Visual Interfaces for Solids Modeling*

Cindy Grimm[†](cmg@cs.brown.edu)
David Pugmire[‡](dpugmire@cs.utah.edu)
Mark Bloomenthal[‡] John Hughes[†] Elaine Cohen[‡]

ABSTRACT

This paper explores the use of visual operators for solids modeling. We focus on designing interfaces for free-form operators such as blends, sweeps, and deformations, because these operators have a large number of interacting parameters whose effects are often determined by an underlying parameterization. In this type of interactive modeling good solutions to the design problem have aesthetic as well as engineering components.

Traditionally, interaction with the parameters of these operators has been through text editors, curve editors, or trial-and-error with a slider bar. Parametric values have been estimated from data, but not interactively. These parameters are usually one- or two-dimensional, but the operators themselves are intrinsically three-dimensional in that they are used to model surfaces visualized in 3D. The traditional textual style of interaction is tedious and interposes a level of abstraction between the parameters and the resulting surface. A 3D visual interface has the potential to reduce or eliminate these problems by combining parameters and representing them with a higher-level visual tool. The visual tools we present not only speed up the process of determining good parameter values but also provide visual interactions that are either independent of the particular parameterizations or make explicit the effect of the parameterizations. Additionally, these tools can be manipulated in the same 3D space as the surfaces produced by the operators, supporting quick, interactive exploration of the large design space of these free-form operators.

This paper discusses the difficulties in creating a coherent user interface for interactive modeling. To this end we present four principles for designing visual operators, using several free-form visual operators as concrete examples.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, Curve, surface, solid, and object representations, Splines;

Additional Keywords: User Interfaces.

1 Introduction

Modeling free-form surfaces is a difficult problem that has attracted a good deal of attention. The difficulty has two primary sources: the mathematics for describing surface types that are sufficiently general to be termed "free-form" is often complicated, and the design space – the set of all possible surfaces that can be made from such a description – is often so huge that the task of selecting an element of this space (or even of narrowing down to subsets of the space that converge on a desired element) is extremely difficult.

Usually the latter problem is addressed in part by providing a *user interface* whose job is to help a user define an element of the design space; such interfaces are often only thinly disguised editors of the parameters of the original mathematical description. Examples are control-point manipulation methods for free-form curves and surfaces and "tension" and "bias" editors for various spline types. While in many cases these parameters have natural geometric interpretations, they may not represent the characteristics that a user wishes to adjust: all too often, the user says "I want *this* basic shape, but I want *this* point to be just a little further over *there*." *Interactive* interfaces – ones providing rapid feedback – have evolved to fill this need, in particular interfaces that let the user interact directly with curves and surfaces instead of their control points.

Building interactive interfaces that support users' goals requires an intimate understanding of the parameters of the design space and their influence on the resulting model. Interfaces that exist in the same 3D world as the surface being modeled have the advantage that the user may get the sense of "shaping the surface directly." Furthermore, 3D interfaces can provide "coordinate-free" interaction, which may match a user's expectations, especially in the context of free-form shape development. Finally, the *interactivity* of an interface can exploit the users' ability to generalize by letting them try small variations of a model and thereby predict what larger changes will generate. This can help give users an intuitive feel for the large design space and the tools that are provided for navigating within the space.

This paper first describes some concrete examples of interactive interfaces for free-form modeling operations; the underlying operations act on spline surfaces. We abstract our experience in building these interfaces to give guidelines for developing "visual tools." We also discuss how the development of such visual tools actually provides feedback into the realm of operator design: a sufficiently powerful interface to an operator may suggest the need for a different operator with enriched expressiveness. Thus the entire process of designing interfaces to free-form modeling becomes not only an interface problem but an operator design problem as well.

*NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219)

†Department of Computer Science, Box 1910, Brown University, Providence, RI 02912. This work was supported in part by grants from Sun Microsystems, ONR Grant N00014-91-J-4052 ARPA Order 8225, NCR, Autodesk, Taco Inc., NASA Apple Computer, ARPA Microsoft

‡Department of Computer Science, University of Utah, Salt Lake City, UT 84112. This work was supported in part by N00014-92-J-4113 the opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

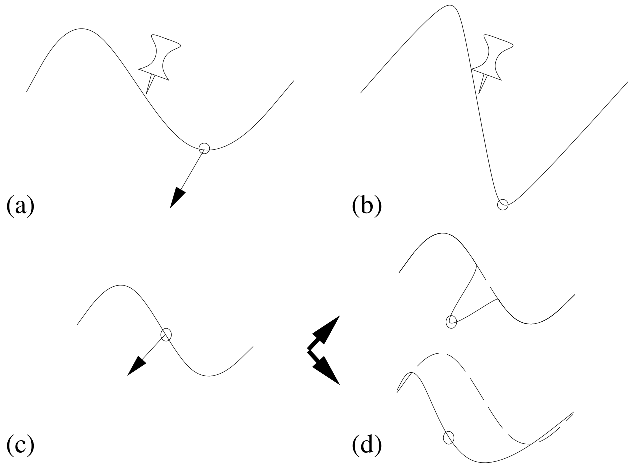


Figure 1: (a) A curve with one point pinned and a vector showing the desired movement. (b) The resulting solution. (c) Curve with desired move vector. (d) Result with two different parameterizations.

We begin by discussing previous work in 3D interaction and modeling. We then discuss in Section 2.1 the prototyping system we use to create and test our visual interfaces. In Section 3 we describe the sweep, warp (a deformation), and blend operators and the interfaces we have created for manipulating them. Section 4 presents four goals appropriate in creating visual interfaces, using the previously discussed visual interfaces as examples. In the final section, Section 5, we discuss possible avenues for future work. An appendix gives some of the mathematics necessary for communication between the visual interface and the underlying operators.

2 Related work

We examine here the two primary methods for interactive operator specification. The first method focuses on direct manipulation of a curve or surface, while the second method involves specifying operators using a geometric object that represents the operation. The work in this paper builds on the latter method.

The direct manipulation of curves and surfaces began with a "push-pull" metaphor in which an arbitrary point on the curve is selected and then dragged using the mouse to another location [FB91] [Fow92]. Extensions to this work include adding other geometric constraints, such as requiring that a particular point of the curve remain fixed, or adjusting the tangents as well as the positions of the curve [WW92]. This form of manipulation simplifies the task of shaping a curve or surface by hiding the dependency upon the control points, but has two main drawbacks that have not been dealt with. The first is that the constraints may lead to some unexpected results. For instance, if the middle of the curve is pinned to a particular location and a nearby point is moved, the curve on the other side of the pinned point may also move (see Figures 1a and 1b). The second problem is that the underlying parameterization of the manipulated object determines the behavior of the manipulation. This is because most direct manipulation techniques are implemented by finding control-point configurations that satisfy the desired constraints. For example, in Figures 1c and 1d the two curves are parameterized differently, yielding two different behaviors when a point is selected and moved. This problem was partially addressed in [HHK92] where a 3D lattice for a free-form deformation was manipulated directly using several different techniques. The effects of these techniques were indicated visually by different geometric shapes, or tools, which were used to sculpt the object in the 3D lattice.

The second area of research has focused on creating geometric representations of modeling operators. We call such representations *visual tools*. The first visual tools, representing the twist, bend, and taper [TPBF87] operators [SHR$^+$92], have geometry that represents the different parameters for the given modeling operator. For example, the beginning and ending points of the twist operator are represented by two 3D points and the amount of the twist is represented by the angle of the twist "handle". This work demonstrated the potential inherent in 3D interfaces and explored basic guidelines for their construction; the design of the visual tools presented in this paper began with these guidelines.

There are, however, problems and issues in the realm of solids modeling that have not been specifically addressed. The first problem is *interactivity*. For a 3D interface to be usable, it must run at interactive rates, i.e, user's actions should elicit immediate feedback. Because many solids modeling operations, such as blending between two surfaces, cannot be computed at interactive rates, we have developed *approximate* techniques, that allow us to run in realtime, with the loss of some resolution (see Appendix A).

The visual tool for the twist, bend, and taper [SHR$^+$92] and our warp tool (Sec. 3) both use a one-to-one correspondence between the parameters of the operator and the geometry of the tool. Although this was possible for these operators, the parameters of many modeling operators do not have such an obvious geometric equivalent, or the values of one parameter cannot be decoupled from those of another. For example, the tools for the blend operator in Section 3 use geometry that indicates the result of the operation, not the parameters.

