Imorph: An Interactive System for Visualizing and Modeling Implicit Morphs

H. Quynh Dinh[1]
Multimedia, Vision and Visualization
Stevens Institute of Technology

Bruno Carvalho[2]
Multimedia, Vision and Visualization
Stevens Institute of Technology

Department of Computer Science
Technical Report CS-2004-8

July 26, 2004

[1]e-mail: quynh@cs.stevens-tech.edu
[2]e-mail: bruno@cs.stevens-tech.edu

Abstract

A metamorphosis, or morph, describes how a source shape gradually changes until it takes on the form of a target shape. Morphs are used in CAD (to construct novel shapes from basic primitives) and in motion picture special effects (to show one character transforming into another). To date, implicit morphing algorithms have been off-line processes. Source and target shapes and any user-defined initial conditions (object positions and warps) are provided as input to these black-box methods. We present an interactive system that constructs an implicit morph in real-time using the texturing hardware in graphics cards. Our solution allows a user to interactively modify parameters of the morph (scaling, translation, rotation, and warps) and immediately visualize the resulting changes in the intermediate shapes. Our approach consists of three elements: (1) the real-time computation and visualization of transforming shapes, (2) the ability to immediately see changes reflected in a morph when source and target shapes are manipulated, and (3) an efficient image-based method for updating the discrete distance field when affine transformations are applied to source and target shapes.

0.1 Introduction

A metamorphosis, or morph, describes how a source shape gradually deforms until it takes on the form of a target shape. Morphs are also called shape transformations, and are used in CAD (to construct novel shapes from basic primitives) and in motion picture special effects (to show one character transforming into another). Morphs can be represented parametrically or implicitly. In the parametric representation, points of the source shape gradually move to the final shape through a linear or other smooth path. Essential to a parametric transformation is a dense set of point correspondences between the shapes. This dense set is often generated from a sparse user-defined set. An intermediate shape to which both the source and target shapes have point correspondence is sometimes required to generate the dense set of correspondences. In an implicit morph, an implicit function describes the changing geometry. Intermediate shapes are acquired by extracting level sets of the function at different time slices. The primary advantages of implicit methods for shape transformation are that they can generate a plausible transition between shapes of differing topology and that they do not require user-defined point correspondences between the source and target shapes. Some parametric methods can handle changes in topology but do so with user intervention or by cutting the surface to obtain a common topology. Many implicit approaches allow additional user-defined constraints (such as the position of the shapes in relation to each other and warps applied to the shape prior to morphing) that influence the resulting shape transformation [8, 2].

The primary drawback of implicit methods is that it is difficult to control the outcome of the transformation – the resulting intermediate shapes in the sequence. To date, implicit morphing algorithms have been off-line processes. Source and target shapes and any user-defined initial conditions are provided as input to these black-box methods. The output is a set of intermediate shapes generated at different slices in time and rendered for visualization in an animation. Extracting the intermediate shapes using an iso-surface extraction algorithm such as Marching Cubes [9] is also an off-line, time-consuming step. The resulting process of modeling morphs consists of trial-and-error wherein the user modifies the input parameters, generates the morph, and extracts intermediate shapes for each set of paramters. This approach is highly non-interactive.

The approach that we present significantly reduces the input-output loop by constructing the implicit shape transformation in real-time using the texturing hardware in graphics cards. Our solution, detailed in Section 0.3.1, allows a user to interactively modify parameters of the morph (such as scaling, translation, rotation, and warps) and immediately visualize the resulting changes in the intermediate shapes. A 2D example is shown in Figure 1. The interactivity our framework provides makes the modeling of transformations akin to modeling geometry. In addition, immediate feedback enables a user to better visualize and control how topology changes in a transformation sequence. Our approach consists of three contributions:

