Paradigms for the Shaping of Surfaces in a Virtual Environment

Steve Bryson†

RNR Technical Report RNR-92-012, January 1992

Applied Research Branch, Numerical Aerodynamics Simulation Division
NASA Ames Research Center
MS T27-A
Moffett Field, Ca. 94035
bryson@nas.nasa.gov

Abstract

This paper describes several paradigms for the directly manipulating a surface in computer graphics. The surface is considered as a collection of points in three-dimensional space which define a surface. These points may be the vertices of polygons, the control points of splines, and so on. Three paradigms are described and evaluated: direct grabbing of the surface at a single point and moving a neighborhood of that point; pushing on the surface with a tool; and pushing on the surface with a model of the user's hand. These paradigms are based on a spatially weighted transformation. This transformation is based on a 'bump' weight function on the surface which is shaped and placed by the user. Construction of the bump function is described and the definition of the weighted transformation is described. The use of the weighted transformation in each paradigm is also described. These paradigms are implemented on a virtual environment system based on a Fake Space BOOM for display and a VPL Dataglove Model II for manipulation. The computation and rendering platform is a Silicon Graphics 380GT/VGX.

† Employee of Computer Sciences Corporation. Work supported under government contract NAS 2-12961

1: Introduction

This paper describes several paradigms for manipulating the shape of computer generated surfaces in virtual environments. The methods suggested enable a user to use natural hand motions to create complex and irregular computer-generated shapes (fig. 1). Three paradigms are described and evaluated: direct grabbing of the surface at a single point and moving a neighborhood of that point; pushing on the surface with a tool; and pushing on the surface with a model of the user's hand. These paradigms are based on a spatially weighted transformation. This transformation is based on a 'bump' weight function on the surface which is determined by the user. The value of the weight function at a point is defined as a function of the distance of that point from some selected point or set of points. It is the choice of the definition of distance and how the selected points are specified that distinguishes the different paradigms.

1.1: Purpose and Motivation

Virtual environments are a powerful new way of approaching computer graphics and user interfaces [2][3]. The problem of manipulating and shaping surfaces in virtual environments is as old as the idea of virtual environments. The goal of presenting the user with a virtual surface and allowing the direct manipulation of that surface by the user's hand is an obvious one. It is unclear, however, what the details of such an interface would be. A surface is a collection of points, and the user's hand can also be thought of as a collection of points. The use of the hand to shape the surface is, at the algorithmic level, some map from the set of points in the user's hand to the set of points in the surface. Many such maps are possible, and so the problem is, in general, very complex. There have been several approaches to this problem using spline-based concepts [4][8] and physical models [9]. This paper describes another approach to the problem of mapping the movement of the user's hand to movements of points in a surface. Our approach has the advantage that it allows manipulation of groups of arbitrary points in space, without requiring underlying data structures. These points can be considered as the control points of splines, or as defining surfaces directly. Several different interfaces to this approach are presented.

In the paradigms for surface shaping, the desired intuition is that the surface act like an infinitely stretchable material. We are intentionally avoiding defining the surface via some physical model to allow the user great versatility in manipulation. The manipulation should be natural in the sense that when a point on the surface is moved via some transformation, other points on the surface follow the motion in a way reminiscent of taffy or putty. We present and discuss three such paradigms for the manipulation of a surface:

Grabbing: selecting a single point of the surface, specified by the position of the user's hand, and moving a predefined neighborhood of that point rigidly with the movement of the hand. The points outside this surface move less and less the further away they are, until beyond some predefined distance on the surface the points do not move at all (fig. 2).

Pushing: Pushing the points of the surface around with a virtual tool of some predefined shape. The position of the tool follows the user's hand (fig. 2).

Kneading: Pushing the points of the surface around with a virtual hand model which precisely follows the motion of the user's hand. This push can be based on the entire hand shape or the position of the fingertips (fig. 2).

The kneading paradigm is really a version of the pushing paradigm in which the virtual tool is the model of the user's hand. The external interface to the kneading paradigm is, however, sufficiently different from that of the pushing paradigm that it will be treated separately.

Underlying these paradigms is the idea that some subset of points on the surface is constrained to move exactly with the user's hand, while some other subset of points is constrained to not move at all. The remaining points on the surface interpolate between these two subsets. This is accomplished by defining a weight function on the surface which is exactly equal to: 1 at those points which are constrained to move exactly with the user's hand, 0 at those point which are constrained to not move at all; and some value between 0 and 1 at the remaining points. A linear transformation is defined by the state of the user's hand and is multiplied by this weight function to define a linear transformation at each point of the surface. Each point of the surface is then mapped to its image under this transformation, defining a new surface which replaces the old one (fig 3).

Note that it is a discrete set of points on the surface that is manipulated. The surface is treated as a collection of points with connectedness, and some subset of these points is moved during the manipulation. The surface is rendered using the points by either treating these points as the vertices of polygons, control points of splines, or some other rendering method. The connectedness of the points determines the topology of the surface.

