

BROWN UNIVERSITY  
Department of Computer Science  
Master's Thesis  
CS-91-M10

A Direct Manipulation Interface to Free-Form Deformations

by  
William M. Hsu

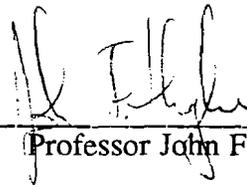
# A Direct Manipulation Interface to Free-Form Deformations

By

William M Hsu

Submitted to the faculty of Brown University as partial fulfillment of the requirements for the Sc.M. degree

May 1991



---

Professor John F. Hughes, Advisor

# 1 INTRODUCTION

Modeling has become one of the predominant research topics in computer graphics.

Though many techniques for modeling three-dimensional objects have been developed, until recently little attention has been paid to user interfaces for these techniques. Many techniques originated in the computer-aided design and manufacturing world, where design engineers used computers to model complex parts precisely. Consequently, typical users were very knowledgeable about the mathematical bases of these techniques, and developers could concentrate on the functionality of the modeling tools, rather than their user interfaces. Now, however, artists, illustrators, educators, animators, and others lacking in the mathematical sophistication have begun to use the computer to model objects. These people need well-designed user interfaces to use sophisticated tools in an intuitive and productive fashion.

A relatively new and complex modeling method is free-form deformation, in which the user deforms objects by adjusting parameters to a polynomial basis in a technique akin to the manipulation of spline surfaces. In many commercial modeling systems, such as TDI's *Explore Design* and Softimage's *Creative Environment*, the user manipulates control points to specify the deformation. If, however, the user is not familiar with splines and their behavior, this may be confusing since the control points do not necessarily lie on the surface of the object. A more intuitive interface would allow the user to manipulate the surface of the object directly. This thesis presents an interface technique that allows the user to define deformations through direct manipulation of the object.

Section 2 reviews the free-form deformation modeling method and the manipulation techniques for this method and other spline surface-modeling methods. Section 3 details

the motivation for and internal workings of our interface. Section 4 describes the implementation of the interface in a 3D modeling and animation system. Section 5 discusses possible future work, and section 6 gives some concluding remarks.

## 2 BACKGROUND

### 2.1 The Free-Form Deformation Modeling Technique

This section summarizes the free-form deformation method described by Sederberg and Parry, and its differences from my implementation. It assumes the reader has some knowledge of splines, in particular B-splines (see [Fari90, Bart87]). We distinguish here between the underlying modeling technique (the mathematics of the deformation) and the interface to the technique, which will be discussed further in section 3.3. Further details on free-form deformations can be found in [Sede86].

#### 2.1.1 Review of the Sederberg and Parry method

The free-form deformation (FFD) method deforms an object by first embedding it, or a region of the object, into a coordinate system local to FFDs, and then applying a function which maps the object back into world coordinates. The local coordinate system is defined by a parallelepiped-shaped lattice of control points, with one corner as the origin,  $(0,0,0)$ , and the opposite corner labeled  $(1,1,1)$ . All object points within this parallelepiped are transformed into the local coordinate system through a mapping from 3-space to 3-space. The function which maps the object from local coordinates to world coordinates uses the lattice of control points to define that mapping. The location of an object point is determined by a weighted sum of the control points, therefore the location of the control points can change the location of the object point. The result of concatenating these functions is a map from 3-space to 3-space.

Figure 1 gives a simplified 2D analogy to an FFD: a square two-dimensional rubber sheet on which a drawing is copied. The square rubber sheet can be stretched to any shape. In (a), the rubber sheet is in its rest state beside a drawing of a smiling person. The rubber sheet is stretched to fit over the drawing, and the drawing is then copied onto the rubber sheet, as shown in (b). Figure 1(c) shows how the drawing looks when the sheet returns to its rest state. Then the sheet can be stretched and bent, as shown in (d), and the drawing copied in an altered state (e). For free-form deformation, the analogue of the rubber sheet is a rectilinear volume.

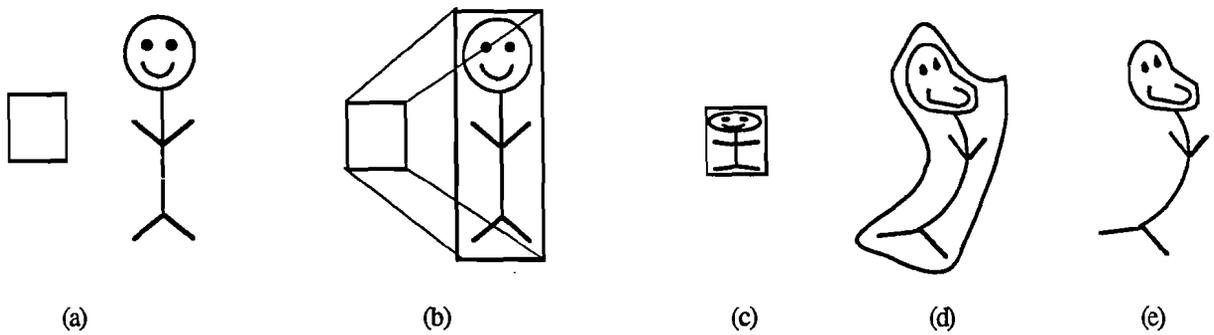


Figure 1

An FFD can also be described by comparison to rendering a computer model onto the screen. An object is described internally in world coordinate space in any unit of measure. The coordinates of this object are transformed into normalized device coordinates and then are mapped into screen coordinates and displayed on the screen. Free-form deformation works in the same manner. First, the object to be deformed is represented in world coordinates; it is then represented in the FFD coordinate system; finally, the object is assigned new world coordinates based on its FFD coordinates.

Now we detail the two mappings mathematically. The FFD acts on a parallelepiped; any point within this parallelepiped is mapped to a new location by the FFD function. We start by defining a local coordinate system on the parallelepiped as follows. Let the vector  $Q_0$  be the origin of the parallelepiped, and  $S$ ,  $T$ , and  $U$  be three orthogonal vectors whose origins are at  $Q_0$  and that span the edges of the parallelepiped. Then any point  $Q$  can be written as the sum

$$Q = Q_0 + sS + tT + uU$$

The numbers  $(s, t, u)$ , called the FFD coordinates of  $Q$ , are simply the ratio between the distance from  $Q_0$  of the object point and the length of the parallelepiped in each dimension. The values of  $s$ ,  $t$ , and  $u$  can be calculated from simple linear equations:

$$s = \frac{T \times U \cdot (Q - Q_0)}{T \times U \cdot S} \quad t = \frac{S \times U \cdot (Q - Q_0)}{S \times U \cdot T} \quad u = \frac{S \times T \cdot (Q - Q_0)}{S \times T \cdot U} \quad (1)$$

Since  $Q$  is within the parallelepiped, we know that  $0 \leq s, t, u \leq 1$ .

We now define a map from FFD coordinates (i.e., tuples of  $(s, t, u)$ ) to world coordinates. This map is defined by taking a weighted sum of control points. In an FFD's rest state, i.e., when the map is the identity, the control points form a lattice that is evenly spaced in each dimension. The control points are denoted by  $P_{i,j,k}$  ( $i = 1..l, j = 1..m, k = 1..n$ ):  $P_{i,j,k}$  is the  $i^{\text{th}}$  control point in the  $S$  direction, the  $j^{\text{th}}$  control point in the  $T$  direction, and the  $k^{\text{th}}$  control point in the  $U$  direction. Figure 2 shows the grid of control points when  $l=2, m=3, n=1$ .

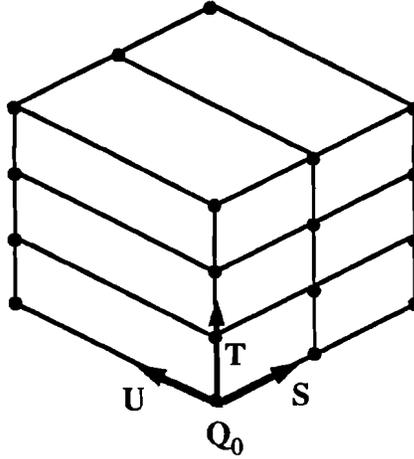


Figure 2

Each control point is assigned a weight, which is determined by the  $(s, t, u)$  values from equation (1). The free-form deformation function used by Sederberg and Parry is defined by a tensor-product Bernstein polynomial, and the deformed position of an arbitrary point with coordinates  $(s, t, u)$ , called  $Q^{ffd}(s, t, u)$ , is given by the formula:

$$Q^{ffd}(s, t, u) = \sum_{i=0}^l \sum_{j=0}^m \sum_{k=0}^n P_{i,j,k} \binom{l}{i} (1-s)^{l-1} s^i \binom{m}{j} (1-t)^{m-1} t^j \binom{n}{k} (1-u)^{n-1} u^k \quad (2)$$

Moving a control point influences the value of  $Q^{ffd}(s, t, u)$ , to an extent determined by its location within the FFD parallelepiped. The control points of the FFD affect the object in the same way as the control points of a Bézier spline — which can be defined by the Bernstein polynomial — affect a curve.

### 2.1.2 B-spline Based Free-Form Deformation Method

My implementation of free-form deformation deviates from the method described by Sederberg and Parry in using piecewise uniform cubic B-splines as the basis, instead of

Bernstein polynomials. Cubic B-splines are much more computationally efficient than Bernstein polynomials for large numbers of control points. When Bernstein polynomials are used as the basis, every control point contributes to the calculation of each deformed point location, giving global control. In contrast, with cubic B-splines only 64 control points (a 4x4x4 lattice) are evaluated to calculate the location of a point, giving local control. Not only is the evaluation of the deformation more efficient, but shaping objects tends to be easier with local control. Piecewise cubic Bézier splines would be just as efficient as piecewise cubic uniform B-splines, but the B-splines maintain  $C^2$  continuity automatically whereas the Bézier splines do not.

