

BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-92-M11

“Volume Warping: A New Technique for Modeling with Volumetric Data”

by
Thomas J. True

Volume Warping: A New Technique for Modeling with Volumetric Data*

Thomas J. True

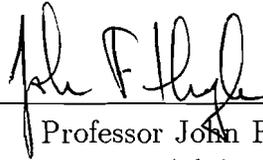
Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of Computer Science
at Brown University

May 1992

*This work was supported in part by grants from NSF, DARPA, IBM, NCR, Sun Microsystems, Hewlett-Packard and Digital Equipment Corporation.

This research project by Thomas J. True is accepted in its present form
by the Department of Computer Science at Brown University
in partial fulfillment of the requirements for the Degree of Master of Science.

A handwritten signature in black ink, appearing to read "John F. Hughes". The signature is written in a cursive style with a large initial "J".

Professor John F. Hughes
Advisor

4/12/92

Date

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 show how to speed up the process to achieve near-real-time speed, and explain the compromises that are made in the resampling of the data to effect such speeds. User interface paradigms based upon this technique are also discussed. This technique further expands the repertoire of volumetric modeling techniques.

Contents

1	Introduction	1
1.1	Volumetric Data and Volumetric Models	1
1.2	Volume Warping	2
1.3	Overview	2
2	Previous Work	3
2.1	Free-Form Deformations	3
2.2	Direct Manipulation of Deformations	4
2.3	Image Warping	6
2.4	Origins of Volumetric Data	7
2.5	Volumetric Sculpting	7
3	Motivation	8
3.1	A Step Towards Unification	8
3.2	Local Shape Control	8
4	The Algorithm	9
5	Implementation	11
5.1	Overview	11
5.2	Rapid Evaluation of B-splines	13
5.3	Implementation Details	15
5.4	Antialiasing	18
5.5	Control Point Placement	20
6	User Interface	22
6.1	Background	22
6.2	How It Works	23
6.3	Extensions	24
7	Usage Examples	24
8	Performance	26

9	Future Work	29
9.1	Performance	29
9.2	Usability	30
9.3	Applications	30
10	Conclusions	30
11	Acknowledgements	31

1 Introduction

1.1 Volumetric Data and Volumetric Models

In recent years, advances in computer hardware technology and computer graphics have led to the development of volume visualization and volumetric modeling. Volumetric modeling, the viewing and modeling of data expressed in three dimensions, which was simply a dream a decade ago, has become a reality today [14] [17]. This new reality requires new and more powerful volumetric modeling techniques.

A volumetric model can be defined simply as a function on 3-space or a subset of 3-space. If f is a real-valued function on 3-space such that the *level set of f* , can be defined as the set of points $S_a = \{x, y, z : f(x, y, z) = a\}$ and f is differentiable, then S_a will be a continuous surface called an *isosurface*. If the function f is chosen with the intent of creating an isosurface with a particular shape, f can be described as a *volumetric model*. Thus for example, the function $f(x, y, z) = x^2 + y^2 + z^2 - 1$ is a volumetric model of a unit sphere where the level set S_0 is the sphere¹.

Various authors have described ways of building models whose isosurfaces have certain desired shapes. Blinn [2] made “blobby objects” by placing “charged points” in space and then computing an isosurface of the potential arising from these charges; Wyvill et al. [4] described an extension of these blobby objects called “soft objects”; Bloomenthal and Shoemake [3] described convolution surfaces that arise from extending soft objects.

There is a subclass of volumetric models called *sampled volumetric models* in which the value of the function f is known only at an array of points in 3-space. In these models, the values at non-lattice points must be inferred from the samples. If the samples are generated by sampling a band-limited function, one can reconstruct the function by convolution with a *sinc* filter, but this is impractical for large data sets. These in-between values are often computed by a simple linear interpolation method. Analogous with two-dimensional pixmaps, these three-dimensional arrays of density values are called *voxmaps*.

Sampled volumetric data has the advantage that the regularity of the

¹Other choices are possible. If, for example $f(x, y, z) = x^2 + y^2 + z^2$ then S_1 is the sphere. Later, we will use functions varying between 0 and 1 and choose $S_{1/2}$ to represent the boundary.

data structure makes the rendering and interchange of data easier in much the same way that the regularity of image data makes image interchange easy and image processing tractable.

1.2 Volume Warping

Volume warping is a new technique for deforming sampled volumetric data. By applying a trivariate B-spline defined mapping function to a volumetric model (defined as a function on 3-space or a subset of 3-space), a new density function on 3-space is built.

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 rubber sheet, the image is deformed. In 3D, a value is assigned to each point of a rubber block; when the rubber block is deformed, the values are carried to new locations, just as deforming the rubber sheet carried along the image values on it.

The warping of volumetrically defined models has an advantage over direct deformations of polygonal models. If, for example, one tries to move one corner of a polygonally defined cube so as to deform it, the few polygons constituting the cube must be subdivided or creases will form. By contrast, applying volume warping to a sampled volumetric model entails automatic re-polygonization, so that no such problems arise.

Volume warping acts on the space in which a model lies rather than on the polygons extracted from the volumetric model. Volume warping however, is not appropriate for a finely polygonalized model or for spline patch models. In both cases, the free-form deformations described by Sederberg and Parry [18] are probably better. By contrast, when one has volumetric data or when the underlying model is not explicitly known, volume warping can be a powerful technique, just as image warping [20] can be useful in deforming photographs of real-world objects, of which no explicit model is available.

1.3 Overview

The basic idea behind volume warping depends on a “continuous” world, but this paper describes an implementation of it in a discrete world using

sampled volumetric models. This implementation in a discrete world leads to certain complications that are addressed in this paper.

The remainder of this paper is divided into ten sections. Section 2 describes the previous work that forms a foundation for volume warping. Section 3 describes motivation for this research. Section 4 describes the continuous mathematical model of warping implemented on discretely sampled volume data and Section 5 describes this discrete implementation. Section 6 describes the current interface to volume warping and describes other user interface possibilities being investigated. The final five sections present examples, performance issues, possible future work, conclusions and acknowledgements.

2 Previous Work

Sampled volumetric modeling is comparatively new, and has not yet received as much attention as that of polyhedral modeling. There is a wide spectrum of modeling techniques for polyhedral models [1] [18]. Similarly, there is a large set of techniques used to modify image data [7] [20] [6]. Since sampled volumetric data shares characteristics of both polygonal modeling and image processing, it seems only appropriate to borrow from each of these domains. Volume warping does just this.

Although modeling with volumetric data is comparatively new, much work has been done in the acquisition and display of volumetric data for medical imaging and scientific visualization [14] [17]. Previous interactive techniques have also been developed for volumetric modeling [10].

2.1 Free-Form Deformations

The free-form deformation (FFD) method of Sederberg and Parry [18] deforms an object by mapping the object from R^3 to R^3 through a map defined in terms of trivariate Bernstein polynomials. This mapping transforms each object point embedded in a local coordinate system back to a new location within the world coordinates.

This local coordinate system in which the object is embedded is a parallelepiped-shaped lattice of control points where one corner is the origin $(0, 0, 0)$ and the opposite corner is $(1, 1, 1)$. Letting Q_0 be the origin of this parallelepiped,

and S , T and U be the three orthogonal vectors from Q_0 which span the edges, any point within this local coordinate system can be written as the sum

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

The local (s, t, u) coordinates of each object point can then be determined using simple linear equations:

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

The FFD function which subsequently maps the local (s, t, u) coordinates of the object back into the world coordinates is defined by taking a weighted sum of control points. These control points are denoted by P_{ijk} ($i = 1, \dots, l, j = 1, \dots, m, k = 1, \dots, n$); P_{ijk} is the i^{th} control point in the S direction, the j^{th} control point in the T direction and the k^{th} control point in the U direction. They are in a grid within the parallelepiped and serve as the coefficients for the Bernstein polynomials.