While the basic idea of the points moving near the hand is an obvious one, the user may also wish for more exotic and non-local possibilities. These may include grabbing the entire edge of a surface, moving a ring on the surface around the point grabbed (without moving the center of the ring), or moving the entire surface as a rigid body. In general, there may be situations where moving an arbitrary subset of the surface is desired. The method described in this paper allows in principle for all of these possibilities, and a limited implementation with many of these features will be described.

The system described in this paper uses concepts that have been discussed by Barr [1] and Parent [7]. Barr anticipated the use of spatially variable transformations, and Parent anticipated the demand of manipulating surfaces as stretchable material.

While the method described in this paper is discussed in terms of the manipulation of two-dimensional surfaces in three-dimensional space, the generalization to higher (or lower) dimensions is straightforward.

1.2 Brief Description of the Algorithm

We now summarize the algorithm behind the spatially weighted transformation. The intuition behind the paradigms described in this paper is to allow the user to 'reach out' and 'grab' a virtual surface with a glove, and as the user moves the glove:

a) some specified region of the surface moves precisely as indicated by that device,
b) some other specified region of the surface does not move at all

Fig 3: Three stages in the manipulation of a surface, involving both translation and rotation.

and

c) the rest of the surface nicely interpolates between these two regions.

This system treats the surface as a collection of vertices with connectedness, and defines a transformation T on these vertices via some input device. A smooth mask m is generated which is equal to 1 in some specified region where the vertices will be transformed by T, 0 in some other specified region where the vertices will not be transformed at all, and some number between 0 and 1 everywhere else, so that these remaining vertices will be transformed by m*T.

In effect, we are defining a weighting function m(v) on each vertex v such that 0   m(v)   1. A weight function which has continuous derivatives to all orders (i. e. is smooth) everywhere except for a one-dimensional curve (where the second derivative is discontinuous) will be used. This means that in practice, since the probability of a vertex falling on this curve is zero, the transformations on the vertices will be smooth everywhere. While many other weight functions that are less smooth will be similarly well behaved in practice, we have chosen one that a) has a nice shape, b) is almost smooth, and c) is not too time consuming to compute (A similar function that is smooth everywhere is defined in terms of integrals which must be evaluated numerically).

The value of the weight function at a vertex is determined by the distance of that vertex from some specified set. The nature of this set and the definition of distance depend on the interaction paradigm being used.

In the grabbing paradigm, a 'grabbed point' on the surface is defined, and the weight function on any other point of the surface is determined by its distance from the grabbed point along the surface. In this way the weight function is guaranteed to be a local function on the surface, even in the case of self-intersection of the surface. This problem will be discussed in more detail in section 2.2.

In the pushing paradigm, a virtual push tool is defined in three-dimensional space. The value of the weight function at a point on the surface is then a function of the three-dimensional distance from that point to the push tool. In the example considered in this paper, the push tool is a single point which follows the position of the user's hand.

In the grabbing paradigm, the weight function is defined much like the case of the pushing paradigm, except that now the tool is a dynamic model of the user's hand. This model has fingers that bend to match the state of bend of the user's hand. Thus this paradigm involves a push tool that changes shape. In the examples considered here, the distance used to define the value of the weight function may be the distance to the closest point on the hand model or the distance to the closest fingertip.

Once a single manipulation of the surface has been performed, the vertices defining the surface are replaced by their images under the masked transform described above, producing a new surface which may again be manipulated. In this sense, this system is iterative: the surface is replaced by its manipulated image.

1.3 The Virtual Environment Hardware

The display component of our virtual environment is the boom-mounted display (fig. 4). This boom supports two small CRTs on a counterweighted yoke attached through six joints to a base. It is manufactured by Fake Space Labs of Menlo Park CA., and fashioned after the prototype developed earlier by Sterling Software, Inc. at the VIEW lab at NASA Ames Research Center [6].

The boom is an alternative to the popular head-mounted LCD display systems that were pioneered at the VIEW lab [2] and are now widely used. The main advantage of the boom is that real CRTs can be used for display in spite of their mass, since none of the weight of the displays is born by the user. CRTs have much better brightness, contrast, and resolution than standard liquid crystal displays.

The CRTs are mounted on the “head” of the boom, along with the wide field optics and two repositionable handles. Six degrees of freedom of motion are provided by the the gimbals and joints of the boom, in a smooth and force-free manner. Within a very wide range, the user can continuously change the three dimensional position and orientation of the head of the boom. The position and orientation information is based on the the current state of the six joints angles. These angles are sensed by optical encoders at the joints and fed into a microprocessor in the base of the boom, which formats the information and sends it out an RS232 port. No magnetic field emitters or sensors are used, and hence the boom information is precise, repeatable, and insensitive to the electromagnetic environment. Currently, the monitors on the boom are monochrome. The boom accepts two RS170 video signals, one for each eye.