One further problem to address is the order of operations, and how one visual tool affects another. With a textual interface, the order of the operators is fairly well-defined, but with an interactive interface there is some ambiguity.

2.1 The prototyping system

The system used to create and examine the visual tools is a hybrid of Brown University's interactive 3D illustration system [ZCv$^+$91] and the University of Utah's modeler, Alpha_1 [EGS91]. Many of the visual operators were constructed using Brown's Toolkit [ZHR$^+$93] [SZH94], a 3D toolkit designed for quick prototyping of three dimensional widgets. The actual modeling operations were performed by Alpha_1 (see Figure 2).

3 The operators

We have seen some of the reasons why a 3D interface can be a powerful design tool, and we have listed some of the problems confronting a visual tool designer. This section describes in detail three different modeling operators and their associated

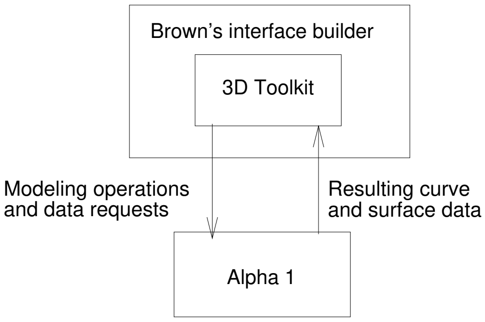


Figure 2: The connection between Brown's interface system and Alpha_1.

visual tools, which were developed by applying the guidelines in [SHR+92] to a solids modeling interface. The following section gives some principles for a solids modeling interface design similar to those in [SHR+92].

The operators presented in this section are used to illustrate the principles given in the following section. We first describe the operator as defined in Alpha_1 and then the visual tools and how they relate to Alpha_1's operators.

The parts of the visual tools fall into three different classes: manipulable geometry (e.g., a vector which can be moved or rotated), explanatory geometry (e.g., a vector indicating a tangent), and geometric parameters (e.g., a curve defined elsewhere). A part is not necessarily restricted to one of the three classes; it can play several roles depending on the context.

The operators we define here exemplify three different types of design interaction; a surface construction operator (the *sweep*), a surface deformation operator (the *warp*), and an operator that smoothly joins two surfaces (the *blend*).

3.1 The sweep tool

A sweep is a curve, surface or volume that is the result of moving a geometrical object (such as a point, curve, surface or volume) through 3D space. Bloomenthal in [BR91] presents a formal framework for the generation of sweep surfaces based on non-uniform rational B-splines. This method sweeps a set of three-dimensional *cross-section* curves along a three-dimensional *axis* curve. Each cross section curve is associated with a parameter value of the axis curve which specifies where the cross section lies. The sweep is the result of blending between successive cross section curves. Figure 3 shows an axis curve, cross-section curve, and the resulting sweep surface.

The *sweep tool* (shown in Figure 4a) is used to add or change a cross-section curve of a sweep. The geometric parameters to this tool are the axis curve and a cross-section curve: the center of the tool is constrained to lie on the axis curve and its orientation is determined by the tangent and normal of the axis curve. The tool slides along the axis curve freely; the center of the tool determines p, the position of the cross-section curve on the axis curve. The cross-section curve can be scaled and rotated using the dark gray point. The light gray vector indicates the tangent of the axis curve at p. The dark

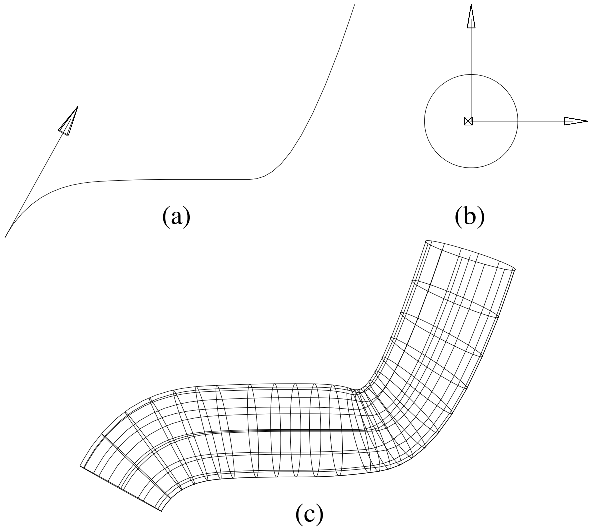


Figure 3: (a) The axis curve and tangent vector (b) The cross-section curve with x and y axes shown; the z axis is out of the paper. (c) The resulting sweep surface: the z axis of the cross-section is always oriented in the direction of the axis curve's tangent vector.

