correction transformations $T_{correction_1} \ldots T_{correction_n}$ such that:

$$\begin{aligned} W_F &= T_1 T_{correction_1} W_{B_1} \\ &\vdots \\ W_F &= T_n T_{correction_n} W_{B_n} \end{aligned}$$

The resulting – frame-padded – composition DAG provides support for two types of domain grouping:

- **selective feature grouping**, that transforms a *complete* set of base domains and *a subset* of their features. This category is synonymous with **contained grouping**, since all the selected features are inside the given set of base domains.

- **selective base grouping**, that transforms a *subset* of base domains and their nested and/or partially overlapping features. If feature F has n bases out of which the first k of them ($k \leq n$) are in the selected group, and F's domain changes via its k base domains, it must update the remaining $n - k$ to-parent transformations, $T_{k+1} \ldots T_n$. This category is similar to, but more accommodating than, **contained grouping**, because it may include features that only partially overlap the set of selected bases.

A good reference on animating models in coordinate free geometry using frame graphs is given in [Hen93]. To avoid loops in the frame dependency structure, the pasting approach focuses only on selective feature grouping and the associate frame relationships in the composition. If the domains were all nested, the frame graph would be a tree whose edges could be interpreted both as structural base coverage

and frame dependency links. The edge duality is hereafter referred to as *poly* versus *frame* respectively.

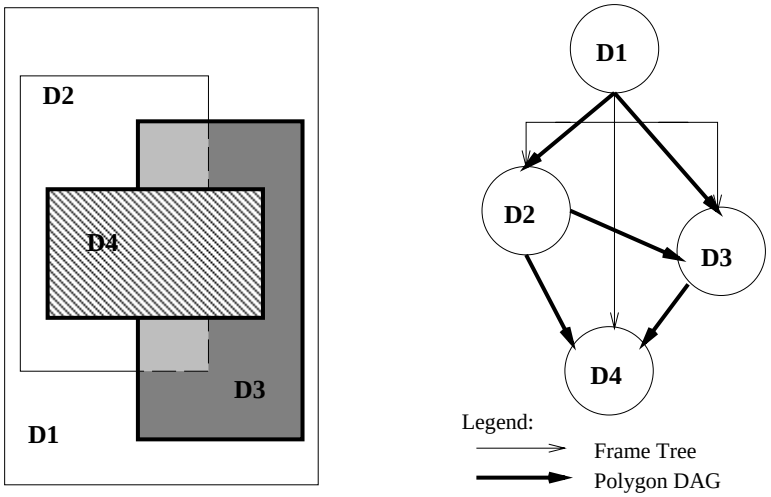

Figure 3.11: The dual frame and polygon graph.

If a feature domain lies over multiple bases, it can be transformed as part of a contained DAG whose root defines the behavior of all its nodes. When pasted onto a base set, the feature depends ultimately on the *shallowest* contained DAG of the base composition; that is, the one having the *shortest* maximal path from root to leaf. The poly edges of the composition DAG will indicate the set of bases it touches directly (as already illustrated in Figure 3.9), while an additional "virtual" edge – a *frame* edge – will bind the feature to its minimum container. The graph of frame edges is called a **correction frame tree**, because all nodes have in-degree *one*, and every node contains a correction transformation relative to its parent. A frame edge does not have a corresponding poly edge unless the end-node lies directly and solely on the tail-node.

Figure 3.11 exemplifies the poly and frame duality on a composition of four domains, $D1, D2, D3, D4$, whose top leaf, $D4$, is a subdomain of root $D1$, but lies

entirely across $D2$ and $D3$. The graph on the right shows the poly edges as thick arrows, and the frame edges as thin arrows. The frame edges link the nodes in a tree-like structure, with one edge per node. The edge between $D1$ and $D4$ is purely virtual because $D1$ does not touch $D4$, yet it is the closest common parent of $D4$'s bases, $D2$ and $D3$.

The frame graph of Figure 3.11 provides further insight into the containment paradigm, a theory whose viability relies on efficient contained DAG and minimum container algorithms. By definition, every node in the CFT is the root of a contained DAG, because each parent node is the minimum container of its children. This implies that the contained nodes and edges of any node x in the structure are given by the subtree rooted at x. This approach requires no tests on the visited nodes, and substitutes the traversal of the possibly dense composition DAG with a search through an N-ary tree.

When a frame moves, the subtree rooted at that frame moves along with it and must have its to-world transformations refreshed recursively. Pasting requires little effort on behalf of the feature composition, since the operation links only the root feature to its minimum container and leaves the other features unchanged frame-wise. Similarly, an unpasted feature composition need only detach the frame edge between the root feature and its minimum container and possibly update its to-world transformation. The complete pasting procedure, involving the contained DAG and the correction frame tree, is assembled into a set of algorithms in the concluding section.

