World Space User Interface for Surface Pasting

by

Leith Kin Yip Chan

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, 1996

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Surface pasting is a composition method that applies B-spline surfaces, called features, to any number of other B-spline surfaces, called base surfaces, in order to add details on the base surfaces. The locations as well as the size of the features are determined by transformations of feature domains. By modifying the domain layout of pasted surfaces, we can manipulate the appearances of features interactively in a *Domain Space User Interface.* However, this domain user interface is inadequate because the user cannot interact with the three-dimensional model directly. In this thesis, I propose a *World Space User Interface* that attempts to map three-dimensional user actions to two-dimensional domain operations and aims at providing a more intuitive and efficient modeling interface for surface pasting.

Acknowledgements

I would like to thank my supervisor, Stephen Mann, for his guidance, advice, continuing support and delicious ice-cream over the years. I would also like to thank my readers, Richard Bartels, who suggested the original idea of this research, and Rick Kazman, for their valuable feedback and suggestions.

I cherish the friendship of people I met in the Computer Graphics Lab and the Chinese Christian Fellowship, which made studying in Waterloo an enjoyable experience. I would like to thank Kit Sum Yuen for her love, patience and support that made my life meaningful.

Financial support for the research in this thesis was provided by NSERC.

I dedicate this work to my parents and my family. Without their unending love and understanding in my life, I could not have accomplished this.

Finally, I would like to thank God for giving love to us.

Trademarks

SGI and Open Inventor are registered trademarks of Silicon Graphics, Inc.

PasteInterface, 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 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.

To my parents

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since the introduction of Bézier curve and surface functions by de Casteljau [dC63] and Bézier [B́66], free-form surfaces have become popular tools in computer aided modeling and animation because of their valuable properties, including *affine invariance*, the *convex hull property* and the *variation diminishing property* [Far93]. By moving control vertices, which define the surface, we can edit the shape of the surface effectively. B-spline tensor-product surfaces, which consist of internally aligned patches, are used to represent sophisticated models and are among the most widely used types of free-form surfaces.

If we want to enhance the detail on a certain region of a B-spline tensor-product surface, we can increase the number of control vertices by splitting patches (Boehm's algorithm [Boe80] and the Oslo algorithm [BBB87]). However, only an entire row or an entire column of patches can be split, thus adding details (patches) to a particular region required the addition of patches across the entire surface. This results in the addition of unnecessary control vertices and computation needed to evaluate the surface. An alternative way to add detail to the surface, aimed at maintaining a low number of control vertices, was proposed by Forsey and Bartels [FB88]. In this approach, detail surfaces, called overlays or features, are applied to 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.

Barghiel [Bar94] generalized Forsey's method, elaborated on a displacement method proposed by Bartels [BF91], and applied it to interactive, real-time modelling. To avoid computing the displacement of every point along the feature surface, he defined an approximate displacement mechanism that involved only the control vertices. This approximation technique is called *surface pasting.* Moreover, he implemented a pasting editor called PasteMaker to paste features on base surfaces hierarchically. The user can control the location as well as the size of the features by manipulating the domain relationship between base and feature surfaces (*Domain Space User Interface*). However, this domain user interface is inadequate because the user cannot interact with the three-dimensional model directly.

In this thesis, I propose a *World Space User Interface* that attempts to map three-dimensional user actions to two-dimensional domain operations so that the user can manipulate the three-dimensional composite surfaces directly. The goal of this research is to find three-dimensional operations that naturally map to the two-dimensional domain functions used in surface pasting. I focus on the mathematics behind these operations rather than the user interface. Thus, although I give a brief qualitative measurement of the user interface at the end of the thesis, I did not perform any user testing.

1.1 Overview of Chapters

Chapter 2 describes the background material with emphasis on surface pasting and domain space user interface.

Chapter 3 discusses the techniques involved to create a world space user interface that I propose in this thesis.

Chapter 4 lists all implementation details of *PasteInterface*, the implementation of the world space user interface.

Chapter 5 compares the world space user interface to the domain space user interface and discusses the limitations of the world space user interface.

Finally, Chapter 6 summarizes the result of this work and lists further work.

Chapter 2

Background

By using a procedure known as pasting, 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. As a result, this technique allows us to model surfaces by interactively changing displacements, control vertices, or the domain layout.

In this chapter, I introduce the basic idea and underlying operations of hierarchical surface pasting [Bar94][BBF95]. I then describe the use of a domain space user interface to manipulate the pasting process interactively. Finally, I briefly describe the Spline Library [VB92] and Open Inventor [Wer94], two libraries on which I based my implementation.

2.1 Surface Pasting

The idea of surface pasting is to attach a feature surface on a base surface in order to provide details on the underlying surface. After pasting, the feature surface retains the character of its unpasted shape and at the same time, reflects the topography of the underlying surface. Moreover, we can have multiple surfaces pasted together, and any particular surface is influenced by all surfaces below and affects all surfaces above it.

This chapter first talks about the basic of B-Spline surfaces and then outlines the Greville Displacement B-Splines and the pasting operations.

2.1.1 B-Spline Surfaces

Surface pasting is based on a widely used type of spline surface, the tensor-product B-spline [Far93]. A three-dimensional tensor-product B-spline surface is defined by a rectangular mesh of three-dimensional control vertices (CVs) and a pair of function bases over a two-dimensional tensor-product domain D.

A B-spline $B_{k,d}$ is a piecewise polynomial basis function of order d or degree $d-1$, and is defined as follow:

$$\begin{aligned} B_{k,1}(u) &= \begin{cases} 1, & \text{if } u_k \le u \le u_{k+1} \\ 0, & \text{otherwise} \end{cases} \\ B_{k,d}(u) &= \frac{u-u_k}{u_{k+d-1}-u_k} B_{k,d-1}(u) + \frac{u_{k+d}-u}{u_{k+d}-u_{k+1}} B_{k+1,d-1}(u), \end{aligned}$$

Each basis function is defined over d subintervals of the total range of u. The selected set of subinterval endpoints u_j is referred as a knot vector.

A common expression of a B-spline surface $S(u,v)$ is a linear combination of control vertices $P_{i,j}$ and basis functions $B_{i,k}(u)$ and $B_{j,l}(v)$, defined in each parametric direction u and v respectively:

$$S(u,v) = \sum_i \sum_j P_{i,j} B_{i,k}(u) B_{j,l}(v),$$

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

Under a diffuse coordinate space (DCS), a vector has an associated local coordinate frame $\mathbb{F}_{i,j} \in DCS$. If we express each control vertex as an origin plus an offset vector in the diffuse coordinate space, we can rewrite our spline as:

$$S(u,v) = \sum_i \sum_j (O_{i,j} + \boldsymbol{d}_{i,j}) B_{i,k}(u) B_{j,l}(v),$$

and all pasting processes operate on this expression of B-spline surfaces.

The characteristics and the quality of the pasting operation largely depend on the choice of $O_{i,j}$ and $\boldsymbol{d}_{i,j}$. Our choice is based on the Greville Displacement B-Splines and will be discussed in the next section.

2.1.2 Greville Displacement B-Splines

Since the surface pasting is built on Greville Displacement B-Splines, we need to convert all B-spline surfaces to Greville Displacement B-Splines surfaces. For each control vertex $P_{i,j}$ of a B-spline surface, there is an associated Greville domain point $\gamma_{i,j} = (\gamma_i, \gamma_j)$, consisting of a pair of Greville abscissae, one for each parametric

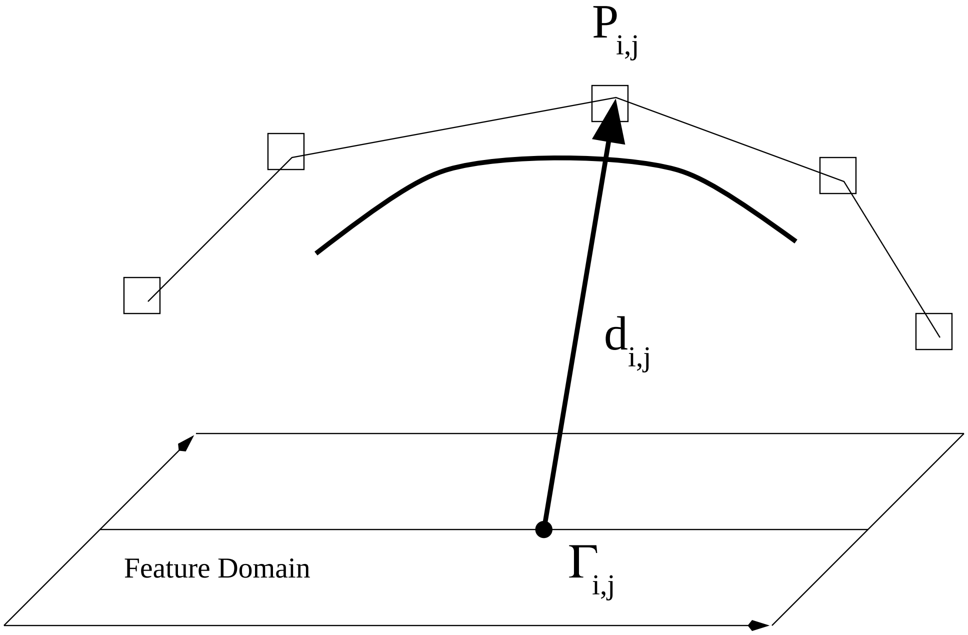


Figure 2.1: Greville Displacement B-Spline

direction, where $S(\gamma_i, \gamma_j)$ is the surface point that is maximally influence by $P_{i,j}$ [Far93]. Thus, if we move $P_{i,j}$, the surface point $S(\gamma_i, \gamma_j)$ will be affected the most. We embed a Greville point into a three-dimensional space by adding a free coordinate $\omega_{i,j}$ and get $(\gamma_i, \gamma_j, \omega_{i,j})$. We identify the subspace $\omega_{i,j} \equiv 0$ as that containing the surface domain, and we denote $\Gamma_{i,j}$ be $(\gamma_{i,j}, 0) = (\gamma_i, \gamma_j, 0)$.

We will use the point $\Gamma_{i,j}$ as $O_{i,j}$, the origin of our local frame.

Then our choice of offset vector $\boldsymbol{d}_{i,j}$ can be written as CV offsets in corresponding frames, with $\Gamma_{i,j}$ as the origin and $(P_{i,j} - \Gamma_{i,j})$ as the displacement $\boldsymbol{d}_{i,j}$. Now we have

$$S(u,v) = \sum_i \sum_j (\Gamma_{i,j} + \boldsymbol{d}_{i,j}) B_{i,k}(u) B_{j,l}(v),$$

as illustrated in Figure 2.1. Notice that $\boldsymbol{d}_{i,j}$ is also embedded in the local coordinate frame . Let the coordinate vectors of the local frame be $\boldsymbol{i}_{i,j}, \boldsymbol{j}_{i,j}$ and $\boldsymbol{k}_{i,j}$, and the original $\boldsymbol{d}_{i,j} = \delta^1_{i,j}\boldsymbol{i} + \delta^2_{i,j}\boldsymbol{j} + \delta^3_{i,j}\boldsymbol{k}$. Then the embedded $\boldsymbol{d}_{i,j}$ is $\delta^1_{i,j}\boldsymbol{i}_{i,j} + \delta^2_{i,j}\boldsymbol{j}_{i,j} + \delta^3_{i,j}\boldsymbol{k}_{i,j}$.

By removing the Γ origins, we have a collection of local displacements from the Greville abscissae and vectorial *Greville Displacement B-Spline*,

$$S(u,v) = \sum_i \sum_j \boldsymbol{d}_{i,j} B_{i,k}(u) B_{j,l}(v),$$

which serves as an essential building block of surface pasting.

2.1.3 Pasting Greville Displacement B-Splines

To understand the pasting process, look at Figure 2.2. It shows a cross section of surfaces before pasting (on the top) and after pasting (on the bottom). Essentially, surface pasting involves only manipulating the location of the feature's control vertices (the displacement vectors remain unchanged). If we look at the feature surface, we first transform the feature domain in order to make it fit inside the domain of the base surface. This transformation T determines the location as well as the size of the feature surface relative to the base surface. Each feature domain point (u,v) becomes $(u',v') = T(u,v)$. The base surface points $S_B(\gamma'_i, \gamma'_j)$ are taken as images of the $\Gamma_{i,j}$ and are used as origins of the displacement vector $\boldsymbol{d}_{i,j}$.

To apply the feature displacement $\boldsymbol{d}_{i,j}$ onto the mapped origin $S_B(\gamma'_i, \gamma'_j)$, we must set up a local coordinate frame $\mathbb{F}$ at $S_B(\gamma'_i, \gamma'_j)$, where $\mathbb{F} = \{S_B(\gamma'_i, \gamma'_j), \boldsymbol{v}(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k})\}$. In order to reflect the orientation of the base surface at the mapped origin $S_B(u',v')$, the vector components $\boldsymbol{i}$ and $\boldsymbol{j}$ are given by the directional derivatives of base surface S_B in the $\boldsymbol{u'}$ and $\boldsymbol{v'}$ direction ($\frac{\partial S_B}{\partial u'}$ and $\frac{\partial S_B}{\partial v'}$), and $\boldsymbol{k}$ is the cross product of $\boldsymbol{i}$

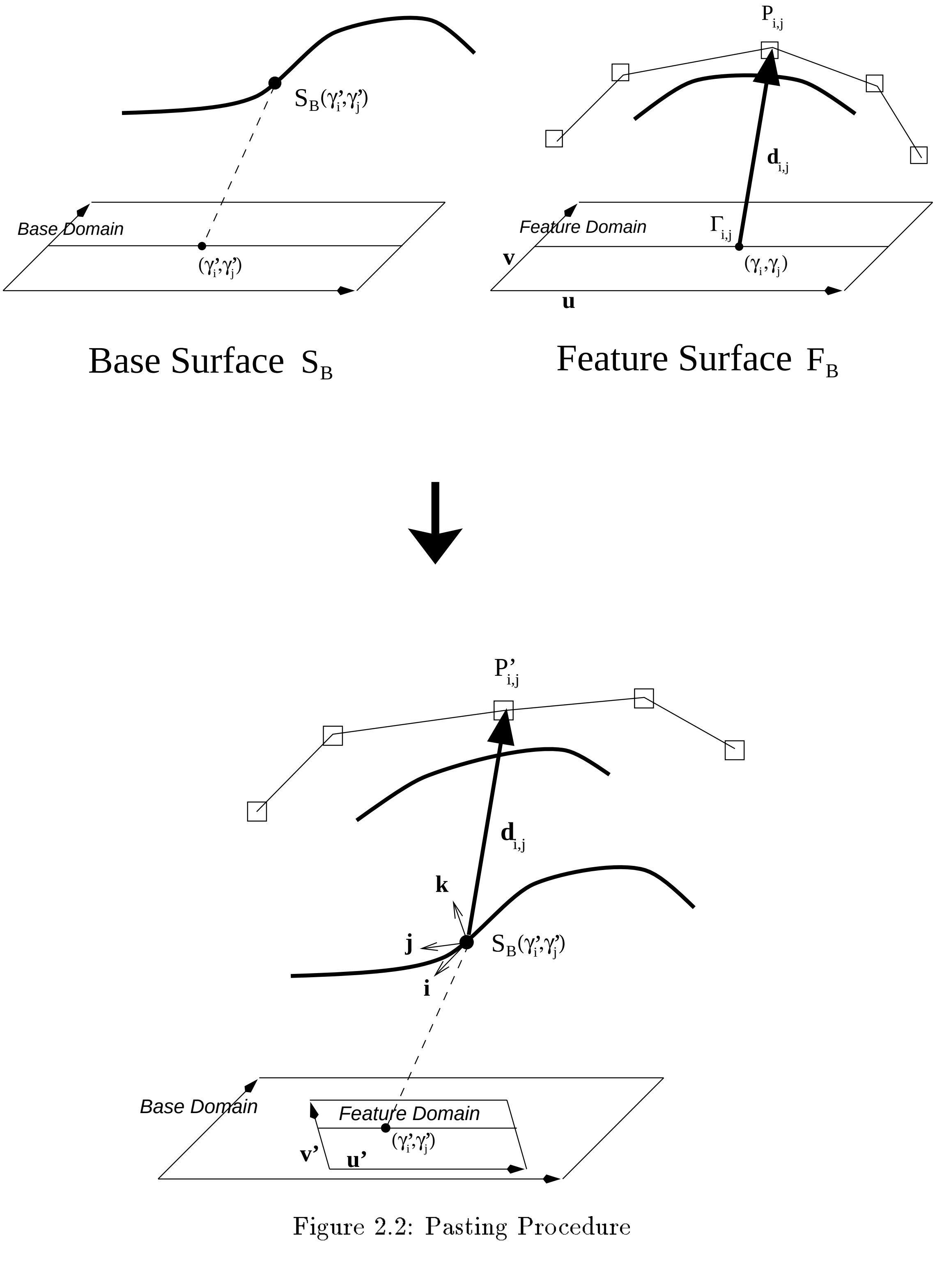


Figure 2.2: Pasting Procedure

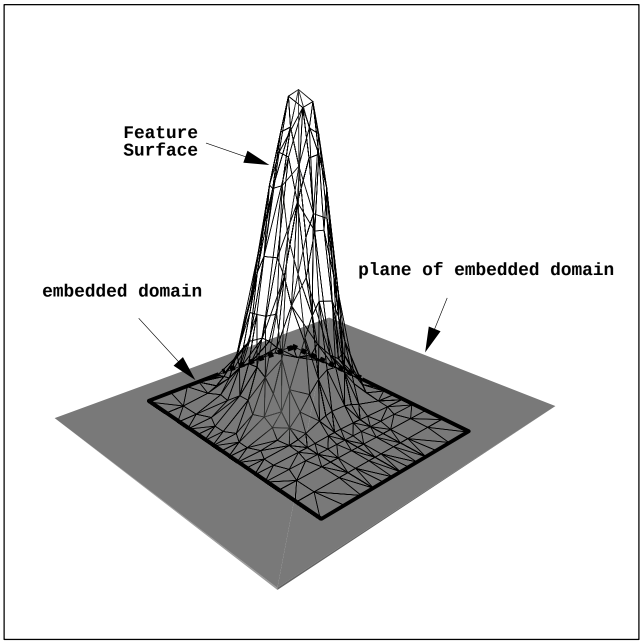


Figure 2.3: A Continuous Feature

and $\boldsymbol{j}$.