The motion of the boom is relatively effortless and completely smooth. When used from a standing position or from a sitting position in a wheeled office chair, it provides ample freedom of motion. There are no straps, no weight on the head, and it allows immediate disengagement to allow the user to work in a desktop environment.

In addition to the boom head position and orientation, the user's hand position, orientation, and finger joint angles are sensed using a VPL dataglove™ model II, which incorporates a Polhemus 3Space™ tracker. The finger joint angles are combined and interpreted as gestures by a low level of the software. These gestures are used to send commands to the software. The finger joint angles are used to construct the virtual hand model in the kneading paradigm.

The keyboard and mouse are also used as input devices to the virtual environment. The boom can be easily swung away from the user's eyes and attention may be refocussed on the normal computer screen. The user, who is seated, can then return to typing and interacting with the computer in the usual way. For small amounts of typing, and for controlling the mouse, the glove need not be removed, since it is quite thin and flexible.

The computational and rendering power for our virtual environment is provided by a Silicon Graphics Iris 380 VGX system. This is a multiprocessor system with eight 33 MHz RISC processors (MIPS R3000 CPUs with R3010 floating point chips). The performance of the machine is rated at approximately 200 Million instructions per second (200 VAX MIPS) and 37 million floating point operations per second (37 64-bit linpack MFLOPS). Our system currently has only 48 MBytes of memory.

The VGX has parallel hardware rendering pipelines. The rated graphics performance of our system is around one million 3D triangles transformed, clipped, projected, lit, shaded, and displayed per second. The system has over 200 bits per pixel of frame buffer memory. We make use of only 48 bits per pixel - two buffers each of eight bits of red and eight bits of blue (double buffering), and 24 bits of Z-buffer.

Stereo display on the boom is handled by rendering the left eye image using only shades of pure red (of which 256 are available) and the right eye image using only shades of pure blue. When the blue (second, right-eye) image is drawn, it is drawn using a “writemask” that protects the bits of the red image. The Z-buffer bit planes are cleared between the drawing of the left- and right-eye images, but the color (red) bit planes are not cleared. Thus, the end result is separately Z-buffered left- and right-eye images, in red and blue respectively, on the screen at the same time with the appropriate mixture of red and blue where the images overlap.

The 1024x1280 pixel RGB video output of the VGX is converted into RS170 component video in real time using a scan converter. While the VGX can put out RS170 directly by setting a software switch, we have found that scan converting the higher resolution 1024x1280 image using dedicated hardware provides spatial antialiasing, and consequently noticeably better image quality when viewing with the boom. The red RS170 component is fed into the left eye of the boom, and the blue RS170 component into the right eye. The synch is fed to both eyes. Since the boom CRTs are monochrome, we see correctly matched (stereo) images.

The configuration of our system operating in virtual reality mode is drawn in figure 5.

1.4 The User Environment

The virtual environment used to test the paradigms described in this paper has several features which enhance the usability of the system. A virtual floor made of cross-hatched lines gives a spatial orientation cure. This greatly enhances the illusion of depth when viewing the virtual surface. A three-dimensional cross shaped cursor always tracks the user's hand. Except when using the kneading paradigm, this is deemed more desirable than the conventional virtual hand model because it does not obstruct the user's view and, less importantly, for speed of rendering.

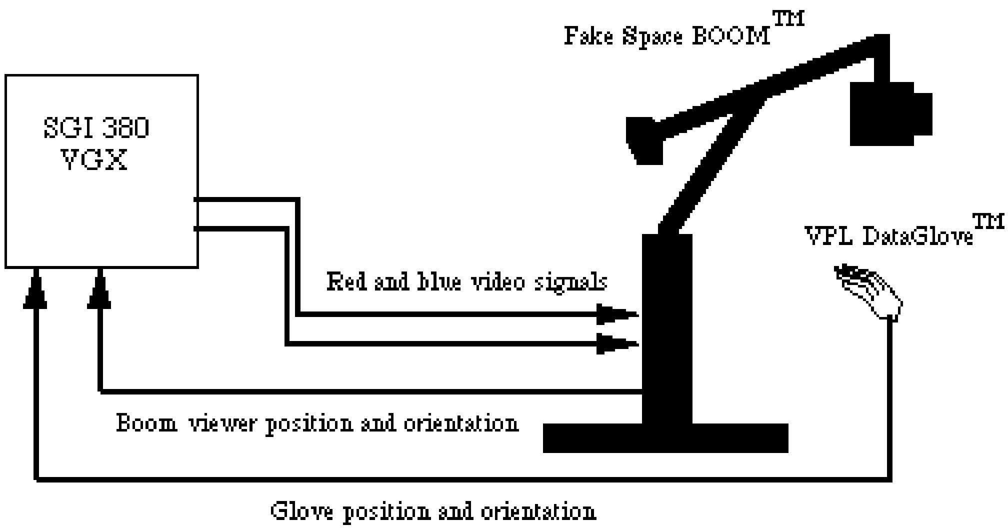


Fig 5: The hardware configuration of the virtual environment system. Position and orientation data from the glove and boom are sent to the Silicon Graphics 380 VGX workstation. This data is used to compute and render various objects in the environment. The rendered stereo images are sent to the boom as video signals.

