Volume Warping *

Thomas J. True
Digital Equipment Corporation
Maynard, MA 01754

John F. Hughes
Brown University
Providence, RI 02912

Abstract

We present volume warping, a technique for deforming sampled volumetric data using B-splines that is related to image warping and to the free-form deformations of Sederberg/Parry and Coquillart. We accelerate the process to near-real-time speed, and explain the compromises that are made to effect such speeds. This technique expands the repertoire of volumetric modeling techniques, and can be applied to any form of volumetric data.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; Curve, Surface, Solid, and Object Representations;

1 Introduction

If f is a smooth real-valued function on 3-space, an *isosurface* of f, is the set of points $S_a = \{x, y, z : f(x, y, z) = a\}$. If the function f is chosen in order to create an isosurface with a particular shape, we call f a *volumetric model.* Thus the function $f(x, y, z) = x^2 + y^2 + z^2 - 1$ is a volumetric model of a unit sphere; the level set S_0 is the sphere.

Various authors have described volumetric models. Blinn [2] made "blobby objects" by computing an isosurface of the potential arising from charged points; Wyvill et al. [4] extended this idea to create "soft objects"; Bloomenthal and Shoemake [3] described convolution surfaces that arise from extending soft objects.[1] In a subclass of volumetric models that we call *sampled volumetric models*, the function f is known only at an array of points in 3-space.[2] Much medical imaging data falls into this category, [3] and there are other examples as well: Galyean and Hughes [10] describe a method of *sculpting* volumetric models.

With sampled volumetric data, one must infer the values at the non-lattice points. If the data comes from sampling of a band-limited function, one can reconstruct by convolution with a *sinc* filter, but this is impractical for large data sets. Instead, one often computes the value at non-lattice points by some simple interpolation method.

Sampled volumetric *modeling*, i.e., altering sampled volumetric data in prescribed ways, is comparatively new and has not received as much attention as polyhedral modeling. There is a wide spectrum of polyhedral modeling techniques [1] [14] and image warping techniques [6] [15] [5]. Here we describe a technique for deforming sampled volumetric models called *volume warping*, which is closely related to image warping. We view this as a step towards unifying volumetric and surface-based models; we look forward to seeing the two forms of modeling become tightly interwoven. This work is also related to scientific visualization, as volumetric data may be best viewed after applying some distortion (to cancel out distortions in the original sampling, crudely compensate for registration errors, expand a region of interest while contracting the rest of the data, etc.)

Volume warping, as described here, is feasible only for relatively small data sets, at least at the interactive speeds necessary to make it practical. Most of our illustrations show models of size $30 \times 30 \times 30$.

To give an example, suppose you want to lower the neck of the teapot in Figure 6, created using the SCULPT program [10]. To do this you would erase the neck and re-sculpt it lower down (just as in a pencil-and-paper drawing you would erase and re-

*This work was supported in part by the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization, and by IBM, NCR, Sun Microsystems, Hewlett Packard and Digital Equipment Corporation.

[1] Perlin [13] describes a volumetric modeling process in which the isosurface is *not* the desired result: the models are intended for volume rendering.

[2] We call such arrays of density values *voxmaps*, in analogy with pixmaps.

[3] Medical imaging data is not collected with the intent of generating a particular isosurface shape; however, once it is collected, one often extracts surfaces from the data.

sketch). With volume warping, however, you could alter the space in which the model resides so as to move the portion containing the neck (see Figure 9). The idea underlying this deformation technique is best understood by analogy with image warping: if you draw an image on a rubber sheet and then deform the sheet, the image is deformed. Our model actually more closely resembles the two-step warping available with Silly Putty: press Silly Putty on newsprint so that some of the newspaper ink sticks to the Silly Putty. Then stretch the Silly Putty and press it down onto blank paper: the ink transferred to this new sheet prints a deformed version of the original picture. In our method, the (3D) analogue of the Silly Putty surface is the *domain voxmap*, that of the original picture is the *source voxmap*, and that of the final picture is the *target voxmap*. The domain-source correspondence is defined by one B-spline map used to "capture" the original data and the domain-target correspondence is defined by another used to "write" the data. These two maps are typically not very different; each time we adjust the writing map, we alter only a small part of the target voxmap, and hence manage to get some computational efficiency from the local-control properties of B-splines.