Finally, the pasted image $P'_{i,j}$ results from embedding the feature displacement $\boldsymbol{d}_{i,j}$ into the local coordinate frame $\mathbb{F}$. The operation is a point-vector addition performed in the given frame context:

$$P'_{i,j} = S_B(u', v') + \boldsymbol{d}_{i,j}$$

2.2 Surface Continuity

The pasting process does not guarantee a smooth attachment between a feature and a base surface. The continuity between pasted feature and the base surface is approximately the same as the continuity between the unpasted feature and the plane of embedded domain (x–y plane), provided the base surface has low curvature. To achieve a rough C^0 continuity, the edge of the unpasted feature has to lie on the edge of the embedded domain's edge (the thick line in Figure 2.3, which lies on the x–y plane), i.e., the displacement vectors of the edge CVs are zero. Therefore, after pasting, the edge of the feature lies near the base surface. To achieve a rough C^1 continuity, the displacement vectors of second inner layer CVs (CVs beside the edge CVs) must also be zero.

However, special effects can also be achieved by adjusting the CVs' positions. For example, lifting the feature above the embedded domain plane creates a flying carpet effect when pasting the feature on a base surface.

2.3 Hierarchical Pasting

By repeating the pasting procedure, we can generate multi-level pasted surfaces, in which the top feature surface may be pasted upon a number of base surfaces and the resulting feature's shape is dependent on all underlying surfaces. Figure 2.4 shows the polygonal view of the domains of a composite surface. Except for B1, which is the root domain, all domains are transformed individually so that they fit inside the root domain. These transformations also determine the size and location of the resulting domains. Figure 2.5 shows the dependencies of the same set of domains, which vary according to the pasting order and may change dynamically. In this example, the order of pasting is that B2 is pasted on B1 first, then B3, F1, F2, and finally F3 pasted last. Therefore, B2 depends on B1, part of B3 depends

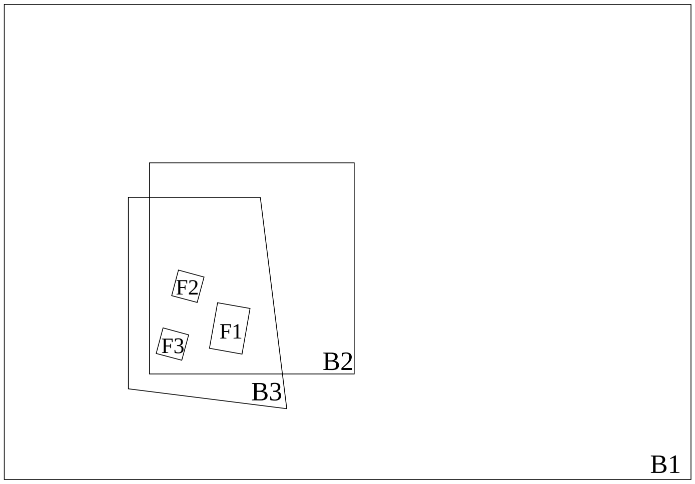


Figure 2.4: Overlapping Domains: the Polygonal View

on B1 and the rest depends on B2, and finally all features (F1, F2, F3) depend on B3. The two shaded polygons above B3 show the shape of the dependent region of B3's domain and combining these polygons gives the complete B3 domain. The original individual surfaces and resulting composite surface are in Figure 2.6 and Figure 2.7 respectively. Notice that surfaces with the domain B3, F1, F2 and F3 are copies of a single surface.

2.4 Domain Space User Interface

Given that a change in the transformation of feature domains can adjust the size and location of the feature surface, it is straight forward to create a *Domain Space User Interface* to manipulate the relative appearance of the overlapping domains, like the one in Figure 2.4. We can translate, rotate, scale or apply any transformation on any domain (except the base domain) in order to manipulate the appearance

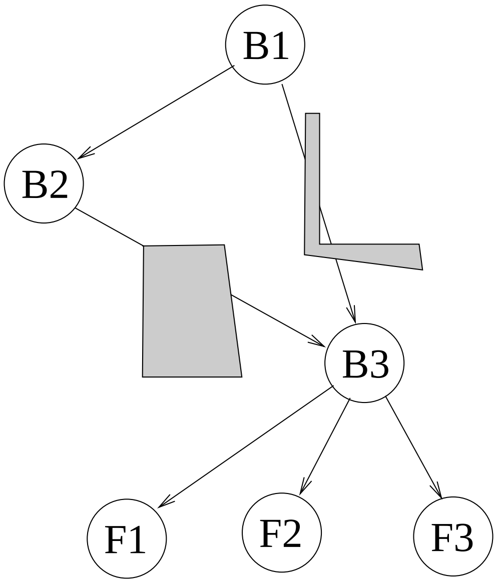


Figure 2.5: Overlapping Domains: the Structure View

of the corresponding feature. For example, if we move (apply a translation transformation to) F1 in Figure 2.4 to somewhere in between F2 and F3, the nose-like feature in Figure 2.7 will move to the top, somewhere in between the two horn-like features. The user interface of Barghiel's PasteMaker [Bar94] is base on this design, as illustrated in Figure 2.8. The lower domain window is the two-dimensional domain view of the composite surfaces. All pasting manipulations are done through this lower window and the top model window shows the resulting surfaces in three dimension.

Although this domain space user interface works, it suffers from the weakness of not having a direct interaction with the 3D model. For example, if we translate a feature domain to the right, the feature surface might move to the left in the model window. In fact, the movement direction of the 3D feature relative to the user can

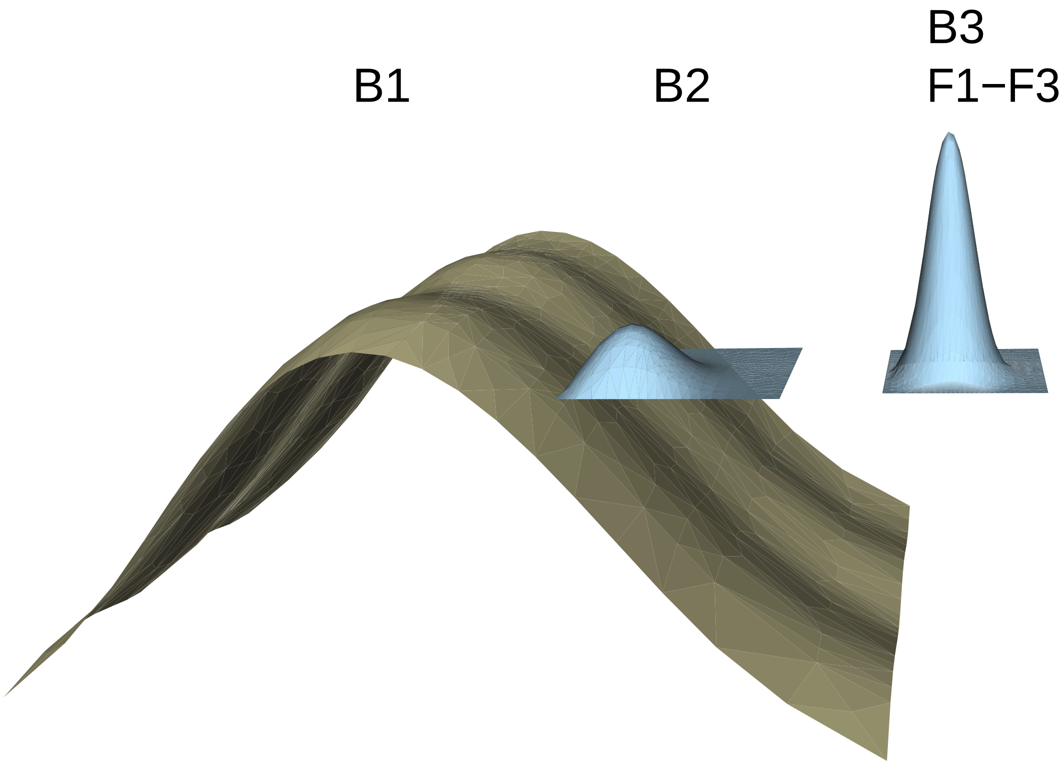


Figure 2.6: Individual Surfaces

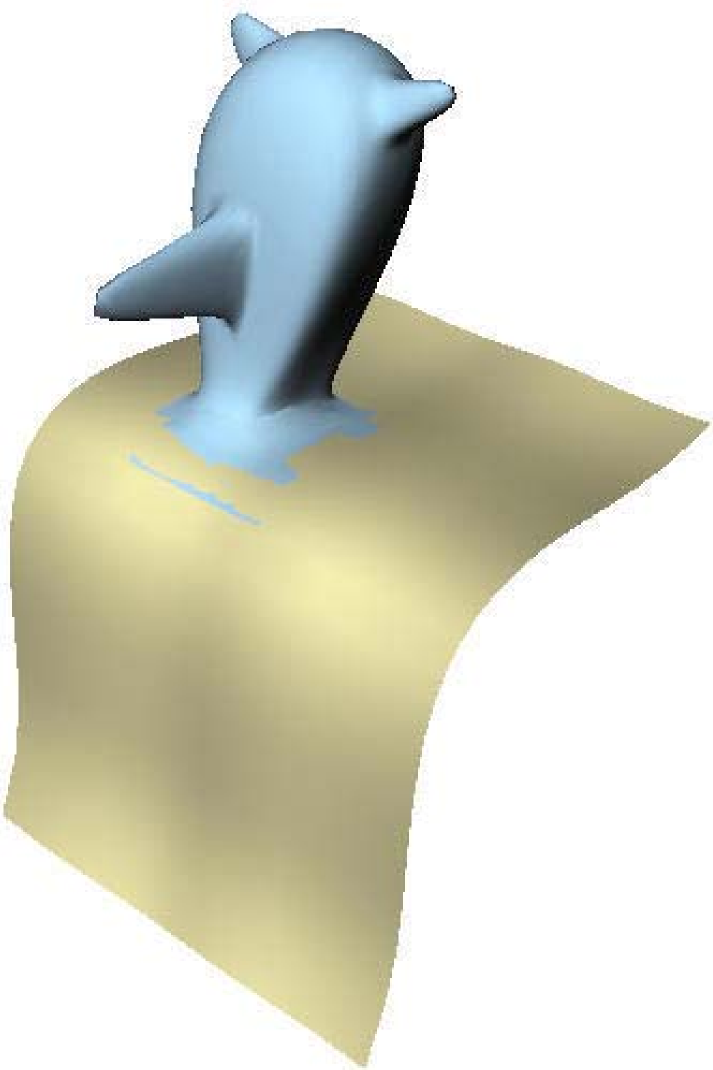
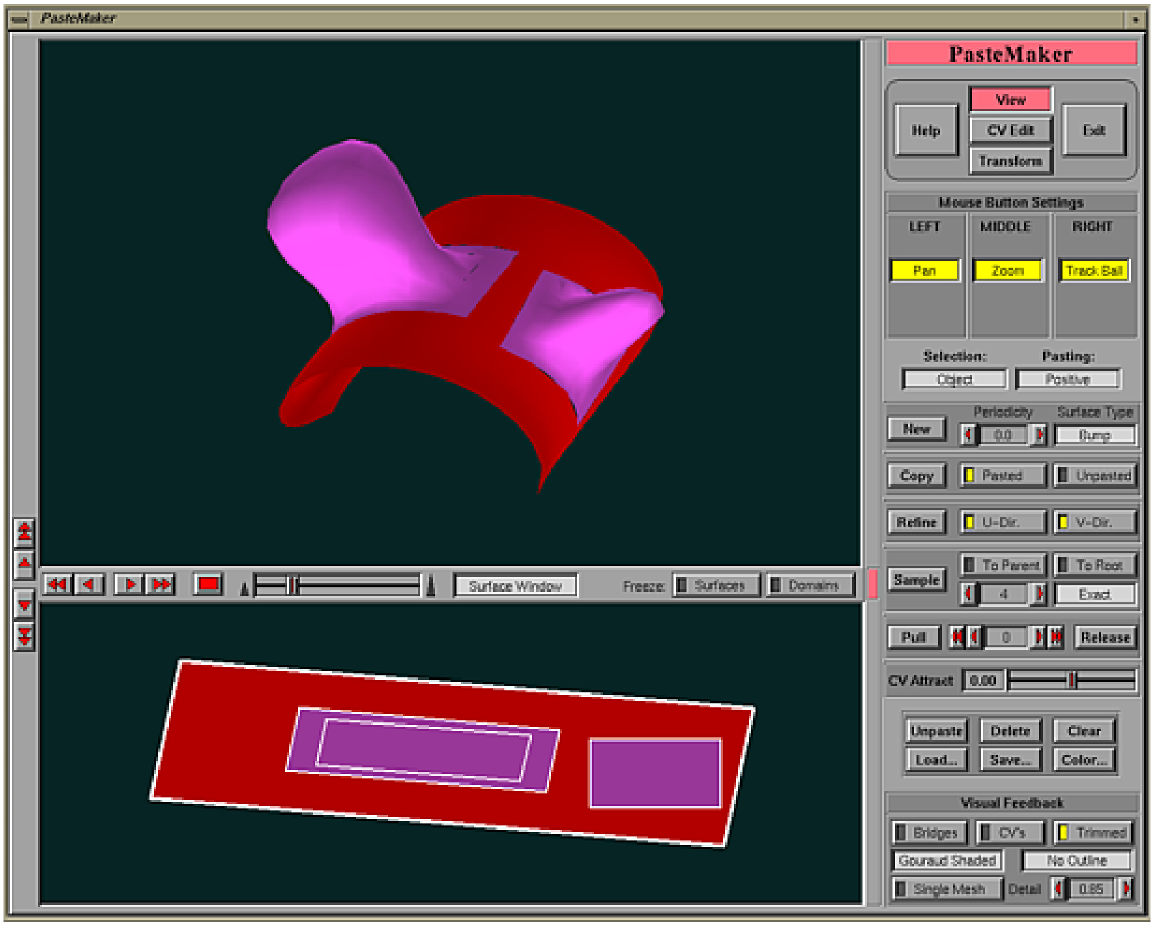


Figure 2.7: Composite Surface



LEFT: Pan
MIDDLE: Zoom
RIGHT: Track Ball

Selection: Object
Pasting: Positive

Periodicity: 0.0
Surface Type: Bump

Sample: 4 Exact

Pull: 0

CV Attract: 0.00

Gouraud Shaded
No Outline

Single Mesh
Detail: 0.85

Figure 2.8: PasteMaker

be arbitrary, depending on the shape of the base surface and the orientation of the camera. Positioning of a feature on the base surface becomes a matter of trial-and-error because we do not know which 3D point on the base surface corresponds to a 2D point in the base domain until we have moved a feature domain onto that domain point to see the result. Therefore, the user has to maintain and relate two sets of data at the same time: the 3D surfaces and 2D domains.