gray vector indicates the orientation of the cross-section curve and is drawn in the plane orthogonal to the light gray vector. Additionally, the cross-section and axis curves can be directly manipulated in place using the push-and-pull method. If the cross-section curve is initially planar, then the location of the curve and the curve manipulation are constrained to the plane orthogonal to the tangent vector.

The user creates a sweep tool, either providing axis and cross-section curves or using the system's default curves. The tool is constrained to lie on the axis curve; initially, a constant-width sweep is produced. The location at which a specific cross-section curve is placed in the sweep is changed by moving the sweep tool to the desired point on the axis curve. The cross-section curve can be deformed, rotated, or scaled as desired.

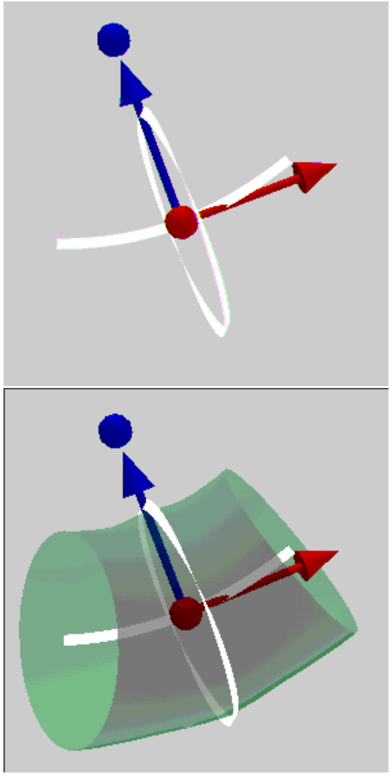
Additional cross-sections are added by unconstraining[1] the tool from the current cross-section and reconstraining it to a new one. Ghosts of these specified cross-sections are drawn in place. At any time the user can constrain the tool to an old cross-section for futher modification or deletion.

3.2 The warp tool

The warp operator is used to introduce bumps of various shapes into a surface [Cob84]. The warping operator takes as input the center of the warp, a warp direction $\vec{d}$, and several unintuitive parameters that specify the shape and region of influence of the warp. In addition, restriction planes may be used to restrict the warp to the part of the surface lying on one side of the plane. Warps are created simply by moving a set of the surface's control points around the warp center in the direction $\vec{d}$.

[1] The Brown University toolkit is a geometrical constraint system; unconstraining one object from another essentially removes the dependency between the two objects.

(a)



(b)

Figure 4: Sweep Tool, consisting of orientation vector (dark gray), scale point (dark gray), tangent vector (light gray), center of tool (light gray point) and cross-section curve. (a) Sweep tool with the axis and cross-section curves in black. (b) Sweep tool with the sweep defined by the axis and the cross-section curve.

A major motivation for developing this operator was to eliminate the tedious and often difficult task of moving the individual control points of the surface. Although this operator simplifies the creation of free-form bumps, the textual interface to the warp operator results in multiple interations of parameter-tweaking. The *warp tool* allows for interactive specification of warps in an intuitive, visual way, thereby eliminating much of the change-view cycle.

The visual tool for a warp is shown in Figure 5a. The warp tool can be moved to any point in three-space; the center of the warp is indicated by the point in the center of the ring. The dark gray vector indicates the direction and strength of the warp. The ring, which can be scaled in and out, represents the region of influence of the warp. (The remaining parameters, those which influence how the warp falls off, are currently set to default values.) The light gray plane and vector in Figure 5b form a restriction plane; any control points "below" the plane are not moved. The location and orientation of the plane are controlled by the light gray vector normal to the plane.