The deformation is then specified by moving the control points from their undisplaced positions on the lattice. The deformed position of an object point Q^{ffd} with coordinates (s, t, u) can then be found by evaluating the trivariate Bernstein polynomial:

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

Because the final mapped location of each object point is determined by a weighted sum of the control points, changing the control points changes the deformation. The FFD control points affect the final shape of the object the same way that the control points of a Bézier spline affect the resulting curve.

2.2 Direct Manipulation of Deformations

Hsu [12] builds upon FFDs by developing a direct manipulation interface for controlling the deformations. Using a B-spline based free-form deformation method instead of the original Bézier approach used by Sederberg and Parry, Hsu's interface allows the user to directly manipulate the surface of an object

without having to directly manipulate the control points. This interface is the starting point on which the user interface for volume warping is based. This interface will be discussed in further detail in Section 6.

Bryson [5] also describes several paradigms for the direct manipulation of surface deformations. Unlike the spline-based free-form deformations described above, these deformations are based on spatially weighted transformations defined using a bump weighting function

$$s(x, a) = \begin{cases} 0 & x \leq 0 \\ e^{-\left(\frac{x}{a-x}\right)^{-2}} & 0 < x < a \\ 1 & x \geq a \end{cases}$$

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

$$bump(x, c, r0, r1) = \begin{cases} step(x, c - r0, c - r1) & x < c \\ step(2c - x, c - r0, c - r1) & x \geq c \end{cases}$$

where c is 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 value of the bump is equal to 0.

Vertices on a surface are then moved to deform a surface by multiplying a specified deformation transformation T by the value of the bump weighting function at each vertex. This defines a new transformation T' . At each vertex, the deformation transformation is then 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 the center of the deformation and ID is the identity transformation.

Treating the surface as simply a collection of vertices, vertices in the region where the value of the bump function is 1 will be transformed by T , vertices in the region where the value of the bump function is 0 will not be transformed at all and vertices inbetween will be transformed by T' . Unlike free-form deformations which require an underlying data structure, this approach allows the manipulation of arbitrary groups of points in space. Note, however, that large deformations will generate creases as in FFDs, because there is no automatic repolygonization.

2.3 Image Warping

Image warping can be defined as the application of geometric transformations to image data [20]. This image processing technique is commonly used for geometric correction, image synthesis and special effects.

Catmull [7] developed an early image warping algorithm based on closed-form mapping functions. Originally developed for texture mapping, Catmull found that by decomposing a 2-D mapping of a raster image into a succession of 1-D mappings, one in horizontal and the other in vertical scanline order, the mapping (an image warp) could be performed quickly.

Image warping can also be described as a mesh warping, where the input and output images are each partitioned into a mesh of patches where each patch delimits an image region over which a continuous mapping function is applied [20]. In this case, mapping between the input and output images becomes simply a matter of transforming each patch. Using this technique, moving the vertices in a mesh defines arbitrary mapping functions with local control.

Smyth [19] has developed a 2-pass mesh warping technique where a bivariate function based on two 2D arrays of control points is used to produce a continuous mapping between the points in the source image and those in the target. The first pass puts each image point into the proper column in the output image by resampling each row and mapping all points (u, v) to their correct (x, v) positions in an intermediate image I . The second pass places each point into the proper output row by resampling each column in I and mapping every point (x, v) to its final (x, y) position in the target image. The mapping functions used in this algorithm are derived from the interpolation of a spline through the x coordinates for pass 1 and the y coordinates for pass 2.

Another technique for image warping is the separable algorithm of Wolberg [20]. This algorithm, which is performed simultaneously on both the input image I and the transpose of the input image I^T , is composed of 4 stages. In the first stage each image is resampled both horizontally and vertically; the coordinate resampler then computes all spatial transformations in the second stage; in the third stage local measures of shearing, perspective distortion and bottlenecking are computed; in the fourth and final stage, a compositor produces the resulting image by selecting the best pixels from I_{xy} and I_{xy}^T .

2.4 Origins of Volumetric Data

Sampled volumetric data from which an isosurface can be extracted arises from a variety of sources. These sources can be grouped into two general categories:

observed Data acquired by the simulation of a mathematical/physical model or by scanning “real-life” objects or phenomenon to produce 2D cross sections.

modeled Data produced by “free-hand” modeling or the voxelization of geometric shapes.

A common application which produces data of the first type is medical imaging [9]. Computed tomography (CT), positron emission tomography (PET) and magnetic resonance imaging (MRI) are all common techniques for the noninvasive imaging of the interior of the human body. Each of the techniques can produce three-dimensional arrays of intensity values, i.e., sampled volumetric data which can be used to generate an isosurface². There also exists irregularly sampled volumetric data from geology, meteorology, etc. which we do not discuss here.

Applications which generate volumetric data of the second type include the SCULPT program developed by Galyean and Hughes [10] (see Section 2.5) and other forms of volumetric modeling [2] [4] [3]. Voxelization of geometric models [13] [15] also falls into this category.

2.5 Volumetric Sculpting

Galyean and Hughes [10] describe an interactive volumetric modeling technique based on the paradigm of sculpting a solid material. Using a 3D input device to control a sculpting tool, the values in an array of voxels are modified much the same way the pixmap values are modified in a traditional 2D “paint” program. This tool, SCULPT, allows the creation of free-form models with complex topology by direct editing of the volumetric data, but lacks the ability to create finely detailed models.

²Not to suggest that medical imaging data was collected with the intent of generating a particular isosurface shape, merely that once it is collected, one can, and often does, choose to extract a surface shape from the data for other uses.

3 Motivation

3.1 A Step Towards Unification

Volume warping is a small step towards the much larger goal of the unification of volumetric and surface-based modeling techniques. This unification can be seen as having two facets:

1. To apply common techniques to both volumetric and polygonally based models.
2. To make the interchange between the volumetric representation of an object and a surface representation of the same object as easy as possible.

Built upon techniques originally developed for polygonal modeling [1] [18], volume warping represents the first part of this unification. Other work is currently underway [11] which will make the second part of the unification a reality and more tightly bind the worlds of volumetric and polygonal modeling.

3.2 Local Shape Control

Unlike the volumetric sculpting of Galyean and Hughes, which provides coarse shape/topology control for a volumetrically defined model, volume warping provides more precise and smooth control over the isosurface. Volumetric sculpting, like a traditional 2D paint program, only allows the user to specify the presence or absence of material at a specific location. Volume warping, on the other hand, by applying a mapping function to the actual volumetric data values changes the values which subsequently influence the isosurface generated. This cubic B-spline mapping function, based on evaluating only 64 control points (unlike the Bézier approach of Sederberg and Parry) provides local control.

To give an example of this local shape control property of volume warping, suppose you have created a model of a teapot using the SCULPT program [10] and you decide, after looking at it, that the neck of the teapot should be a little lower. To make this alteration by sculpting would require

that you erase the neck and re-sculpt it lower down (just as in a pencil-and-paper drawing you would have to erase and re-sketch). With volume warping, however, you could ask to alter the space in which the model resides so as to move the portion containing the neck. Volume warping can therefore be seen as providing an extension to volumetric sculpting.