3.5 Pasting: The Complete View

From the static perspective of motionless features, the pasting algorithm is reduced to the domain and surface mapping procedure described in Chapter 2. Once the relationships between overlapping entities are permitted to change, issues related to efficient algorithms and supporting data structures require serious attention. The degrees of freedom allowed for any transformable cluster is limited, in this approach, by the simplifying requirement that the composition be circuit-free. The model records its current state, determined by the node and edge configuration, and ignores the historic sequence. The steps of the dynamic pasting mechanism include graph adjustments and surface/domain mapping. The operations performed on the data structures are centered on the notion of a *contained DAG*, and require a local frame to be associated with each entity. An efficient description of the model represents the composition as a pair of graphs: the composition DAG reflecting structural and possibly polygonal base coverage, as well as the tree of frames and correction transformations.

The algorithms described below take advantage of the dual model representation and minimize the need for graph updates by operating only on the root of the contained DAG where possible; also, they assume that the entities involved satisfy the root-inclusion condition.

Algorithm 3.5 The Pasting Sequence

given feature composition $\mathcal{F}$ *and base composition* $\mathcal{B}$ *do*
find base coverage $SBC_{\mathcal{F}}$ *and min container* $\Omega_{\mathcal{F}}$
for each Greville point P_i *in the domain of root feature* F_0 *of* $\mathcal{F}$ *do*
find domain $B_j \in SBC_{\mathcal{F}}$ *such that* $P_i \in B_j$

end

compute and store F_0*'s correction transformation*

attach feature CFT to base CFT with edge from $\Omega_{\mathcal{F}}$ *to* F_0

insert feature DAG into base DAG with edges from $SBC_{\mathcal{F}}$ *to* F_0

call **The Basic Pasting Procedure**[6] *for* F_0

call **The CV Pasting Procedure**[7] *for all of* F_0*'s descendents*

end

The *unpaste* operation is the reverse of pasting and is applicable to any grouping of pasted entities except the root base. The algorithm listed below operates at the contained DAG level, which consists of at least one entity and a sub-DAG at most, and does not include entities partially overlapping its components. If such entities exist, they must be "lifted" as *high-bridges* before operating on the contained DAG, then "lowered" (i.e. repasted) on the base composition in pasted order.

High-bridge nodes may themselves be roots of contained DAG's that intertwine with other DAG components, and therefore must be lifted as *contained* entities in pasted order. Algorithm 3.6 describes the recursive procedure of lifting the high-bridge set of a contained DAG and returning an ordered set of contained DAGs, denoted by Φ_{high}^{cont}.

Algorithm 3.6 Lifting the High-Bridges

given high-bridge set V_{high}^{cut} *do*

for each $x \in V_{high}^{cut}$ *do (in pasted order)*

[6] Algorithm 2.2

[7] Algorithm 2.3

remove x's incoming edges (remove poly edge)

remove CFT edge between Ω_x *and x (remove frame edge)*

compute high-bridge DAG G_x^{cont} *and its high-bridge set* H_{high}^{cut}

add G_x^{cont} *to* Φ_{high}^{cont}

let $V_{high}^{cut} = V_{high}^{cut} - H_{high}^{cut}$

call **Lifting the High-Bridges** *for* H_{high}^{cut}

end

end

The lifted DAGs have had their cutsets removed, but have not been fully extracted from the composition DAG or from the correction frame tree. When lowered, they only need to reestablish the links with the underlying structure and remap their surfaces to reflect the morphology of their new base coverage:

Algorithm 3.7 Lowering the High-Bridges

given an ordered list of high-bridge DAG's Φ_{high}^{cont}

for each $G_i^{high} \in \Phi_{high}^{cont}$ *do*

apply **The Pasting Sequence** *with no DAG insertion*

end

end

Unpasting derives easily from pasting as a complementary operation. The unpasting algorithm shows the steps of extracting the contained DAG rooted at x from the graph G, and turning it into an independent structure:

Algorithm 3.8 The Unpasting Sequence

given DAG G *and* $G_x^{cont} \subset G$ *do*

let $\Phi_{high}^{cont} =$ **Lifting the High-Bridges** *for* G_x^{cont}

remove graph G_x^{cont} *from* $G \Rightarrow$ *stand-alone feature composition* $\mathcal{F}$

detach x*'s CFT (frame) edge from its minimum container in* G

compute x*'s correction transformation if required*

for each Greville point $\Gamma_{i,j}$ *in* x *do*

associate $\Gamma_{i,j}$ *with* x

end

call **The CV Pasting Procedure** *for all of* $\mathcal{F}$*'s surfaces*

call **Lowering the High-Bridges** *for* Φ_{high}^{cont} *onto the remaining* G

end

Domain transformation or "shuffling" modifies the layout of the pasted domains. It is applied to contained DAG's assumed to remain within the area bounded by the root of the entire composition. If that root coincides with the root of the contained DAG, the algorithm requires that the high-bridges remain inside the root; also, it does not paste the contained DAG onto anything. The procedure resembles the three-step unpasting sequence: raise high-bridges, manipulate selected cluster, and lower high-bridges, and is illustrated in Plates 11, 13, and 14.