3.3 The blend

Surface blending is a powerful design tool for making smooth C^1 transitions between two surfaces. Blends are used for a variety of reasons: to physically strengthen the join between two objects, to model objects to be milled with a ball-end cutter, to clean up the sharp edges after a boolean operation, and to increase visual appeal. The blending operator [Kim92], [Fil89] is a function of three parameters:

- Two primary surfaces, $\sigma_1(u, v)$ and $\sigma_2(u, v)$, where D_i is a rectangular subset of $\Re^2$

$$\sigma_i(u, v) \; : \; D_i \; \subset \; \Re^2 \; \rightarrow \Re^3 \tag{1}$$

- two curves defined in the parametric space of the surfaces, $\gamma_1(t) = (u_1(t), v_1(t))$ and $\gamma_2(t) = (u_2(t), v_2(t))$, where

$$\gamma_i(t) \; : \Re \rightarrow \; D_i \tag{2}$$

- two tangent curves $\lambda_1(t)$ and $\lambda_2(t)$ describing the direction and magnitude of the tangents along the boundary of the primary surface and the blend surface.

The parameter space curves $\gamma_1(t)$ and $\gamma_2(t)$ are symbolically *composed* onto the respective primary surfaces to produce curves $\Gamma_1(t)$ and $\Gamma_2(t)$ that lie *exactly* (to machine tolerance) on the surfaces:

$$\Gamma_i(t) = \sigma_i(\gamma_i(t)) = \sigma_i(u_i(t), v_i(t)). \tag{3}$$

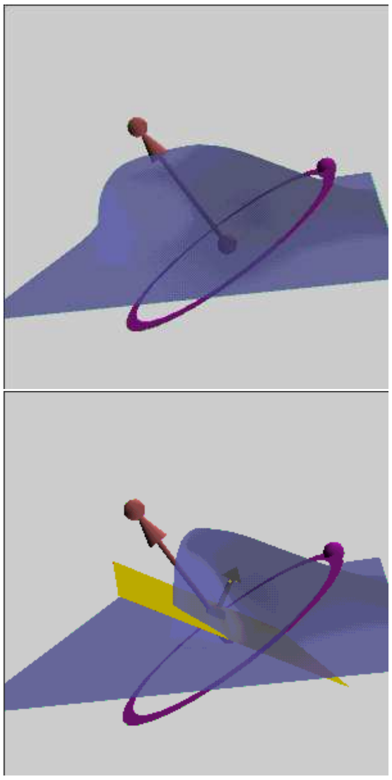
Figure 6a, b and c gives an example of $\sigma(u, v)$, $\gamma(t)$ and $\Gamma(t)$, respectively. The composed curve $\Gamma(t)$ becomes the *rail* curve or blend-surface boundary. With $\Gamma_1(t), \Gamma_2(t), \lambda_1(t)$ and $\lambda_2(t)$, a Hermite blending surface [Far92] can be constructed which is C^1 to both primary surfaces (see Figure 7a and b).

Unfortunately, it is currently impossible to perform the symbolic curve-surface composition interactively due to its computational expense. We have therefore developed a fast approximation to composition to let us explore blending operations interactively, the details of which are given in Appendix A.1. With these approximated rail curves, we can create a blend surface that is C^1 to a certain tolerance. We use these approximations during interaction in order to provide feedback to the user. This approximation gives the user a good idea of how the final blend will appear after the interactive design. After interaction, the exact, more time consuming blend can be computed. This technique provides a good tradeoff between interactivity and correctness.

We have created several different visual tools that interact with the blend operator. Our first tool simultaneously specifies the two rail curves of a blend operation (see Figure 8a); we call this the *rail-tie tool* because it looks like a railroad tie. The second tool is for altering a rail curve once it has been constructed. This tool is a direct manipulation tool and has no associated geometry (see Figure 9). The last tool is for altering the tangents of the blend surface and is called the *tangent tool* (see Figure 10).

The rail-tie tool takes as parameters the two surfaces between which the blend surface is to be defined. The idea of the rail-tie tool is to specify some number of contiguous points on each surface through which the rail curve for that surface will pass. The rail-tie tool creates two points that are constrained to lie in the two surfaces. One of the major difficulties in specifying rail curves is getting a good correspondence between the two rail curves; the parameterization of the top half

(a)



(b)

Figure 5: Warp Tool. (a) Warp applied to a flat surface. (b) The same warp with a restriction plane.