4 The Algorithm

A volumetric model is just a function on 3-space or a subset of 3-space. We call this function the *density* function because it describes where material is: a density of one means the material is there, a density of zero indicates that 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 think of the density function as being defined primarily on the unit cube; its values outside the unit cube are everywhere zero. Thus we have a function

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

that denotes the presence or absence of material.

If we have a function from the unit cube to itself,³

$$B : (x, y, z) \mapsto (B^x(x, y, z), B^y(x, y, z), B^z(x, y, z)) \quad (1)$$

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

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

Figure 1 depicts this situation. The domain and codomain of B are drawn as two separate cubes and the original density d is a real-valued function on the domain. The new function, $d' = d \circ B$, becomes a function on the codomain of B , and is indicated by the dashed line. We can call this new density function the *pushforward* of the original density function by the map B . If B

³If P is a point in 3-space, the notation P^x , P^y , P^z denotes the x , y , and z coordinates of P .

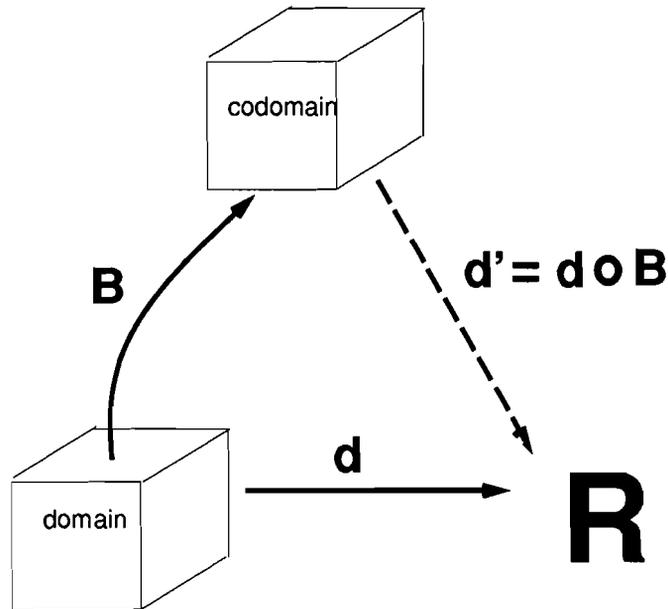


Figure 1: The pushforward of a density function.

is injective, we can look for the point (x, y, z) that is sent to each point W in the codomain by the map B . The value of the new density function at W is just the value of the original function at the corresponding source point (and is zero if W is not in the image of the map B). If B is not injective, some sort of average over all the points in the preimage must be used to adjust the definition. In our case, it suffices to deal only with the case where B is injective, because the other cases can be handled during the filtering process (see Section 5.4).

The volume-warping process, as shown in Figure 2, uses the pushforward technique described above. We start with a volumetric model, i.e., a density function d on a unit cube. We call this cube the *source* and denote it by S . We create a particular B-spline map B from the source to another unit cube, which we call the *target* T . We push forward the density d from S to get a new density function d' defined on T .

The density on the target T is the *warped version* of the original density function on the source S . If the B-spline map differs only slightly from the identity, then the isosurfaces of the source and target density functions are

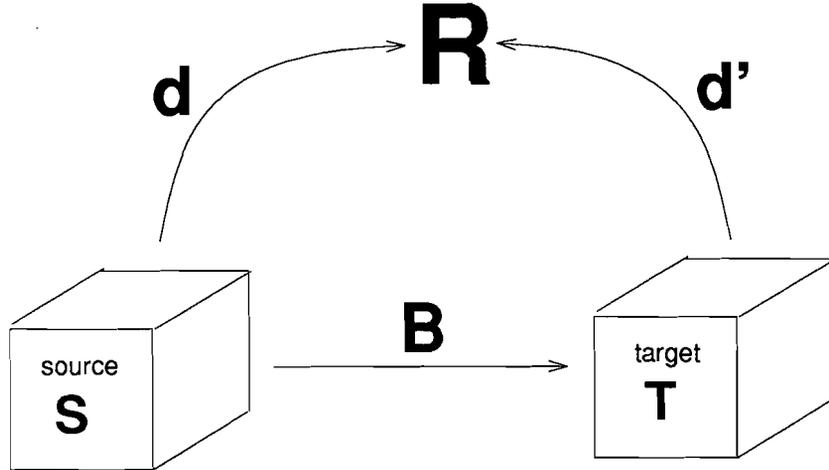


Figure 2: The mapping process.

quite similar and the term “warped” is justified.

5 Implementation

5.1 Overview

The description in the previous section was based on functions defined on an entire cube. With sampled volumetric data, we have only *samples* of the functions of interest. Since convolution with an appropriate reconstruction kernel to recover the whole function is impractical, we work with approximations of the functions represented by the samples. Rather than proper reconstruction with the *sinc* function, we use box filters repeatedly. We have not found the aliasing thus introduced to be a substantial problem.

The constants used in the program are

- **ISIZE**: the original model to be warped is an $\text{ISIZE} \times \text{ISIZE} \times \text{ISIZE}$ array of unsigned bytes (values between 0 and 255); by division by 255, these represent density values between 0 and 1, and the object described by the model is an isosurface for value = 127 (i.e., density = 0.5). In our program, $\text{ISIZE} = 30$.

- **TSIZE**: the target voxmap is an $\text{TSIZE} \times \text{TSIZE} \times \text{TSIZE}$ array of unsigned bytes. The warped model is placed into the center of this array (*TSIZE* must be greater than *ISIZE*) and the remainder of the target voxmap is padded with zeros. In our program, $\text{TSIZE} = 38$.
- **CSIZE**: The B-spline maps that we define are determined by an array of control points of size $\text{CSIZE} \times \text{CSIZE} \times \text{CSIZE}$. (We generally use $\text{CSIZE} = 9$, but other values may be appropriate for finer control.) 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 $\text{CSIZE} - 3$. The ratio of TSIZE to $\text{CSIZE} - 3$ must be an integer for the rapid spline evaluations of Section 5.2 to work. In our program, this ratio is $60/(9 - 3) = 10$. We call this ratio the *granularity* G .

Our B-spline map is defined on the source with the help of the B-spline basis functions and the $\text{CSIZE} \times \text{CSIZE} \times \text{CSIZE}$ array of control points P_{ijk} . We first establish a correspondence between points in the source 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 source array whose index is (i, j, k) corresponds to the point in 3-space whose coordinates are

$$(s, t, u) = (i/G, j/G, k/G) \tag{2}$$

Recall that the B-spline basis functions are

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

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

$$\begin{aligned} a &\leq s < a + 1 \\ b &\leq t < b + 1 \\ c &\leq u < c + 1 \end{aligned}$$

of the source (which we call a *region*) by using the control points P_{ijk} , for

$$\begin{aligned} a &\leq i < a + 3 \\ b &\leq j < b + 3 \\ c &\leq k < c + 3 \end{aligned}$$

The general form for the mapping function on this region is therefore

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

Note that there are $G \times G \times G$ regions and that altering one of the control points alters the values of the function in only a subset of the regions. Thus although our map is analogous to the 3D Bernstein polynomials used by Sederberg and Parry [18], it provides more localized control.

The B-spline map between the source and the target voxmap is specified by the location of the control points. We can alter the map by moving the control points that define this map. Having defined the altered map, the density is pushed forward from the source to the target to produce the warped sampled volumetric data set. For each point in the domain of the altered 3D B-spline we will evaluate the B-spline function to get another 3D point (Equation (3) shows how to compute $B(s, t, u)$), and record the resulting locations in a 3D array of 3D points called the *deformation mapping array*.