Equation (2) is revised to reflect the change in basis in the FFD function:

$$Q_{i,j,k}^{\text{ffd}}(s,t,u) = \sum_{l=-3}^0 \sum_{m=-3}^0 \sum_{n=-3}^0 P_{i+l,j+m,k+n} B_l(s) B_m(t) B_n(u) \quad (3)$$

where  $i$ ,  $j$ , and  $k$  are the spline segments enclosing the point and  $B_l$ ,  $B_m$  and  $B_n$  are the B-spline blending functions used in each direction, namely

$$\begin{aligned} B_{-0}(u) &= \frac{1}{6} u^3 \\ B_{-1}(u) &= \frac{1}{6} (1 + 3u + 3u^2 - 3u^3) \\ B_{-2}(u) &= \frac{1}{6} (4 - 6u^2 + u^3) \\ B_{-3}(u) &= \frac{1}{6} (1 - 3u + 3u^2 - u^3) \end{aligned}$$

It should be noted that equation (3) is evaluated for each of the  $x$ ,  $y$ , and  $z$  components of  $\mathbf{Q}$  and  $\mathbf{P}$ , and that the set of blending functions is identical for each direction.

With B-splines as the basis, however, finding the  $(s, t, u)$  values of the object points is not as simple as before. To calculate the local coordinate values for an object point, we must first find which spline segments contain the point. Then the  $s$ ,  $t$ , and  $u$  components are calculated by finding the roots of the cubics in equation (3). Also, the control-point lattice is no longer completely uniform. In order to outline the deformation region exactly, each outer control point is given a multiplicity of three.

### 2.1.3 Features of Free-Form Deformations

There are several important features of FFDs:

- They do not rely on the data type of the object: they can deform polygonal data, implicit surfaces, and parametric surfaces.
- Parametric curves and surfaces remain parametric.
- There is a class of FFDs that are volume-preserving.
- Continuity between two abutting deformation volumes is easily achieved.

Detailed explanations can be found in [Sede86].

Modeling with an “indirect” method such as free-form deformation has several advantages over modeling methods that directly define the geometry of an object. Ideas like “stretch that part a little here” and “bend that section a little there” can be handled easily with FFDs. This type of modeling is especially useful in animation sequences for creating squash and stretch effects, as well as in caricaturing objects and exaggerating motion. Since the underlying data type of the object to be deformed is of little consequence, FFD is also useful in augmenting objects already created with different modeling techniques that rely on

different data types. For example, with FFD an object composed of a polygonal object connected to a trimmed surface can be deformed as one cohesive object.

## **2.2 Review of Extended Free-Form Deformation**

One restriction on free-form deformation is the control points must initially be configured as a uniform grid. This limitation means that some shapes, such as circularly symmetric deformations, cannot be obtained. Coquillart extends free-form deformation by allowing the lattice of control points to be arbitrarily shaped before the deformation process [Coqu90], permitting a new class of deformations to be created and making the process more intuitive. The user, however, must first shape the lattice into the desired form and then move the control points to achieve a deformation. Coquillart concedes that the initial shape of the lattice is “of paramount importance.” To shape the lattice, the user must manipulate each control point explicitly, and thus must have some knowledge about the underlying deformation method.

## **2.3 Current FFD and Spline-Based Modeling Interfaces**

The shape of free-form deformations is controlled through the placement of control points. To date, this placement has been done either by directly manipulating individual control points or by moving groups of them according to a simple function (e.g., rotating a group of control points around a point). Since the free-form deformation and spline-based modeling techniques share the same mathematical foundations and are similar in concept,

we can predict future enhancements to the free-form deformation interface by examining the interfaces developed for current spline-surface modelers.

Some spline-surface modelers have high-level tools that move groups of control points in a predictable manner. These tools include functions such as *bend*, *twist*, *group warp*, *flatten*, and *bulge* [Cobb84] [Ries89]. *Bend* and *twist* modify an object precisely as their names indicate. *Group warp* moves a group of control points in response to the movement of a center or target point. The relative positioning of the control points in the group can be warped by applying a function to them as they are moved. The function can be a decay function, say, giving the surface manipulated an elastic behavior, the control points can be weighted so as further to vary the effects of the applied function. The *flatten* operation aligns a set of control points to a plane. The direction of movement is usually along the plane normal, but can be user-defined. The *bulge* tool disperses control points outward in a uniform manner, so that the surface balloons out. These tools give the user fairly intuitive means of modeling for certain operations, but note that they all operate directly on the control points. An exception is the technique Forsey and Bartels use in their hierarchical B-spline editor [Fors88]. Points on the surface can be manipulated, but only those directly beneath a control point.

## 3 A Better Interface to Free-Form Deformations

### 3.1 What Makes a Good User Interface?

Foley [Fole84] describes a criteria for a good interface: “An effective interface design is one in which a user carries out his work with minimal conscious attention to his tools (the paraphernalia of the interactive terminal and the command language) and maximal effectiveness. It is free of distractions and reasonably ‘friendly.’” Translated to the cognitive level, an interface’s effectiveness is inversely related to the amount of effort users must expend to accomplish their goals: the less mental energy necessary to manipulate the tools to accomplish a task, the better the interface.

This cognitive effort can be measured by evaluating the two aspects of human-computer communication: (1) the communication from human users to the computer: the commands and operations the user executes and performs; (2) the information the computer communicates to users: the output, visual or otherwise, from the computer. Hutchins defines the gaps between the user’s goals and a system’s capacities as the Gulf of Execution and the Gulf of Evaluation [Hutc85]. The Gulf of Execution is the difference between user intent and what the interface allows; the Gulf of Evaluation is the difference between what the user expects to see and what the interface displays. To accomplish their goals with the available tools, user’s must bridge the Gulf of Execution by constructing a correct sequence of commands in the language of the interface. The easier this is, the better the interface design.

Without proper feedback, the user is unsure what the results of her actions will be. If the feedback is not what the user expects or lacks pertinent information, the user must exert cognitive effort to interpret and analyze what feedback there is. If, on the other hand, the feedback is what is expected and helpful, the user can concentrate on the problem task and use the feedback as an affirmation of progress.

In many cases, actions and feedback work in conjunction to present a coherent picture to the user. Consider for instance, deleting a file. In some graphical user interfaces, to delete a file, the user drags its iconic representation into a trash bin on the screen. In this single process, there is a dialog between the user and the computer. First, the user selects an icon that represents the actual internal file. Picking the icon highlights it, confirming the user's choice. Next comes a continuous action/feedback loop as the user drags the pointer across the screen and the icon follows. When the pointer reaches the trash bin, the bin is highlighted, telling the user the results of a potential action. The user releases the mouse button and the file disappears into the trash bin. Finally, the trash bin bulges to show that the file has been deposited there. Since this process relates closely to how people actually throw things away, the cognitive effort in learning the task is minimal. Deleting a file in a UNIX shell with the *rm* command, on the other hand, requires the user to remember the file name, the name of the command for deleting the file, and the proper syntax for that command. Furthermore, there is no feedback showing that the file is actually gone: to be sure, the user must issue another command to ask about the status of the file. Clearly, the direct-manipulation interface is more intuitive.

### **3.2 Deficiencies of Current Free-Form Deformation Interfaces**

Free-form deformations are defined by control points in the same way as spline curves and surfaces are. The interfaces to both methods also rely on direct control-point manipulation. A system may provide high-level tools for moving groups of control points, as mentioned in section 2.3, but the focus remains on control-point manipulation. The difficulty is, of course, that users unfamiliar with splines may not know what the control points are. The difficulty is exacerbated by the fact that control points of free-form deformations do not approximate the surface of the object, as they do for spline surfaces. Thus, it is unclear what, if anything, moving a control point will do to the object. Even when moving a control point located on or very near the surface, the surface usually does not move as far as the control point.

These apparent discrepancies arise because parameters of the function that deforms the object are being manipulated, not the object itself. Moving a control point does not shape the object as the user desires, nor is the visual feedback what the user expects. For example, if the control points originally lie on the surface of an object, then a likely assumption by those unfamiliar with splines is that they will continue to do so. However, when the user moves a control point, the surface moves towards the control point but does not stay in contact with it. To make an educated guess about where to place the control point, the user must analyze the feedback to construct some sort of relationship between the control point and the surface of the object. The tool requires far more than Foley's "minimal conscious attention."

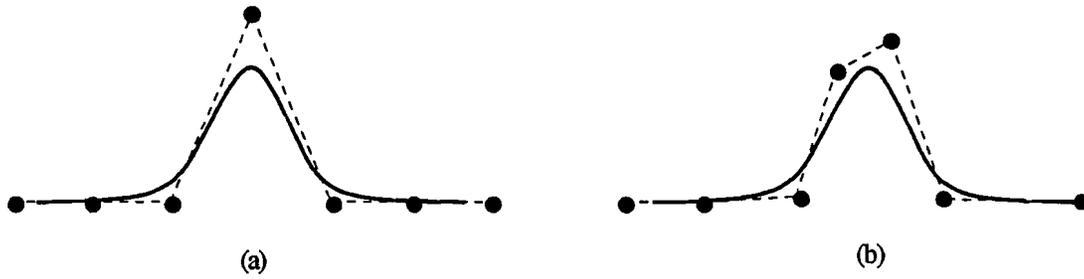


Figure 3. The same shaped curved is defined in different ways depending on the position of the hump.