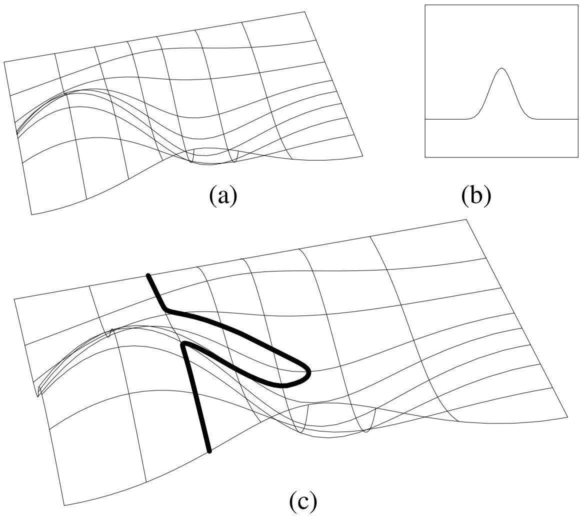


Figure 6: Composition. (a) Surface, $\sigma(u, v)$. (b) $\gamma(t)$, curve defined in parametric space of $\sigma(u, v)$. (c) Composite curve $\Gamma(t) = \sigma(\gamma(t))$ in bold.

of the blend surface is defined by the top rail curve, while the parameterization of the bottom half is defined by the bottom rail curve. If these two parameterizations do not match nicely, the blend surface "twists" (see Figure 8b). The rail-tie tool provides explicit control over the parameterization of the blend surface by defining a correspondence between the parameterization of the two rail curves while defining the geometry of the rail curves.

To specify rail curves, the user first creates a rail-tie tool over the two primary surfaces. The two points of the rail-tie tool are automatically constrained to default points on the two surfaces. The user then positions the two spheres to indicate the first two points of the rail curves. Once they are positioned, the user requests two new points. Gray spheres now appear on the surface to indicate the locations of the previously specified points. The user is now free to move the new points to indicate the positions of the next two points on the rail curves. The system continuously updates the rail curves to pass through all the given points.

The *rail-curve manipulation tool* allows us to manipulate the composite rail curve $\Gamma(t)$ on the surface, instead of the curve $\gamma(t)$ defined in the parameter space of the surface. The behavior of most direct manipulation techniques depends upon the parameterization of the curve; in this case, the difficulty is compounded because the behavior also depends on the parameterization of the surface. To solve this problem, we developed a curve manipulation technique that does not depend upon the parameterization of the curve or the surface (details are given in Appendix A.2).

To delineate the extents of the curve manipulation we use two spheres that are constrained to the curve. Figures 9a and b show the same curve with two different curve extents marked out. When the curve is grabbed and pulled, only the section of the curve between the two spheres moves.

The *tangent tool* takes as parameters a rail curve and its corresponding blend region. The tool is constrained to lie on the rail curve, but is free to move along it in order to indicate the point $p = \sigma(\gamma(t))$ at which to scale the cross-boundary tangents. The vector points in the direction of the cross-boundary tangent at p. We can scale the tangent of the blend at p by stretching and shrinking the vector. The corresponding tangent curves are altered using a least-squares technique to achieve the desired tangent values.

4 Principles of visual tool design

Designing 3D interfaces presents several problems not found in 2D or textual interfaces. The principles below begin to address the problem of devising successful interfaces. They are similar to the guidelines in previous work [SHR+92] on designing 3D interfaces in general, but are tailored for the solids modeling domain. Their purpose is to give the 3D interface designer a framework within which to pose individual problems. Although designing successful interfaces is still more an art than a science, these guidelines may help expose where and how the power of 3D user interfaces can be used.

We begin by stating the four principles and then examine each of them individually in light of our examples.