Typically, the user normally does not care about the surface domains. He/She is interested in only the 3D model. Thus, it would be nice to let him/her manipulate the 3D model directly by hiding the 2D domains. This motivates the development of a *World Space User Interface*, the topic of this thesis.

2.5 Spline Library and Open Inventor

My work is based upon two software packages: The Spline Library and Open Inventor. The Spline Library is an object oriented library under continual development at the Computer Graphics Lab, University of Waterloo [VB92]. It provides many convenient object classes such as NURBS Curves and Surfaces so that one can develop a prototype spline application quickly. Moreover, it contains a pasting class for surface pasting [Bar94]. The pasting class allows one to perform surface pasting and maintains a dependency graph of pasted surfaces. In order to perform pasting, a user has to provide an invertible transformation to transform the feature domain into the base domain, as described in Section 2.1.3.

Open Inventor is an object oriented library for interactive 3D graphics [Wer94]. It provides a high level rendering mechanism and building blocks for 3D user interfaces. Inventor focuses on creating 3D objects. All information about these objects (their shape, size, coloring, surface texture, location in 3D space) is stored in a scene graph. The scene graph consists of one or more nodes, each of which represents a geometry, property, or grouping object. Hierarchical scenes are created by adding nodes as children of grouping nodes, resulting in a directed acyclic graph. Moreover, the scene graph can also be used for creating a 3D user interface by using 3D objects called 3D manipulators[1]. Open Inventor also provides several geometric objects such as spheres, cubes and NURBS surfaces. By using C++ inheritance, it is possible to add more geometric objects. A more detailed description of the Open Inventor components used in the implementation of world space user interface can be found in Chapter 4.

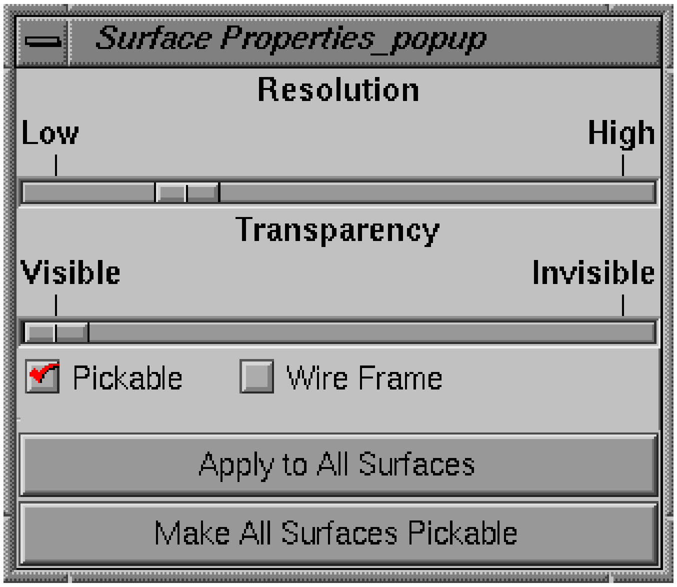
[1] A 3D Manipulator consists of a set of 3D objects (called draggers) co-existing with 3D models that provide an user interface to perform certain actions. Users can click and drag on a dragger to activate and control a specific action.

Chapter 3

World Space User Interface

The goal of the world space user interface is to hide the domains from a user so that he/she can directly manipulate the three-dimensional objects. However, no matter what is displayed to the user or how the user manipulates the objects, operations such as feature translation and rotation have to be performed in the domain space. Therefore, the world space user interface maps all user actions into domain operations. This chapter mainly focuses on the techniques employed to allow a user to perform operations in world space, while at the same time converting all operations into corresponding domain space operations. These operations include surface selection, surface pasting/unpasting and feature manipulation (translation, rotation, scaling, fine tuning and grouping).

The world space interface is implemented in *PasteInterface*, which will be described in more detail in the next chapter.



☑ Pickable ☐ Wire Frame

Figure 3.1: Surface Properties Dialog Box

3.1 Selecting a Surface

In order to manipulate surfaces, the user must specify a particular surface out of many possible surfaces. In PasteInterface, surface selection is done by 3D picking, which sends a ray radiated from the camera viewpoint through the mouse location on the view plane.

The first surface the ray meets is the surface selected. 3D picking is a very effective surface selection method, except when the user wants to pick a surface hidden or covered by another surface. In this case, the user cannot see the hidden surface and is therefore unable to pick it. One solution to this problem is to allow the user to make the top surface transparent and unpickable in order to unveil the hidden surface (Figure 3.1 shows a surface properties dialog box created by using

Motif). However, making a surface pickable again is a bit problematic, since we cannot pick it. My solution is a button (bottom button in Figure 3.1) that makes all surfaces pickable. Other solutions such as having an additional special picking method are also possible.

3.2 Pasting/Unpasting

After selecting a surface, the user may want to paste it on a base surface. Before pasting a feature on a base surface, we need to map the feature domain into the base domain. This transformation determines the location and the size of the pasted feature on the base surface. In my system, for the user to specify the pasting location, he/she must move and scale the feature close to the target location on the base surface. We then project the edge of the feature on the base surface and show it to the user. This projected image is the final position of the feature after pasting. If the user is satisfied with the position, a transformation is calculated and applied to the feature domain, and finally the feature will be pasted onto the base surface. The following sections talks about the pasting and unpasting process in more detail.

3.2.1 Moving the Unpasted Feature

The user manipulates the unpasted feature via a 3D manipulator called a *transformation box* [Wer94]. This transformation box consists of many handles, which are called draggers, as shown in Figure 3.2. There are three sets of draggers: one for translation, one for scaling and the remaining one for rotation. Their functions are

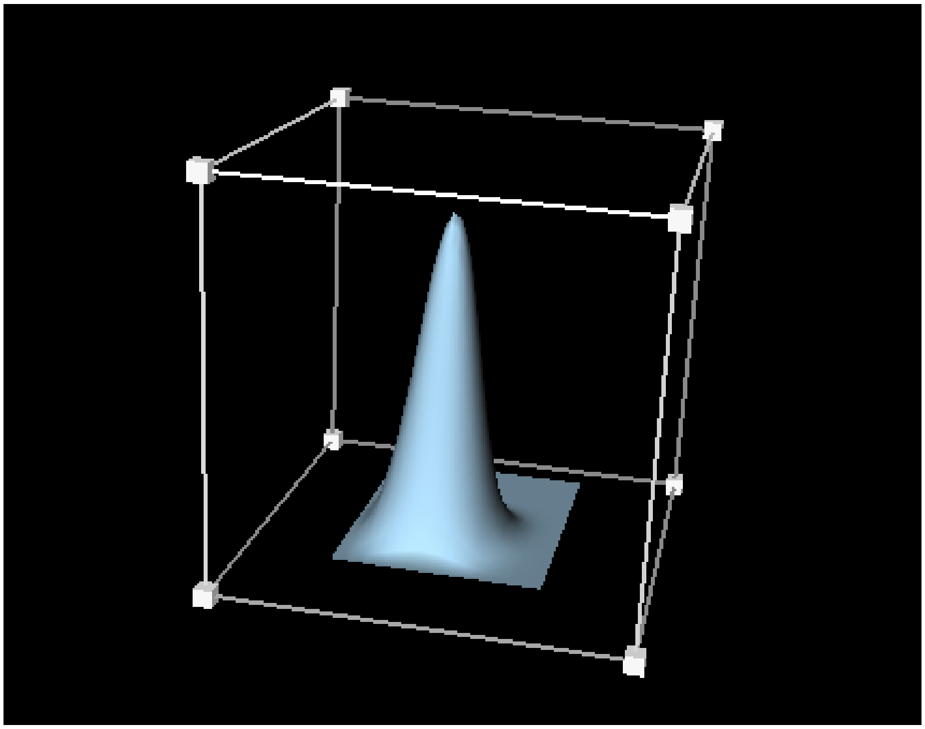


Figure 3.2: Transformation Box 3D Manipulator

shown in Table 3.1. Notice that manipulations of the unpasted feature are done solely to find an embedding of the feature domains in the base domain, as discussed in the next section. Once this embedding is performed, the three-dimensional transformation of the feature is discarded.

3.2.2 Projecting the Feature

After moving the feature, the user may want to preview the pasting location of the feature. This is done by projecting the four corner points of the feature onto the base surface. The projection of each corner is in the direction of the normal to the plane that best fits the four corner points of the feature. The intersection points

Draggers	*Functions*
6 faces of the box	move the object along the corresponding plane
8 small cubes at corners	scale the object uniformly
12 edges of the box	rotate the object around the axis at the center and parallel to the corresponding edge

Table 3.1: Transformation Box 3D Manipulator's Functions

will be the outline of the pasting position, as in Figure 3.3a (Figure 3.3b shows the feature after pasting).

If the user decides to paste the feature to the preview location, we have to transform the feature domain so that the feature can paste onto the preview location. First we need to find the corresponding domain points of the four preview points on the base surface, as shown in Figure 3.4. Then we need to transform the original domain polygon $D = F_1F_2F_3F_4$ to the projected polygon $\bar{D} = \bar{F}_1\bar{F}_2\bar{F}_3\bar{F}_4$. Since the target polygon is not necessarily a rectangle, a linear transformation will not work, but a *bi-linear transformation* will be sufficient. Moreover, if both polygons are convex[1], the bi-linear transformation has an unique inverse from $\bar{D}$ back to D, which is a requirement for surface pasting.

By using a bi-linear transformation, all points inside $F_1F_2F_3F_4$ can be repre-

[1] If the projected polygon is not convex, we can either ignore one corner point, or simply disallow pasting.

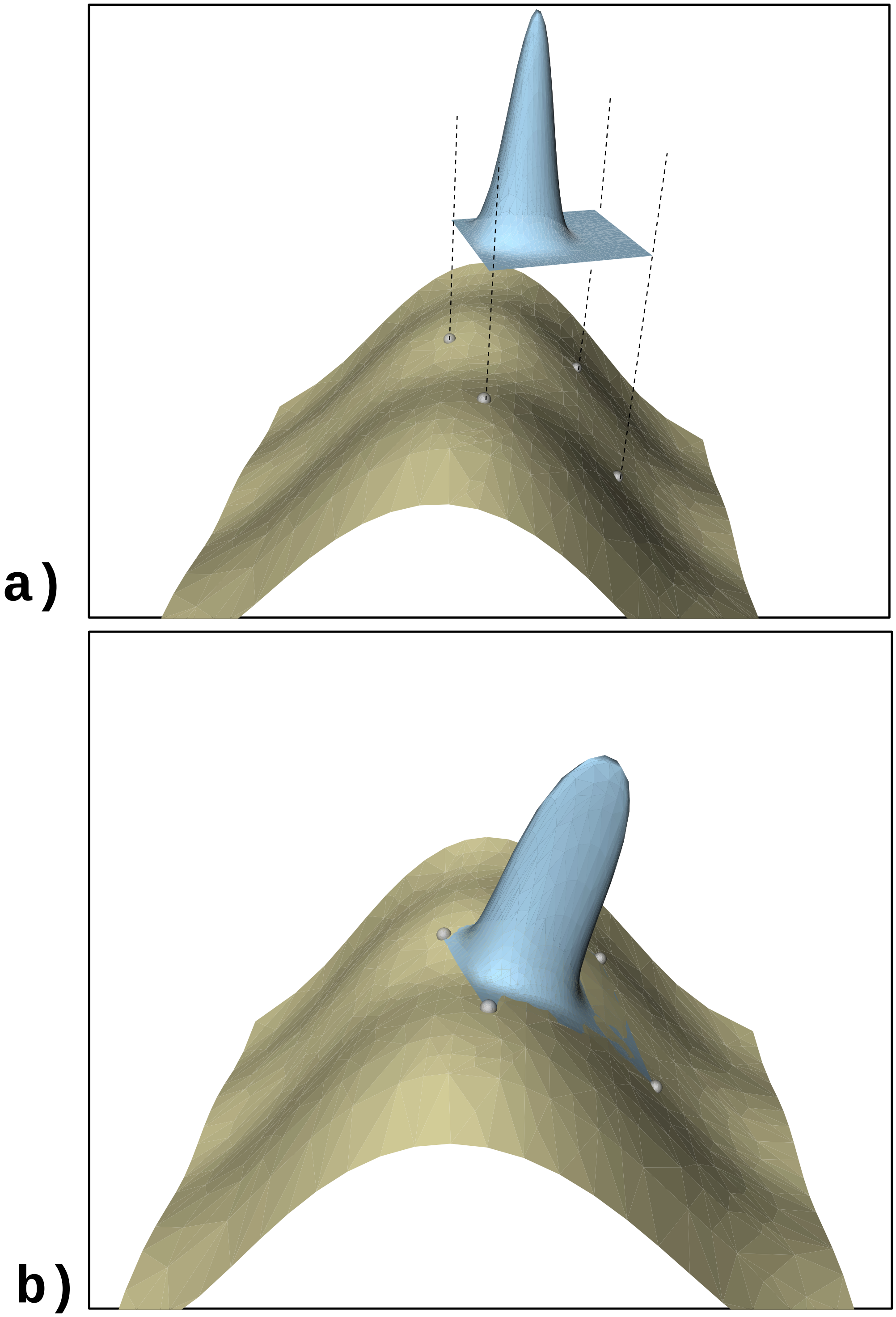


Figure 3.3: Projecting Feature on Base Surface and Pasting

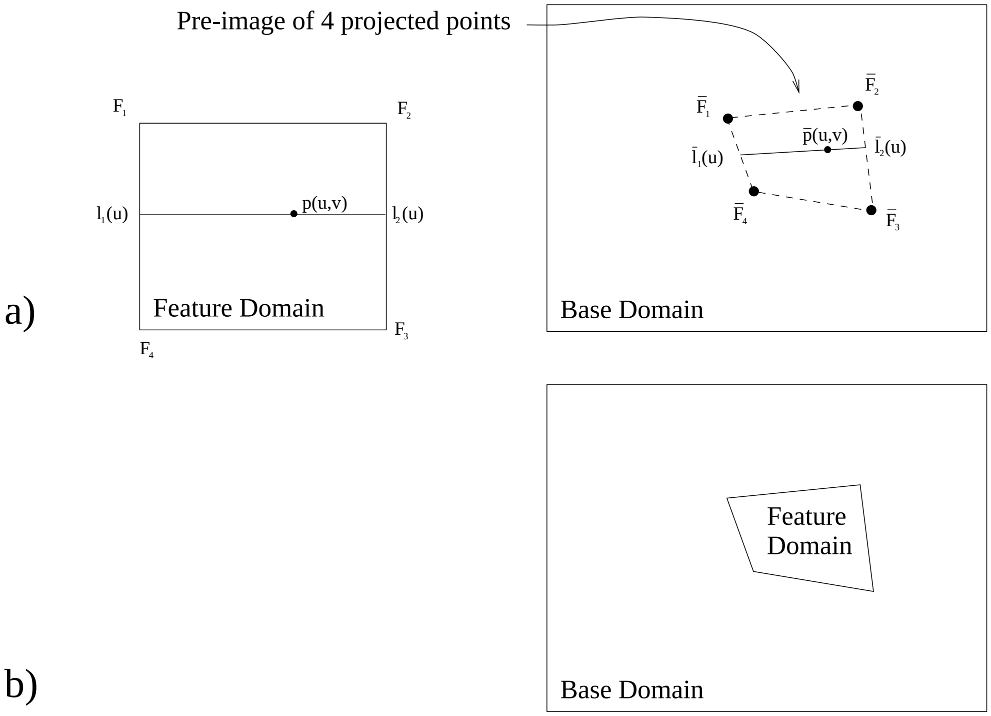


Figure 3.4: Pasting - Domain View

sented as:

$$\begin{aligned} p(u,v) &= l_1(u)\cdot v + l_2(u)\cdot(1-v) \qquad (3.1)\\ \text{where}\quad l_1(u) &= F_1\cdot u + F_4\cdot(1-u)\\ l_2(u) &= F_2\cdot u + F_3\cdot(1-u)\\ 0\le u\le 1, &\qquad 0\le v\le 1 \end{aligned}$$

Similarly, all points inside $\bar{F}_1\bar{F}_2\bar{F}_3\bar{F}_4$ can be represented as:

$$\begin{aligned} \bar{p}(u,v) &= \bar{l}_1(u) \cdot v + \bar{l}_2(u) \cdot (1-v) \\ \text{where} \quad \bar{l}_1(u) &= \bar{F}_1 \cdot u + \bar{F}_4 \cdot (1-u) \\ \bar{l}_2(u) &= \bar{F}_2 \cdot u + \bar{F}_3 \cdot (1-u) \\ 0 \leq u \leq 1, & \qquad 0 \leq v \leq 1 \end{aligned} \tag{3.2}$$

To transform any point from a polygon to the other polygon domain, we only need to find the u and v values of the point inside a polygon and then evaluate the above equations to find the transformed point inside the other polygon. For example, in Figure 3.4a, given a point $p(u,v)$ inside $F_1F_2F_3F_4$, and the observation that $p(u,v)l_1(u)$ is parallel to $l_2(u)l_1(u)$, we have:

$$(p(u,v) - l_1(u)) \times (l_2(u) - l_1(u)) = 0 \tag{3.3}$$

Solving Equation 3.1 and 3.3 gives u and v, and hence we can get $\bar{p}(u,v)$ from Equation 3.2.