The rendering of the surface which the user manipulates is done in a checkered fashion, as this gives good cues to the shape of the surface. This extra cue was found to be very useful in the perception of the detailed shape of the surface. Other options are available to the user. In addition to shaping the surface, the user can, via various gestures, move, orient and scale the surface in virtual space as desired. Several surfaces can be inserted into the environment and manipulated independently. The resolution of the surface, the number of vertices in the definition of the surface, can be changed on the fly. Finally, each of the paradigms described in this paper can be accessed at any time by the user.

2: The Spatially Weighted Transformation

2.1 Definition of the bump weighting function

When the user manipulates the surface, three disjoint subsets of the surface need to be specified:

a) The part of the surface that moves rigidly according to the user's input

b) The part of the surface that does not move at all (possibly empty)

c) The complement of the above two subsets, where the motion is interpolated (possibly empty)

These subsets will be supplied with a weighting function, defined relative to these subsets, which will control the motion with which the points in these subsets move. The motion will be defined via a transformation T, and the weighting function will multiply T to define a new transformation T' on each vertex. Thus a weighting function is needed that is constant and equal to one in some set of numbers corresponding to subset a), is constant and equal to 0 outside some larger set of numbers corresponding to subset b), is monotonically decreasing from 1 to 0 in the numbers corresponding to subset c), and is smooth. A function which is versatile, easy to control, and has a nice shape is the bump function.

The construction of this function is based on the function

f(x) = Blc{(Aalhs5co1(0x 0, eSup5(-Bbc((xSup5(-2)))Sup5(x>0)))

f(x) has well-defined derivatives to all orders at 0 (its derivatives are of the form (polynomial in x)*(polynomial in (1/x))*f(x), and f(x) will go to zero faster than the polynomial in (1/x) diverges as x goes to zero) [5, exercise 1.1.18]. This function is used to construct the smooth step function

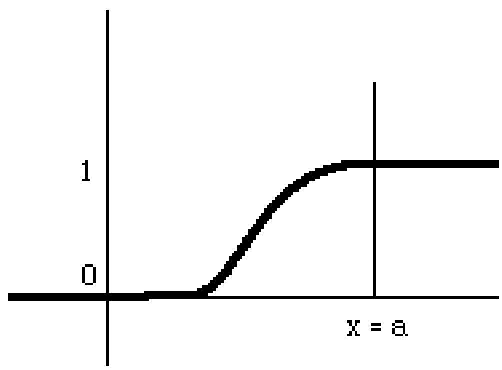
s(x, a) = Blc{(Aalhs5co1(0x 0, eSup9(-Bbc((F(x,a-x))Sup9(-2))Sup5(0<x<a), 1x a))

which has continuous derivatives to all orders at x=0 (fig. 6). At x=a, where a is any positive number, this function has discontinuous second and higher derivatives. By translation, the function

step(x, r0, r1) = s(x-r0, r1-r0)

is a function that is equal to 0 for x r0, equal to 1 for x r1, and is smooth between r0 and r1, where 0 < r0 < r1. Finally, one can define the bump function, given by

bump(x, c, r0, r1) = Blc{(Aalhs5co1(step(x, c-r0, c-r1)x < c,step(2c-x, c-r0, c-r1)x c))



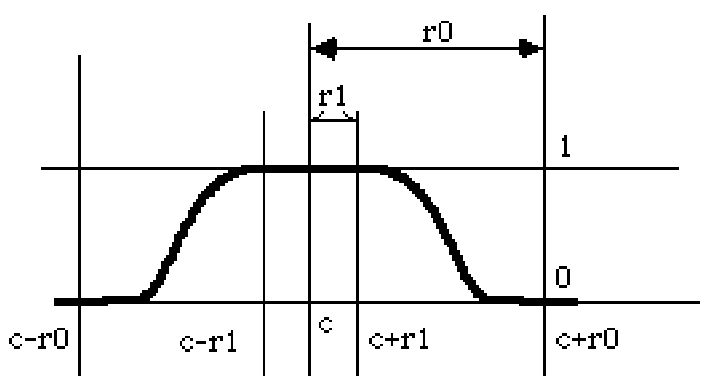


Fig 6: The basic step function Fig. 7: The bump function, showing the

s(x, a). parameters c, r1, r0 which define the bump.

where c is any number which gives the center of the bump, r1 is the distance from the center within which the bump is equal to 1, and r0 is the distance from the center beyond which the bump is equal to 0 (fig 7).