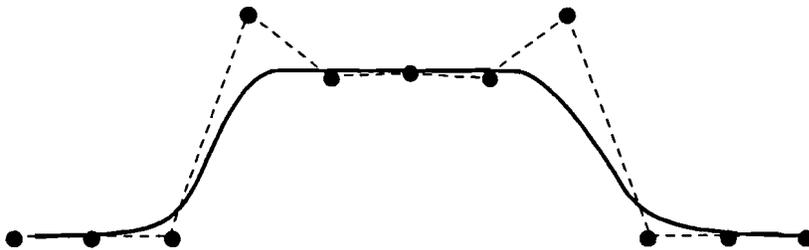


Figure 4. A flat hump. Control points on the outer edge of the plateau are above the hump and those in the mid-section are at or slightly below it.

Figure 3 shows an instance of potential confusion on the user's part. A single hump is easily generated by moving a control point upward, as shown in (a). But creating that same hump in the slightly shifted position in (b) involves positioning more than one control point, and in a non-uniform manner. The same non-intuitive behavior arises in trying to create a plateau-like hump, as shown by figure 4. One might expect to create a flat hump by moving a series of control points upward and aligning them horizontally, but instead, the control points at the hump's edge must be lifted further than the ones in the middle.

### 3.3 A Direct Manipulation Based Solution

The difficulties encountered in deforming an object with current interfaces arise from the apparent lack of correlation between the surface of the object and the control points moved. Direct manipulation of the surface itself would be far more intuitive, and thus is what my interface allows. Direct manipulation of 3D objects is a part of many polygonal modelers [Pare77] [Carl82] [Alle89]; in fact, many of the high-level tools developed for spline-based modelers originated from the polygonal modelers. It was recognized that manipulating individual points for large-scale changes can become frustrating, tedious, and error-prone. Using the proper high-level tools allows users to manipulate objects in a way more in tune with their mental description of how the object is shaped. Unfortunately, spline-based modelers have shifted the focus from manipulating object points to adjusting control points, adding a layer of indirection. Carlson [Carl82] points out that “The main problem that exists in the free form surface design systems is the lack of a suitable intuitive user interface. On the other hand, most polygonal systems have a fairly good user interface, but lack any degree of generality.”

To gain the best of both worlds, I have developed a new interface to free-form deformations that allows direct manipulation of the object, and in the process have eliminated the need for the user to be aware of the control points. The user gains the power of free-form deformations with a simple click-and-drag interface.

### **3.3.1 The Magnetic Tool**

Section 2.3 described several high-level tools that aid in the modeling process. All of those tools can be represented by one unified model, a magnetic tool, that is general enough to allow both pulling on and pushing against an object, and to act on a single point or on a

large section of an object. The general concept of magnets and magnetism is a familiar one, so the tool can be quickly learned and used in an intuitive manner.

To enhance visual feedback, the magnetic tool is represented as a visible object within the modeling environment. Several different pieces of information can be conveyed by this representation. The shape of the magnet can indicate the direction in which the surface of an object will move in as it comes under the magnetic tool's influence. For instance, a cone-shaped magnetic tool implies that a region of the object will be attracted towards a single point, the apex of the cone, and a box-shaped tool indicates that all object points will move in the direction parallel to the magnetic tool's normal. (See figure 5) The size of the magnetic tool can suggest the size of the area the magnetic tool will affect. Once the tool starts to deform the object, the deformed region can easily be highlighted to tell the user precisely what will change, including fringe areas (the area affected by the deformation function, but not directly affected by the magnetic tool).

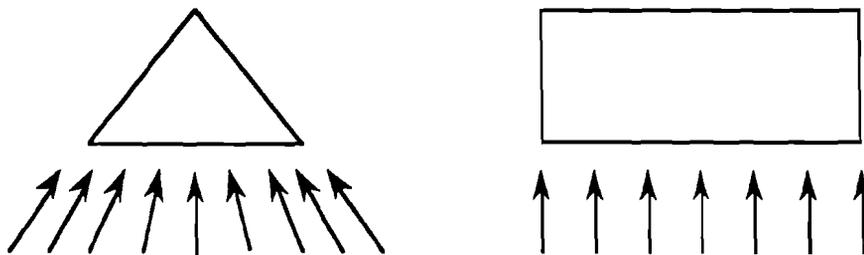


Figure 5. The arrows indicate the direction of attraction towards the magnetic tools.

Like real magnets, too, the magnetic tool need not actually touch the points it affects. The points affected by the tool depend on the tool's *realm of influence*: the area in which points distant from the primary point selected still come under the influence of the magnetic tool.

representation of the realm of influence should be, to some degree, transparent, so the user can see what is happening to the surface of the object as it is being deformed.

Users often would like a high-level tool that is a slight variant of one already provided. To take this into account, the tools in this interface also have an associated *strength of magnetism* that can be used as a parameter to augment the magnetic function of the tool. For example, if the magnetism exerted by a tool is a function  $1/r$  where  $r$  is the radius of the realm of influence, then the strength parameter can be used to make it a function of  $s/r$ , where  $s$  is the strength value. The greater the strength, the greater the magnitude of attraction.

### **3.3.1.1 Mathematical Technique for Direct Manipulation**

In order to manipulate the object surface directly, the user must have control over the points on the object. The deformed positions of the points on the object, however, are determined by the positions of the control points. Thus the problem is to move the control points based on how the user moves an object point. Once the desired location of a key object point is determined, the control points must be positioned so that the surface defined by the deformation moves the key object point to that location, at least within some tolerance. Since the deformed location of the key object point is a function of many control points, there are many configurations that will yield the same location for the deformed object point. Instead of determining all such control-point configurations, we find the configuration that requires the least amount of control-point movement.

Let us first note how the position of the deformed object point is obtained. Recall from equation (3) that the deformed object point location,  $Q$ , is a function of the control points,

$\mathbf{P}$ :  $Q = F(\mathbf{P})$ . A new location for point  $Q$ ,  $Q_{\text{new}}$ , is then  $Q_{\text{new}} = F(\mathbf{P} + \Delta\mathbf{P})$ , where  $\Delta\mathbf{P}$  is the change in position of the control points. Since  $F$  is a many-to-one function,  $F$  is not invertible, but if  $F^{-1}$  were the inverse of the function  $F$ , we could determine an equation for  $\Delta\mathbf{P}$  on the basis of  $\Delta Q$ , the change from  $Q$  to  $Q_{\text{new}}$ .

$$\begin{aligned}
 Q &= F(\mathbf{P}) \\
 Q_{\text{new}} &= F(\mathbf{P} + \Delta\mathbf{P}) \\
 F^{-1}(Q) &= \mathbf{P} \\
 F^{-1}(Q_{\text{new}}) &= \mathbf{P} + \Delta\mathbf{P} \\
 F^{-1}(Q_{\text{new}}) &= F^{-1}(Q) + \Delta\mathbf{P} \\
 \Delta\mathbf{P} &= F^{-1}(Q_{\text{new}}) - F^{-1}(Q) \\
 \Delta\mathbf{P} &= F^{-1}(Q_{\text{new}} - Q) \\
 \Delta\mathbf{P} &= F^{-1}(\Delta Q)
 \end{aligned} \tag{4}$$

We are able to condense  $F^{-1}(Q_{\text{new}}) - F^{-1}(Q)$  to  $F^{-1}(Q_{\text{new}} - Q)$  because the function  $F$ , and therefore  $F^{-1}$ , is an affine transformation.

To find out more about the “inverse” of  $F$  let us first examine  $F$  more closely. The function  $F$  transforms the control point locations,  $\mathbf{P}$ , into a deformed object point location,  $Q$ . In other words,  $F$  regulates how changes to the control points affect a change in the location of the deformed object point. This can be expressed in a mathematical structure called the Jacobian, which is a matrix of partial derivatives of a function’s outputs with respect to its inputs. Equation (3) tells us that the position of a deformed object point is a function of the  $x$ ,  $y$ , and  $z$  components of the control points, therefore

$$\mathbf{F}(\mathbf{P}) = \begin{bmatrix} f_x(\mathbf{P}) \\ f_y(\mathbf{P}) \\ f_z(\mathbf{P}) \end{bmatrix}$$

The Jacobian of  $F$  is then  $(\partial F / \partial \mathbf{P})$  or, for the 64 control points that influence a single object point in an FFD,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_x(\mathbf{P})}{\partial P_{0x}} & \frac{\partial f_x(\mathbf{P})}{\partial P_{0y}} & \frac{\partial f_x(\mathbf{P})}{\partial P_{0z}} & \dots & \frac{\partial f_x(\mathbf{P})}{\partial P_{64z}} \\ \frac{\partial f_y(\mathbf{P})}{\partial P_{0x}} & \frac{\partial f_y(\mathbf{P})}{\partial P_{0y}} & \frac{\partial f_y(\mathbf{P})}{\partial P_{0z}} & \dots & \frac{\partial f_y(\mathbf{P})}{\partial P_{64z}} \\ \frac{\partial f_z(\mathbf{P})}{\partial P_{0x}} & \frac{\partial f_z(\mathbf{P})}{\partial P_{0y}} & \frac{\partial f_z(\mathbf{P})}{\partial P_{0z}} & \dots & \frac{\partial f_z(\mathbf{P})}{\partial P_{64z}} \end{bmatrix}$$