Finding the inverse point is the same process, except that we solve for (u,v) from polygon $\bar{F}_1\bar{F}_2\bar{F}_3\bar{F}_4$ and evaluate $p(u,v)$ from Equation 3.1.

3.2.3 Vertical Scaling After Pasting

After finding the required transformation, we can transform the feature domain (Figure 3.4b) and paste the feature on the base surface. If the feature surface has been scaled up or down before pasting, we must adjust the vertical height of the pasted feature by that scaling factor because the pasting process does not affect the vertical height of the feature. We scale the feature surface's height by scaling its control vertices along the feature's vertical direction, resulting in Figure 3.3b.

3.2.4 Unpasting

Unpasting a feature is relatively easy compared to pasting, since the only concern is the location of the unpasted feature. A reasonable choice is to put the unpasted feature spatially close to its pasted location so that the user can pick the feature up easily without wondering where it has gone. After unpasting, the feature domain transformation is reset and therefore the surface size is restored. Note that an immediate paste will not in general return the feature to the same location on the base surface, which should not be a problem since unpasting is usually performed to move a feature to a different composite surface.

3.3 Translating a Pasted Feature

After pasting, the user may want to adjust the position of the feature surface on the base surface. One way is to translate the feature across the base surface.

The challenge of the translation is to move the pasted feature surface with the mouse cursor in a way that the user has a direct manipulation of the feature. However, it is hard to define a direct manipulation of a pasted feature. A loose description would be: we want the user to manipulate the feature as if there is a real three dimensional model of a feature on a base surface sitting in front of him/her so that he/she can use his/her hand to slide the feature back and front, left and right, or set the feature at an arbitrary position. It would be ideal if the user has that kind of feeling when he/she is using the mouse to translate the feature. For example, having the feature strictly follow the position of the mouse cursor may not be the best solution because sometimes it destroys the user's mental model. This situation is illustrated in Figure 3.5. In this figure, a user tries to move the shaded feature (the one closer to us) on the top of the bump (base surface) with the mouse cursor moving from right to left. Since the feature stays under the mouse cursor as the user moves the mouse, the feature moves across the top of the bump and then suddenly jumps to the edge of the base surface (the left shaded feature). This movement is desired if the user really wants to take the feature to the edge; however, the user may only want to slide the feature to the back of the bump.

In this section, two approaches for simulating the direct manipulation will be discussed: projection and picking.

3.3.1 Projective-Translation

The main purpose of the projective-translation is to give the user the feeling of sliding a feature. When the user moves the mouse, the feature should slide on the base surface by following the direction of the mouse movement. Thus, we have to transform the mouse movement $(\Delta x, \Delta y)$ on the screen to a translation $(\Delta u, \Delta v)$ in the base domain. This translation is then used to translate the feature domain within the base domain.

I chose the following approach: given a mouse movement vector produced by the mouse motion in a short time span, we first find the center point P of the feature to represent the location of the feature surface. Second, we find the tangent plane

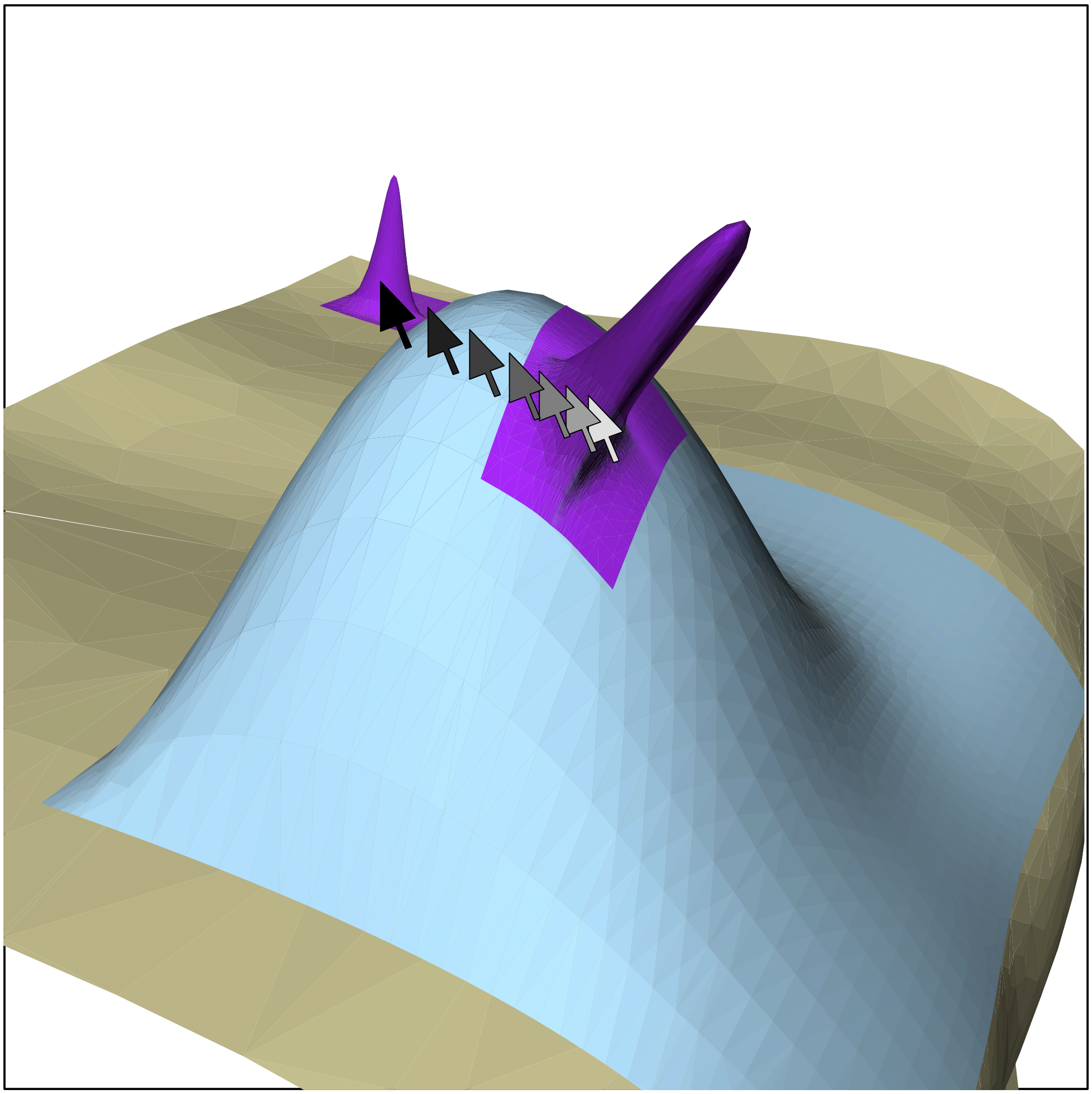


Figure 3.5: Moving a Feature with the Mouse

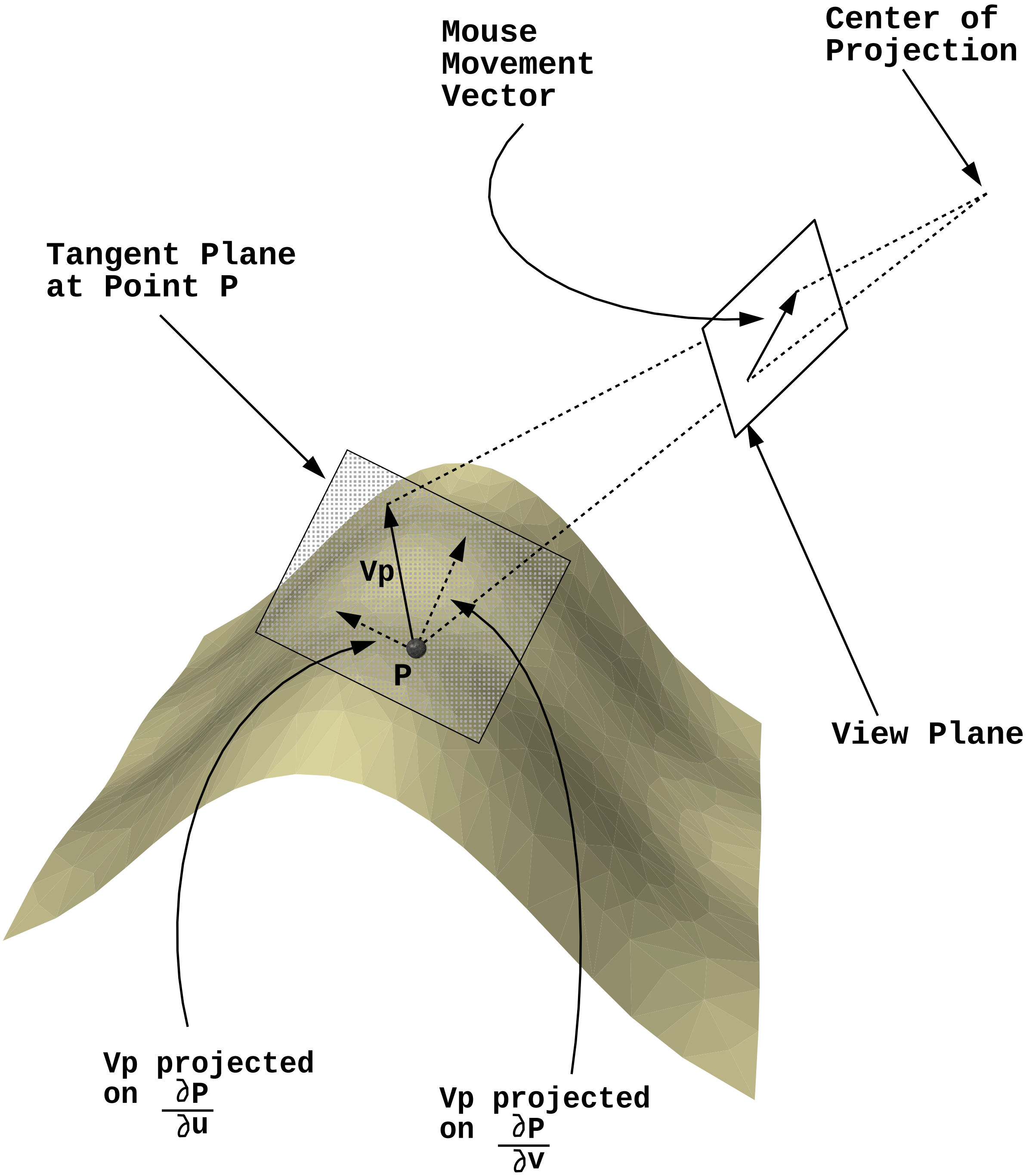

Mouse Movement Vector
Center of Projection
Tangent Plane at Point P
Vp
P
View Plane
Vp projected on $\frac{\partial P}{\partial u}$
Vp projected on $\frac{\partial P}{\partial v}$

Figure 3.6: Projecting Mouse Movement Vector on the Tangent Plane

at that point, which is defined by two partial derivative vector $\frac{\partial P}{\partial u}$ and $\frac{\partial P}{\partial v}$, as in Figure 3.6. Then by projecting the mouse movement vector from the view plane to the tangent plane, we get the vector v_P. Finally, we get the domain displacements Δu and Δv, which are the magnitudes of vectors obtained by projecting v_P onto $\frac{\partial P}{\partial u}$ and $\frac{\partial P}{\partial v}$ respectively.

There is a special case we must handle: if the tangent plane is perpendicular to the view plane, the projected vector v_P will have infinite length. This problem is handled by not updating the tangent plane with the changing feature location within a click-drag-release cycle[2], since the feature is easy to move across a position that produces this perpendicular tangent plane. Moreover, the user should be aware that if the feature is not facing him/her but close to facing the top or bottom direction, the feature translation tends to be more sensitive because the projection will create a longer vector.

There is an additional advantage of not updating the tangent plane in real time with the movement of a feature: the feature will not go backward while the mouse is moved along one direction within a click-drag-release cycle. Figure 3.7 shows the desired movement of the feature resulting from the mouse movement. When the mouse moves upward, we would like the feature go up at the beginning, and then pass over the top and continue to go “forward”.

If the tangent plane is updated in real time, the feature moves upward at the beginning of the motion because the tangent plane is facing the view plane. Once the feature moves across the top and to the back of the base surface, we would like

[2] The user clicks the mouse button, starts moving mouse cursor to move the feature, and finally stops moving by releasing the button

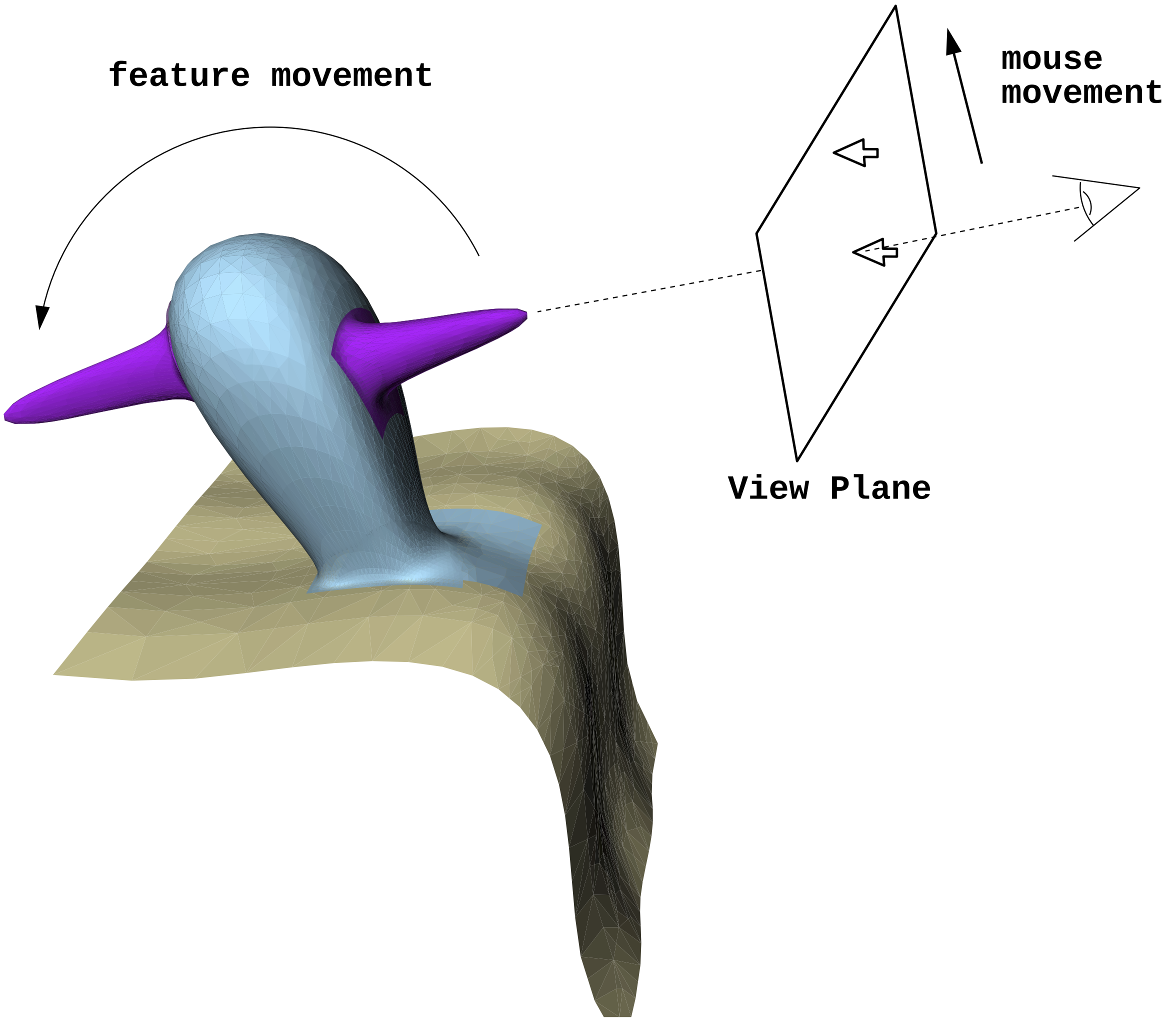


Figure 3.7: Sliding a Feature with a Mouse

the feature to move down as we continue our mouse motion. However, note that now the tangent plane has flipped so that its back faces the view plane. Because the mouse movement vector now projects to the back of the tangent plane, moving the mouse further upward will actually move the feature backward[3].