This basic idea depends on a "continuous" world, but this paper describes an implementation of it in a discrete world, using sampled volumetric models. This leads to certain complications addressed below.

Before we give the details of the method, we want to mention several points:

- This paper presents only a method for implementing volume warping, and not a user interface for it. See Section 7, however, for a simplified approach to describing a warp.

- The technique is not particularly fast, because volume data sets tend to be huge. We give some timing data in Section 8.

- The technique has an advantage over direct deformations of polygonal models, in which boundaries between large polygons tend to crease: since our isosurfaces comprise polygons of approximately constant size, creases do not tend to appear.

- Volume warping acts on the space in which a model lies rather than on the things used to define the model. Volume warping is not appropriate for spline patch models, for instance. By contrast, when one has implicitly volumetric data, volume warping can be quite powerful.

2 Mathematical Description

We call the function f defining a volumetric model the *density* because it describes where material is: a density of one means the material is there, a density of zero means the space is empty, and the isosurface where the density is 1/2 is the boundary between the inside and the outside of the material. In this section, we consider the density function as defined on the unit cube; its values outside the unit cube are everywhere zero. Thus the function

$$d : [0,1] \times [0,1] \times [0,1] \to [0,1] : (x,y,z) \mapsto d(x,y,z)$$

denotes the presence or absence of material.

If we have a function from the unit cube to itself,[4]

$$B : (x,y,z) \mapsto (B^x(x,y,z), B^y(x,y,z), B^z(x,y,z))$$

then we can build a different density function on the cube as follows: the new density is computed by first applying the map B and then evaluating the original density function at the resulting point. The new density is therefore

$$d'(x,y,z) = d(B^x(x,y,z), B^y(x,y,z), B^z(x,y,z)),$$

which we call the *pullback* of the original density function by the map B (see Figure 1).

There is another way to combine a map B and a density function d: we can define a new density function on the codomain of B as a *pushforward* of a density on the domain, as shown in Figure 1. Here, however, the description is not quite so simple. If B is injective, we can look for the point (x,y,z) that is sent by the map B to each codomain point W. The value of the new density function at W is just the value of the original function at the corresponding domain point (and is zero if W is not in the image of the map B). This technique is fine if the map B is injective. If not, however, some sort of average over all the points in the preimage is needed. This is adequately handled during filtering (see Section 6).

The volume-warping process, as shown in Figure 2, uses both of these techniques. We start with a volumetric model, i.e., a density function d on a unit cube called the *source* S. We create a particular B-spline map B from another unit cube, the *domain* D, to S, and pull back the density function from S to D to get a new density function d'. We then define another B-spline map $\bar{B}$ from the domain to a third unit cube, the *target* T. We push forward the density d' from D to get a new density function d'' defined on T.

[4]If P is a point in 3-space, P^x, P^y, and P^z denote the x, y, and z coordinates of P.

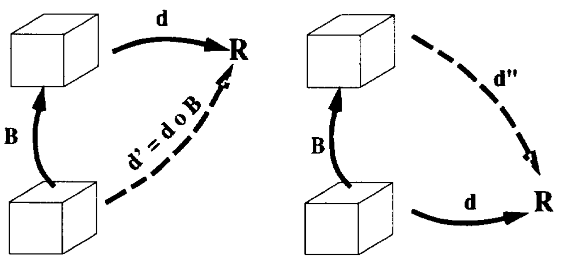

Figure 1: Pullback and pushforward of a density function.

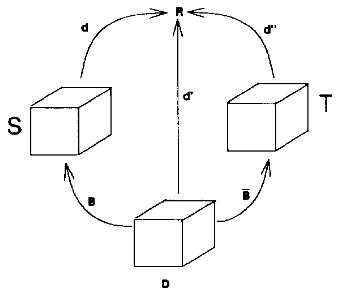

Figure 2: The entire mapping process.

The density on the target T is the *warped version* of the original density function on S. If the B-spline maps B and $\bar{B}$ differ only slightly, then the isosurfaces of the source and target density functions are quite similar and the term "warped" is justified.

3 Warping Sampled Volumetric Data

The previous section described functions defined on an entire cube. With sampled volumetric data, however, we have only samples of the functions. We therefore work with approximations of the functions represented by the samples: rather than proper reconstruction with the *sinc* function, we use box filters repeatedly. The aliasing thus introduced has not been a substantial problem.