1. The visual tool should exist in the same space as the object or objects it manipulates.

2. The visual tool should eliminate the need to understand the particular implementation details of a modeling operator.

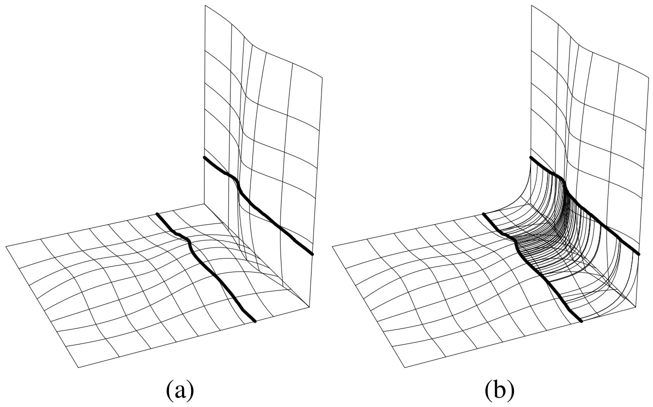


Figure 7: Rail Curves. (a) Two primary surfaces with rail curves shown in bold. (b) Primary surfaces with blending surface.

3. The visual tool should provide visual clues on its function and use.
4. The design of the visual tool should be based on the user's intuition of how the operator should behave, not on the parameters to the operator.

4.1 Operator space

The visual tool should exist in the same space as the object or objects it manipulates.

This principle has to do with understanding what happens to an object or operator when its values or parameters are changed. Often, there is an inherent abstraction or hidden assumptions in an operator's parameters. By defining a visual tool in the same space as the result of the operator, we can reduce these abstractions and assumptions. Consider positioning one cube next to another in three space. With a text interface, the user alternates between typing in different positions and examining the locations of the cube. If the location of the first cube is known, the user can calculate the location of the second. Note, though, that not only is the location of the first cube needed, but where that location is relative to the cube, since cubes can be defined with their origin at a corner instead of at the middle. (This is what we mean by hidden assumptions: the normal assumption is that a cube's origin is at its center.)

Now consider a "tool" that exists in the same space as the cubes and moves a cube in the direction in which the mouse moves. Now a cube can be picked up and moved directly to its location, without knowledge of the exact numeric value of that location, the size of the cube, or how cubes are defined. This type of tool is called an *object handle*, and is explained in detail in [SHR$^+$92].

Our first example of this principle is the sweep tool. Traditionally, placing a cross-section on an axis curve required knowledge of how the axis curve is parameterized. With the sweep tool in the space of the axis curve, we can specify the location of the cross section by its desired location, without needing to know the parameterization of the axis curve. This tool simplifies placing multiple cross sections on the axis curve because their relative scales and rotations are immediately apparent: if a cross-section is oriented incorrectly on the axis curve, we merely rotate it in place without needing to know how much or in which direction to rotate.

Another example of this principle is the warp tool. To create a warp, the warp tool is placed in the location and direction of the desired warp. The actual parameter values are unimportant to the user: what matters is the particular shape the user is trying to achieve. The warp tool lets the user alter the shape of the warp by adjusting geometry that indicates the effect of a parameter, without concern for actual values.

4.2 Independence of operator implementation

The visual tool should eliminate the need to understand the particular implementation details of a modeling operator.

This principle has several aspects. The first is that the user should not have to know the effect of the implementation details of an operator on the result. For example, in Alpha_1's sweep operator, the location of a cross section can be given by a parameter value or an arc length value. In the sweep tool, the method by which the cross section is placed is independent of these issues and of the parameterization of the axis curve. In the rail-curve manipulation tool, not only is the parameterization of the curve and surface hidden, but the user need not know that the rail curve is actually defined in the domain of the surface. Instead, the user alters the geometry of the curve as it appears on the surface.

Another aspect is portability. For example, the warp tool defined here is currently used to apply Alpha_1's warp operator to a surface. Suppose a different warp operator is defined that operates in a similar manner but with different effects or on a different representation, such as a polygonal mesh. The warp tool could be used without outward change with either of these warp operators.

Another way to hide implementation details is to make their effects explicit to the user. The rail-curve manipulation tool is an example of this: the user defines not only which point on the curve should move where, but how much of the curve to move. This allows the user to control the rail-curve geometry explicitly.

This principle extends to the number and type of parameters as well as the individual parameters. In the rail-tie tool, all the parameters to the blend are tied up in one tool. Since with this tool the rail curves are specified on the surface, we need not know that the rail curves are actually defined in the 2D parameter space of the surface and then composed. The parametric correspondence between the two rail curves is defined implicitly because they are defined at the same time. This prevents the common problem of orienting one curve incorrectly with respect to the other.