Notice that the user should be aware of the rough direction of the tangent plane underlying the feature surface in order to use the projective-translation effectively. For example, after moving the feature to the back of the base surface in Figure 3.7, if the user clicks the mouse button and tries to move the feature again, moving the mouse upward will move the feature uphill and vice versa, which is opposite to the previous motion.

No matter how the user slides the feature, it does not make sense to move the feature off the base surface since it destroys the consistency of the user's mental model. Therefore the feature will stop moving on the edge of the base surface.

3.3.2 Picking-Translation

An alternative translation method is picking. Picking lets the user pick and drop a feature to anywhere the user can see on the screen. By attaching a 3D manipulator on the feature surface, a user can drag the manipulator to pick a point on the base surface, as shown in Figure 3.8. Then we can find the domain coordinates of the point and translate the feature so that the feature's domain center lies on that domain point. A picking dragger for translation, which is a part of the manipulator,

[3] Notice that this case assumes that the mouse movement vector does not project to a perpendicular or close-to-perpendicular tangent plane; the feature runs over these positions within a short elapse time. Otherwise, the feature will jump to the edge of the base surface, which is worse than going backward.

is shown in Figure 3.9. Moreover, the user can also use the corner draggers, which represent four corners of the feature, to translate the feature. The translation mechanism of these corner draggers is the same as the picking dragger except corner draggers use corners as their reference points for translation.

The picking-translation will fail if the picking dragger does not pick a point on the base surface. This happens if nothing is picked or if the user picks a surface that is not a part of the current composite surface. In this case, the user has to move the picking dragger again to pick on a valid surface.

Similar to projective-translation, the user cannot move the feature out of the base surface by using picking-translation. The feature can at most move to the edge of the base surface.

3.4 Rotation and Scaling

Rotation and scaling of a pasted surface is done with a 3D manipulator. In Figure 3.9, there are two draggers for scaling and one for rotation. The rotation dragger has one degree of freedom (the angle of rotation) and the horizontal scaling dragger has two degrees of freedom (the scaling ratios of two perpendicular directions). The values of these draggers are directly mapped to the underlying domain polygon. For example, if the rotation manipulator rotates 90 degrees, the feature domain will also make a right angle rotation. The remaining height dragger has one degree of freedom; however, unlike the above two manipulators, its scaling ratio maps to the z-component of all the feature's CVs in the feature's local coordinate frame. Moreover, this height adjustment is permanent and an unpasting will not

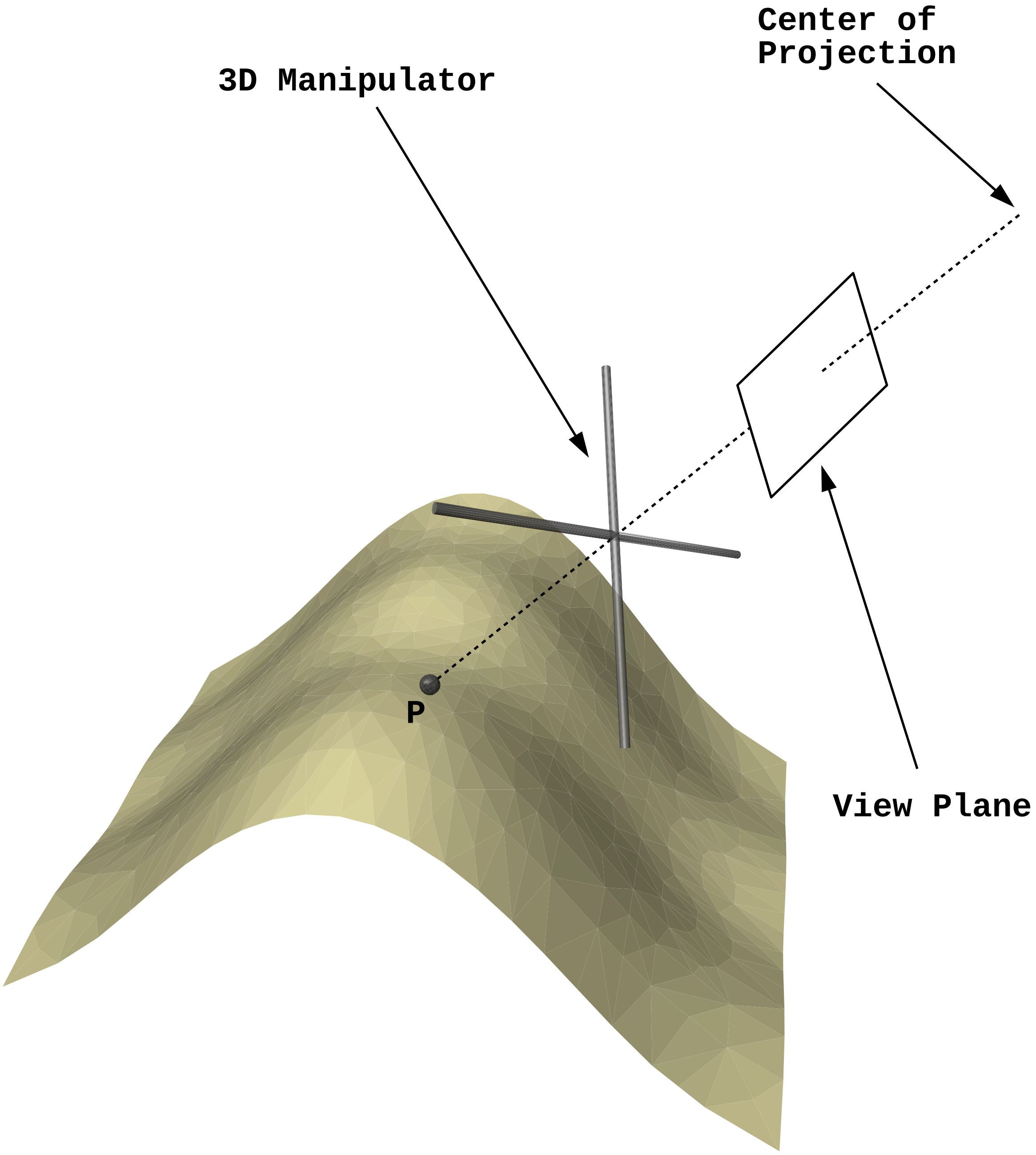


Figure 3.8: Using a 3D Manipulator to Pick a Point on the Surface

reset it.

It is important to put the manipulator in an appropriate position, orientation and scale for the user to view and relate the manipulator to the current feature: the center of the manipulator should be the center of the feature, the manipulator and the feature should face the same direction, and the manipulator should be big enough to surround the feature. In this case, the orientation is defined by the normal of the plane that best fits the four corner points of the feature. The location of the manipulator is a point at the center (on the base surface) of the four corner points with displacement x along the normal direction, where x equals the distance between the center point on the feature surface (the image of the feature domain center) and the plane. And finally the size of the manipulator matches the size of the feature's bounding box.

3.5 Fine Tuning

In order to fine tune the feature's location and appearance, an additional function is loaded into the four corner draggers (Figure 3.9). A user can translate a individual single corner by shift-clicking[4] a corner dragger, without moving the whole feature. However, the corner is not allowed to move out a region of validity as shown in the domain view of Figure 3.10, because the convexity of the feature domain has to be maintained. To help the user identifying the valid region, the region's boundary is drawn on the surfaces whenever the corner dragger is activated, as shown in the model view of the same figure.

[4] Clicking the dragger while pressing the shift key

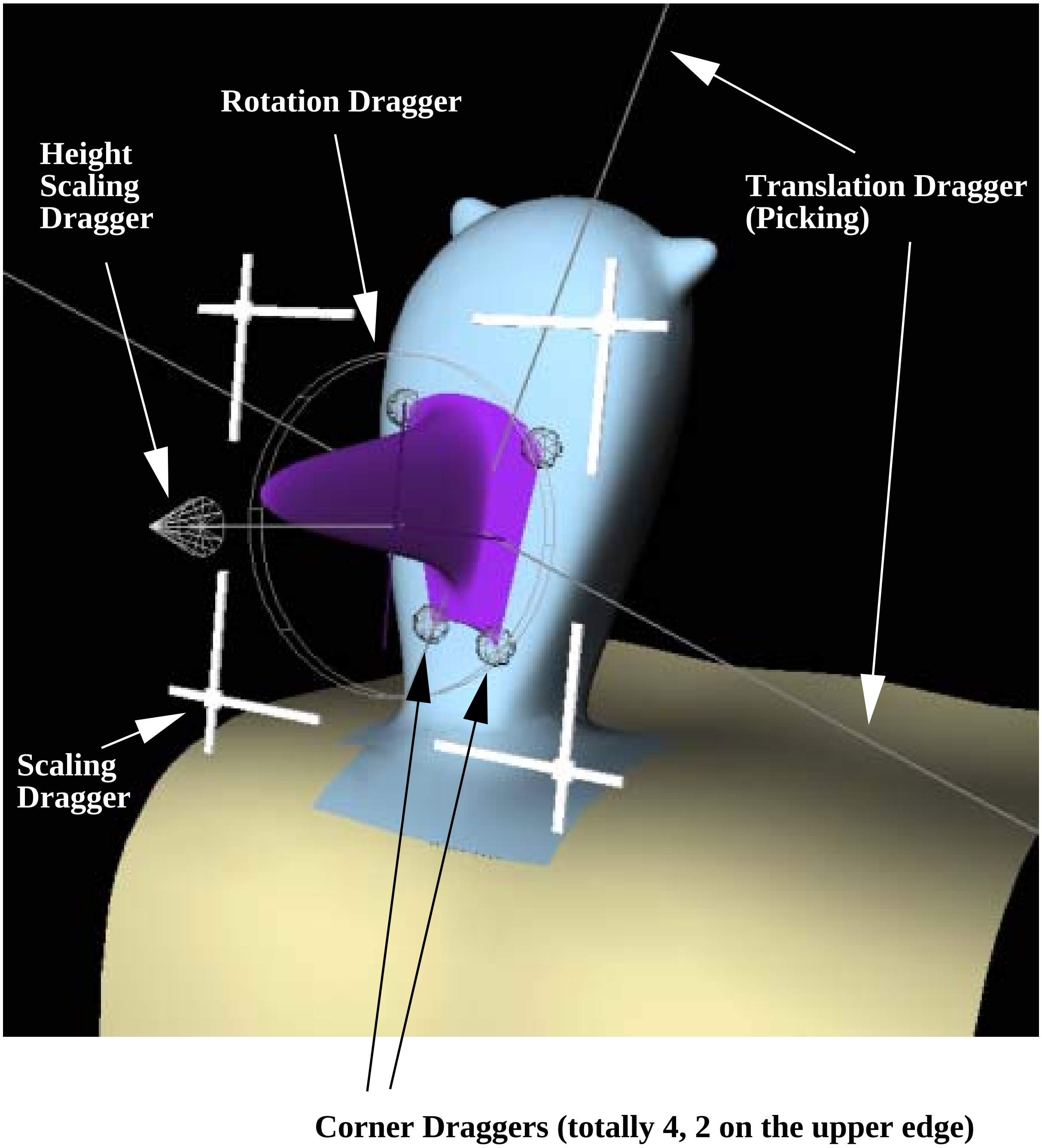

Rotation Dragger
Height Scaling Dragger
Translation Dragger (Picking)
Scaling Dragger
Corner Draggers (totally 4, 2 on the upper edge)

Figure 3.9: 3D Manipulators

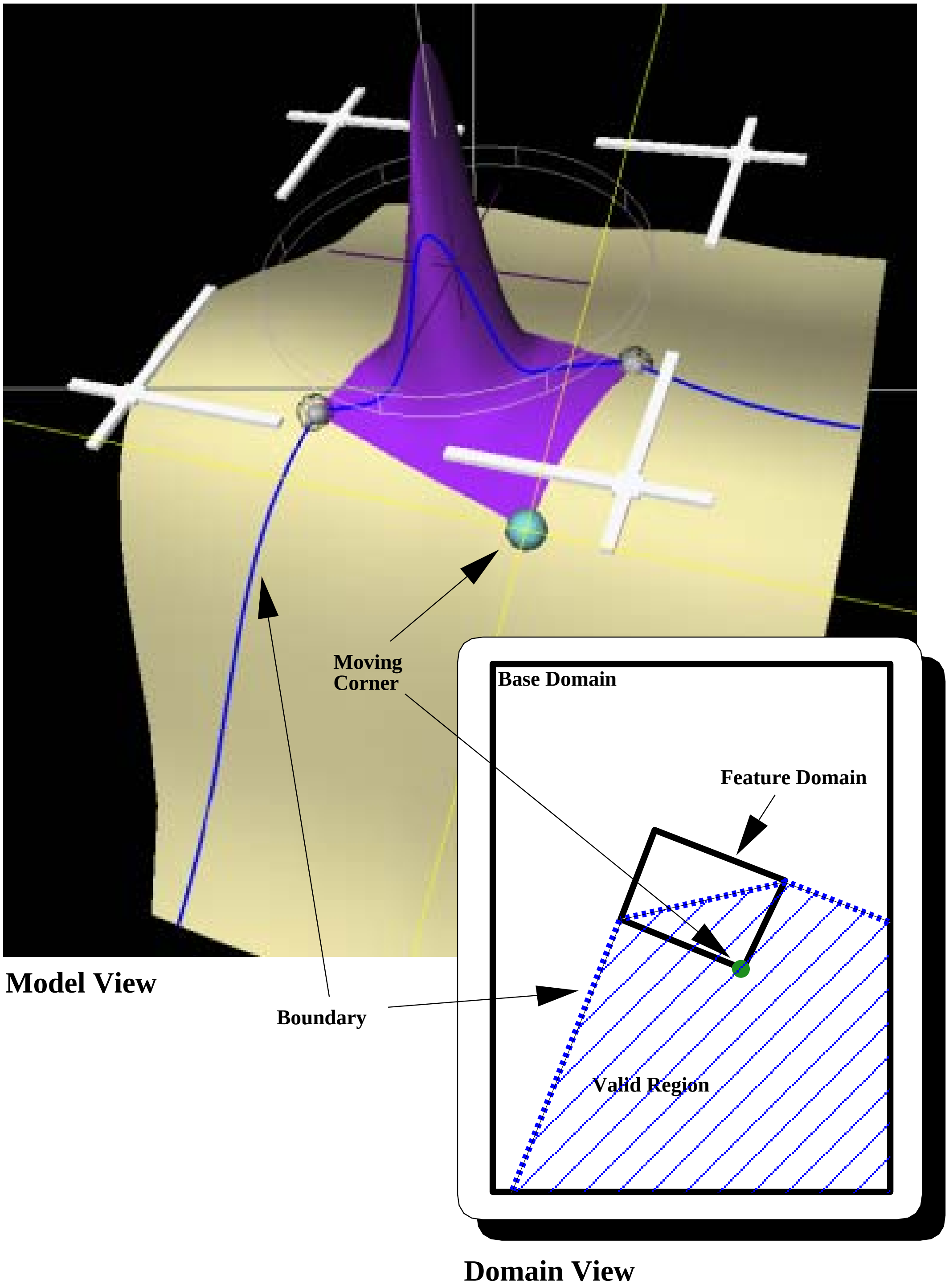


Figure 3.10: Translating a Corner of a Feature

Corner dragging enables the user to place individual corners at precise locations on the base surface, allowing the user to make changes to the feature than cannot be accomplished with only translation, rotation, and scaling.