Algorithm 3.9 The Domain-Transformation Sequence

given composition $\mathcal{C}$ *and* $G_x^{cont} \subset \mathcal{C}$ *do*

if $x \equiv$ *root of* $\mathcal{C}$ *then*

apply transformation to G_x^{cont}

for each Greville point $\Gamma_{i,j}$ *in* x *do*

recompute $\Gamma_{i,j}$ *and associate it with* x

end

call **The CV Pasting Procedure** *for all of* $\mathcal{C}$*'s surfaces*

else

call **Lifting the High-Bridges** *for* G_x^{cont}

apply transformation to G_x^{cont}

call **Lower the High-Bridges** *onto* $\mathcal{C}$

end

end

Pasting lends itself to a variety of modeling operations, and is potentially useful in animation. Its range of applicability is discussed in the next chapter.

Chapter 4

Applications

This chapter examines the applicability of the pasting operation to key areas of computer graphics, such as rendering, free-form surface modeling, and animation. **PasteMaker**, a test of the principles introduced so far, is presented next, followed by a brief discussion on user interface issues.

4.1 Areas Of Applicability

Although initially derived from the need for a superior refinement scheme, the feature-oriented pasting mechanism has emerged as a feasible instrument for rendering, industrial previewing, geometric design, and computer animation.

4.1.1 Rendering and Previewing

In the classic approach, adding complexity to a surface requires an excessive number of control vertices and a new rendering of the whole surface, regardless of the extent

of the change. The hierarchical pasting mechanism, however, maintains a low CV count overall, and calls upon the renderer only on the areas affected by the editing operation. Consequently, rendering becomes less expensive and better suited to interactive inspection by the industrial previewer or computer artist.

4.1.2 Geometric Design

Perhaps the most intuitive use of pasting is in the modeling area: indeed, it defines a method for making local changes and layering detail arbitrarily, and allows for editing at the feature or composition level. Courtesy of the domain-surface duality, features can be manipulated in the domain space or in the surface space either pasted or unpasted.

Since feature changes are independent of the bases and non-overlapping features do not interact with each other, modeling that affects remote parts of the composition requires only few local updates. This optimization contributes to the interactive modeling speed and enforces a tighter control on the locality of change.

Often, it is sufficient to design a single *standard feature* that applies "well" to several base surfaces: instead of designing N features shaped as already pasted details, it suffices to construct one unpasted feature that produces the N pasted images when mapped on the various bases. The surface design ratio improves from a costly $1:1$ to an efficient $1:N$, and even if the pasted feature does not look perfect initially, it requires less adjustment than needed when designing from scratch. This is the case of the side-mirror in the car industry: the mirror has one unpasted shape meeting the design needs of a whole car family when pasted onto each car body. Further adjustments of the mirror configuration include sliding the pasted feature along the car panels, and adjusting the control vertices of the composition.

The pasting mechanism provides two main handles to the geometry of the model, both of which contribute equally to the shape of the composite surface:

- the **domain handle** is the underlying domain of each surface. When a domain is transformed, it may change the layout of the composition and redistribute the contribution of the bases; as domains shuffle, the effect may be twofold: if the DAG edges do not change to reflect radical modifications in the overlapping strata, the *polygonal base coverage*[1] must have been affected, requiring that feature displacements be reapplied to the known set of bases. This happens when the features slide only partially from under their bases. If, however, the shuffle is more dramatic to incur changes in the *structural base coverage* and modify the layout of the paste hierarchy, the new bases must be determined prior to reapplying the feature displacements.

- the **CV handle** is the control vertex matrix associated with each surface. When one or more CVs move, they leave the structural (domain) setup unchanged, but affect the topography of the model from the owning surface of the CV to all its descendants. The control vertex handle has a two-way applicability to design editing: to alter the shape of a feature, assuming no change in the critical domains[2], one can either manipulate the CVs of the feature (direct change), or those of the bases it lies on (indirect change).

  - **direct changes** apply identically to the pasted and unpasted image: as shown in Chapter 2, the control vertices of the pasted feature are the mapped images of the original Greville displacements. Therefore, a

[1] Both polygonal and structural base coverage were defined in Chapter 3.

[2] The feature domain and the domains of the underlying bases.

change in the pasted CV has a correspondent in the unmapped displacement; similarly, a CV change in the unpasted feature will be reflected in its pasted image. Indeed, once the displacement vectors have been established, it is no more expensive to edit the displacement than it is to edit the regular spline.

– **indirect changes** are propagated from bases to features recursively, and require that the feature surface and all its descendants be repasted onto their bases.

Additional editing handles are provided by the length and orientation of the feature displacement vectors.

- the **orientation handle** is determined by the cross-product order of the x and y Greville displacements. $x \times y$ maps the result onto an elevation on the base surface. $y \times x$ is useful in generating depressions into the base. The use of the orientation handle is illustrated in Plate 10, which depicts two blue surfaces, one elevated and the other depressed, pasted onto a green base.

- the **offset handle** is the normal component of the unmapped displacement, one of $\| x \times y \|$ or $\| y \times x \|$ depending on the orientation choice. Useful applications would involve changing one or more offsets by a constant or variable delta, as shown in Plate 2.

Since the handles operate on contained DAGs – clusters of one or more surfaces – the extent of the change depends on the size of the cluster chosen for the operation. Thus, the contained DAG acts as a means of local deformation control. *Nota bene*: if present, high-bridge surfaces may extend the deformation area beyond the boundaries of the contained DAG.