The precise details of the pushforward of the discrete data values are given in Section 5.4. Before addressing this however, we describe how the initial values of a B-spline map are computed for each point in the source voxmap.

5.2 Rapid Evaluation of B-splines

It is clear that we must be able to compute the B-spline map rapidly, since it is applied to each voxel in the source voxmap. The standard way to

do this with B-splines would be to use incremental computation [8]. Once we realize, however, that the parameter values at which the B-splines are computed do not change when the control-point positions are changed, we see that a pre-computation scheme can save substantial time. Moreover, by giving the relative sizes of the source and the control mesh an appropriate relation, we achieve a substantial speedup.

Look once again at Equations (2) and (3). Since the source size is an integer multiple of the control-mesh size, the fractional parts of s , t , and u used in the computation of the B-spline in Equation (3) are the same for points in any region. We use this fact as follows. If we denote by b_{ij} the coefficient of t^j in the i th basis function of Equation (3), then we can rewrite the B-spline evaluation formula as

$$\begin{aligned}
B^x(s, t, u) &= \sum_{i,j,k=0}^3 B_i(s-a)B_j(t-b)B_k(u-c)P_{a+i,b+j,c+k}^x \\
&= \sum_{i,j,k,p,q,r=0}^3 b_{ip}(s-a)^p b_{jq}(t-b)^q b_{kr}(u-c)^r P_{a+i,b+j,c+k}^x \\
B^y(s, t, u) &= \sum_{i,j,k=0}^3 B_i(s-a)B_j(t-b)B_k(u-c)P_{a+i,b+j,c+k}^y \\
&= \sum_{i,j,k,p,q,r=0}^3 b_{ip}(s-a)^p b_{jq}(t-b)^q b_{kr}(u-c)^r P_{a+i,b+j,c+k}^y \\
B^z(s, t, u) &= \sum_{i,j,k=0}^3 B_i(s-a)B_j(t-b)B_k(u-c)P_{a+i,b+j,c+k}^z \\
&= \sum_{i,j,k,p,q,r=0}^3 b_{ip}(s-a)^p b_{jq}(t-b)^q b_{kr}(u-c)^r P_{a+i,b+j,c+k}^z
\end{aligned}$$

where the values of $s-a$, $t-b$, and $u-c$ are always $0/G, \dots, (G-1)/G$. In fact, for the point in the source voxmap indexed by e, f, g , the values of $s-a$, $t-b$, and $u-c$ will be $(e \bmod G)/G$, $(f \bmod G)/G$ and $(g \bmod G)/G$. Letting $\bar{e} = e \bmod G$, and similarly for \bar{f} and \bar{g} , we can write the formula

$$B^x(s, t, u) = \sum_{i,j,k,p,q,r=0}^3 b_{ip}(\bar{e}/G)^p b_{jq}(\bar{f}/G)^q b_{kr}(\bar{g}/G)^r P_{a+i,b+j,c+k}^x$$

$$\begin{aligned}
B^y(s, t, u) &= \sum_{i,j,k,p,q,r=0}^3 b_{ip}(\bar{e}/G)^p b_{jq}(\bar{f}/G)^q b_{kr}(\bar{g}/G)^r P_{a+i,b+j,c+k}^y \\
B^z(s, t, u) &= \sum_{i,j,k,p,q,r=0}^3 b_{ip}(\bar{e}/G)^p b_{jq}(\bar{f}/G)^q b_{kr}(\bar{g}/G)^r P_{a+i,b+j,c+k}^z
\end{aligned}$$

We therefore precompute the values

$$c[\bar{e}][\bar{f}][\bar{g}][i][j][k] = \sum_{p,q,r=0}^3 b_{ip}(\bar{e}/G)^p b_{jq}(\bar{f}/G)^q b_{kr}(\bar{g}/G)^r \quad (4)$$

for $0 \leq \bar{e}, \bar{f}, \bar{g} < G$ and $0 \leq i, j, k \leq 3$. Computing the value of the spline function at particular values of e, f , and g now requires only that we determine the values of a, b , and c (which are just the integer parts of $e/G, f/G$, and g/G , respectively) and the values of \bar{e}, \bar{f} , and \bar{g} . We then calculate the sum in Equation (4) by the simple form

$$\begin{aligned}
B^x(s, t, u) &= \sum_{i,j,k=0}^3 c[\bar{e}][\bar{f}][\bar{g}][i][j][k] P_{a+i,b+j,c+k}^x \\
B^y(s, t, u) &= \sum_{i,j,k=0}^3 c[\bar{e}][\bar{f}][\bar{g}][i][j][k] P_{a+i,b+j,c+k}^y \\
B^z(s, t, u) &= \sum_{i,j,k=0}^3 c[\bar{e}][\bar{f}][\bar{g}][i][j][k] P_{a+i,b+j,c+k}^z \quad (5)
\end{aligned}$$

which is just a sum of 64 terms. Note that in C and Fortran, the 64 entries of the coefficient array used are stored in sequential memory, so that access to them can be particularly rapid.

5.3 Implementation Details

We now can complete the overall structure of the program.

Initialization. We begin by reading the original density function into the source voxmap; we then set up the lattice of control points as described in Section 5.5 and thus define a B-spline map from the source voxmap to the target voxmap. This will be the map used to push forward the density

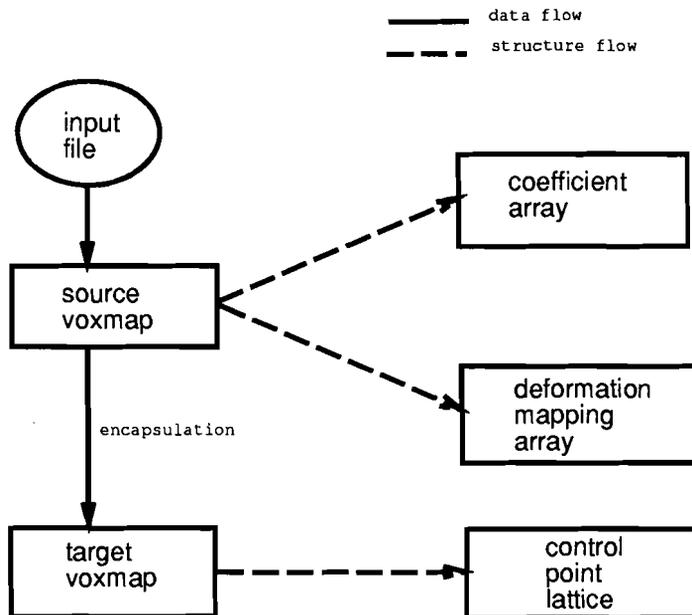


Figure 3: Block diagram of initialization process.

function from the source voxmap to the target voxmap. Note that we choose our control points so that the initial B-spline map is a scalar multiple of a translate of the identity so that pushing forward the original density function through the initial control point configuration will produce a target voxmap which is identical to the source.

Once the lattice of control points is set up, we calculate the s , t , and u values at each point of the source voxmap, and use these values to determine the four B-spline basis function values on each of s , t and u ; these are saved for later use (see Section 6). Then we compute the $G \times G \times G \times \text{CSIZE} \times \text{CSIZE} \times \text{CSIZE}$ array of coefficients described in Equation (4), for use in determining the values of the spline map. Finally, we initialize the memory required for the target voxmap and the deformation mapping array. Figure 3 pictorially illustrates this initialization of the various data structures described above.