The mechanism of these draggers is the same as the picking manipulator shown in Figure 3.8, except they move individual corners instead of the whole feature.

3.6 Grouping

The grouping operation relates surfaces in a composite surface together so that all operations such as rotation, translation and pasting/unpasting apply to all grouped surfaces at the same time with the same magnitude. Surface grouping is not only a convenient function, it is essential for pasting a composite surface (a set of pasted surfaces) to another base surface. For example, suppose a user has created a set of pasted surfaces of a head with nose and ears (composite surface), and wants to paste them on a body (a base surface). In order to paste all of the pasted surfaces on the new base surface, they have to be grouped and treated as a single unit.

Unpasting a group of surfaces results in another composite surface, and the dependency relationship between each group member is maintained. In order to create a new composite surface, we need to have a surface domain that is big enough to hold all of the other grouped surfaces' domains so that all other surfaces can become features of this new base surface. For this reason, there must be a surface in the group whose domain contains all other group members' domains, and therefore this condition has to be fulfilled when adding surfaces into a group.

Chapter 4

Implementation Details

I developed a program called *PasteInterface* to test the ideas and techniques mentioned in the previous chapter. In order to have a better understanding of PasteInterface, this chapter provides implementation detail and issues that have not been covered before.

4.1 Data Structure

PasteInterface is built upon two libraries: The Splines Library [VB92], which contains classes for surface pasting (SPaste), and Open Inventor [Wer94], which provides a high level rendering and 3D user interface toolkit. These two libraries have their own data structures, which are incompatible with each other. Therefore, PasteInterface has to maintain and connect these two sets of data. Figure 4.1 shows the combined data structure of the SPaste and Open Inventor. On the left, SPaste contains a dependency graph that keeps track of surfaces dependency (same

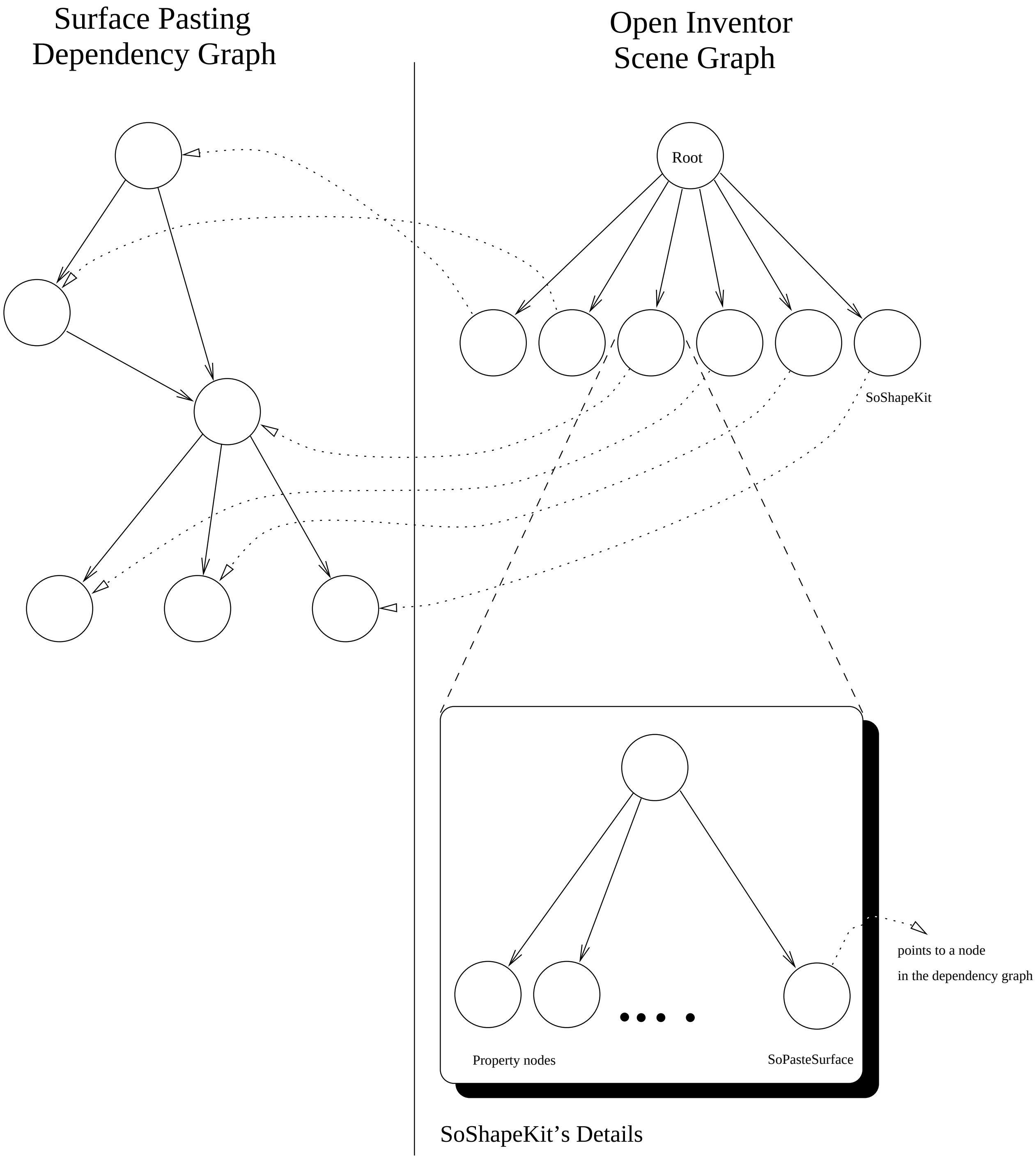


Figure 4.1: PasteInterface's Data Structure

as Figure 2.5), with each node representing a surface. On the right, Open Inventor maintains a scene graph that holds all of the shape nodes called SoShapeKit ('So' stands for scene node). Within each SoShapeKit, there are property nodes and a surface node. The property nodes define the appearance and properties of the surface, such as draw style, complexity, colour and pickablity. The surface node, called SoPasteSurface, is created by C++ inheritance from SoNurbsSurface (a NURBS surface node, one of many geometric objects provided by Open Inventor) and customized to surface pasting by containing a pointer to the corresponding surface node in SPaste's dependency graph. When a user picks on and translates a surface, the pointer acts as a channel to inform SPaste's dependency graph of the changes the user wants to make, and then updates the SoPasteSurface with the modified surface's control vertices. Moreover, a table containing the pairing information of these two graphs' nodes is also available for finding these scene graph's nodes from the dependency graph's nodes. This is useful if we want to update a set of surfaces for rendering by giving a set of nodes in the dependency graph.

4.2 Outline Mode

Performance issues arise when updating a grouped feature or a feature that has many dependent surfaces. For example, if a user picks a feature that has a number of dependent surfaces, the system has to update all of these surfaces while the user is translating/rotating/scaling the feature. This updating can be very slow. In order to get interactive speeds, an outline mode is provided. Under this mode, only the outline (four corners) of the selected feature is updated within a click-drag-release-cycle of projective-translation, rotation and scaling. The feature and related surfaces are updated only at the end after the user releases the button.

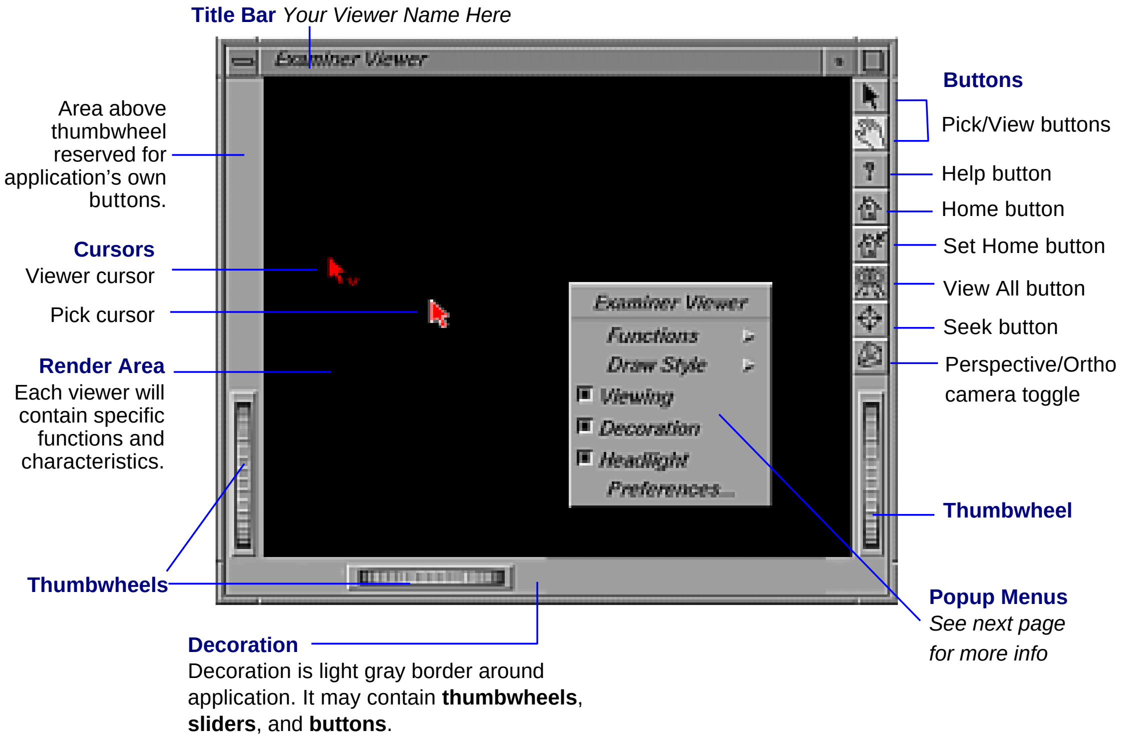
The performance problem is even worse for picking-translation (Section 3.3.2). In Open Inventor, ray-picking is a slow process. Translation would be too slow if we ray-pick during a drag-release-cycle to update the new position of the feature and show the outline to the user, and even slower if we update of the feature and its dependent surfaces. Therefore, ray-picking and surface updating has to be performed at the end of the drag-release-cycle. Fortunately, the visual feedback in picking-translation is not as important as in projective-translation because the user knows precisely where the feature will go – wherever the mouse cursor points to.

4.3 User Interface

PasteInterface uses Open Inventor's examiner viewer to provide a model viewing environment. Examiner viewer provides viewing facilities such as virtual trackball, dolly and panning, as illustrated in Figures 4.2 and 4.3. These figures are copies of help screens from Open Inventor (the examiner viewer in Figure 4.3 is an extension of the base viewer in Figure 4.2). The two most frequently used buttons are at the top right corner of Figure 4.2: The pick and view buttons. The pick button changes the system to picking mode so that a user can pick on the 3D manipulators and interact with the model. The view button sets the current state to be viewing mode to allow a user changing the viewing parameters of the camera (e.g., zooming, panning, virtual track ball). A user can also press 'ESC' key to toggle between these two modes.

Anatomy of a Viewer

General Keyboard Use
'home' key for Home function (Reset to Home View)
'Esc' key toggles between viewing/picking
'Alt' key puts the viewer into temporary VIEW mode (while down)
's' key toggles Seek function On/Off
Arrow Keys for translation of camera in viewer plane: Up, Down, Right, Left



Title Bar *Your Viewer Name Here*

Area above thumbwheel reserved for application's own buttons.

Cursors
Viewer cursor
Pick cursor

Render Area
Each viewer will contain specific functions and characteristics.

Thumbwheels

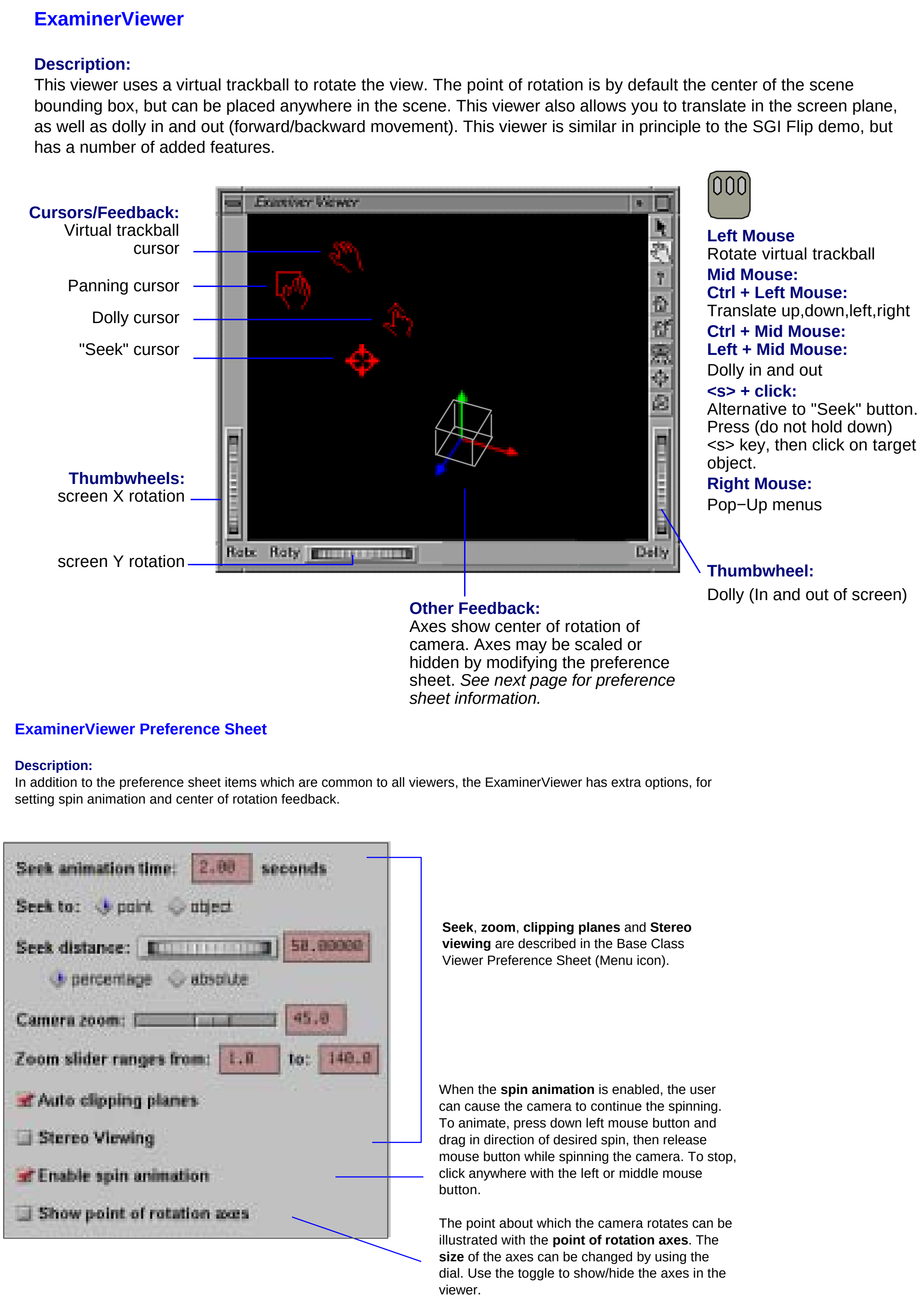
Buttons
Pick/View buttons
Help button
Home button
Set Home button
View All button
Seek button
Perspective/Ortho camera toggle

Thumbwheel

Popup Menus
See next page for more info

Decoration
Decoration is light gray border around application. It may contain **thumbwheels**, **sliders**, and **buttons**.