4.1.3 Animation

The pasting operation fits well into the animation scheme as an alternative solution to key-frame trajectories and other path-definition techniques. To this end, the object(s) subject to animation must be pasted onto a base composite that is not represented visually, but serves as a path generator.

The primary contribution to the animation realm stems from the inherent shape-motion duality of the domain and CV handles, yielding what we call *organic motion*: a blend of sweeping motion and smooth molding of the features in their passage between layers of arbitrarily shaped surface clusters. Imagine, for example, the behavior exhibited by a tear drop as it trickles down the cheek, or the sluggish crawl of a snail on uneven land. Since features change their shape according to the non-planar base surface(s) on which they are mapped, the pasting technique applies primarily to the animation of morph-assisted motion.

4.2 PasteMaker: A C++ Pasting Editor

PasteMaker is a spline surface editor that demonstrates the concepts presented in the previous chapters, and explores ways in which the theoretical findings apply to modeling and animation. The editor runs on an SGI Onyx machine, and performs all the animated tasks interactively, in real time. The main features of the editor appear in the color plates shown in Appendix B.

4.2.1 An Object Oriented Exercise

The program is written entirely in C++ using the Splines library [VB92] under development at the University of Waterloo. Many of the object oriented princi-

ples translated here into code originate from [Mey88]. Valuable references on the intricacies of the C++ programming language are found in [ES90] and [Str91].

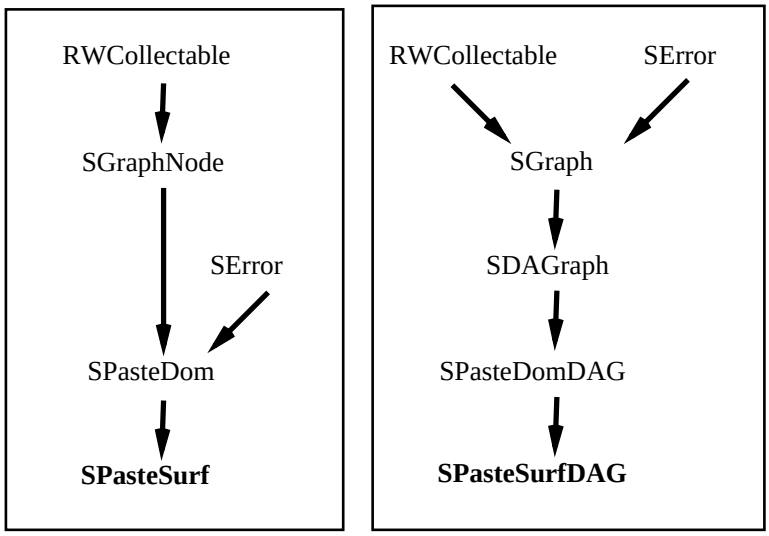

Figure 4.1: The inheritance hierarchies of the SPaste classes

The underlying component of the application is the *SPaste* library, which implements the concept of feature-mapping and provides handles to pasting-related operations. *SPaste* is device-independent. The core of the package is *SPasteDom*, a two-dimensional domain class that specifies a domain by a rectangle and an affine transformation to be applied to the rectangle. Domain objects are manipulated from within *SPasteDomDAG*, a DAG class responsible with pasting, transforming affinely, and unpasting its components. A third class, *SPasteSurf*, inherits *SPasteDom*'s properties and adds generic B-spline information.

Inserting a surface into an *SPasteSurfDAG* instance is equivalent to pasting it onto a base DAG. *SPasteSurfDAG* is derived from *SPasteDomDAG*, and controls the surface aspect of the pasting operation by redefining a small set of virtual functions. The class hierarchy diagram and the class interface listings are presented

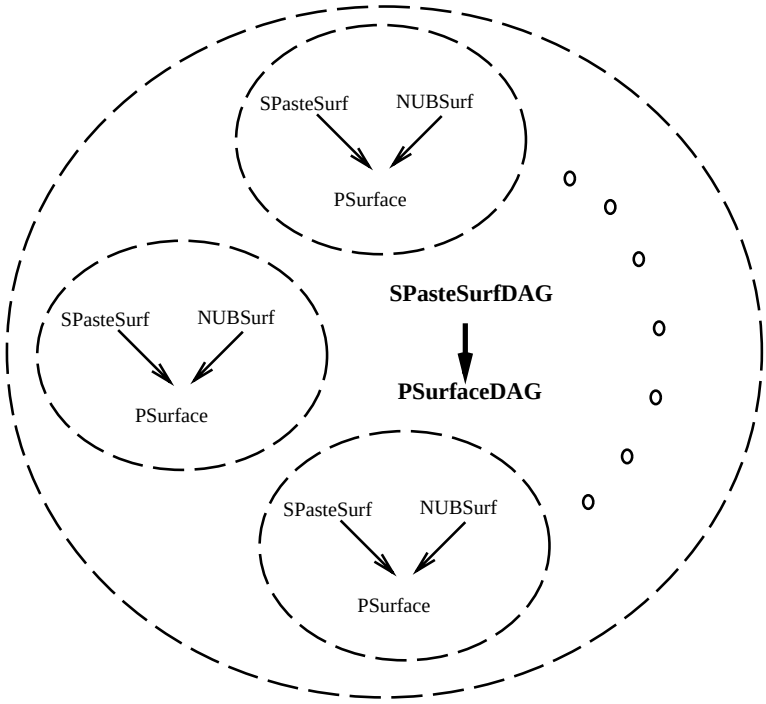

Figure 4.2: Inheritance diagram of the PasteMaker class architecture Classes

in Figure 4.1.