There are 64\*3 input values in  $\mathbf{P}$  for each  $x$ ,  $y$ , and  $z$  direction, resulting in a 3x192 matrix. Since  $F(\mathbf{P})$  is a linear function, the Jacobian is simply the matrix of coefficients of the control points. These coefficients are the basis functions evaluated at the  $s$ ,  $t$ , and  $u$  values of the object point. Since  $s$ ,  $t$ , and  $u$  are constants and  $f_x(\mathbf{P})$ ,  $f_y(\mathbf{P})$ ,  $f_z(\mathbf{P})$  are linearly independent, the Jacobian reduces to

$$\mathbf{J} = \begin{bmatrix} B_{.3}(s)B_{.3}(t)B_{.3}(u) & 0 & 0 & \dots & 0 \\ 0 & B_{.3}(s)B_{.3}(t)B_{.3}(u) & 0 & \dots & 0 \\ 0 & 0 & B_{.3}(s)B_{.3}(t)B_{.3}(u) & \dots & B_{0}(s)B_{0}(t)B_{0}(u) \end{bmatrix}$$

and need be computed only once for each object point. Since the weight of each control point is the same for each component, many values in the matrix are identical, so the 3x192 matrix has only 64 distinct values. Equation (3) can now be expressed in matrix form as

$$\mathbf{Q} = \mathbf{JP} \quad (5)$$

To compute  $\Delta\mathbf{P}$  we would need to find the inverse of  $\mathbf{J}$ . But  $\mathbf{J}$  is not a square matrix, and matrix inverses are defined only for square matrices (this corresponds to the fact that our original function is not invertible). Instead, we find the *pseudoinverse (generalized inverse)*  $\mathbf{J}^+$  of  $\mathbf{J}$ : given the system of linear equations  $\mathbf{JX} = \mathbf{Y}$ , the pseudoinverse  $\mathbf{J}^+$  is the matrix where  $\mathbf{X} = \mathbf{J}^+\mathbf{Y}$  [Nobl77]. This solution is the best solution in the least-squares sense, which in our case is the solution with the minimum amount of change, exactly what we are looking for. The pseudoinverse is computed by first representing the  $m \times n$  matrix  $\mathbf{J}$

in the form  $\mathbf{J} = \mathbf{BC}$ , where  $\mathbf{B}$  is  $m \times k$  and  $\mathbf{C}$  is  $k \times n$ , such that all three matrices  $\mathbf{J}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$  have rank  $k$ . Then the general formula for the pseudoinverse  $\mathbf{J}^+$  of  $\mathbf{J}$  is given by

$$\mathbf{J}^+ = \mathbf{C}^T(\mathbf{C}\mathbf{C}^T)^{-1}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T \quad (6)$$

For free-form deformations, the rank of  $\mathbf{J}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  is 3,  $(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$  reduces to the identity matrix, and  $\mathbf{C} = \mathbf{J}$ . Let  $d_{i,j}$  be the elements of  $\mathbf{C}$ , note that  $c_{i+1,j+1} = c_{i,j}$ , and let  $\mathbf{C}'$  be a row of  $\mathbf{C}$ . Then  $(\mathbf{C}\mathbf{C}^T)^{-1}$  is the reciprocal of the magnitude of  $\mathbf{C}'$  squared times the identity matrix, and  $\mathbf{C}^T(\mathbf{C}\mathbf{C}^T)^{-1}$  reduces to  $\mathbf{C} / \|\mathbf{C}'\|^2$ . So, the pseudoinverse of  $\mathbf{J}$  can now be found using the following equation:

$$\mathbf{J}^+ = \frac{1}{\|\mathbf{J}'\|^2} \mathbf{J}^T, \quad (7)$$

where  $\mathbf{J}'$  is a row of  $\mathbf{J}$ .

Combining equations (4) and (7), we find that the appropriate location of the control points based on the location of the deformed object point is determined by

$$\mathbf{P}_{\text{new}} = \mathbf{P} + \Delta Q \mathbf{J}^+, \quad (8)$$

where  $\mathbf{P}_{\text{new}}$  is the matrix of new locations for the control points, provided all the control points are free to move independently.

On occasion, when the deformed object point lies at or near the border of the control-point lattice, some control points are forced to coincide with other control points because the outer control points have a multiplicity of three. To compute the pseudoinverse, the coincident control points must move together, so their respective weights are summed and

the control points are moved as one. To formulate the pseudoinverse equation properly, a matrix,  $S$ , which selects the proper control point position is added to equation (5), so that the deformed object point location is defined by

$$Q = JSP$$

The 192x192 matrix  $S$  is the identity matrix if all control points are allowed to move freely. If some of the control points must be coincident, then the one in the row for each control point so constrained is shifted to the column that corresponds to the control point it must follow. For example, in the one-dimensional case, if  $P = [P_{-2} P_{-1} P_0 P_1]^T$ , where  $P_{-2}$  and  $P_{-1}$  do not exist and  $P_0$  has a multiplicity of 3, then

$$S = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The equation for the pseudoinverse  $J^+$  is now

$$J^+ = (CS)^T(CSS^T C^T)^{-1}(B^T B)^{-1} B^T \quad (9)$$

For computational and space efficiency, the matrices  $J$ ,  $J^+$ ,  $C$ , and  $S$  can all be stored and computed as vectors, and  $B$  need not be computed at all.

High-level tools that move several object points at one time are computed by adding the appropriate rows and columns to  $J$ ,  $P$ , and  $\Delta Q$ , and to the matrices derived from them.

The only problem arises when the problem becomes over-determined, i.e. when there are more unknowns than equations; in that case an exact solution may be impossible to find.

The simplified pseudoinverse equation (7) cannot be used, and equation (9) must be used

instead. This yields the best possible answer in the least-squares sense and is the most reasonable given the circumstances.

### 3.3.1.2 Limitations

This interface has two limitations. First, the problem can be over-constrained. Second, aliasing can occur so that some control points are not moved when they ought to be.

Figure 6 shows an over-constrained situation in which no exact solution exists. The object points (open circles) are to be positioned by the control points (filled circles). If all the object points are constrained to remain stationary except for the middle one and a new goal position (shown by the cross) for the middle point is assigned, it is evident that there are too few control points to yield the undulations requested.

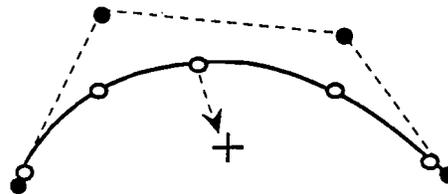


Figure 6.

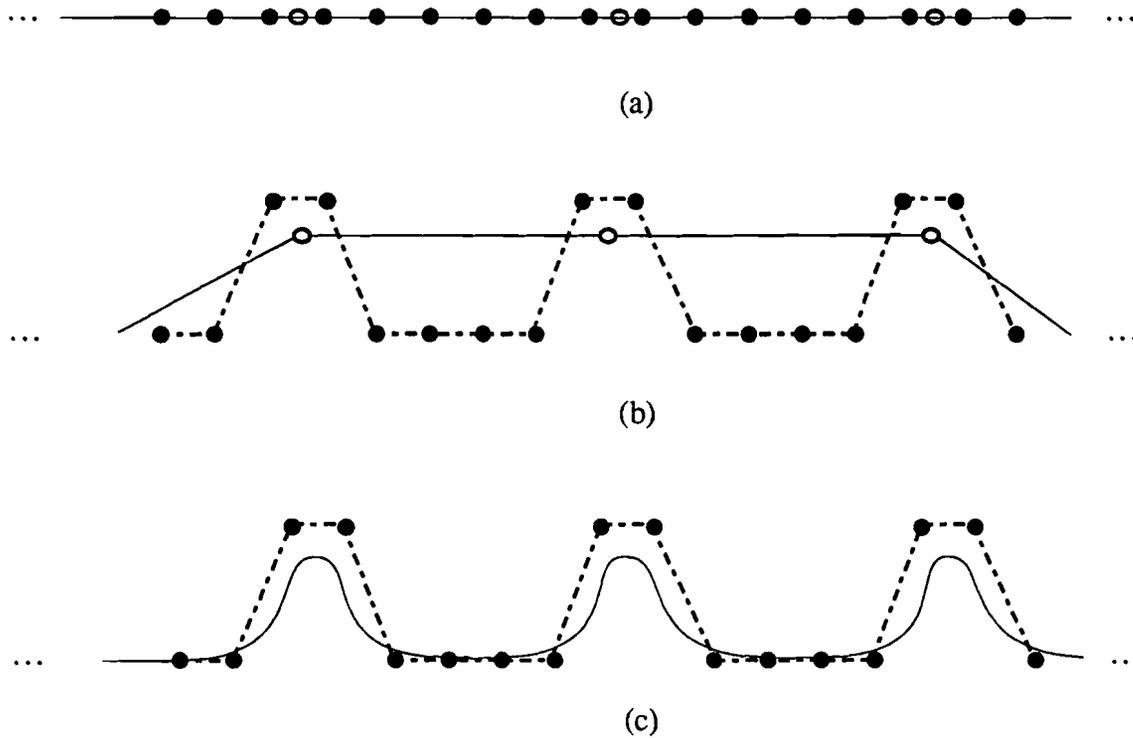


Figure 7. Aliasing because of too many control points relative to the number of object points. Open circles denote object points, filled circles denote control points. No control polygon is drawn. (a) Initial condition. (b) Resulting control-point location when the three object points are moved upward. (c) Possible aliasing if the resolution of the object is dramatically increased.

Over-constrained problems always arise because of a lack of control points. The second limitation has the opposite cause: there are too many control points in relation to the number of object points. Though a solution for the goals can certainly be reached, it is desirable for that general shape of the object remain the same when the tessellation or resolution of that object is increased. When there are too many control points, as in figure 7, not all the control points in the region being modified can move, due to the local control of B-splines. The results look correct at the resolution of the object being modeled, but aliasing occurs when the resolution is increased.

Both of these limitations arise from an inappropriate ratio between the number of control points and number of object points. The interface can attempt to adjust the resolution of the object to achieve a better ratio, but sometimes that is not desirable. In the over-constrained case the best least-squares solution is given, i.e. a best approximation to the movement desired is given, considering the constraints. To eliminate the aliasing problem, the system may create “false” goal points to make sure all control points affecting the surface within the realm of influence are moved. These “false” goal points are simply goal locations that are interpolated from the real deformed object points being moved.