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 density function using moved control points
    add pushed forward values
  }
}

```

Each of the steps in this loop has its own subtlety. The first part, choosing which control points to move, can be as simple as reading a file containing a list of control-point indices and the new control-point locations. We have actually implemented a more intuitive method of control as the basis for an interactive user interface, as described in Section 6. For now, it suffices to say that certain control points are scheduled to be moved.

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. For each region 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 to simply 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 source to the target voxmap.

We must now push forward the density function from the source to the target voxmap. The simplest way to do this is to compute where each source voxel center is sent by the mapping, round off to integer coordinates, and then place the density from the source voxel into the target voxel at those coordinates. This produces severe aliasing, however, and we are therefore compelled to do some filtering. To this end, we compute the (floating-point) coordinates of the point to which each source voxel-center is sent and place the results in the $\text{TSIZE} \times \text{TSIZE} \times \text{TSIZE}$ deformation mapping array. Computing the results of the map in this deformation mapping array is expensive, but using the rapid B-spline computation described in Section 5.2 reduces the time involved substantially. Figure 4 illustrates the flow of data from the source to the target voxmap.

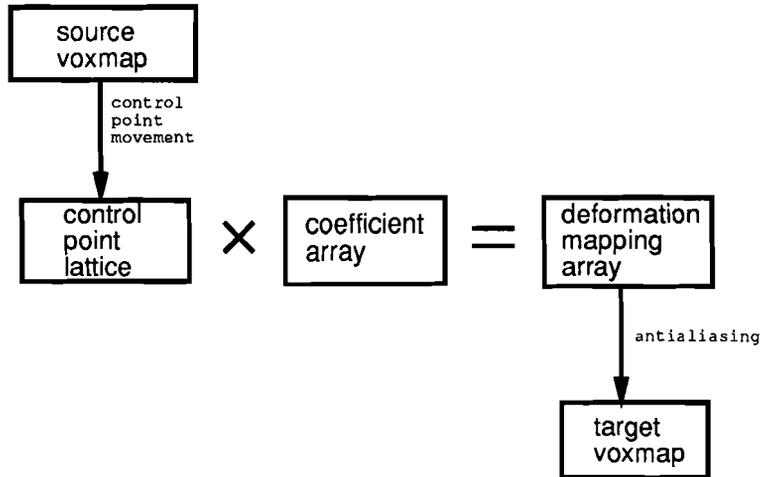


Figure 4: Block diagram of warping process.

5.4 Antialiasing

As noted in the introduction, we ought to exactly reconstruct the complete density function on the source voxmap with a *sinc* filter, and then push forward this reconstructed function. We could then filter the result with a *sinc* function on the target space to remove all high frequencies before sampling the function into the target voxmap.

Rather than doing anything so computationally expensive, however, we use box filtering to approximate the function. We reconstruct the density function on the source with a box filter by taking the value at each voxel center and assuming 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 voxel in the target, compute the approximate overlap of the transformed box with the voxel and record it as a fraction of the target voxel’s volume that we call the *weight*. We then take the value from the source voxel and multiply it by the weight, to get the *weighted value*. We tally these weights and weighted values for each target voxel. In pseudocode:

```

for each source 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 source voxel is prohibitively expensive. Once again, we compromise and instead compute an approximation of the transformed source voxel: we compute the approximate transformed locations of the centers of the six faces of the source voxel and take the bounding cube of these six points as our proxy for the actual transformed source voxel. The effect (in an analogous 2D case) is shown in Figure 5: the bounding boxes of adjacent pixels may overlap. Thus a single target pixel (in the analogous 2D process) 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 source pixels. If not, then it is reasonable to count the portion left uncovered as contributing 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 source voxel center in this array. To find the center of the face lying between the voxel center at (e, f, g) and the one at $(e + 1, f, g)$, we simply average the points stored in the deformation mapping array at these two indices.⁴ Figures 6 and 7 show the results of volume

⁴We 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 having the locations of the transformed voxel centers lets us do the very inexpensive pushforward described at the end of Section 5.3 in almost real time.

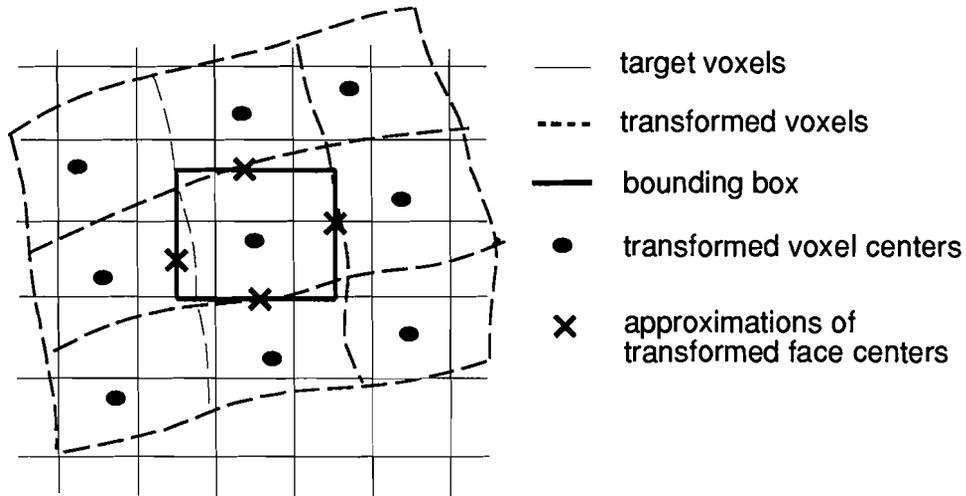


Figure 5: The bounding box of 6 points in a transformed voxel is used to approximate the transformed voxel.

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 the isosurface of the resulting density field and render it.

5.5 Control Point Placement

In order to determine the placement of each control point, we start by dividing the central $ISIZE \times ISIZE \times ISIZE$ subarray of the target data space into $CSIZE - 3$ pieces in each direction; each division is then a cube with an edge of length $ISIZE / (CSIZE - 3)$ (which may not be an integer, of course). The control points are then placed in an evenly spaced lattice based upon this subdivision of the target voxmap.

Since we have chosen an evenly spaced control lattice and the values in each segment being B-splines require 4 control points, the control points on the end in each direction will lie outside of the central $ISIZE \times ISIZE \times ISIZE$ subarray of the target voxmap. This even spacing of the control points aids in the mapping and B-spline evaluation by making the parametric steps in s , t and u uniform throughout the volume. Note that because this is a B-spline,

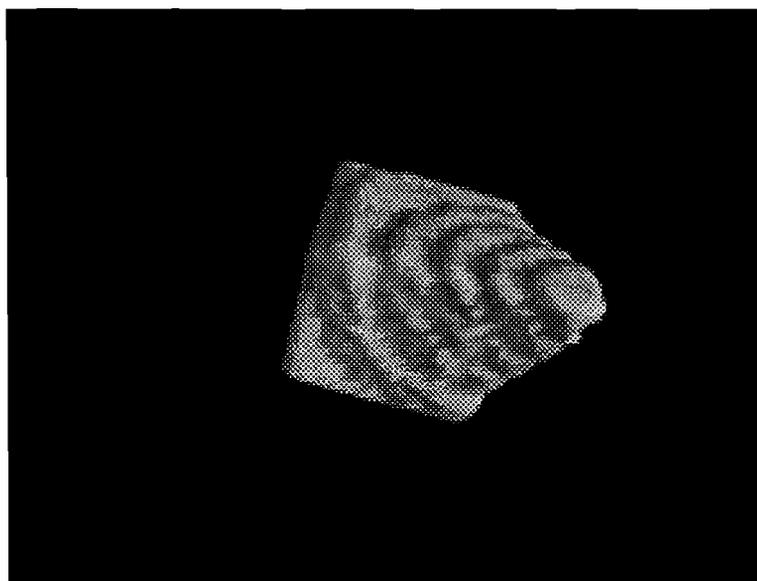


Figure 6: A warped volume in which the pushforward was done with the simple “map and round off” scheme.