PasteMaker uses the *SPaste* library to define its own pasting classes: *PSurface*, which inherits from *SPasteSurf* and a B-spline tensor-product class, and *PSurfaceDAG*, derived from *SPasteSurfDAG*. Figure 4.2 shows the inheritance graphs of *PSurface* and *PSurfaceDAG*, and the relationships between objects of the two classes.

The other application-specific classes are presented from an architectural standpoint in the next section.

4.2.2 Architectural Overview

PasteMaker is structured on the *Model-Viewer-Controller* (MVC) paradigm, illustrated in Figure 4.3.

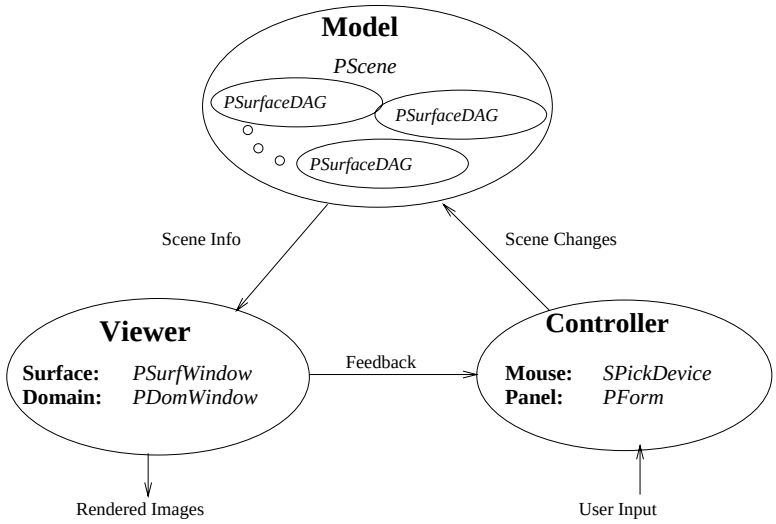

Figure 4.3: The PasteMaker Architecture Represented As An MVC Diagram

The **controller** is user-driven and is responsible for updating the model and processing the viewer feedback. It has two components, each dedicated to a style of interaction: the control panel on the right-hand side of the screen (Plate 1) maintains a set of buttons and settings to adjust the model indirectly via mouse clicks; outside the panel, the mouse buttons interact with the model directly, in sequences of pick-and-edit operations. *PForm* is the class that implements the panel component, while *SPickDevice* is a major element of the mouse component.

The **model** is a collection of B-spline surface clusters (or DAGs) organized as a scene. The controller classes interact only with *PScene*, the maintainer of the entire model. *PScene* delegates editing tasks to its components, the *PSurfaceDAG*s, which in turn maintain pasted nodes of type *PSurface*. The changes affecting the model

are reflected instantly in the viewer.

The **viewer** addresses the issue of determining suitable visualization and interaction handles, considering the dual nature of the pasting mechanism. Since both the domains and the CV displacements are mapped during the pasting operation, and both are valuable modeling handles, the viewer offers a split image of the model by rendering the domains and surfaces in separate windows. Classes *PDomWindow* and *PSurfWindow*, both derived from the abstract *PWindow* class, implement the viewer in its surface and respectively domain instantiation. The domain view provides a clear picture of the structural relationships in the model, and offers an intuitive handle to layout editing.

4.2.3 Functional Overview

PasteMaker operates in one of three modes, *pasting*, *control vertex editing*, or *viewing*, with specific operations bound to the mouse buttons in each mode.

Pasting and domain operations can be accomplished by direct manipulation in either view. The cluster of surfaces subject to change is determined by the current selection, which may contain a single object, a contained DAG, or a whole graph. For example, depressing the left mouse button on a domain selects the contained DAG rooted at that domain; then, as the selection is dragged in domain space, establishing new structural dependencies in the DAG, the surface view reflects the repasting process at the control vertex level as a smooth animation of surface morphing and sliding.

The control panel commands are active in all three modes. Among them, PULL/RELEASE and POSITIVE/NEGATIVE PASTING implement the displacement vector handles presented earlier in this chapter. Pulling a surface by an

amount changes the length of the original normal displacements (Plate 2), while releasing it resets the displacements to the original offset. Depending on the number of displacements affected and the type of pull – constant or variable – the effects of the deformation may vary significantly. Three such examples, a magic carpet, a blow-up, and an extrusion effect, are illustrated in Plates 8, 2 and 5 respectively. Plate 10 and its single-color counterpart, Plate 9, demonstrate the use of positive and negative pasting (notice the two blue surfaces), which converts depressions into elevations and vice-versa by reversing the displacement orientation.