The function bump(x, c, r0, r1) defines a bump which is controlled by three parameters: c, which is where on the real line the bump is centered; r0, the width of the nonzero parts of the bump; and r1, the width of the plateau where the bump is constant and equal to one. The task of controlling the bump is the task of controlling these three parameters.

In this way, the bump function can specify the desired subsets of the surface with only three parameters.

2.2: Definition of the vertex transformation for the local manipulation of a surface.

The bump function defined in section 2.1 is very useful for specifying the way in which a surface is manipulated. Intuitively, when a user specifies that a surface has been grabbed at a particular point p and moves the controller to define some transformation T, then the surface close to p should be fully transformed by T, moving with the controller. The effect of the transformation should fall off gradually with increasing distance from p on the surface, until far away, the surface does not move at all. This is accomplished by using the bump function with x = distance from p and taking c to be zero. At each vertex of the surface, the transformation at that vertex is defined as:

T'(x) = bump(x, c, r0, r1)*T + (1-bump(x, c, r0, r1))*Id

where x is the distance of the vertex from p and Id = the identity transformation. Where bump(x, c, r0, r1) = 1, T'(x) = T, and where bump(x, c, r0, r1) = 0, T'(x) = Id. Note that T need not be a linear transformation, but can be any map from three-dimensional space to three-dimensional space.

Fig 8: Three basic transformations: translation, rotation, and scaling. The scaling is performed after a translation.

Examples of transformations that would be used in a typical manipulation application include translation, rotation, and scaling. These transformations would be controlled with mouse and keyboard combination commands. In this case, when the user indicates that a vertex p (with components (px, py, pz)) on the surface is to be grabbed, the value of the bump function centered at p is computed at each vertex v of the surface and is stored in the array bval[v]. Then, by moving the input device, the user indicates the translation (tx, ty, tz), rotation **R**, and scale transformation (scalex, scaley, scalez) that are to be applied to the surface (fig. 8). The transformation at each vertex v is weighted by the value in bval(v) at that vertex as follows:

T'(v) = bval[v] Bbc((Aalhs5co3(scalex,0,0,0,scaley,0,0,0,scalez)) **R** + Bbc((Aalhs5co3(1-bval[v],0,0,0,1-bval[v],0,0,0,1-bval[v]))

This transformation is then applied to the current components of vertex v = (vx, vy, vz) as

Bbc((Aalhs5co1(vx',vy',vz')) = **T**'(v) Bbc((Aalhs5co1(vx-px,vy-py,vz-pz)) + Bbc((Aalhs5co1(px+tx*bval[v],py+ty*b-val[v],pz+tz*bval[v]))

Thus the transformations are applied to each vertex centered around p. In particular, the rotations and the scaling occur centered at p.

There is one subtlety in this scheme: the definition of distance on the surface. One could use the euclidean three-dimensional distance between the vertices, but there are cases, such as in the grabbing paradigm, when one wishes that the manipulation is local with respect to the surface, not how the surface is embedded in three-dimensions. This is a problem, for example, when the surface is folded over on itself, say folded corner to corner. If the three-dimensional distance between points is used, grabbing one corner will cause the opposite corner to move (See fig. 9). The points in the center of the surface, which we usually call closer to one corner than the opposite corner, will move less, and the effect could be that we have grabbed the two corners at once. This is not what is intuitively meant by "grabbing the surface at a point".

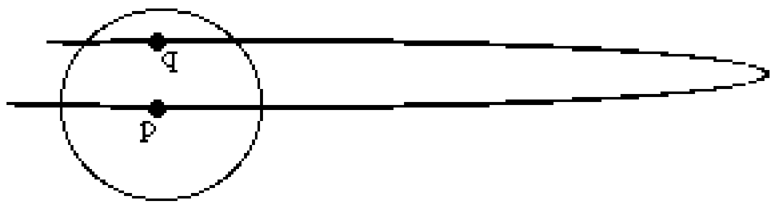


Fig 9: A cross section of a folded surface. The circle shows where the weight function is nonzero if the weight function is a function of the distance between the vertices on the surface. Then if vertex p is grabbed and moved, vertex q will also be moved, even though p is far from q with respect to distance measured on the surface.

In this case, what is needed is a definition of distance along the surface that is both computationally reasonable and respects the topology of the surface. The solution to this problem is to define a 'prototype surface' which contains vertices and connectedness that are in a 1-1 correspondence with the manipulated surface, and so has the same topology as the manipulated surface (fig 10). The distance between two points on the manipulated surface is defined as the three-dimensional distance between the corresponding two points on the prototype surface. For example, if the manipulated surface has the topology of a sphere, the prototype surface will be the geometrical sphere. If the manipulated surface has the topology of a torus, the prototype surface will be the geometrical torus.