The constants used in the program are

- ISIZE: the model to be warped is an ISIZE × ISIZE × ISIZE array of values between 0 and 255; by scaling, these represent density values between 0 and 1, and the object described by the model is an isosurface for value = 127.
- SSIZE: the source voxmap is an SSIZE × SSIZE × SSIZE array of unsigned bytes. The original model is read into the center of this array (SSIZE must be greater than ISIZE) and the remainder of the source voxmap is padded with zeros.
- DSIZE: The domain voxmap is DSIZE × DSIZE × DSIZE; we choose DSIZE to be 2 ∗ ISIZE, for reasons explained in Section 7.
- CSIZE: Our B-spline maps are determined by an array of control points of size CSIZE × CSIZE × CSIZE. (We generally use CSIZE = 9, but other values may be appropriate for finer control. Figure 9 was even made with a noncubical control point array.) A B-spline $B(s,t,u)$ based on such an array of control points is defined for values of s, t, and u between 0 and CSIZE − 3. The ratio of DSIZE to CSIZE − 3, which we call the *granularity*, must be an integer for the rapid spline evaluations of Section 4 to work.

Our two B-spline maps are defined on the domain with the help of the B-spline basis functions and the CSIZE × CSIZE × CSIZE array of control points P_{ijk}. We first establish a correspondence between points in the domain and points in the cube $[0, \text{CSIZE} - 3] \times [0, \text{CSIZE} - 3] \times [0, \text{CSIZE} - 3]$. The correspondence is simple: the entry in the domain array whose index is (i,j,k) corresponds to the point in 3-space whose coordinates are

$$(s,t,u) = (i/G, j/G, k/G) \tag{1}$$

Recall that the B-spline basis functions are

$$\begin{aligned} B_0(t) &= (1-t)^3/6, & B_1(t) &= (3t^3 - 6t^2 + 4)/6 \\ B_2(t) &= (-3t^3 + 3t^2 + 3t + 1)/6, & B_3(t) &= t^3/6 \end{aligned}$$

These basis functions are used to blend the control points, taken in groups of 64, as follows. For integers a, b, and c between 0 and $G-1$, we define the B-spline function on the subcube $s \in [a, a+1]$, $t \in [b, b+1]$, $u \in [c, c+1]$ of the domain (which we call a *region*) by using the control points P_{ijk}, for $a \le i < a+3$, $b \le j < b+3$, and $c \le k < c+3$. The general form for the function on this region is

$$\begin{aligned} & B^x(s,t,u) \qquad (2) \\ &= \sum_{i,j,k=0}^{3} B_i(s-a)B_j(t-b)B_k(u-c)P^x_{a+i,b+j,c+k} \end{aligned}$$

with similar terms for B^y and B^z. Note that there are $G \times G \times G$ regions and that altering one of the control points alters the functions values in only a subset of the regions, so that we have local shape control.

To specify the B-spline maps, we need to give the locations of the control points. Suffice it for now to say that, to define the map used to "capture" the original data into the domain voxmap from the source voxmap, we place the points so that the image of the entire domain voxmap under the B-spline is exactly the central ISIZE × ISIZE × ISIZE subcube of the source voxmap.

The capture map is used to pull back the original density function from the source voxmap to the domain voxmap. This step is done only once. After this, the array in which the source data was stored ceases to act as the source and becomes the array in which the target data is stored. (This is the reason the original data was put in the center of a larger array: we want to let the model be stretched along some direction.)

We now use the same control points to define a B-spline map from the domain to the target voxmap. We can alter the map by moving its defining control points. Then the density is pushed forward from the domain to the target to produce the warped sampled volumetric data set. Precise details of pushing forward the discrete data values are given in Section 6. Here, we describe how to compute the values of a B-spline map at each point in the domain, since these maps are used both to pull back the original data from the source and to push it forward into the target.

Note: for each point in the domain of the altered 3D B-spline (the "writing map"), we evaluate the B-spline function to get another 3D point (Equation (2) shows how to compute $B(s,t,u)$). We record the resulting locations in a 3D array of 3D points called the *deformation mapping array.*

4 Rapid Evaluation of B-splines

We need to compute the B-spline maps rapidly, since they are applied to each voxel in the domain. The standard way to do this with B-splines would be incremental computation [9], but there is a faster method available in our situation.