### **3.3.2 Control Points**

Although the user no longer needs to manipulate the control points directly in order to control the deformation, the location of these points are still necessary parameters to the deformation function. The lattice of control points serves other purposes as well. Its size and position as a whole define the volume of space to undergo the deformation. And the resolution of the lattice, that is the number of control points along each axis, determines the extent of the deformation created by each control point.

If the control points are not displayed, these aspects of the control points must nevertheless somehow be conveyed to and controlled by the user. We do this by outlining the parallelepiped region under the influence of the control points in a bounding box that can be manipulated and positioned over the object to be deformed. An entire object need not fall within the bounding box. We can deform only that portion of the object within the bounding box, clipping the rest of the object. Once the bounding box is placed on the desired object, the portion of the object within it can be highlighted, telling the user which part of the object can be deformed.

The influence of each control point spans four segments in each dimension of the lattice, except for those near or at the ends. The more control points there are, the smaller the region affect by any single control point. However, if users are unaware of the control points, as they will be with our interface, then specifying the resolution of the control-point lattice makes no sense. Instead, we use a metaphor; the effect of changing lattice resolution changes both the size of the area being deformed and the acuteness of the curvature. So that the consistency of the object itself appears to change, and we speak of altering the *taffiness* of the object. Increasing the taffiness increases the resolution of the lattice, so that pulling on the object creates thin spikes with high curvature. Low taffiness creates a more gradual rise (or fall) in the deformation, and makes the object seem more rigid. (See figure 8) Since the resolution of the lattice can be different on each axis, each dimension can have a separate taffiness valuator. The only problem with this is that the lattice need not be parallel with the object surface; indeed, it cannot be for objects-like spheres. In this case, it is confusing if the object has a different taffiness in three arbitrary directions, and it is therefore recommended that the resolution be uniform.

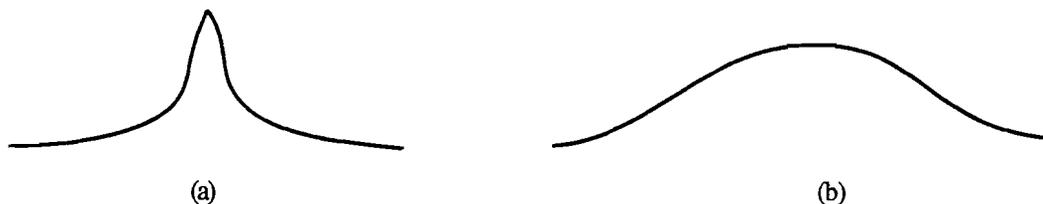


Figure 8. (a) is a curve with high taffiness. (b) is a curve with low taffiness. Both are pulled upward from the center.

Multiple lattices can be applied to a single object so that the user can apply several different types of deformations. Each lattice is separate from the others, and can have a different size, placement and resolution. Likewise, a single lattice can be used to deform multiple objects, so long as a portion of each object lies within the lattice boundaries.

### 3.3.3 System Compensation Due to Interface Abstraction

Hiding the control points from the user implies that limitations visible before under user control must now be handled by the system. For instance, if one can see the large gap between control points created by extending them during a deformation, one can see that a thin hump between them could not be made. However, when the control points are not displayed one has no reason to believe this limitation exists.

To compensate for this loss of information, the interface refines the lattice of control points, filling in the large gaps between them, so that the user can continue to manipulate the object without restriction or surprise. The refinement can be done in several ways, each with its own disadvantages. One might first think of using the Oslo algorithm [Cohe80] (or the “knot insertion algorithm” developed independently by Boehm [Boeh80]) to subdivide the piecewise cubic B-splines by inserting new control points in the areas needed. Any number of control points can be inserted, with any interval, but in order to retain the proper global topology for the entire lattice, duplicate sets of control points must be added in all directions to maintain the grid structure.

Figure 9 illustrates the effect of subdividing the lattice of control points using the Oslo algorithm. Portions of the lattice gain a higher control-point resolution than desired. Because of this, future deformations in those areas will have a different curvature from unaffected regions, leading to unsatisfactory results.

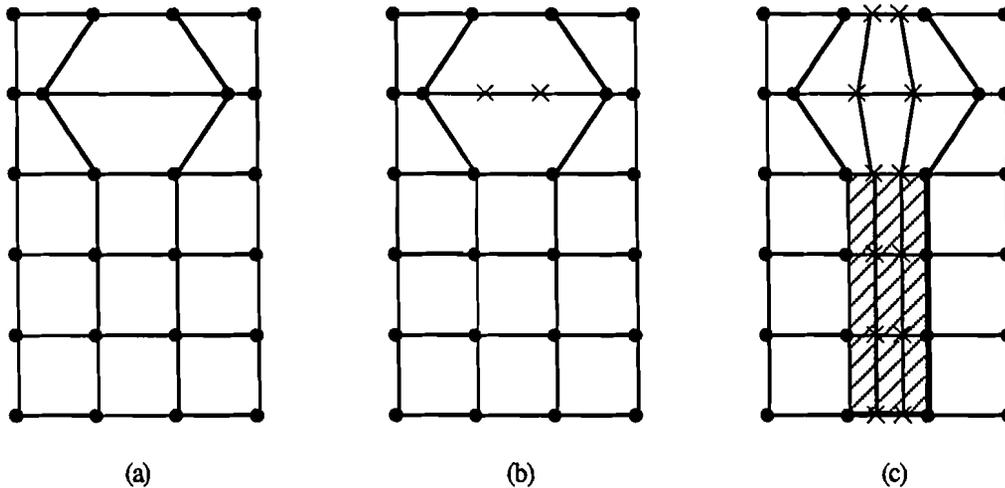


Figure 9. Refinement using the Oslo algorithm. (a) shows the control points (filled circles) right after a deformation. (b) shows where new control points are desired (crosses). (c) shows all the control points added. Shaded area is the portion of lattice that was not supposed to be refined.

The second method would be to extend Forsey and Bartels' hierarchical B-spline model [Fors88] to three dimensions. With this solution the control points are generally added only in the region where they are needed, but it is nevertheless unsuitable for two reasons. First, the subdivision is exponential: that is, each level of refinement subdivides the region twice as much as the previous level. Restricting the resolution to powers of two does not give the flexibility needed to maintain consistent taffiness throughout the object. Second, some areas may be refined when they should not be. For instance, when one area to be refined overlaps an existing area of refinement, the two areas must be consolidated into one rectilinear region, thus creating a larger than desired area of refinement. The gray areas in figure 10 show unintended areas refined due to consolidation.

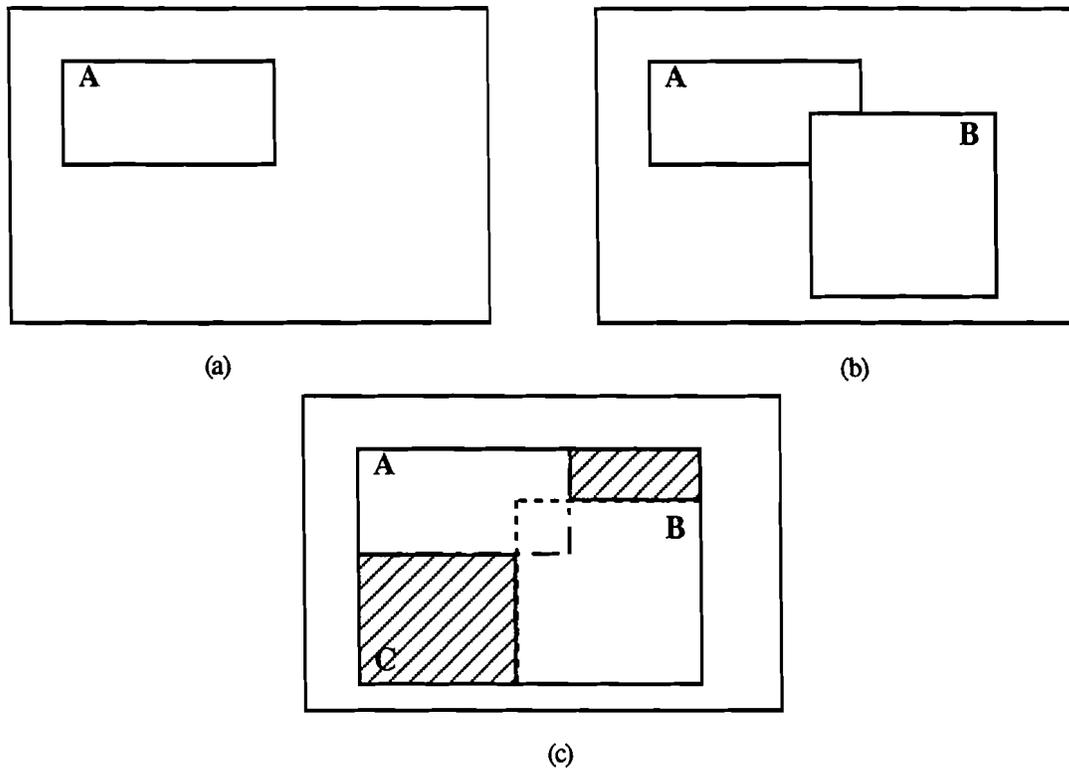


Figure 10. Refinement using hierarchical B-splines. (a) Region A is a refined portion of the lattice. (b) B is a refined area added later, that overlaps region A. (c) Consolidation of the two regions, C, creates unintended refined regions (shown shaded).

The method ultimately employed is quite simple. The lattice of control points is refined by replacing or overlapping it with a new one. The new lattice is a regular parallelepiped of control points that is extended to cover the same region of the object as the old lattice did and uses the same spacing between control points as the old lattice. Now the entire object has the same taffiness as it did before, without the side effects of the other methods and with a minimum of new control points. Figure 11 shows how the old control-point lattice is replaced with the new one.