When the parameter c in the bump function is taken to be non-zero, then the weight function will form a ring about the point p. This allows for interesting non-local manipulations of the surface. An example of such a manipulation in the grabbing paradigm is shown in fig. 11. If $c > r0$, there will be an area in the center of the ring, centered at p, that will not move at all.

Fig 10: Left: A manipulated surface with the topology of a sphere. Right: The corresponding prototype surface.

Fig 11: A translation of a portion of the surface defined by a bump function with a non-zero value of c. The surface is grabbed in the center of the ring.

Note also that the definition of $T'(x)$ can be generalized to any weight function. The bump function is treated specially because its parameters correspond to those aspects of the manipulation over which we desire the most control, namely the size of the region of the surface effected.

2.3 Implementation and Control Interface

The system described above was implemented based on parameterized two-dimensional surfaces in three dimensions. The vertices on the surface are defined by three functions (called components of the vertex) of two parameters (called parameters of the vertex). The connectedness of the surface is defined by the ordering of the parameters, and in fact the coordinates of a vertex are stored in arrays indexed by the parameters of that vertex. Note that this implementation trivially extends to a parameterized surface of any dimension in a space of any dimension, so the method described here can be used to manipulate, for example, volumetric objects.

In an implementation of the theoretical structure described in section 2 above, three logically independent tasks have to be performed, given a transformation to be applied at that point:

- Define the shape of the bump weighting function on the surface as a function of distance from some point or set
- Compute of the weighting function for all vertices on the surface
- Compute the weighted transformation of each vertex in the surface.

The interface for the control of the shape of the basic bump function is given by drawing a graph of the current bump function in virtual space (fig 12). By using various gestures, the user can change the values of the corresponding parameters by moving the hand. The position of the hand during these gestures continuously determines the values of the bump parameters. During this operation, as the bump is redefined the graph is redrawn, giving the user feedback on the new shape of the bump.

When the manipulated surface is initially defined, the vertex components of the parameterized surface prior to any deformation are stored in the array of 3D vectors init[s][t]. These vertices form the prototype surface. Once the user has indicated a vertex p, which is considered the grabbed vertex, the parameters of p, (ps, pt) are determined. Then for all values of the parameters (s, t) for which there are vertices, the value of the bump function are computed as bump(r, c, r0, r1), where r is the euclidean three-dimensional distance from init[ps][pt] to init[s][t]. In this way, if two vertices coincide on the initial surface, they will be have the same bump value. Thus they will transformed identically and so always coincide. The value of bump(r, c, r0, r1) is stored in an array bval[s][t] (with s and t appropriately converted to indices), which was denoted as bval[v] in section 2.2, and contains the values of the bump function for the vertex with parameters (s, t).

The user must be given some indication of the strength of the bump on the surface. The possibility of exotic bump shapes (such as rings and radial modulations) requires that the user have good feedback as to exactly where on the surface the bump function will enable vertices to move. In the current implementation, this problem was solved by coloring the vertices on the surface according to the value of the bump. The color of each vertex is defined to be an interpolation between two colors, using the value of the bump function at that vertex as the interpolation parameter (fig 12).

In addition to the control over the shape of the bump function described above, an ability to enrich the shape of the bump was implemented by replacing the r computed in the above paragraphs by $r/f(\theta)$, where $f(\theta)$ is some (preferably periodic) function, and θ is an angle parameter defined as $\arctan(ds/dt)$ where $ds = ps - t$, $dt = pt - t$. In this way irregular, radial variations in the bump can be easily implemented. In the author's implementation, $f(\theta)$ is a simple ellipse with variable eccentricity and phase (fig. 9). The interface to the eccentricity and phase is also through gestures and hand position.

In the current implementation, the bump parameters may also be controlled via the mouse in a conventional screen environment.

3: Implementation and Evaluation of the Paradigms

3.1 The Grabbing Paradigm

In the grabbing paradigm, the user indicates a point on the surface as the grabbed point. Each vertex of the surface is then assigned a weight determined by the distance of that vertex from the grabbed point and the current values of the bump parameters. The distance in this case uses the distance between two points on the 'prototype surface' as described in section 2.2. The grabbed point is defined as the point on the surface closest to the current hand position when the grab was initiated. The user indicates that the surface is to be grabbed at the grab point by performing and holding a gesture. As long a this gesture is held, the position and orientation of the hand determines the transformation T, which which is used as described in section 2.2 to determine a transformation at each point of the surface. In a similar way, another gesture uses the position of the hand to determine T as a scale transformation. In this implementation, the hand is not actually touching the surface. The transformation T is based on the position and orientation offset of the user's hand position and orientation at the time the grab gesture is first executed.