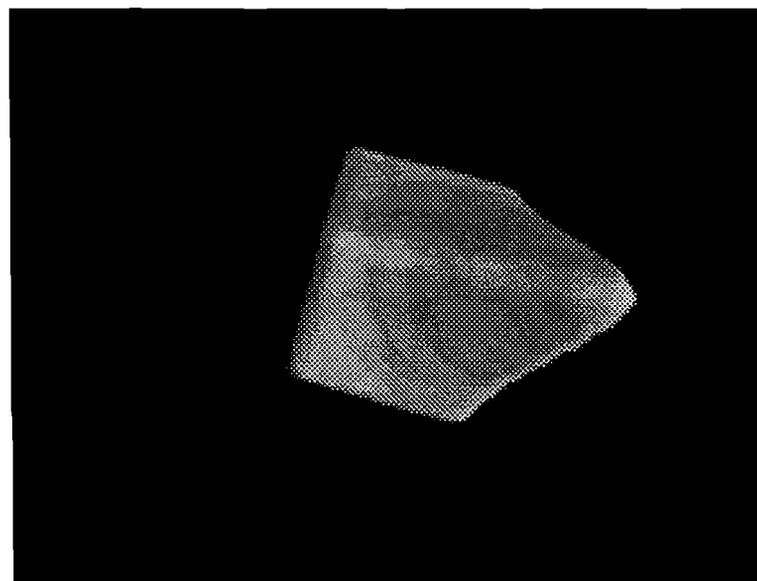


Figure 7: A warped volume in which the bounding-box filtering technique was used.

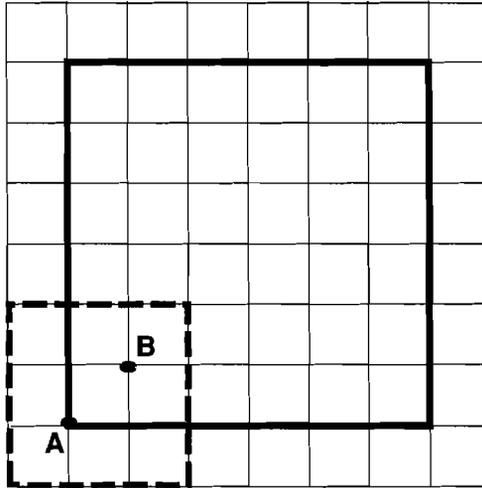


Figure 8: The placement of the control points for the B-spline map in 2D, for a 9×9 control mesh; note that the data is divided into a 6×6 grid of regions.

the image of the function defined by the control points is a proper subset of the convex hull of the control points.

An example of the control point layout in 2D when $CSIZE = 9$ can be seen in Figure 8. In this figure, notice how the data area denoted by the bold rectangle is divided into $CSIZE - 3$ (i.e., 6) equal regions and how some of the control points lie outside the bounds of the data. As an example of how the control points affect each region, in this 2D example data values in region $A_x \leq x < B_x$ $A_y \leq y < B_y$ are influenced by the 16 control points in the rectangular region denoted by the dotted line.

6 User Interface

6.1 Background

Hsu [12] develops a direct manipulation interface to FFDs which allows the user to directly manipulate the surface of an object without having to directly manipulate the individual control points which govern the surface. It is upon this direct manipulation paradigm that the user interface for volume warping

is built.

Traditional interfaces to FFDs like those to spline curves and surfaces have typically relied on direct control-point manipulation. An interface of this type however is unnatural and hard to use and understand for a user that is unfamiliar with the principles of splines. For example, the fact that the control points most often do not approximate the surface of the object makes direct control-point manipulation difficult since one can often not exactly determine the effect that moving one particular control point will have on the surface.

Hsu's interface eliminates the need for the user to be aware of the control points providing a simple and easy to use click-and-drag interface based upon the paradigm of a "magnet tool" that allows both the pulling on and pushing against a single object point or a larger section of the object. By varying the size, shape and "realm of influence" of this tool different effects can be achieved. Although Hsu's method was originally applied to polygonally defined models, it can be applied directly to the warping of volumetrically defined models.

6.2 How It Works

In the users view, to warp volumetric data, all that is required is the selection of a point within the volumetric space and the movement of this point to a new location within the volume. Once the point is placed in its new location, the data warps accordingly. In reality things are not that simple however. Even though the user no longer has to think about and directly move the control points, the control points are still the governing factor in the deformation. We therefore implement a scheme where the control points are moved based upon how the user moves the selected point or a group of selected points within the volumetric model. This scheme of moving the control points based upon the movement of data points can be characterized as an underconstrained system. Hsu describes in detail the general solution for an underconstrained system of B-splines so we will not discuss it further here.

6.3 Extensions

This interface, which allows the user to directly manipulate the volumetric data by indirectly moving the control points based upon the movement of a selected point or group of points can be the basis for several more complex interaction methods. Among those that we can imagine are:

Arbitrary Geometry Allowing the user to specify arbitrary geometries for the warping tool with different spheres/areas of influence provides an unlimited number of achievable warping effects.

Particle Accelerator A tool that when moved within a certain range of the volume sucks in points on one side and then spits them back out on the other side.

Start and Goal Curves Given user defined starting and ending 3D curves, the warp of the volumetric data interpolates between the curves over a sequence of frames.

Object Avoidance Given a user defined object, the volumetric data warps by being pulled or pushed around or through a specified object or group of objects. An example of this would be a torus accelerator where the model is warped by pulling it through a torus.

7 Usage Examples

Comparison between the unwarped and warped volumetric models in Figures 9–14 demonstrates some of the results attainable with our technique.

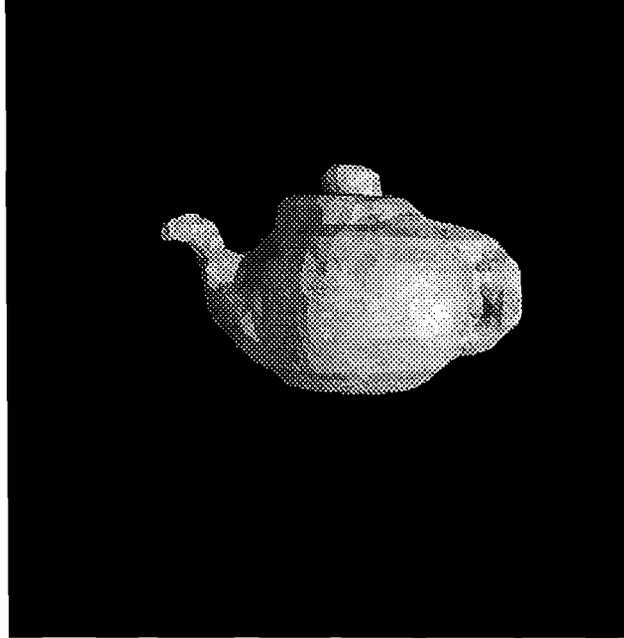


Figure 9: Teapot, original sculpture.