Consider Equations (1) and (2). Since the domain size is an integer multiple of the control-mesh size, the fractional parts of s, t, and u used in the B-spline evaluation in Equation (2) are the same for points in any region (they in fact have values $0/G, ..., (G-1)/G$.). Thus if we denote by b_{ij} the coefficient of t^j in the ith basis function of Equation (2), we can rewrite the formula for $B^x(s,t,u)$ as

$$\sum_{i,j,k,p,q,r=0}^{3} b_{ip}(s-a)^p b_{jq}(t-b)^q b_{kr}(u-c)^r P^x_{a+i,b+j,c+k}$$

and we see that only powers of the fractions above are involved in the computation.

We therefore precompute the values

$$c[\bar{e}][\bar{f}][\bar{g}][i][j][k] = \sum_{p,q,r=0}^{3} b_{ip}(\bar{e}/G)^p b_{jq}(\bar{f}/G)^q b_{kr}(\bar{g}/G)^r \tag{3}$$

for $0 \le \bar{e}, \bar{f}, \bar{g} < G$ and $0 \le i,j,k \le 3$. Computing the value of the spline function at a domain location indexed by (e,f,g) now requires only that we determine the values of a, b, and c (the integer parts of e/G, f/G, and g/G, respectively) and the values of $\bar{e}$, $\bar{f}$, and $\bar{g}$ (which are the residues of e, f, and g mod G). We then calculate the sum in (2) by the simple 64-term sum

$$B^x(s,t,u) = \sum_{i,j,k=0}^{3} c[\bar{e}][\bar{f}][\bar{g}][i][j][k] P^x_{a+i,b+j,c+k}.$$

5 Filling in the Details

We now can complete the overall structure of the program.

Initialization. We begin by reading the original density function into the center of the source voxmap; we set up the control points as described in Section 7, and thus define a map from the domain voxmap to the source voxmap. We use this map to pull back the density function from the source voxmap to the domain voxmap: for each index (e,f,g) in the domain voxmap, we evaluate the B-spline and round to get a triple of integers between 0 and SSIZE, which we use to index into the source voxmap; the density value found there is placed in domain[e][f][g].

This approximates the pullback process described in Section 2. The rounding appears not to introduce aliasing errors, probably because the domain has about twice the resolution of the source.

We also set up a three-dimensional array of region Booleans, which will be used to indicate which B-splines to evaluate. Next we calculate the s, t, and u values at each point of the domain voxmap and use them to determine the four B-spline basis function values on each of s, t and u; these are saved for later use. Then we compute the $G \times G \times G \times$ CSIZE × CSIZE × CSIZE array of coefficients described in (3) for use in determining the values of the spline map. Finally, we rename the source array to be the target array.

The main loop. The main loop of the program is basically just

```
repeat {
   get user input to determine
      which control points to move
   determine which regions are affected
   for each such region{
      pushforward density function using
         unmoved control points
      subtract pushed forward values
      pushforward the density function using
         moved control points
      add pushed forward values
   }
}
```

If control point P_{ijk} is to be moved, the regions labeled by indices (a, b, c), where $i - 3 \leq a \leq i$ and similarly for j and k, are influenced, so their Boolean flags are set to TRUE. Now for each region whose Boolean flag is set, we must first erase from the target voxmap the contribution of the pushforward of the density on this region. We could do this as described in the loop, by pushing forward the values and subtracting them from the values stored in the voxels of the target voxmap. Instead, we have found that for modest deformations, it suffices simply to write zeros into the target voxmap at the pushed-forward locations. At this point, we actually change the control points by moving them to their new locations, and hence define a new B-spline mapping from the domain to the target voxmap.

We must now push forward the density function from the domain to the target voxmap. Simply pushing forward voxel centers and round produces severe aliasing. We must therefore do some filtering, so we compute the (floating-point) coordinates of the point to which each domain voxel-center is sent and place the results in a DSIZE x DSIZE x DSIZE deformation mapping array. Computing this array is expensive, but the rapid B-spline computation of Section 4 reduces the time substantially.