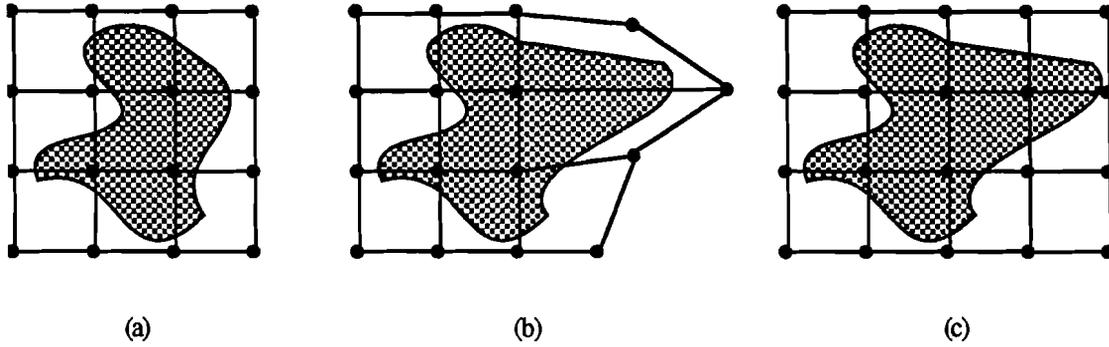


Figure 11. (a) An object (shaded region) and its initial lattice. (b) After a deformation. (c) A new lattice replaces the initial lattice with the same spacing between control points.

### 3.4 Graphical User Interface

We have described up to this point a direct surface manipulation technique to deform objects and a model for its behavior. Two further elements would be necessary to complete the user interface, a 2D graphical user interface for parameter specification and methods for positioning and visually representing the lattice and magnetic tool in 3-space. However, graphical user interface design and 3D positioning are not within the scope of this thesis, and we do not discuss them further here.

## 4 IMPLEMENTATION

The free-form deformation method is only one of many modeling tools that should be at the user's disposal. It is more useful for modifying objects already created than for initial object creation, and is also very useful for animation. For these reasons, this modeling method and the interface described in this thesis have been integrated to some extent into BAGS, the Brown Animation Generation System [Zele91]. BAGS is a unified modeling and animation system that allows the creation of objects, such as spheres, cubes, superquadrics, and ducts; more complex objects are modeled using constructive solid geometry (CSG). The placement, size, and orientation of those objects can be specified for an instant of time or over an interval of time (for creating animations). BAGS also serves as the testbed for research in graphics and user interface techniques

Before implementation in BAGS, all interface functionality was tested in a 2D prototype environment. In the 2D test program the deformation method at first used piecewise uniform cubic B-splines to deform a square object consisting of 100 to 400 connected points. The direct manipulation system for single and multiple constraints (object point positions) was then implemented, and this revealed the need for adaptive lattice refinement and adaptive object refinement. Both were implemented and tested to validate the algorithms used. The lattice is adaptively refined by replacing the old lattice of control points with a new lattice of control points shaped in a parallelepiped, with the same initial control-point spacing as the old lattice (see §3.3.3).

The adaptive object refinement method used increased the number of vertices defining the surface whenever the Euclidean distance between neighboring vertices grew too large.

This served two purposes. First, it maintained smooth curvature throughout the object, no matter how acute the deformation. More importantly, however, it provided more evenly spaced vertices for the user to manipulate after a deformation. Without this refinement, stretching an object would produce long faces with few vertices for the magnetic tool to tug on, and the user would be unable to manipulate certain sections of the object, as shown in figure 12.

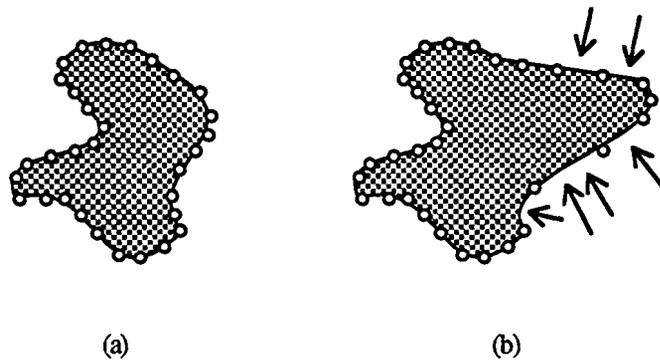


Figure 12. (a) An object (shaded region) and the vertices (open circles) that define its surface. (b) After a deformation, the vertices have large gaps between them; the arrows indicate the places where the user cannot pick a vertex.

New object point locations are computed by first calculating their FFD coordinates, and then using those coordinates to compute the points' world coordinates. The FFD coordinates are computed by linearly interpolating the FFD coordinates of neighboring points. The deformation function is then applied, yielding the world coordinates for the point. The 2D test program also allowed the user to change the resolution of the lattice at any time during the deformation process.

A useful subset of what was implemented in the 2D test program has been implemented as part of BAGS. The standard free-form deformation method, with piecewise cubic B-splines as the basis, has been totally integrated into BAGS as a standard modeling method. Any object in BAGS can be deformed. The traditional method of explicitly moving control

points to specify deformations has been implemented, and those familiar with splines may prefer this method over the direct surface manipulation method. Some of the direct surface manipulation interface has also been implemented into BAGS. The multiple-constraint system is in place, and three tools have been developed to demonstrate the system: there is currently a tool that positions a single object point, a tool that flattens points to a plane defined by the tool, and a tool that moves a section of an object intact to one side of the tool (that is, it applies the same translation vector to all points within its realm of influence).

Some aspects of the interface described here, however, are either not implemented in BAGS at all, or are implemented with stopgap measures. For example, the lattice of control points is not visually represented as a bounding box, but rather by a cube of the appropriate size. The lattice is positioned on the basis of a dependency between the lattice and the cube. The magnetic tool and its realm of influence are represented with an ordinary object, using different colors (through CSG) for the magnet proper and the realm of influence. For the most part, relevant feedback was given by these stopgap measures. For those parts not implemented, the 2D variation implemented in the 2D test program gave a proof of concept and workable methods that can be extended to 3D.

## 5 FUTURE WORK

The foundations for a direct manipulation deformation tool have been laid, but many areas must be expanded in order to achieve a complete, robust interface. A graphical user interface that handles specifying the resolution of the lattice (taffiness of the object), the type of tool the user wants, etc., should be built. More sophisticated 3D manipulation and visualization methods for the magnetic tool can be employed. For instance, 3D input devices such as the Polhemus, the Space Ball, or the VPL DataGlove could be used for 3D manipulation of the tool, instead of the 2D mouse or tablet. Better visualization techniques can be used to help the user see where the magnetic tool is in 3-space and in relation to the object to be deformed. Those techniques can include projecting shadows onto a stage, showing the world or object coordinate axes emanating from the magnetic tool, and (if the tool is confined to a plane) showing the plane and how it intersects the other objects. The realm of influence could also be represented more intuitively, say by a translucent disc or square around the tool.

Adaptive lattice refinement and adaptive object tessellation have been implemented in the 2D test program, and should be expanded to the 3D environment. Close attention must be given to the time at which the lattice of control points is refined in the 3D environment, since the final shape of a deformation is unlikely to be made with a single stroke of the mouse, in the 2D case. Instead, since the pointer usually operates on a single plane, a series of “tug on the object, move the camera” operations may be required for a single deformation. In this case, it is undesirable to refine the lattice after each tug, as was done in the 2D test program, and a more flexible method will be required.

Adaptive object tessellation is also recommended, in order to give the new extended portion of the object enough object points for picking. Otherwise, making large deformations cause rifts between neighboring vertices which can be much wider than the tool. Simply splitting the face and interpolating over the location of the existing vertices is not sufficient, because it does not eliminate the faceting caused by curves in low-resolution objects. Instead, the vertex locations are computed by applying the deformation function to interpolated  $(s, t, u)$  values from the neighboring vertices. Aesthetically, it is important to show the curvature created by the deformation. Without the tessellation, the deformation may look nothing like its representation at higher resolutions, and thus may confuse the user.

As users become more sophisticated in using the tools at their disposal, they often want to expand the functionality provided. If a small editor and parser were provided, the capabilities of the magnetic tool could be extended during run time. Users would write mathematical expressions for the functionality of the new tools they desired. If the editor or parser could read and write files, a complete library of tools could quickly be developed, and a brand-new function could be added without having to recompile or relink the application.

## 6 CONCLUSION

Several advanced modeling techniques have been developed recently, but they are slow to reach the general user, in part because of their poor interfaces. Properly designed interfaces will allow these modeling techniques reach a wider audience and will become more important as the complexity of new modeling techniques increases. The free-form deformation modeling technique is used as an example of these techniques, and we have described an interface developed to show how improvements can be made. With current interfaces, users can become confused by the myriad of control points, and must expend undue cognitive energy to analyze how to move them properly. The interface developed visually eliminates the control points by allowing direct control over the shape of the object. The direct manipulation interface employed lets users shape objects in an intuitive fashion, thereby reducing their cognitive load. This technique can easily be applied to spline surface and curve editors, allowing direct manipulation of those modeling methods as well.

## 7 ACKNOWLEDGMENTS

I would like to thank Henry Kaufman for his invaluable technical advice, encouragement, and interest in my work, and professor John Hughes for his review of my writing and clear explanations of many mathematical details. I could not have accomplished the integration of my project into BAGS without the willing assistance of the BAGS guru, Bob Zeleznik. My attendance at Brown could not have been possible without the financial support provided by Digital Equipment Corporation's Graduate Engineering Education Program. I am also grateful for all the help, comments, and cavorting from the wonderful people in the graphics group and professor Andries van Dam. Most of all, I am thankful for my loving wife, Shari, whose sacrifice, patience and encouragement over the past year and a half has been immeasurable.