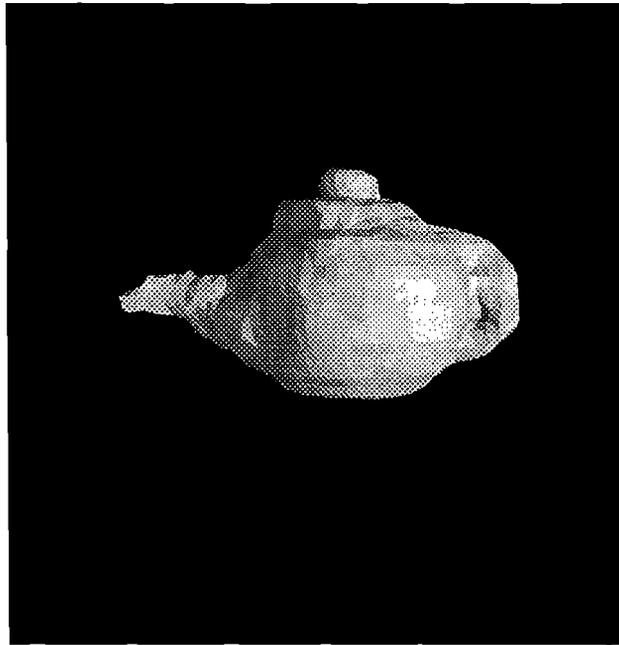


Figure 10: Teapot, warped to adjust the spout position.

8 Performance

To be truly interactive, any modeling technique requires realtime performance. We have worked hard to achieve this performance, feeling that volume warping would only be useful if this interactive performance could be attained.

The interactive manipulation and viewing of volumetric models is numerically intense and realtime performance on currently available graphics workstations is hard to achieve. The pushforward mapping and antialiasing inherent in volume warping makes interactive performance of our new volumetric modeling technique even harder to achieve. Our rapid evaluation of B-splines (see Section 5.2) and our ability to localize a deformation have not been enough to give us realtime volume warping. Currently, one warp of a volumetric model where ISIZE= 30, TSIZE= 38 and CSIZE= 9 requires approximately 1 second on an HP 9000/730 high performance graphics workstation. Analyzing this performance, we have found the following bottlenecks in our implementation.

antialiasing Computing the bounding volume of the transformed source voxel and the percentage of overlap calculations for each target voxel as described in Section 5.4 have shown to consume a large part of the actual deformation time (independent of rendering and redisplaying the model).

rendering We use the marching cubes algorithm [16] to generate the individual polygons which comprise the isosurface of the model. For even relatively small models (ISIZE= 30) however, this algorithm generates on the order of 6000-8000 triangles each time the model is re-rendered after a warp.

screen refresh Independent of the actual rendering of the model discussed above, we have found screen refresh to also to be a major bottleneck. Although an HP 9000/730 graphics workstation is rated at 300,000 polygons/sec, simple redisplay of a volumetric model composed of 6000 triangles can't be done in real time (30 frames/sec.). The reason for this is that the 300,000 polygon/sec statistic is based on the display of triangle strips where adjacent triangles share common vertices and the adding of an additional triangle simply requires the addition of a single

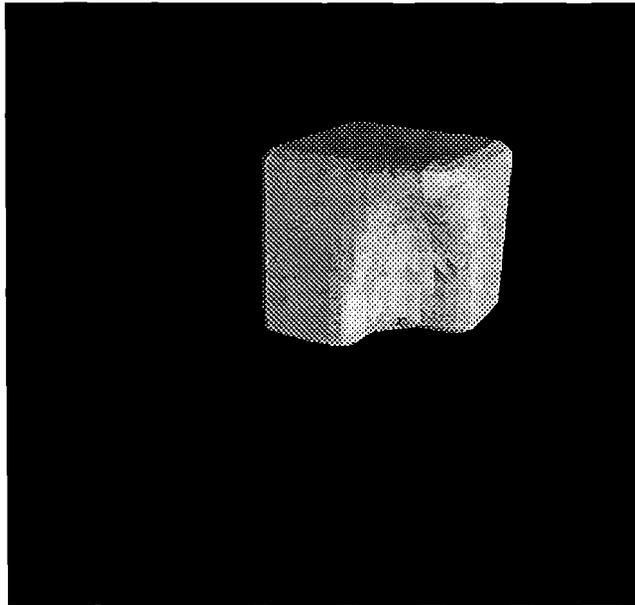


Figure 11: Clay block, corner warped inward.

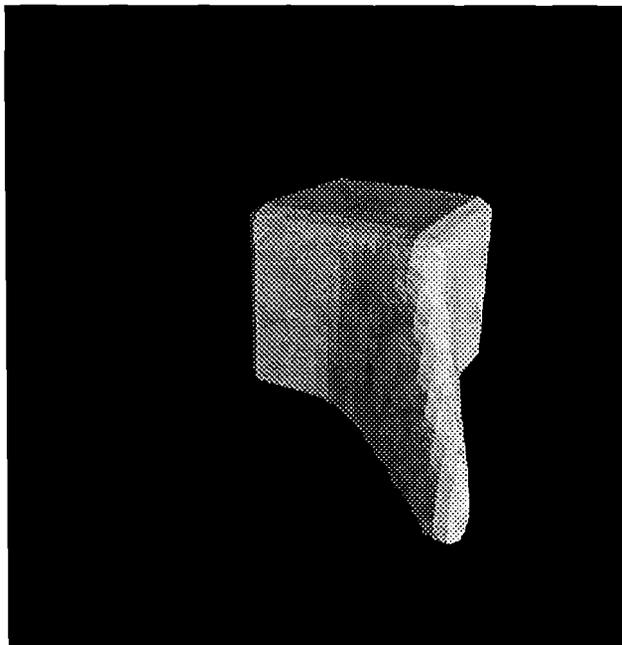


Figure 12: Clay block, corner warped outward.

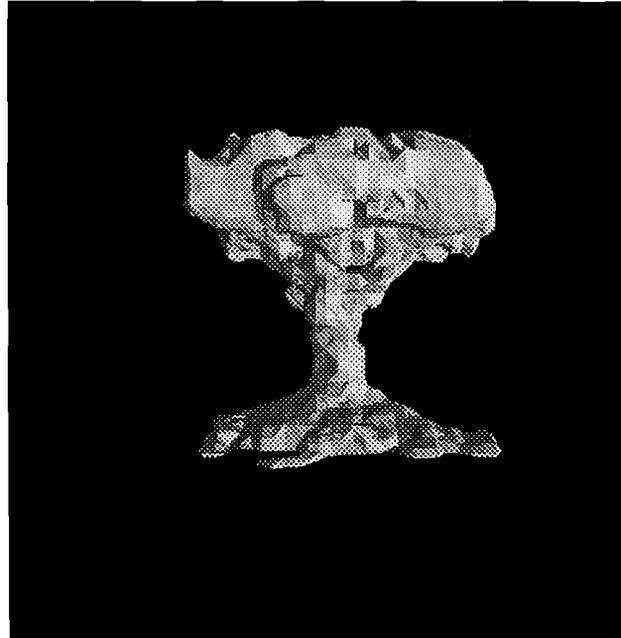


Figure 13: Tree, original sculpture.