Figure 4.2: Inventor's Base Viewer (From the Help Menu of Open Inventor)



ExaminerViewer

Description:
This viewer uses a virtual trackball to rotate the view. The point of rotation is by default the center of the scene bounding box, but can be placed anywhere in the scene. This viewer also allows you to translate in the screen plane, as well as dolly in and out (forward/backward movement). This viewer is similar in principle to the SGI Flip demo, but has a number of added features.

Cursors/Feedback:
Virtual trackball cursor
Panning cursor
Dolly cursor
"Seek" cursor

Thumbwheels:
screen X rotation
screen Y rotation

Left Mouse
Rotate virtual trackball
Mid Mouse:
Ctrl + Left Mouse:
Translate up,down,left,right
Ctrl + Mid Mouse:
Left + Mid Mouse:
Dolly in and out
<s> + click:
Alternative to "Seek" button. Press (do not hold down) <s> key, then click on target object.
Right Mouse:
Pop−Up menus

Thumbwheel:
Dolly (In and out of screen)

Other Feedback:
Axes show center of rotation of camera. Axes may be scaled or hidden by modifying the preference sheet. *See next page for preference sheet information.*

ExaminerViewer Preference Sheet

Description:
In addition to the preference sheet items which are common to all viewers, the ExaminerViewer has extra options, for setting spin animation and center of rotation feedback.

Seek animation time: 2.00 seconds
Seek to: ● point ○ object
Seek distance: 50.00000
● percentage ○ absolute
Camera zoom: 45.0
Zoom slider ranges from: 1.0 to: 140.0
☑ Auto clipping planes
□ Stereo Viewing
☑ Enable spin animation
□ Show point of rotation axes

Seek, **zoom**, **clipping planes** and **Stereo viewing** are described in the Base Class Viewer Preference Sheet (Menu icon).

When the **spin animation** is enabled, the user can cause the camera to continue the spinning. To animate, press down left mouse button and drag in direction of desired spin, then release mouse button while spinning the camera. To stop, click anywhere with the left or middle mouse button.

The point about which the camera rotates can be illustrated with the **point of rotation axes**. The **size** of the axes can be changed by using the dial. Use the toggle to show/hide the axes in the viewer.

Figure 4.3: Inventor's Examiner Viewer (From the Help Menu of Open Inventor)

In addition to the viewer, PasteInterface has its own buttons and pulldown menu, as shown in Figure 4.4. The existence of buttons is for convenience, as all of their operations can be activated through the pull down menu. The first two buttons are for pasting and unpasting of the selected feature. In order to select a feature, a user should first click on the "select feature" button, and then click on a feature surface. Then the feature will be selected and automatically highlighted. The "Select base" button is used for selecting a base surface so that all other feature surfaces will paste on it.

All pull down menu items are described in the following table:

Pull Down Menu	*Items*	*Functions*
File	Open..	Pop up a file selection dialog box to open or import a file
	Save..	Pop up a file selection dialog box to save current surfaces to a file
	Export Inventor Format..	Export the current surfaces to Open Inventor file format
	Quit	Quit the system
Edit	Start Grouping	Start grouping a set of surfaces by clicking them one by one
	End Grouping	Group the selected surfaces
	UnGroup	UnGroup the current selected group of surfaces

	Delete Selected	Delete the selected surface or selected group of surfaces
	Delete All	Delete all surfaces
Selection Mode	Base	Switch to base selection mode (same as “Select Base” button)
	Feature	Switch to feature selection mode (same as “Select Feature” button)
	None	Switch to none selection mode so that the user can interact with 3D manipulators (automatically switch to this mode after selecting a base or feature)
Pasting	Paste	Paste the selected feature to the base surface (same as “Paste” button)
	Unpaste	Unpaste the selected feature from the base surface (same as “Unpaste” button)
Surface	Surface Properties..	Pop up the surface properties dialog box (Figure 3.1)
Option	Domain Window	Pop up the domain window which shows the polygonal view of all surface domains of the active composite surface

	Outline Mode	If it is ON, outline of the feature will be updated within a click-drag-release cycle of translation/scaling/rotation. Otherwise, the feature will be updated in real time (slow)
	Show Projection	If it is ON, show the projection (preview location) of the selected unpasted feature on the base surface
	Trim Surfaces	It it is ON, trim the regions on base surfaces that covered by other surfaces. It only supports simple trimming (refer to Section 5.2.5 for details).

Notice that the domain viewer under the option menu is provided for debugging purposes and for the expert user who wants to know what happened behind the scene. It uses the standard examiner viewer from Open Inventor so that the user can use facilities such as zooming and panning.

To activate the projective-translation (Section 3.3.1) on the selected feature, the user can press the middle mouse button and slide the mouse.

The grouping and ungrouping operation is a single-level operation, i.e., ungrouping always gives the user ungrouped individual surfaces, no matter how the user grouped the surfaces before.

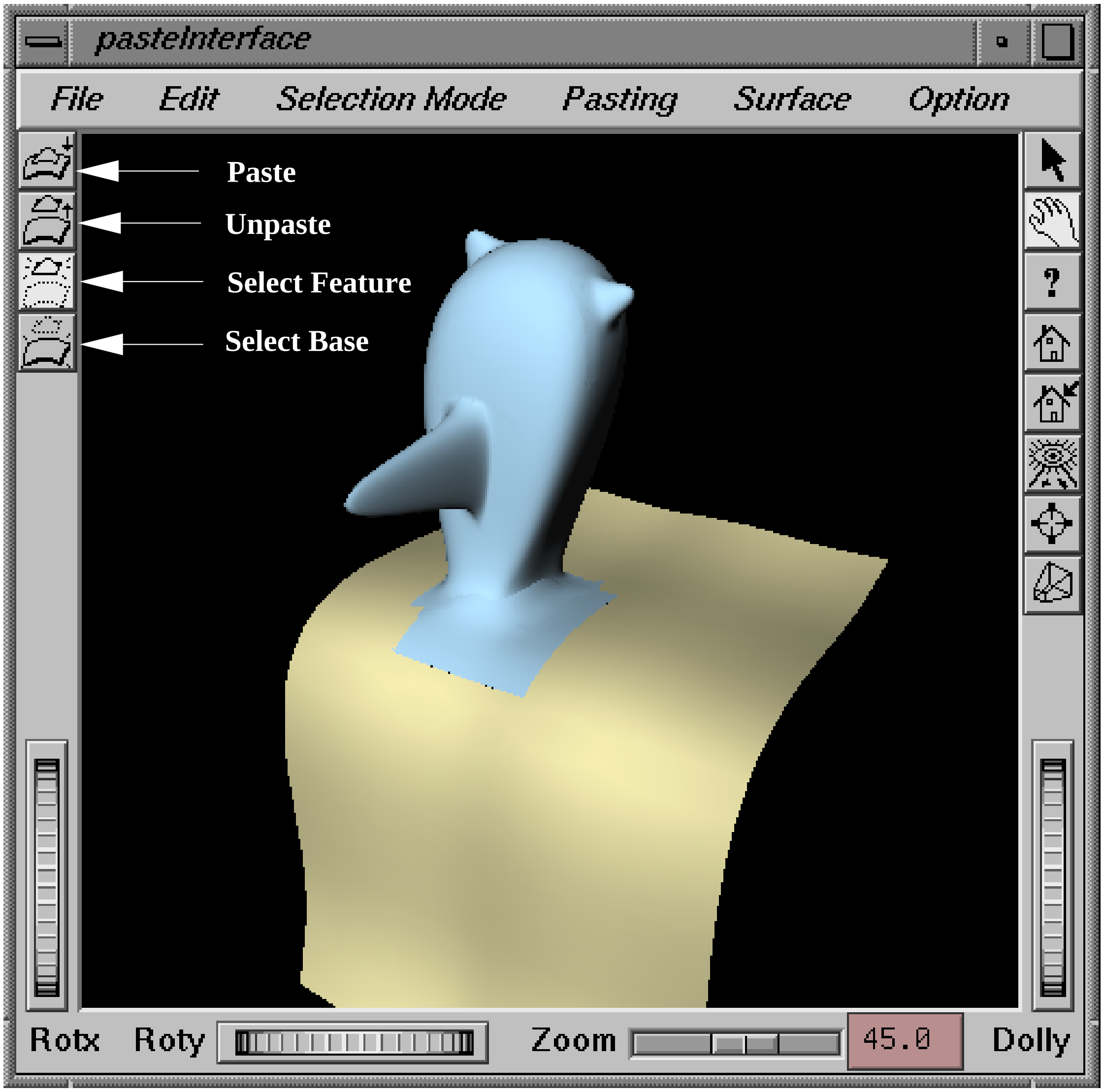


Figure 4.4: PasteInterface

4.3.1 Selecting the Base Surface

Although multiple composite surfaces can exist at the same time, there can be only one active base surface at a time in the system. If the user wants to manipulate the features of a composite surface, the base surface of this composite surface must be the active base surface. To select a new base surface, the user first has to press the "select base" button to change selection mode. Pressing this button groups the active composite surface (the composite surface that contains the active base surface) and unsets the current active base surface. Then the user can select any composite surface to set its base surface to be the active base surface.

However, there can be no transformations applied to the active composite surface due to the design of the system (in Section 5.2.4, I discuss what is required to allow transformations). Therefore, if the selected composite surface has previously been translated/rotated/scaled, the system will pop up a message box to ask for the permission of untransforming the composite surface. The user can select "OK" to proceed or "Cancel" to abort base surface selection.

4.4 File I/O

Surfaces may be written to a file whose format consists of the pasting order of all surfaces, their knot vectors and CVs, and their domain transformation. To rebuild the composite surface from the file, the program follows the pasting order to transform each surface's domain (except the base) and paste it to the base surface.

Chapter 5

Evaluation

PasteInterface was developed and tested on a SGI Onyx machine. With the help of the outline mode, the interaction of the world space user interface is smooth enough to allow the user to manipulate composite surfaces interactively, in real time. Surface updating occurs at the end of each manipulation and therefore the potential performance problem of updating speed due to moving grouped surfaces or moving a surface that has many dependent surfaces is minimized.

Based on the implementations of PasteInterface and PasteMaker, this chapter evaluates the world space user interface. First it compares the world space user interface and domain space user interface, and then it points out some limitations of the design of the world space user interface.

5.1 World Space User Interface Vs Domain Space User Interface

Although no user experiments have been conducted to compare PasteInterface and PasteMaker, the world space user interface is better than the domain space user interface in several ways, including:

- Providing all domain operations. The world space user interface performs all domains operations provided by the domain space user interface. These are pasting/unpasting, translation, rotation and scaling.

- Working with 3D model directly. The world space user interface allows a user to interact with the composite surface directly, without having to look at the domains layout. The domain space user interface forces the user to work in the domain space, even though the user may be only interested in the 3D model.

- Corner draggers. Corner dragging enables the user to place individual corners at precise locations on the base surface, allowing the user to make changes to the feature that cannot be accomplished with only translation, rotation, and scaling. Corner dragging appears to give the user significantly more control over the shape and location of the features.

- Precise pasting and translation. The world space user interface allows the user to position a feature on the base surface precisely in three dimensions when pasting or translating a feature, by using a variety of projection and picking techniques. On the other hand, in the domain space user interface, the user has to first try moving the feature domain in the domain window, and then see where the corresponding feature surface has translated in three-dimensional space. Since the moving direction of feature domain and the feature surface is not necessary the same in the screen, the user has to perform the above trial-and-error method many times until the feature surface arrives the desired position.

- 3D manipulators. Use of 3D manipulators in the world space user interface reduces the number of buttons or menu items that would have been used to change the system state to translation/rotation/scaling mode. Moreover, it allows the user to activate operations without clicking on the menus or buttons, so that the user doesn't need to shift his/her attention from the 3D model.

- Simplified user interface. The world space user interface has a much simpler layout than the domain space user interface. Without the domain window and with the menu items or buttons replaced by 3D manipulators, the user will not be confused by the domain window and too many menu items or buttons. Therefore, the world space user interface is easier to learn than the domain space user interface.

- Better User Performance. PasteInterface has roughly the same speed as PasteMaker when translating a pasted feature on a base surface. However, in PasteInterface, the user can specify a rough location on the base surface in three-dimensional space when pasting the feature. After pasting, the user only need to fine tune the position or the size of the pasted feature. On the other hand, in PasteMaker, the user can only specify the domain location on the base domain when pasting a feature. After pasting, the feature is not necessary close to the desired three-dimensional location on the base surface (since it is not easy to guess the two-dimensional domain location from the position on the three-dimensional surface). Then the user has to waste time on translating/rotating/scaling the feature back to the desired position using domain operations. Moreover, the outline mode provided in PasteInterface also speeds up the feature manipulation. As a result, the user in PasteInterface should spend less time than in PasteMaker in doing the same task.

Notice that some of the above differences are due to the implementation of PasteMaker (too many buttons resulting in a crowded interface and needing to click multiple buttons to manipulate features). We could build a better and more natural domain user interface by, for example, taking advantage of Open Inventor's manipulators and viewer. However, even in an improved domain user interface, the user still cannot work on the 3D composite surfaces directly and hence the inherent problem of the domain space user interface remains.

Although the world space user interface is more natural to use than the domain space user interface, it is unclear if this results in higher speed or better accuracy for various tasks. Experiments on user testing would be needed to determine this, but it is likely that the domain space user interface would be better for domain based operations (e.g., placing two features adjacent to one another) while the world space user interface would be better for world space based operations (e.g., positioning a feature at a particular bump on the base surface).

5.2 Limitations and Extensions

Although the world space user interface has several advantages over the domain user interface, it has some limitations. This section discusses the limitations and suggests ways for improvement.

5.2.1 Translation of a Pasted Feature

There are two translation modes in the world space user interface: projective-translation and picking-translation. Both of them have their limitations and strengths. Projective-translation works well if the user starts moving the feature on a relatively flat portion of the composite surface. However, if the feature starts on a bumpy surface, the user has to guess the direction of the sampled tangent plane underlying the feature since it may not reflect the orientation of the feature accurately (refer to Section 3.3.1 for more detail), and therefore the feature may move in the reverse direction to the one that the user intended. On the other hand, picking-translation has no such problem because the translation is done by picking and is independent of the shape of the composite surface. However, the limitation of the picking-translation is that the user cannot use it to move the feature to the back of the composite surface, as the projective-translation does. Moreover, unlike picking-translation, projective-translation shows the outline (in outline mode) of the feature continuously with the mouse motion before committing the translation. The outline is very helpful for previewing the final shape of the feature after translation.

As a result, projective-translation is good for moving a feature from a relatively flat surface and for fine tuning the feature's position with its previewing capacity. Picking translation, on the other hand, has a consistent behavior when moving a feature on any surface shape, with the drawbacks of limited mobility and the lack of previewing capability. In fact, the original intention of providing these two translation mechanisms was to have them compensate each other's weakness. However, it is unclear whether a general user can use these two techniques interchangeably in an effective and natural manner because they require substantially different mental models and provide different feedback. More research on the human computer interaction aspect of this user interface has to be done.