6 Forward Mapping with Antialiasing

Rather than doing computationally expensive *sinc* reconstruction of the data to be pushed forward, we make some of the usual approximations: we use box filtering in both the domain and codomain. We reconstruct the density function on the domain with a box filter, i.e., we take the value at each voxel center and assume that the value of the original density function is constant throughout the unit cube about the voxel center (henceforth called the voxel). We then look at the image of this cube under the B-spline mapping and, for each target voxel, compute its approximate overlap with the transformed box and record it as a fraction of the target voxel's volume that we call the *weight*. We then multiply the value from the domain voxel by the weight to get the *weighted value*. We tally these weights and weighted values for each target voxel. In pseudocode:

```
for each domain voxel V{
   compute the transformed voxel B(V)
   for each target voxel Q that overlaps B(V){
      compute weight =  volume of the overlap
      compute weighted value = weight * d(V)
      add weight to Q.weight
      add weighted value to Q.weighted_value
   }
}

for each target voxel Q {
   if weight < 1 then d''(Q) = Q.weighted_value
   else d''(Q) = Q.weighted_value / Q.weight
}
```

Unfortunately, computing the exact overlap of the target voxel with the transformed domain voxel is prohibitively expensive. We use instead an approximation of the transformed domain voxel: we compute the approximate transformed locations of the centers of the domain voxel's six faces and take the bounding cube of these six points as our proxy for the actual transformed domain voxel. Figure 3 shows the analogous effect in 2D: the bounding boxes of adjacent pixels may overlap. Thus a single target pixel may find itself with an accumulated weight of more than 100%; this is the reason for the division in the last line of the pseudocode above.

On the other hand, a pixel may not be completely covered by the transformed domain pixels. If not, then we can say that the portion left uncovered contributes zero to the density function: adding this in brings the total weight to 1 and does not change the weighted values at all. This explains the second-to-last line of the pseudocode.

The approximate locations of the six transformed points are computed with the deformation mapping array. Recall that we computed the target locations of each mapped domain voxel center in this array. To find the center of the face lying between the voxel center at (e, f, g) and that at $(e + 1, f, g)$, we simply average the points stored in the deformation mapping

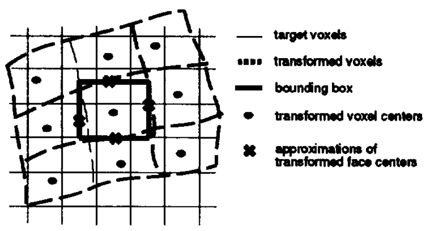
target voxels
transformed voxels
bounding box
transformed voxel centers
approximations of transformed face centers

Figure 3: The bounding box of 6 points in a transformed voxel is used to approximate the transformed voxel.

Figure 4: Comparison between "map and round off" and filtered pushforwards.

array at these two indices.[5] Figure 4 shows the results of volume warping with and without the antialiasing provided by this approximate box filtering.

This completes the warping of the volume data; to make an actual picture, we compute and render the isosurface of the resulting density field. All images here were generated by the AVS$^{\text{TM}}$ isosurface tiler.

7 Placement of Control Points

The control-point placement used to capture the source data into the domain is as follows: we take the central ISIZE^3 subarray of the source data and divide it into CSIZE $-$ 4 pieces in each direction. (Figure 5 shows the analogous 2D subdivision.)

We place the control points P_{ijk} using this subdivision. In each direction, the first and last control

[5]We could have computed the transformed locations of these face centers instead of those of the voxel centers in the first place, but chose not to because we use the locations of the transformed voxel centers to do the very inexpensive pushforward described at the end of Section 5 in almost real time.

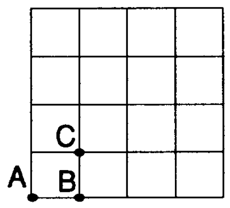

Figure 5: Placement of the control points for the 2D capture map, for a 9×9 control mesh for a few sample points; note that the source region is divided into a 4×4 grid. Point A is $P_{00} = P_{01} = P_{02} = P_{10} = P_{20}$, B is $P_{30} = P_{31} = P_{32}$, C is P_{33}.

points are tripled and the rest are evenly spaced (see Figure 5). Thus, for example, the points $P_{i,0,0}$ for $i = 0, 1, 2$ all go to the front lower left corner of the central ISIZE^3 subcube of the source voxmap.

This placement of control points makes the control mesh (and the image of the B-spline) enclose exactly the same region as that occupied by the original data. Moving a point of the control mesh influences points near that control point, which gives an accurate and intuitive way to define warps.