## 8 APPENDIX

This section is written to document the integration of new research into BAGS, with the intent of making subsequent integration attempts easier and guiding future enhancements to BAGS. This section assumes that the reader is familiar with the internal workings of BAGS; see [Hubb91], [Conn91], and related documents referenced therein.

### 8.1 Integration with BAGS

The free-form deformation method and interface have been integrated into BAGS as a collection of five change operators, or *chops*:

- *ffdeform* chop - performs the actual free-form deformation to an object.
- *resolution* chop - specifies the resolution of the lattice of control points.
- *move\_cntrl\_pt* chop - specifies how a specific control point is translated.
- *mag\_range* chop - acts as a switch by toggling the objects the magnetic tool will affect, if any.
- *mag\_scope* chop - allows the direct manipulation of the object by computing the placement of relevant control points.

The first three chops, *ffdeform*, *resolution*, and *move\_cntrl\_pt*, are all that is needed to deform an object with the traditional control-point manipulation interface. The latter two chops, *mag\_range* and *mag\_scope*, facilitate direct surface manipulation. We discuss first how a deformation is set up and executed, and then how *mag\_range* and *mag\_scope* provide the direct manipulation.

An object is defined by the change operators (chops) contained in its state. An `ffdeform` chop in an object's state deforms the object. The deformation is specified by a control-point lattice, which is an object in BAGS. The `ffdeform` chop actually takes two lattices as parameters, one as the initial lattice configuration and the other as the final lattice configuration. The two lattices are compared against one another to determine how the object is deformed.

The control-point lattice is initially configured using the resolution chop and linear transformation chops. The resolution chop defines how many control points there are in each dimension; for example, the lattice in figure 2 has a resolution of (2,3,4). The position, size, and orientation of the lattice are defined by the linear transformation chops translate, rotate, and scale. These chops move the control points as an aggregate, i.e. they affect the lattice as whole. Individual control points are manipulated by the `move_cntrl_pt` chop, which positions the control points with relative offsets.

Script ordering is important to create the desired effect, since only those `move_cntrl_pt` chops that occur before the `ffdeform` chop are used in the deformation. Figure 13 shows the relative placements of the chops in their respective object states. The arrow indicates at which point the `ffdeform` chop uses the lattice object (In the current implementation, the `ffdeform` chop only uses the CTM of the initial lattice, so the initial configuration lattice can be the same as the final configuration lattice. The initial configuration lattice parameter is a hook for Extended FFDs).

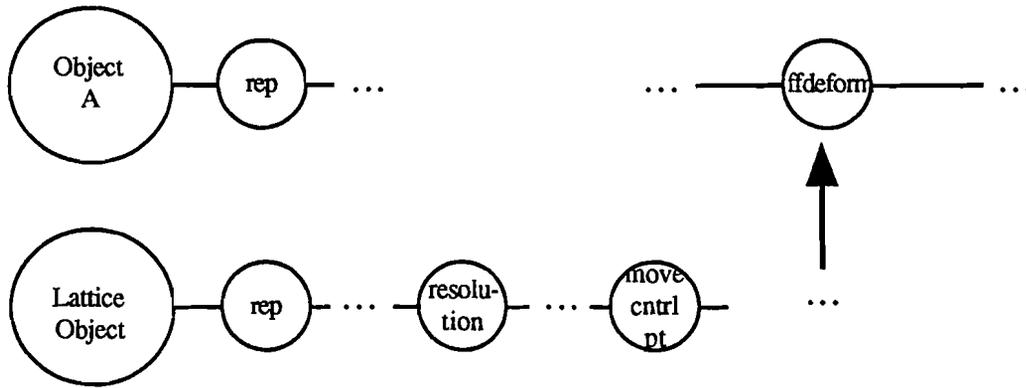


Figure 13. The states of Object A and the control-point lattice. Script priority increases to the right, the small circles represent chops.

For direct manipulation of the object, the magnetic tool is introduced, a new object that computes the location of control points using the `mag_scope` chop. The `mag_scope` chop's parameter list includes the object to be deformed, the lattice it changes, and the group to which the lattice belongs. In order for the `mag_scope` chop to work properly, the deformed object must include changes made by previous iterations of the `mag_scope` chop, and thus the `mag_scope` chop must come after the `ffdeform` chop in script order. The `mag_scope` chop cannot change the control points of a lattice directly because changes to the lattice must come before the `ffdeform` chop in order to take effect. Instead, the `mag_scope` chop writes out `move_cntrl_pt` chops into the state of the lattice before the `ffdeform` chop.

Finding the proper position for a set of `move_cntrl_pt` chops during each iteration of the `mag_scope` command would require state traversals through potentially large states. To narrow the scope of the search, the lattice affected is added as a member to the group that is passed as a parameter to the `mag_scope` chop. The `move_cntrl_pt` chops are added to the group, which relays them to the lattice. This has the additional benefit of duplicating the `move_cntrl_pt` chops for every lattice in the group, which is useful for creating multiple deformations of similar form. Defining the lattice's membership to the group at the proper

time guarantees proper placement of the `move_cntrl_pt` chops. Figure 14 shows how the chops from different objects interrelate with each other to perform the direct manipulation for FFDs.

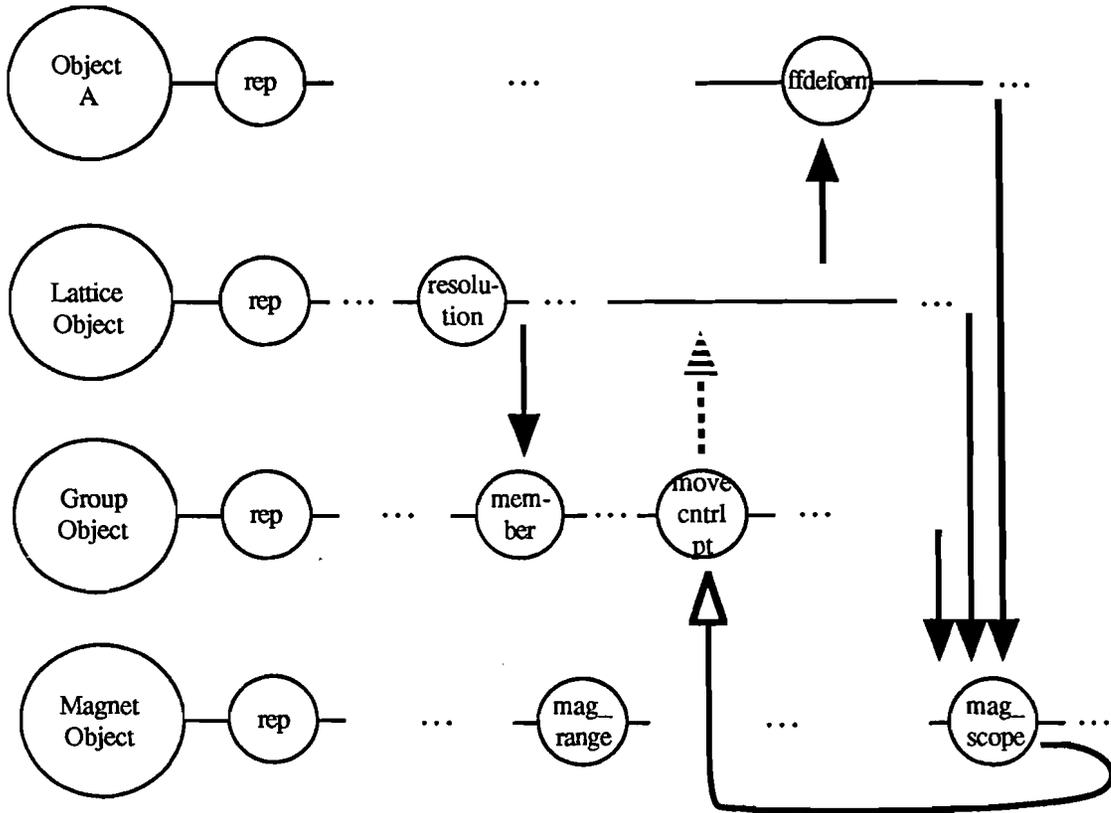


Figure 14. The states of Object A, the control-point lattice, the group, and the magnet. Filled arrows represent parameters to various chops. Open arrow indicate chop written out by `mag_scope`. Dotted arrow represents how the `move_cntrl_pt` chops are passed to the lattice.

A single object can undergo several different deformations, each defined by a different lattice of control points. In the event that such multiple lattices overlap in space or come very close to one another, one must be able to determine which lattice of control points the magnetic tool is to affect. To distinguish the lattices, each is given an associated id that is specified in a parameter to the `mag_scope` chop. Which lattice the tool affects is specified in the `mag_range` chop as a bit mask of lattice ids. (The bit mask is used to let the tool

affect more than one lattice at a time.) The `mag_range chop` must precede the `mag_scope chop` in script order.

This interface utilizes two caches used solely by the free-form deformation operators, an *applied polyhedron* cache and a *vertex id* cache. The applied polyhedron cache is a cache of the polyhedron object, represented by a TRIP object, and all linear transformations applied to it so far. The vertex id cache stores the id of the last vertex moved by the magnetic tool. Like most caches in BAGS, the applied polyhedron cache is used strictly to increase speed, and is created immediately before the `ffdeform chop`. The main purpose of the vertex id cache, however, is its functionality. If the function of the tool is to manipulate a single point until that point is explicitly released, then the id of that point must be stored and fetched for each iteration through the `mag_scope chop`. Without the vertex id cache, the closest point to the magnetic tool would be sought at each iteration, and this might not be the same point each time.

## 8.2 Integration Obstacles

The FFD technique and the direct manipulation interface are implemented through a collection of chops. This, however, is not the only way research is integrated into BAGS. The FFD technique and direct manipulation interface could have been implemented together as a controller, for example. BAGS was still being developed concurrent to the integration of this research, and the strict use of chops can be attributed to BAGS's lack of maturity, no documentation, and the incremental design of the FFD project. Indeed, this implementation brought to light several deficiencies in the current framework of BAGS for this type of implementation. Different implementations, however, may or may not have the same problems.