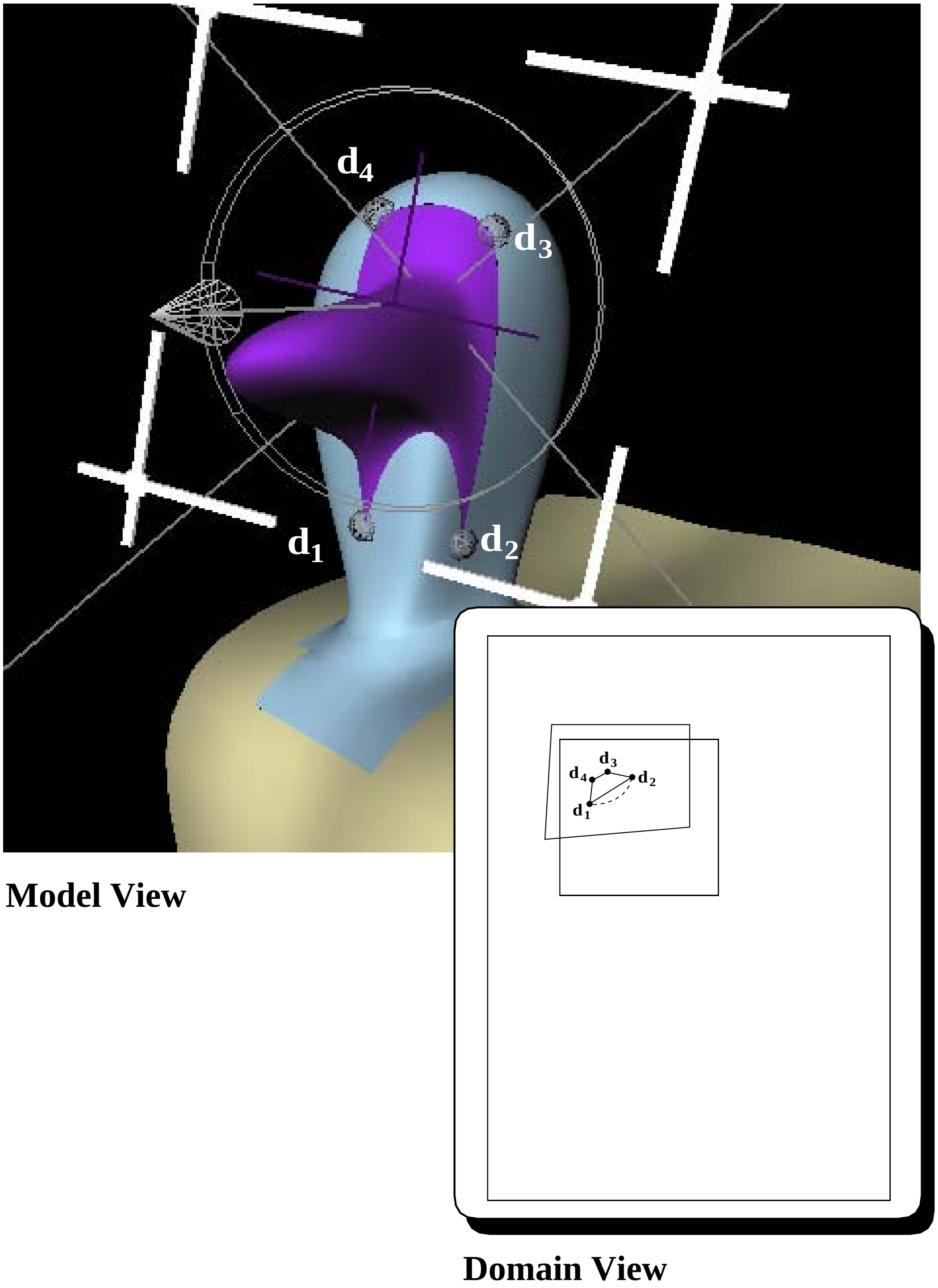
5.2.2 Rotation and Scaling

In PasteInterface, the rotation degree and scaling factor of the corresponding draggers are directly mapped to the feature domain's rotation and scaling (Section 3.4). However, this does not guarantee that the three-dimensional feature surface rotates or scales with the same degree. For example, rotating the rotation dragger 90 degree causes a right angle rotation of the feature domain, but the feature surface may not rotate exactly 90 degree, the rotation effect of the feature surface also depends on the shape of the base surface. More advance mapping scheme will help synchronizing draggers and feature surface's movements more accurately (for example, sampling two non-parallel directional derivatives can help making the rotation more accurate). However, from the user's perspective, when the user is manipulating the rotation or scaling dragger, it is very likely that he/she focuses on the outline feedback and stops manipulation when he/she is satisfied with the outline shape. Therefore, the accurate synchronization between draggers and feature surface becomes less important as long as their movement are roughly the same.

5.2.3 Domain Transformation

Currently a bi-linear transformation is used to transform the feature domain. It maps a feature's rectangular domain polygon to a convex quadrilateral. We can control the location of the quadrilateral's corner points in order to change the shape of the feature surface. However, the effectiveness of moving four corner points is limited. In Figure 5.1, the highlighted feature in model view has the domain $d_1d_2d_3d_4$ in domain view. Because of the curvature of the base surface, one edge (d_1d_2) of the feature surface is highly curved. If we want to straighten this surface edge, we have to change the domain edge d_1d_2 to be the dotted line shown in the domain view. Unfortunately, this is impossible with bi-linear transformations because all the edges must be straight lines in the domain.

To solve this problem, we can sample more edge points of the domain and then find another invertible transformation method to transform all the points in the original domain polygon to the target polygon, as shown in Figure 5.2b (Figure 5.2a is the bi-linear transformation). Or, more ideally, all the edges of the domain polygon can map to spline curves, which are controlled by a number of control vertices, as shown in Figure 5.2c. However, this method needs a sophisticated



Model View

Domain View

Figure 5.1: Straight Domain Edge Resulting in Curved Surface Edge

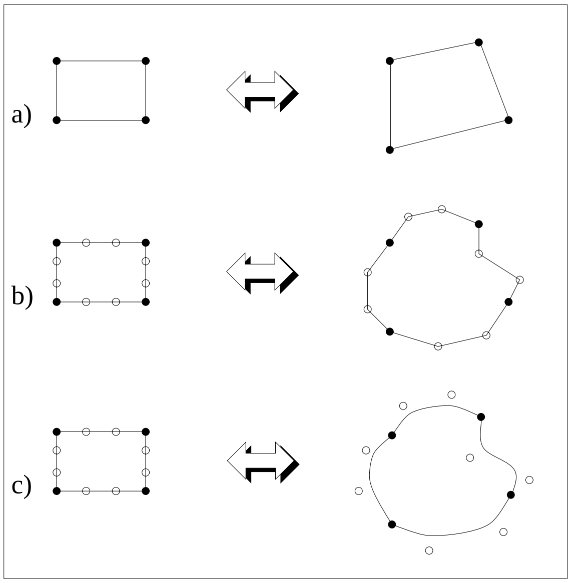


Figure 5.2: Different Domain Transformations

invertible transformation algorithm to transform all domain points between the original rectangular domain to the target curved domain.

5.2.4 Base Surface

In PasteInterface, the user is not allowed to transform the active composite surface, and if the user selects a new base surface from a transformed composite surface, the composite surface has to be untransformed (Section 4.3.1). This is the result of an assumption I made in the early state of development that the composite surface will not be transformed.

If we want to have the capability of transforming an active composite surface, we need to have a tighter integration of data structures than the one shown in Section 4.1 of Chapter 4. The Open Inventor scene graph should represent the base-feature

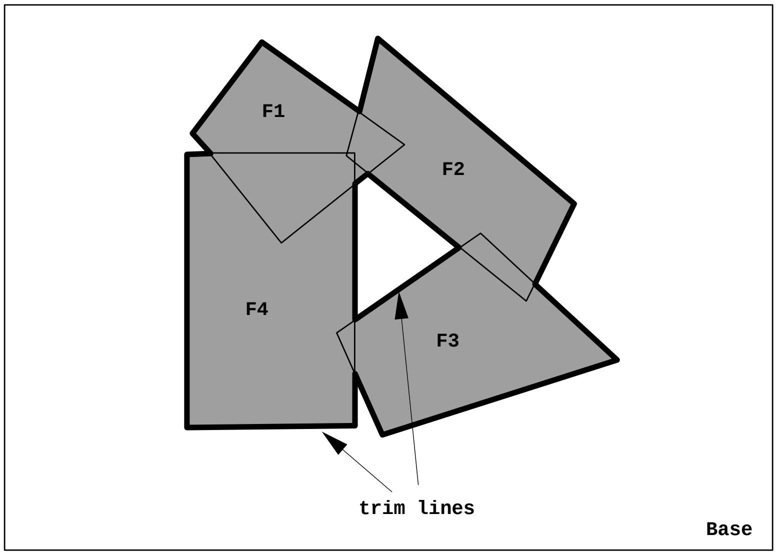


Figure 5.3: Trimming a Base Surface

relationship of all the surfaces so that if the user transforms a base surface, all its features will be transformed as well.

5.2.5 Surface Trimming

Surface trimming is the ability to cut out the region on the base surface that is covered by features. This ability allows us to see a depression created by a feature (as opposed to a bump) without being blocked by the base surface. Figure 5.3 shows a base domain with four feature domains overlapping each other. The shaded area is the trim region that we want to cut from the base surface. To specify this trim region in the Open Inventor, we need to create two trim lines and pass them to the Open Inventor (we cannot just pass four domain polygons because Open Inventor required non-overlapping trim region). However, finding the correct trim lines requires a polygon intersection algorithm [NP82] that has not been implemented in PasteInterface. Instead, PasteInterface supports simple trimming that only trims the base surface when the feature domain is totally contained in the base domain and does not intersect other feature domains, which does not require finding polygon intersections.

5.2.6 Input Device

The current input device used in PasteInterface is a two dimensional mouse. All the operations are activated with it. Although the surface model is in three space, with the help of 3D manipulators (such as the transformation box), this inherently two dimensional device is sufficient for manipulating surfaces in three dimensional space. Therefore, using three dimensional input device probably will not bring a lot of improvement in the user interface. However, a haptic computer interface[1] may bring a significant improvement on the translation of the pasted feature because the user can feel the shape of the base surface and move the feature around as if there is a real model sitting in front of the user.

[1] A haptic computer interface allows the user to feel virtual objects. By exerting an external force on the user's finger tips, the interface creates the illusion of interactions with solid virtual objects.

Chapter 6

Conclusion

Surface pasting is a composition method that applies B-spline surfaces, called features, to any number of base surfaces. The locations as well as the size of the features are determined by the transformations of feature domains. By modifying the domain layout of pasted surfaces, we can manipulate the appearances of features interactively in a *Domain Space User Interface.* This thesis has argued that the domain space user interface is inadequate and presented a *world space user interface* that allows a user to directly manipulate the three-dimensional composite surfaces interactively. By using a variety of techniques like projective-translation and the use of 3D manipulators, all the user's actions on the 3D models are mapped to underlying two-dimensional domain operations. Therefore, the user does not need to pay attention to the two-dimensional domain layout and can instead concentrate on modeling in a three-dimensional environment. Since the user can specify the pasting or translation location on the three-dimensional composite surface precisely without using the trial-and-error methodology as in the domain space user interface, user performance is actually improved. As a result, the world space user interface provides a more intuitive and efficient modeling interface for surface pasting.

6.1 Future Work

Future work can go into two directions: finding a better world space user interface and finding a way to create better composite surfaces.

Since my research focuses on the mathematical aspect of mapping three-dimensional operations into two-dimensional domain functions, no user interface experiment has been conducted. The look and feel of current user interface is only based on the advise of a few users. Further research should focus on human computer interactions (e.g., the issue discussed in Section 5.2.1) with usability analysis and testing with real users and aim for a user-oriented world space interface.

Another direction could be in improving the surface quality. Surface pasting is only an approximation technique. It optimizes performance by trading off surface quality. Irregularities in the approximation may also arise from the different tensor-product alignment of the overlapping surfaces. For example, A bicubic surface has a sixth degree behavior unless the surfaces are equiparametic. A feature's margin varies as a cubic, while its base varies as a sixth degree. Thus, there may be noticeable mismatches along the junctions between the feature and the base surface, especially if the base surface has high curvature around the junction. Therefore, it would be preferable to have a post-processor that takes the information of pasted surfaces created from an efficient approximated modeler like PasteInterface and creates high quality composite surfaces (particularly along the margins of features).

This post-processor can optimize the surface quality rather than the performance because it does not need to be interactive, although how to improve the surface quality is still unclear.

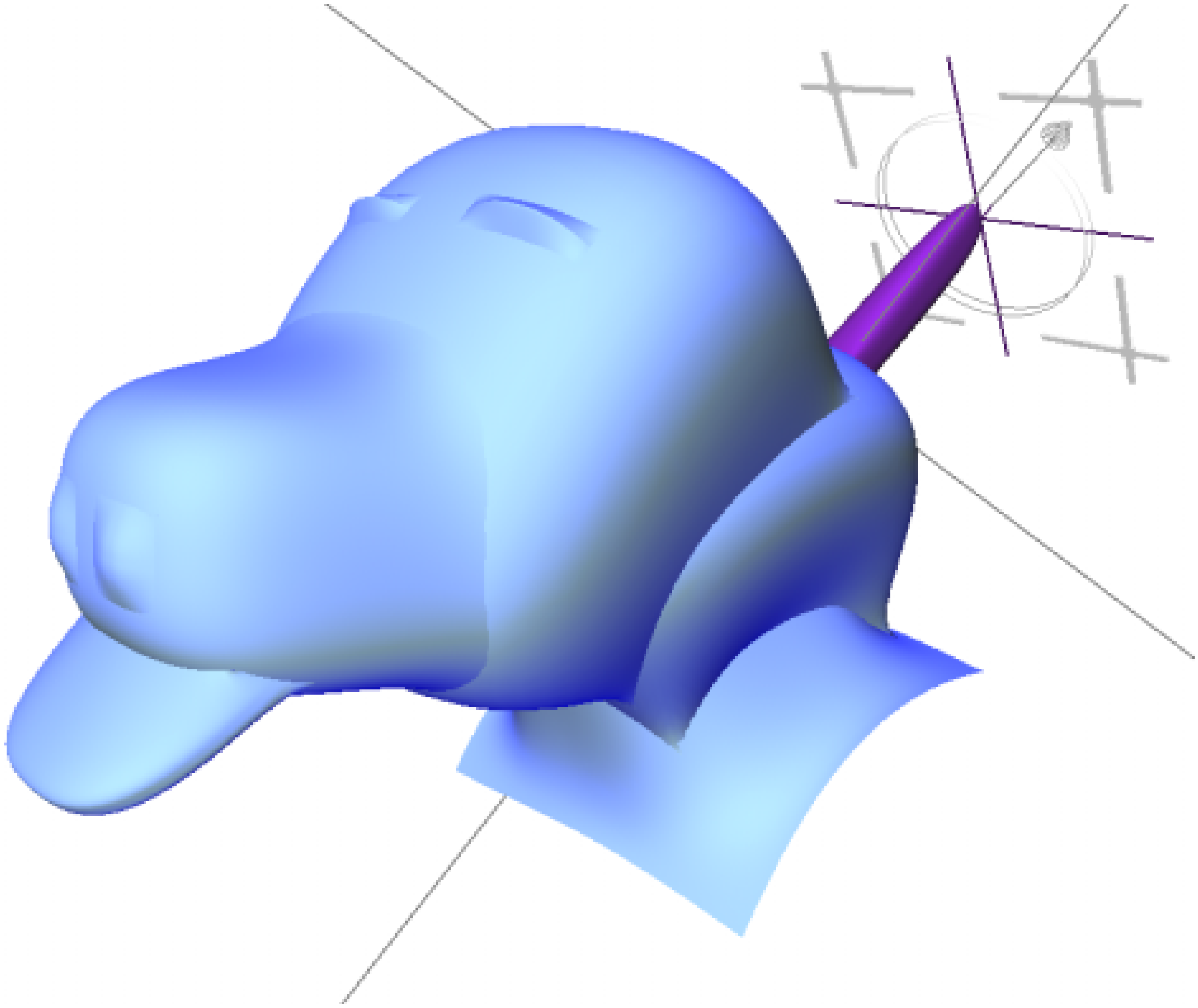


Figure 6.1: A Model Created Using PasteInterface

Bibliography

[B́66] P. Bézier. Définition numérique des courbes et surfaces i. *Automatisme*, XI:625–632, 1966.

[Bar94] C. Barghiel. Feature oriented composition of b-spline surfaces. Master's thesis, Univeristy of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1994. (Available as Computer Science Department Technical Report CS-94-13).

[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.

[BBF95] C. Barghiel, R. Bartels, and D. Forsey. Pasting spline surfaces. In L. Schumaker M Daehlen, T. Lyche, editor, *Mathematical Methods for Curves and Surfaces*, pages 31–40. Vanderbilt University Press, 1995.

[BF91] R. Bartels and D. Forsey. Spline overlay surfaces. Technical Report CS-92-08, Univeristy of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1991.

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

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

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

[FB88] D. Forsey and R. Bartels. Hierarchical b-spline refinement. *Computer Graphics*, 22(4):205–212, august 1988.

[NP82] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting germetric figures. *Communications of the ACM*, 25(10):739–747, October 1982.

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

[Wer94] J. Wernecke. *Programming Object-Oriented 3D Graphics with Open Inventor.* Addison-Wesley Publishing Company, 1994.