PasteMaker's COPY command generates pasted or unpasted clones according to the user's choice. Plate 6 shows the green pasted surface on top of its purple base, and its unpasted clone on its left. The blue and pink composition on Plate 10 is an identical copy of its twin, both of which are pasted onto a green surface. The clone of a pasted surface uses the pasted CVs of its original as displacements.

The editor explores the use of linear as well as non-linear region transformations in the NEW command, which can instantiate surfaces whose domains are transformed by a sinusoidal transformation (Plate 12).

To verify the statement that knot insertion improves the precision of the pasting, **PasteMaker** contains a recursive REFINEMENT command that operates in either parametric direction. Plate 4 shows the red feature piercing its blue base before refinement, when it has 64 control vertices; after u and v refinement resulting in a total of 225 control vertices, the refined feature of Plate 3 is pasted more accurately.

The editor contains a set of *visual feedback* options to test the integrity of the hierarchy changes, and to examine new visualization techniques. The BRIDGES option monitors the formation of high-bridge surfaces, rendered in orange; the OUTLINE option highlights the surfaces whose domain could accommodate the

selection if it were to be pasted, as indicated by the yellow contour on the rightmost green surface in Plate 6. To represent the composition of surfaces as a single entity, the editor offers two alternatives: TRIM the bases recursively along the feature junctions (Plates 7 and 8), or render the DAG as ONE MESH (Plate 9). The application also provides a handle to the *alpha* value for transparency editing.

Direct manipulation seems to be the best type of interaction in the modeling of feature-oriented compositions. Ideally, the domain transformation would be transparent to the user, with all the interaction being performed in control vertex space.

Chapter 5

Final Comments

This chapter reviews the ideas and principles introduced so far, and sheds light on the strengths and limitations of the pasting operation. Finally, it suggests new areas of applicability, and provides directions for future research.

5.1 Summary

As the performance of the graphics hardware approaches the age of feasible virtual environments, the emphasis on real-time, interactive modeling and animation has emerged as a powerful trend. Although surface modeling has long benefited from Bézier's innovation on parametric functions, sophisticated design requirements maintain the need for fast and flexible modeling tools. Moreover, B-spline surfaces have become commonplace in the production studios of today's entertainment capitals, caught up in a competition for an ever greater arsenal of special effects.

Conventional spline surfaces are popular modeling instruments, yet lack important qualities when applied to models whose shape and complexity are subject to

ongoing change. Even when surfaces are specified as tensor products that preserve the continuity between patches, there is a limit to how much control vertices can move without affecting areas that should remain unchanged. The locality of the change is determined by the degree of the curves defining the given area, and the only reasonable way (using conventional methods) of reducing the size of the affected area is to increase the number of control vertices. As a result, multiple details added to a single, overly specified, spline surface reduce editing speed, increase storage requirements, and are hard to remove or adjust. If, instead, each element of detail were an individual surface somehow applied to the underlying topography, the resulting composition would be easier to maintain and would perform better under tight speed and storage requirements.

This thesis has presented a composition method that applies B-spline surfaces, called *features*, to any number of *base* surfaces, in an attempt to meet what we perceive to be major performance criteria in a dynamic modeling environment: local editing, convenient detail repositioning, and interactive execution speed. Our solution, known as the *pasting procedure*, derives from a technique proposed by Forsey [FB88]. We represent each feature surface as a *vector spline*, where each vector $\mathbf{v}_{i,j}$ is a three-dimensional displacement of control vertex $\mathrm{CV}_{i,j}$ relative to its Greville abscissa $\Gamma_{i,j}$. The matrix of control vertices defining a spline surface is therefore expressed as a scattered coordinate system consisting of *diffuse CVs*, suitable for pasting on a series of arbitrarily shaped surfaces.

The pasting operation initially identifies the underlying base(s) and maps the feature domain onto them; the resulting image points and the local curvatures of the base surface(s) form local coordinate frames which serve as mapping support for the feature's CV displacements. Finally, the pasted image of each control vertex is the point that results from adding the associated displacement vector to the local

coordinate frame. The matrix of pasted control vertices and the feature domain fully determine the shape of the pasted surface.

Aside from its original use as a space-saving refinement instrument and time-efficient rendering aid, the multi-layered composition of pasted surfaces provides a suitable ground for further processing, which might involve trimming, blending, warping, and so on[CB89, SP86]. Pasting is a versatile modeling instrument that supports arbitrary domain shuffling provided the compositional structure remains a DAG (Plates 10 and 12), and lends itself well to path-definition and motion control problems in animation. The inherent motion-shape duality transcending the pasting algorithm applies itself to the animation of organic forms, tissue, cloth, or composites whose finer details yield under changes affecting the underlying structure.