Alternatively, we can use an inverse-problem approach [11] [8]. We describe a warp by saying "I want to move *this* point *there*." More mathematically, we have the capture B-spline map B_1, pick a point in its image, say $B_1(s_0, t_0, u_0) = Q_1$, and then try to alter B_1 to design a new B-spline B_2 such that $B_2(s_0, t_0, u_0) = Q_2$. We are asking "Where should we move the control points of B_1 in order to make the map B_2 satisfy $B_2(s_0, t_0, u_0) = Q_2$?" A least-squared motion solution to this problem is easy to determine.

First we take the point Q_1 and determine (solving some cubics) the point (s_0, t_0, u_0) in the domain such that $B_1(s_0, t_0, u_0) = Q_1$.. In matrix form, we then have

$$Q_1^x = BP^x$$

where B is the array of coefficients of the P_{ijk} and P^x is a vector of the x-coordinates of the P_{ijk} in Equation 2. (We write similar equations for y and z.) We wish to change the control points by an amount ΔP so that

$$Q_2^x - Q_1^x = B\Delta P^x$$

and, as mentioned above, we wish to find the smallest such change. We compute ΔP^x using the pseudoinverse of B [12] to get

$$\Delta P^x = B^+(Q_2^x - Q_1^x)$$

which solves the problem.

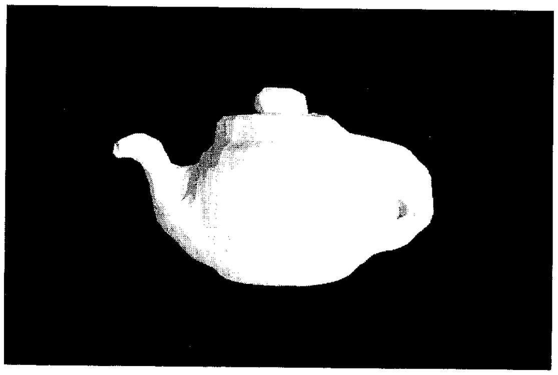

Figure 6: Teapot, original sculpture.

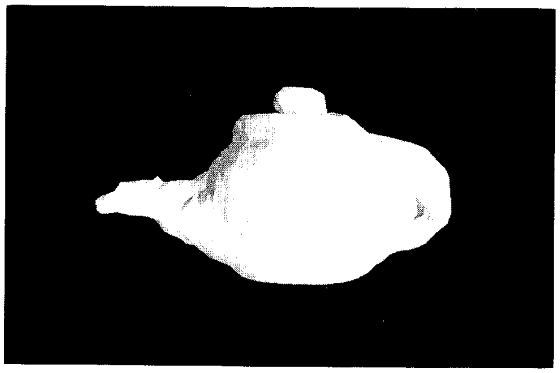

Figure 9: Teapot, warped to adjust spout position.

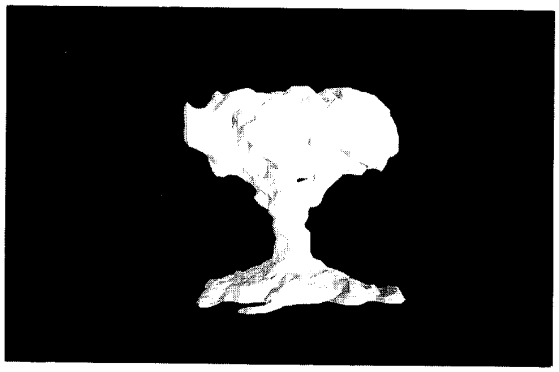

Figure 7: Tree, original sculpture.

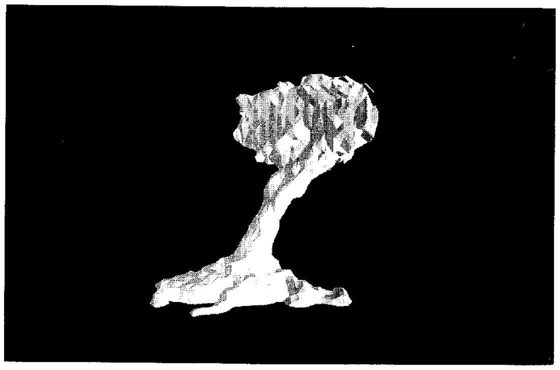

Figure 10: Tree, warped as if caught in a strong gust of wind.