One of the drawbacks of this implementation is the lack of persistent computed data. In order to get a representation of the object at a particular stage, the system must traverse the entire state of the chop and accumulate the changes made to it up to the point of the request. This can become extremely inefficient for complex interactive modeling tasks like FFD, since each change specified in the FFD command requires a state traversal and accumulation of changes. For example, each time the mouse moves to change the location of a control point, the entire state of the object leading up to the deformation, and the deformation itself, are recomputed. For a complex, highly tessellated model, this computational expense is prohibitive. Fortunately, BAGS's caching mechanism retains intermediate stages of some commands and objects, and this eliminates some, though not all, of the recomputation. For free-form deformation, one particular cache contains the polyhedral object in the state just before it is deformed. The deformation command (the `ffdeform chop`) takes as one of its parameters a reference to a lattice of control points whose position defines the shape of the deformation. Since the cached polyhedral object does not retain any of the deformation information caused by this deformation command for any previous iteration, all deformations are calculated, even those done before and unaffected by the current changes to the lattice of control points. This potentially eliminates any computational efficiency gained from the local-control attribute of the cubic B-splines.

If implemented as a controller within BAGS, the FFD interface could have operated with fewer state traversals. The controller would be a mini-modeler within BAGS, able to do whatever it wanted. However, isolating the FFD interface from the rest of the system creates other problems. For instance, modeling would be done without the context of other objects, and care would be needed in writing and updating chops so changes would be saved in the script. Since chops need to be written anyway, there is nothing to prevent creating a controller layered on top of the existing FFD implementation.

The change operators in BAGS are assumed to be atomic operations: the processing of a particular chop must complete before any other operation can begin. This prohibits displaying the intermediate results of a chop, which could be helpful to the user. For iterative constraint systems that may take significant time to compute, it would be desirable to show the user intermittent progress.

Although the BAGS architecture documentation claims that caching is done only for speed, this is not entirely true. For example, the magnetic tool that allows the user to drag a point first finds the point by looking for the closest one to the tool; it then continues to use that point until the user turns the magnetic tool off. To continue to move the same point, the vertex id is cached and retrieved on each iteration through the `mag_scope` command.

Without the cache, the entire object would be searched each iteration for the closest vertex, and different ones might well be found as the tool moved along the surface of the object. Although, the vertex id can be stored elsewhere, e.g., as part of the TRIP object, using a cache was the most appropriate option at the time.

In general, the polyhedral representation used in BAGS makes associating data with objects difficult. If the data is stored separately, then if on the next iteration the object is compressed, the link between the object and the data will be out of sync. If the data is stored with the object itself, then any invalidation of that object causes the data to be lost, even if the reason for the invalidation does not apply to the extra data. For example, if a color was associated with each vertex and the geometry of the object changed, the color information would be lost, even though it had no correlation with the geometry of the object. The invalidation conversion facility can avert such problems, but its abilities are limited and it requires the programmer to become very intimate with the invalidation mechanism. Because of these invalidation problems, passing some types of data from one

chop to another can be precarious. For instance, the `ffdeform` chop generates extra data and attaches it to the polyhedral object being deformed. This data is also used by the `mag_scope` chop and is passed along through the polyhedral form. It is essential that the data come from the `ffdeform` chop associated with this `mag_scope` chop, and no other, since the `mag_scope` chop cannot generate the data on its own. If a chop that compresses the TRIP object comes in between the `ffdeform` chop and the `mag_scope` chop, the data will be lost. This causes an interleaving of chops that the script writer must be aware of when dealing with multiple deformations to the same object.

Figure 15, which shows a portion of a SCEFO script used to model a head, indicate the importance of script ordering. We note that the `mag_range` chop appears before any of the `ffdeform` chops and that the `mag_scope` and `ffdeform` chops are interleaved. A closer look at `eyelat2` shows how delicate the structure is. `Eyelat2` deforms an eye socket of the head, but is not directly manipulated by the tool. Since we want the eyes to be symmetrical, we simply deform the other eye and place both `eyelat1` and `eyelat2` in the same group, so that its `move_cntrl_pt` chops affect `eyelat2` as well by placing. We cannot, however, sequentially group together their `ffdeform` chops and place them before the `mag_scope` chop that affects the `eyelat1` lattice. Although the magnetic tool appears to deform both sockets simultaneously, two separate deformations are in fact involved. If the deformation for `eye2` is placed in between that for `eye1` and the `mag_scope` chop for `eyelat1`, the `ffdeform` chop for `eye2` will corrupt the data passed from the `ffdeform` chop for `eye1` to the `mag_scope` chop. Thus the deformation must be moved after the `mag_scope` chop. It could also be placed before the `ffdeform` chop for `eye1`, so long as the affected areas do not overlap.

The restriction in placing the `ffdeform` chop for `eyelat2` can be lifted if the TRIP object carries multiple extra data columns, one for each deformation, but that leaves open the possibility of increasing the size of the TRIP object without bound.

In summary, a firm understanding of BAGS is needed before a well-thought-out integration plan can be formed. Clear and comprehensive documentation is necessary for all those attempting to integrate their research in BAGS. The object-oriented design of BAGS gives you just enough rope to hang yourself, if you're not careful.

```

eyelat1      : rep      0 = [latticeClass, 0]
               resolution 0 = [5, 5, 2]
               scale      0 = [1.0, 1.0, 1.0] : (linear, 1)
               translate  0 = [0.0, 0.0, 0.0] : (linear, 1)
               ctm        0 = temp1.ctm ;

eyelat2      : rep      0 = [latticeClass, 0]
               resolution 0 = [5, 5, 2]
               scale      0 = [1, 1, 1] : (linear, 1)
               translate  0 = [0, 0, 0] : (linear, 1)
               ctm        0 = temp2.ctm ;

mouthlat     : rep      0 = [latticeClass, 0]
               resolution 0 = [7, 7, 2]
               scale      0 = [1, 1, 1] : (linear, 1)
               translate  0 = [0, 0, 0] : (linear, 1)
               ctm        0 = temp3.ctm ;

head         : rep      0 = [sphereClass, 0]
               resolution 0 = [7]
               scale      0 = [3, 4, 3]
               translate  0 = [0, 0, 0] ;

headlat      : rep      0 = [latticeClass, 0]
               resolution 0 = [9, 9, 4]
               scale      0 = [1, 1, 1] : (linear, 1)
               ctm        0 = head.ctm ;
               translate  0 = [0, 0, 0] ;

headgroup    : rep      0 = [groupClass, 0]
               member     0 = [Add, [headlat]] ;

eyegroup     : rep      0 = [groupClass, 0]
               member     0 = [Add, [eyelat1, eyelat2]] ;

mouthgroup   : rep      0 = [groupClass, 0]
               member     0 = [Add, [mouthlat]] ;

tool         : rep      0 = [sphereClass, 0]
               mag_range  0 = [0.5, (0*2)]
               scale      0 = [0.5, 0.5, 0.5] ;

head         : ffdeform  0 = [headlat, headlat, 1] ;

tool         : mag_scope 0 = [1, head, headgroup, headlat, 1, 2];

head         : ffdeform  0 = [eyelat1, eyelat1, 1] ;

tool         : mag_scope 0 = [2, head, eyegroup, eyelat1, 1, 2];

head         : ffdeform  0 = [eyelat2, eyelat2, 1]
               ffdeform  0 = [mouthlat, mouthlat, 1] ;

tool         : mag_scope 0 = [4, head, mouthgroup, mouthlat, 1, 2];

```

Figure 15. SCEFO script fragment for modeling a head through deformations.

## References

- [Alle89] Allen, J.B., Wyvill, B., and Witten, I.H., "A Method for Direct Manipulation of Polygon Meshes", *Proceedings of Computer Graphics International '89*, Earnshaw and Wyvill, eds., pp. 451-469, 1989.
- [Bart87] Bartels, R.H., Beatty, J.C., and Barsky, B.A., *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, 1987.
- [Boeh80] Boehm, W., "Inserting New Knots into B-spline Curves", *Computer-Aided Design*, Vol 12 No 4, pp. 199-202. July 1980.
- [Carl82] Carlson, W.E., "Techniques for the Generation of Three Dimensional Data for Use in Complex Image Synthesis", PhD dissertation, Ohio State University, September 1982.
- [Clar76] Clark, J.H., "Designing Surfaces in 3D", *CACM*, Vol 19 No 8, pp. 454-460. ACM, August 1976.
- [Cobb84] Cobb, E.S., "Design of Sculptured Surfaces Using the B-spline Representation", PhD dissertation, University of Utah, June 1984.
- [Cohe80] Cohen, E., Lyche, T., and Riesenfeld, R., "Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics", *Computer Graphics and Image Processing*, Vol 14 No 2, pp. 87-111. October 1980.
- [Conn91] Conner, D.B. and True, T.J., "A Reference Manual for SCEFO", BAGS documentation, Brown University, 1991.
- [Coqu90] Coquillart, S., "Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling" *Computer Graphics (Proceedings of SIGGRAPH '90)*, Vol 24 No 4, pp. 187-196. ACM August 1990.
- [Fari90] Farin, G., *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.
- [Fole84] Foley, J.D, Wallace, V.L., and Chan, P. "The Human Factors of Computer Graphics Interaction Techniques", *CG&A*, pp. 13-48, IEEE November 1984.
- [Fors88] Forsy, D.R. and Bartels, R.H. "Hierarchical B-spline Refinement" *Computer Graphics (Proceedings of SIGGRAPH '88)*, Vol 22 No 4, pp. 205-212. ACM August 1988.
-