This paradigm is least intuitive, but it allows the most precise control over the shape of the surface during manipulation. It has the advantage that the manipulations of the surface are direct, as if one were grabbing the surface itself. The disparity caused by not directly grabbing the surface with the hand is quickly overcome by learning. Further, any manipulation can be accomplished with comparatively few concepts, all of which involve the specification of the weight function on the surface. All the surfaces in fig. 1 were drawn with the grabbing paradigm. This paradigm also fits well into a conventional mouse-screen environment.

3.2 The Pushing Paradigm

In the pushing paradigm, a push tool is defined as a point in three-dimensional space. The location of this point is that of the user's hand as returned by the Polhemus tracker. In this paradigm the distance used in the computation of the weighting function for a vertex is the three-dimensional distance from that vertex to the push tool. The prototype surface is not used. To use the push tool, the user makes and holds a gesture. The push tool is then used to push points of the surface, which move only when they are close enough to the push tool to have a non-zero value of the weight function. Thus although the push tool is defined as a point, it has a size which corresponds to the parameters of the bump function. This is graphically displayed by nested semi-transparent spheres (fig. 2). One of these spheres, corresponding to the bump parameter r0, is rendered as a skeleton. The other, inner sphere corresponds to the bump parameter r1 and is rendered as a semi-transparent solid sphere. These spheres are semi-transparent so that they do not block the user's view of the surface.

Care must be taken so that the surface appears to be 'pushed' by the push tool, rather than becoming 'stuck' to it. This is a problem because while the push tool acts on the surface, those points closer to the center of the push tool than the bump parameter r1 would move rigidly with the push tool until the push gesture is released. This is not what pushing appears to do. The illusion of pushing is accomplished by using the distance from the push tool to the surface points when the push gesture is first initiated to define the weighting function on the surface. Thus when the push tool is backed off, the points of the surface return to their pre-push position.

The pushing paradigm has great intuitive appeal. It is very reminiscent of the manipulation of real-world tools with our hands. Generalizations of this paradigm with static push tools of other shapes could produce quite a powerful surface shaping tool set. Pushing, however, does not gives as much control over the manipulation of the surface as does the grabbing paradigm.

3.3 The kneading Paradigm

The kneading paradigm is much like the push tool paradigm, except that the push tool is now a skeletal model of the user's hand. This model somewhat faithfully matches the position and orientation of the user's palm and fingers, including the angle of bend of their joints. As the hand tool is more than a single point, there are several possibilities for a definition of distance.

In an attempt to shape the surface using the shape of the fingers, the distance used to define the weight function at a point on the surface is taken to be the distance to the nearest point on the hand skeleton. The bump function is still used, so the effective shape of the hand tool will be a collection of cylinders around each segment of the hand skeleton. In this case, the values of r0 and r1 must not be too large, or the shape of the hand would be lost. Once the weight function on the surface is computed, the surface is pushed exactly as described for the push tool in section 3.2 above.

There are several problems with the kneading paradigm using the entire hand skeleton. The most serious problem is that the time to compute the distance from every point on the surface to the segments of the skeleton is prohibitively long. On our system, described in section 1.3, this computation drops the frame rate to unacceptably low levels, so that the system is too sluggish for good interaction. There are other problems, but they are generic to the kneading paradigm and will be discussed at the end of this section.

Another approach that has been implemented is to use the distance from the points on the surface to the tips of the four fingers. This in effect is using four push tools described in section 3.2, where the location of each push tool is at the location of the user's finger tips. This is further refined by considering only those fingertips that are extended. This runs at a usable frame rate, and is a rather enjoyable way to manipulate the surfaced.

There are several problems with the kneading paradigm which do not occur in the other two paradigms.

- As the kneading paradigm is based on the position of the fingers, while shaping with the hand one cannot use gestures for control. Thus one is required to have other inputs, or to toggle gestures, to enable and disable the pushing operation.
- While the versatility of using one or more fingers is useful for quickly varying the size of the shaped region, the standard VPL dataglove delivers only the bends of two joints per finger. In particular the separation of the fingers is not available. Thus spreading one's fingers to change the shape of the shaped region is not an option. This would be a highly desirable feature.
- The most serious problem generic to the kneading paradigm is that the manipulation of the surface faithfully tracks the shape of the hand. Thus while pushing the surface with the hand, one must be unusually attentive to one's hand position. This is rather fatiguing, even for those user's highly experienced with glove operations. As the hand tires, control over the manipulation is lost. Tactile feedback in the glove would be expected to help this problem.
- Noise in the hand tracker becomes very significant and annoying, especially noise in the orientation data.

Thus from the point of view of precision and control, the kneading paradigm leaves a lot to be desired. The kneading paradigm, on the other hand, is the closest to the goal of shaping surfaces in a virtual environment "with one's bare hands". This is a very attractive goal, and further development of the kneading paradigm is warranted.