5.1.1 Strengths

- Provides a low cost, space-efficient refinement mechanism with local deformation control by representing additional detail as a separate surface.
- Facilitates the modeling and animation of multi-layer detail surfaces in real time, and enables an interaction with the composition as a whole or as a set of self-standing surfaces.
- Uses a general and efficient displacement scheme applicable to non-uniform rational B-splines, (NURBS). The displacement length and orientation provide additional modeling handles for depression and elevation control. Low curvature base surfaces with as few as one or two patches perform very well under the Greville displacement scheme, and produce pleasingly accurate approximations.

- Relaxes the nesting requirement for the surface regions, allowing them to overlap and change arbitrarily, individually or in groups, as long as they relate in a DAG fashion and have invertible, linear, or non-linear domain transformations.

- Supports editing at the control vertex level and at the domain level: CV editing operates directly in the surface space, while domain changes – caused by structural region shuffles or independent transformations – influence the shape of the surface indirectly, through the shape and layout of the underlying bases.

- Extends naturally to n-dimensional B-spline surfaces and to other surface definitions as long as they provide support for an efficient displacement method. Essentially, one must be able to identify those points that are most likely to influence the behavior of the surface under modeling transformations, and use them as origins of the displacement vectors.

5.1.2 Limitations

- Despite its qualities of speed, flexibility, and broad applicability, pasting remains an *approximation* technique, subject to the choice of displacement scheme, control vertex density, and surface curvature. In cases when precise measurements are important, the exact feature displacements must be computed for every point on the surface, regardless of the cost of the task.

- Irregularities in the approximation may also arise from the different tensor product alignment of the overlapping surfaces. A bicubic, for example, has a sixth degree behavior unless the surfaces are equiparametric. A feature's mar-

gin varies as a cubic, while its bases vary as a sixth degree surface. Since the pasting procedure does not address the issue of precise base-feature intersection, slight mismatches might be noticeable along the junctions, particularly on surfaces of sharp curvature around the junction zone.

- The smooth blending between feature and base has not been addressed, under the assumption that features adjust to the shape of their base(s) given the originally computed displacements. To achieve rough C^0 continuity, the feature displacements around its edges should be zero. For a higher degree of continuity, the curvature of the feature in the proximity of its edges must approach zero: the slower the feature "takes off" from the base, the closer it will follow the shape of the underlying base. Even if the model fails to impose exact continuity, the concern for preciseness can be relaxed under the consideration that both resolution and machine-cutting are finite.

5.2 Future Work

As the number of control vertices and overlapping surfaces increases, the mapping costs impact the performance of the algorithm significantly. Further research should focus on optimization issues, such as identifying the smallest area that needs to be updated after a pasting change.

Consider the case shown in Figure 5.1a. Feature domain F rests on root domain A, and on domains B and C. When B moves slightly to the left, as in Figure 5.1b, the feature must identify the base associated with each displacement vector, and remap its displacements accordingly. To optimize these operations, one can ignore the parts of the base coverage that do not change, as is the case with domain C,

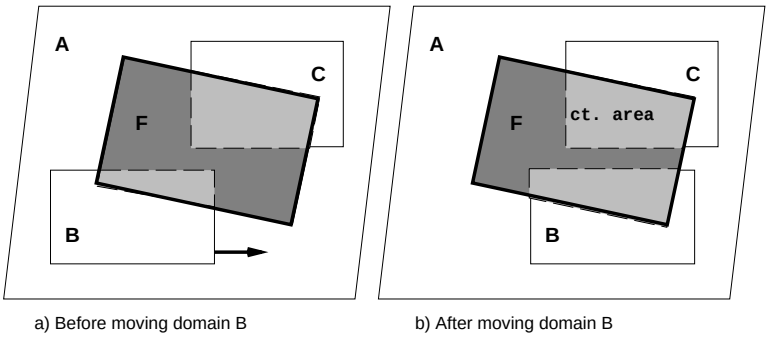

Figure 5.1: Base coverage before and after a domain change

and remap only the displacements affected by the change. Referred to as *selective mapping*, this method must compute the polygonal base coverage of each pasted feature, and perform updates that propagate along "thinner" paths through the hierarchy. To a limited extent, selective mapping is capable of dealing with *loops* in the structure graph.

In this thesis we have proposed a displacement scheme that is easy to construct, and yields satisfactory result in most cases. There are certainly other schemes, more or less sophisticated, whose performance ranges from poor to excellent, depending on the data at hand. A worthwhile goal is to develop displacement techniques suitable for specific surface configurations or spline representations. Moreover, the entire pasting process may need to be adapted to other types of splines – multivariates, box splines, Coons patches, etc. – in response to the demand for custom modeling tools.

One of the "rough edges" of the pasting method is the little concern for junction smoothness. Leaving approximation and misalignment considerations aside, the method cannot *guarantee* seamless detailing. It can, however, examine the feature and base curvatures along the merging area, and possibly *bend* the feature's ends to brush smoothly against the surface of the underlying base(s). To avoid gross

imperfections of the pasting algorithm even well inside the feature domain, one could base the step-size of the derivative calculation, defined in Chapter 2, on the curvature of the base surface: the sharper the curvature around the image point, the smaller the step-size would be.