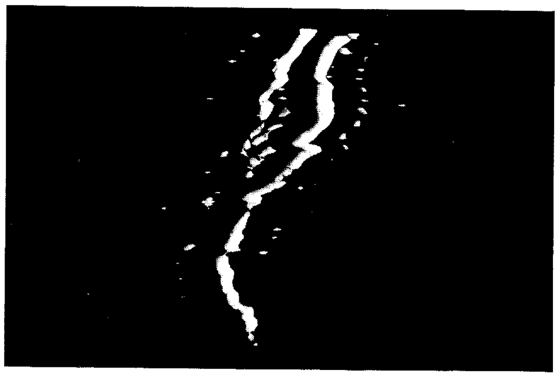

Figure 8: Differentially scaled angiogram. Courtesy of E.K. Yucel, M.D., Massachusetts General Hospital.

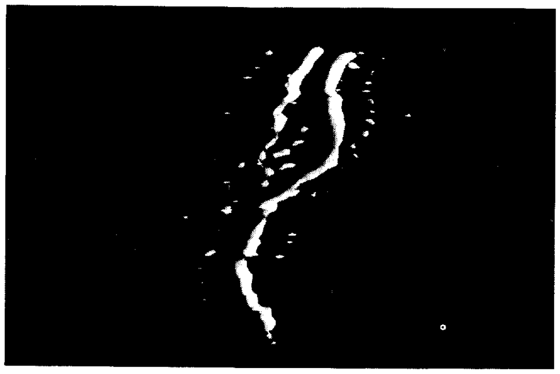

Figure 11: Angiogram warped to remove kink in artery.

(See color plates, p. CP-33.)

8 Performance

Although volumetric data sets are usually large and computationally intensive to work with, we have obtained respectable performance using the rapid evaluation of B-splines described in Section 4 and the antialiasing technique described in Section 6. Using the constant values ISIZE = 30, SSIZE = 38, DSIZE = 60 and CSIZE = 9 and running on an IBM RISC6000 workstation, after an initial startup time of approximately 16 seconds, a typical warp takes about 6 seconds (see Figures 6, 7, 9 and 10). Unlike warping a polygonal surface model, warping a volumetric model is independent of model complexity because sampled volumetric data is regular. Since any B-spline warp influences at most 64 regions, as determined by the control points, the timing for any warp is dependent only upon the granularity of the data points within a region.

9 Future Work

Our future plans for work on this technique include an interactive user interface to allow the easy specification of these deformations. One possible interface paradigm for this type of volume modeling is the direct-manipulation interface described in [11]; another is the 3D widget set described in [7]. We are also examining further ways to speed up this modeling technique, since such speedup is crucial if volume deformations are to become a useful interactive modeling technique.

Volume warping is an ideal application for fine-grained parallelism, since its performance could be improved by farming out the B-spline mapping of each region to a different processor.

We have begun to apply volume warping to MRI data (see Figures 8 and 11, in which a kinked artery has been straightened by warping). We look forward to applying it to other data as well, perhaps in removing distortions introduced during data acquisition.

10 Acknowledgments

Much of this work was done by the first author as his Master's project under the direction of the second. We appreciate the support of the Brown Computer Graphics Group, especially George Reilly, Ken Herndon, and Dan Robbins. The first author acknowledges the support of the Digital Equipment Corporation Graduate Engineering Education Program.

References

[1] A. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3):21–30, July 1984.

[2] J. F. Blinn. A generalization of algebraic surface drawing. *ACM TOG*, 1(3):235–256, 1982.

[3] J. Bloomenthal and K. Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, July 1991.

[4] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109–116, March 1990.

[5] Carolco Productions. Terminator 2, 1991.

[6] E. Catmull and A. R. Smith. 3-d transformations of images in scanline order. *Computer Graphics*, 14(3):279–285, 1980.

[7] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. *To appear in Computer Graphics*, April 1992.

[8] S. Coquillart. Extended free-form deformation: A sculpting tool for 3d geometric modeling. *Computer Graphics*, 24(4):187–196, 1990.

[9] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics : Principles and Practice.* Addison Wesley, second edition, 1990.

[10] T. Galyean and J. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics*, 25(4):267–274, July 1991.

[11] William M. Hsu. A direct manipulation interface to free-form deformations. Master's thesis, Brown University, May 1991.

[12] B. Noble and J. W. Daniel. *Applied Linear Algebra.* Prentice-Hall, 2nd edition, 1977.

[13] K. Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics*, 23(3):253–262, July 1989.

[14] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, August 1986.

[15] G. Wolberg. *Digital Image Warping.* IEEE Computer Science Press, 1990.