4: Conclusions

The paradigms discussed in this paper all, to some extent, succeed in allowing a user to 'directly' manipulate a virtual surface. From the point of view of control, the grabbing paradigm is, at this stage of development, the preferable paradigm. From the point of view of intuitive power, the kneading paradigm is very attractive. The kneading paradigm suffers, however, from severe problems of control due to fatigue. The pushing paradigm is somewhat of a compromise, but it does not have the control of the grabbing paradigm.

There are times, however, when each of these paradigms is appropriate. A system that allows the quick selection of paradigms has proven to be very nice. Further development of all three paradigms is, in our opinion, clearly warranted.

There are several areas where the interface to the specification of the weight function described in this paper can be extended. The system described above is built on the concept of a mask on the surface. The bump function was used to define this mask because the bump is relatively easy to specify, as it is defined in terms of three parameters. As mentioned previously, the concept of the mask on the surface can be generalized beyond the bump function. This would allow the specification of arbitrary neighborhoods to be manipulated. Generalizing the bump function in this way introduces no new conceptual structure, but would involve considerable development of the user interface.

Acknowledgements

The Author would like to thank his colleagues in the Advanced Research Office of the NAS division at NASA Ames for their many helpful comments and discussions, particularly Sam Uselton, Creon Levit, Mike Yamasaki, Al Globus, Kyra Lowther, and Horst Simon. Thanks also to Jeff Hultquist for help with Postscript and John Krystynak for enthusiasm. Special appreciation goes to Profs. Andrew Hanson, Louis Kauffman, and Louis Crane for being sources of inspiration and for their very warm encouragement.

References

1 Barr, A., Global and Local Deformations of Solid Primitives, *Computer Graphics,* Vol 17, #3, July 1984, pp 21-30

2 Fisher, S. et. al., Virtual Environment Interface Workstations, *Proceedings of the Human Factors Society 32nd Annual Meeting*, Anaheim, Ca. 1988

3 Fisher, S., Virtual Environments, Personal Simulation and Telepresence, *Implementing and Interacting with Real Time Microworlds*, Course Notes, Vol. 29, SIGGRAPH 1989

4 Forsey, D., and Bartels, R., Hierarchical B-Spline Refinement, *Computer Graphics,* Vol 22, #4, August 1988, pp 205-212

5 Guillemin, V. and Pollack, A., *Differential Topology,* Prentice-Hall, Englewood Cliffs, NJ., 1974

6 MacDowall, I., Bolas, M., Pieper, S., Fisher, S. and Humphries, J., Implementation and Integration of a Counterbalanced CRT-based Stereoscopic Display for Interactive Viewpoint Control in Virtual Environment Applications, *Proceedings of the 1990 SPIE Conference on Stereoscopic Displays and Applications*, Santa Clara, Ca. 1990

7 Parent, R., A System for Sculpting 3-D Data, *Computer Graphics,* Vol 11, #2, July 1977, pp 138-147

8 Sederberg, T. and Parry, S., Free-Form Deformation of Solid Geometric Models, *Computer Graphics,* Vol 20, #4, July 1986, pp 151-160

9 Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K., Elastically Deformable Models, *Computer Graphics,* Vol 21, #4, July 1987, pp 205-214

Fig 1: Examples of surfaces sculpted using the method described in this paper. Clockwise from upper left: two views of a cup; a boar's head; a cat mask.

Fig. 2: Illustrations of the paradigms described in this paper. Upper left: The grabbing paradigm. Note the shading of the surface, indicating the area currently grabbed. Lower left: The pushing paradigm, showing the push tool. The push tool is rendered opaque for purposes of illustration. In actual use it is semi-transparent. Lower right: The push tool in action, pushing indentations into the surface. Upper right: The kneading paradigm, here showing the use of the hand tool pushing on the surface with the fingertips.

Fig. 4: The hardware interface to the virtual environment used in this paper. The large black device is the Fake Space Boom. It supports two small CRTs on a counterweighted yoke attached through six joints to a base. These CRTs are used to produce a wide-field, stereoscopic display. Sensors at the joints allow head-tracking so that the graphics scene may be continually rendered from the user's point of view. The VPL Dataglove on the user's hand allows control over objects in the environment.

Fig 12: The user's surface manipulation environment. In each figure, the line drawing under the surface is a graph of the bump function as a function of distance. The colored dots are the handles that the user moves to control parameters of the bump. The circle in the upper left allows the specification of radial distortions in the bump. The white area on the surface is the region where the bump function is non-zero. Upper left: The plane prior to manipulation. Note the cursor surrounded by white area. Upper Right: The surface after translation upwards weighted by the bump function indicated by the white area in the upper left figure. Lower left: a radially distorted bump weighting a translation upwards. Note from the graph below the figure that the bump is now very broad. The circle in the upper left of the figure has become an ellipse. The white area is now elliptical on the surface, so an elliptical hump is lifted by the transformation. Lower Right: An example of many translations and rotations weighted by many differently shaped bump functions, with the bump editing tools turned off.