The surfaces on which we have based our presentation have had a three dimensional geometric representation. The same methodology applies seamlessly to entities exhibiting other properties, visual, acoustic, or both. The pasting mechanism could serve as an inexpensive texture-mapping tool, allowing for textures to be composited in color space, then applied to an object in geometric space. Further research will probe into less obvious, yet suitable areas of applicability, and extend the pasting concept to n-dimensional splines. If n is the dimension of the space, *selective n-pasting* can denote the mapping of at most n coordinates.[1]

Higher dimension splines may store color, brightness, or transparency information in the extra $n-3$ dimensions. For instance, if color and transparency are quantified attributes of the control mesh, they can undergo the same pasting transformation as the geometry component. It is thus possible to define and manipulate opacity at the detail level, and create special effects such as floating crystal-like engravings on glass or stained glass figures roaming intriguingly within the narrow framework of a gothic church window. Hyperspace pasting offers modeling a new challenge and opens avenues of research in n-space compositing.

[1] This thesis has focused on *complete 3-pasting*, in which all three dimensions are mapped.

Appendix A

The C++ SPaste Classes

The classes listed in this appendix implement a fully functional pasting hierarchy, based on the theoretical concepts presented in the thesis. The SPaste classes belong to the Splines project currently under development at the Computer Graphics Laboratory, University of Waterloo.

The following pages list the header files of four templated classes: two node classes, SPasteDom and SPasteSurf, and two graph classes, SPasteDomDAG and SPasteSurfDAG. The class hierarchies are illustrated in Figure 4.1. Most Rogue Wave classes encountered in the listing are prefixed by "RW". [Wav91a, Wav91b]

Appendix B

Color Plates

The illustrations shown in this appendix were created with PasteMaker, a C++ Pasting editor described in Chapter 4. The photos were produced on a Silicon Graphics Onyx with a Focus Graphics Camera.

Bibliography

[Bé66] P. Bézier. Définition numérique des courbes et surfaces I. *Automatisme*, XI:625–632, 1966.

[BBB87] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[BF91] R. Bartels and D. Forsey. Spline overlay surfaces. Technical report, University of Waterloo, Computer Science Department, Waterloo, Ontario, 1991. CS-92-08.

[Boe80] W. Boehm. Inserting new knots into B-spline curve. *Computer-Aided Design*, 12:199–201, 1980.

[Cat74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, 1974.

[CB89] M. Casale and J. Bobrow. A set operation algorithm for sculptured solids modeled with trimmed patches. *Computer Aided Geometric Design*, 6(3):235–248, 1989.

[dB78] C. de Boor. A practical guide to splines. In *Applied Mathematical Sciences*. Springer-Verlag, New York, New York, 1978. Volume 27.

[dC63] P. de Casteljau. Courbes et surfaces à poles. Technical report, A. Citroen, Paris, 1963.

[ES90] M.A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley, 1990.

[FAG83] H. Fuchs, G.D. Abram, and E.D. Grant. Near real-time shaded display of rigid objects. *Computer Graphics*, pages 65–72, August 1983. Proceedings of SIGGRAPH '83.

[Far93] G.E. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, Inc., 1993. Third edition.

[FB88] D. Forsey and R. Bartels. Hierarchical b-spline refinement. *Computer Graphics*, 22(4):205–212, August 1988. Proceedings of SIGGRAPH '88, Atlanta, Georgia, August 1-5.

[For90] D. Forsey. *Motion Control and Surface Modeling of Articulated Figures in Computer Animation.* PhD thesis, University of Waterloo, 1990.

[FvDFJ90] J.D Foley, A. van Dam, S.K. Feiner, and Hughes J.F. *Computer Graphics. Principles and Practice.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[Hen93] F.J. Henigman. *Articulated Figure Animation In Coordinate-Free Geometry ?????* PhD thesis, University of Waterloo, 1993. In progress.

[Mey88] B. Meyer. *Object-oriented Software Construction.* Prentice Hall, 1988.

[NNT72] M.E. Newell, R.G. Newell, and Sancha T.L. A solution to the hidden surface problem. *Proceedings of the ACM National Conference*, pages 443–450, 1972.

[SP86] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 151–160, August 1986.

[Str91] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1991.

[Vat92] B.R. Vatti. A generic solution to polygon clipping. *Comunications of the ACM*, 35(7):56–63, July 1992.

[VB92] A. Vermeulen and R. Bartels. C++ splines classes for prototyping. *Curves and Surfaces in Computer Vision and Graphics II*, 1610:121–131, 1992. SPIE Proceedings, SPIE '92, Bellingham, Washington.

[WA77] K. Weiler and P. Atherton. Hidden surface removal using using polygon area sorting. *Computer Graphics*, pages 214–222, August 1977. Proceedings of SIGGRAPH '77.

[War69] J. Warnock. A hidden-surface algorithm for computer generated halftone pictures. Technical report, University of Utah, Salt Lake City, UT, 1969.

[Wav91a] Rogue Wave. *Math.h++ Class Library: Introduction and Reference Manual.* Rogue Wave Associates, 1991. Version 4.0.

[Wav91b] Rogue Wave. *Tools.h++ Class Library: Introduction and Reference Manual.* Rogue Wave Associates, 1991. Version 4.0.