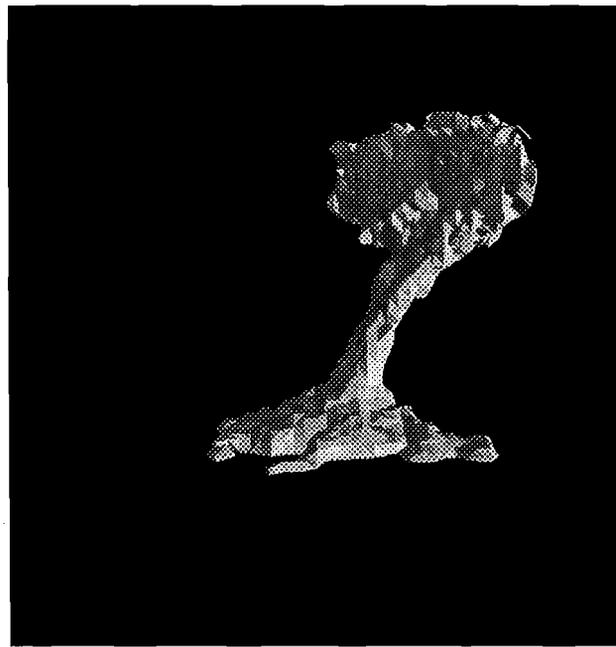


Figure 14: Tree, warped as if caught in a strong gust of wind.

point. Our rendering algorithm produces individual triangles which cannot be displayed as quickly by the graphics hardware.

9 Future Work

9.1 Performance

One of the many ways in which we are looking to improve this modeling technique is to improve the performance. We are looking into solutions to the bottlenecks discussed in Section 8.

By changing our rendering algorithm to output triangle strips or polygonally defined meshes, we feel that we might be able to achieve an order of magnitude speedup in both rendering and screen refresh. A limitation in most currently available graphics packages which may not make this feasible however, is the fact that most implementations of triangle strips and polygon meshes where vertices are shared do not allow vertices to have more than one color and normal per vertex for lighting and shading calculations. We are currently looking into ways to work around this limitation.

The interactive performance of the user interface could be improved if the graphics hardware supported overlay planes which share the z-buffer with the image planes. This would allow the interactive movement of the warping tool without having to redraw the complete underlying model each time.

An iterative refinement approach to rendering a warped model could also be used to improve the interactive performance. If we skipped the antialiasing step described in Section 5.4 (producing results similar to Figure 6) the computation of the bounding volume and overlap for each transformed source voxel would no longer be significant in the warping time. This unantialiased rendering could then be used during an interactive “rubberbanding” phase of warping a model; the antialiased rendering could be used to exhibit the final warped results once the “rubberbanding” is complete.

We also feel that volume warping is an ideal application for fine-grained parallelism. Performance could be improved by farming out the B-spline mapping of each region to a different processor.

9.2 Usability

As a standalone modeling technique, volume warping isn't all that useful. In the future we would like to integrate volume warping with volumetric sculpting so that the user could use the techniques together. As stated in Section 3.2, we feel that volume warping would prove to be an excellent postprocess to volumetric sculpting by providing the user local shape control for making minor alterations to a sculpted model.

We would also like to see volume warping integrated into an environment in which the user can easily switch between volumetric and polygonal modeling techniques. Within an environment of this type, a user could easily warp any modeled object.

We also plan to add the capabilities for using three-dimensional input devices to control the deformations. Specific devices which we are considering to control the movement of the warping tool include a three-dimensional mouse, space-ball, and hardware dials.

9.3 Applications

In the future we would like to apply volume warping to scientific data. One application of this technique may be the removal of distortions and inconsistencies introduced during the data acquisition process, although one would have to be careful to regulate the data errors introduced by the warping process itself.

Another potential application may be in the field of morphometrics. Scientists working in this area can apply volume warping to volumetrically defined bone structure models to graphically simulate bone structure growth and decay during the aging process.

10 Conclusions

We have described volume warping, a new volumetric modeling paradigm derived from techniques in polyhedral modeling and image warping. The results let one deform volumetric models in a convenient manner that is moderately fast and easy to implement. We have also described techniques for rapid evaluation of B-splines in the particular case where the control points are to be changed but the evaluation points remain constant, and a

technique for manipulating the control points of a 3D B-spline automatically so as to effect a particular change in the map. We have also described several user interface paradigms based upon this technique. We see this as one small step towards the much larger goal of the unification of polygonal and volumetric modeling techniques.

11 Acknowledgements

I would like to thank my advisor John Hughes for his initial suggestion to undertake this research and endless help overcoming the mathematical and technical stumbling blocks encountered along the way. His clear explanations of the mathematical details, review of my writing, and overall interest in my work has been immeasurable. I also could not have completed this research without the help, support, and encouragement of all the great people in the Brown University Computer Graphics Group led by Andries van Dam and John Hughes. In particular I would to thank George Reilly, who assisted in improving the performance of the code, Ken Herndon, who made the first model of the teapot, and Dan Robbins, who created the model for the tree. Other folks in the group worthy of special thanks for their guidance and assistance include Bob Zeleznik, Nate Huang, Brook Conner, and Scott Snibbe. Mary Andrade of Brown also deserves special thanks for her friendship, moral support, and proofreading skills. I'd also like to thank Ingrid Carlbom and William Hsu of the Digital Equipment Corporation Cambridge Research Lab; Ingrid for numerous words of wisdom regarding volume visualization and suggestions of future applications for my research and William for sparking my interest in free-form deformations. I would also like to acknowledge the financial support provided by Digital Equipment Corporation's Graduate Engineering Education Program which made my studies at Brown University possible. Most of all, I'd like to thank my family for their unending support and my special friend Thérèse Mersereau for her encouragement and patience over the past 18 months.

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] Steve Bryson. Paradigms for the shaping of surfaces in a virtual environment. In *Proceedings of the Hawaii International Conference on System Sciences*, pages 649–658, January 1992.
- [6] Carolco Productions. Terminator 2, 1991.
- [7] E. Catmull and A. R. Smith. 3-d transformations of images in scanline order. *Computer Graphics*, 14(3):279–285, 1980.
- [8] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics : Principles and Practice*. Addison Wesley, second edition, 1990.
- [9] H. Fuchs, S. Levoy, and S. M. Pizer. Interactive visualization of 3d medical data. *Computer*, pages 46–51, 1990.
- [10] T. Galyean and J. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics*, 25(4):267–274, July 1991.
- [11] C. Grimm and J. Hughes. S-patching implicit surfaces. In preparation.
- [12] William M. Hsu. A direct manipulation interface to free-form deformations. Master’s thesis, Brown University, May 1991.
- [13] A. Kaufman. Efficient algorithms fo 3d scan-conversion of parametric curves, surfaces and volumes. *Computer Graphics*, 21(4):171–179, 1987.

- [14] A. Kaufman. *Volume Visualization*. IEEE Computer Science Press, 1990.
- [15] A. Kaufman and E. Shimony. 3d scan-conversion algorithms for voxel-based graphics. In *ACM Workshop on Interactive 3D Graphics*, pages 45–75, October 1986.
- [16] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.
- [17] G. M. Nielson and B. Shriver. *Visualization in Scientific Computing*. IEEE Computer Science Press, 1990.
- [18] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, August 1986.
- [19] D. B. Smyth. A two-pass mesh warping algorithm for object transformation and image interpolation. *ILM Technical Memo 1030*, 1990.
- [20] G. Wolberg. *Digital Image Warping*. IEEE Computer Science Press, 1990.