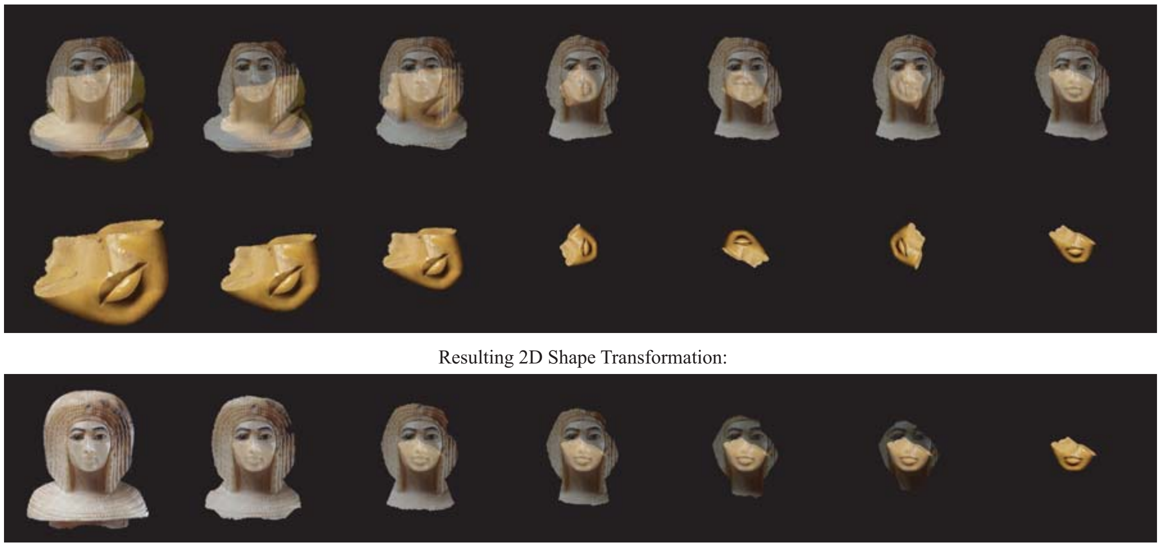

Figure 1: Top: Midpoint in the 2D morph from a stone bust to a broken half face. 2D shape changes as the target shape is scaled, translated, and rotated. Center: Manipulation of the target shape (half face). Bottom: Final morph from stone bust to half face.

1 Real-time calculation and visualization of implicit morphs.

2 Interactive manipulation of source and target in an implicit morph modeling system.

3 Image-based method for calculating distance fields that allows for faster updates of the distance field as source and target shapes are manipulated.

As with many image-based approaches, our method can handle complex shapes of arbitrary topology, subject to resolution. The performance of our system is dependent on the resolution of the input and improves with additional texture memory.

The remainder of this paper presents related work in Section 0.2, our new real-time computation and visualization of 2D and 3D morphing shapes in Section 0.3, our framework for interactively applying affine transformations and non-linear warps to source and target shapes in Section 0.4, the image-based method to computing distance fields in Section 0.5, and timing results in Section 0.6.

0.2 Related Work

In this section, we briefly review existing methods for constructing and controlling implicit shape transformations, and methods that use hardware-accelerated computations.

0.2.1 Implicit Shape Transformation

A wide variety of approaches to performing shape transformation using implicit functions have been published in the graphics literature. These methods require little user-input and are able to handle changes in topology without user intervention. However, few of these methods offer much user control over the resulting shape transformation, nor are they interactive. Instead of covering the many implicit approaches here, we refer the interested reader to [6] where the authors present a comprehensive survey of morphing algorithms. The interactive framework that we present uses interpolated distance fields to generate the morph. In [11], Payne and Toga use linear interpolation between signed distance transforms of source and target shapes to produce smooth morphs. The magnitude of the signed distance field indicates the distance from the surface, while the sign indicates whether a point is inside or outside of the object. At any point in time, the intermediate shape is the zero level-set of the interpolated distance function. Their method has become a standard in implicit transformations.

None of the previous implicit shape transformation algorithms generate the morph in real-time. If a resulting transformation is unsatisfactory, a user must modify the initial parameters of the transformation and apply the morphing algorithm once again in an off-line process. To view the intermediate shapes in an implicit transformation, the intermediate shapes must first be generated using an iso-surface extraction algorithm such as Marching Cubes [9]. Both the creation of the implicit morph and the extraction of the iso-surfaces of intermediate shapes are time-consuming processes. In contrast, the interactive framework we describe enables a user to manipulate the source and target shapes of a morph and immediately visualize the changes reflected in the intermediate shapes without an iso-surface extraction step.

0.2.2 Controlling Shape Transformations

A number of the methods described above enable user control over the shape transformation by allowing the initial conditions of the source and target shapes to be changed, or by incorporating deformations into the morph. In [8], Lerios et al. provide user control of the transformation through user identification of corresponding features on the source and target shapes. They construct a warp function from this sparse sampling of corresponding points by weighted averaging of all corresponding pairs, where the weights are determined by the inverse squared distance to identified feature points. In [2], the signed distance transform is used to perform morphing, similar to the approach of Payne and Toga. Cohen-Or et al. add user control to this process by allowing spatial deformations in order to produce better alignment of the objects, and the results are superior those created using non-deformed objects. The deformation is defined by a warp function that is applied to the signed distance field. This warp function consists of a rigid rotational component and an elastic component, and is controlled by a set of corresponding anchor points between the source and target shapes in the transformation.

Some implicit morphing methods modify the initial conditions of the source and target shapes. These initial conditions consist of the position and orientation of the shapes, and their relative proximity to each other. For example, the amount of spatial overlap between the shapes often determines the intermediate shapes in the transformation sequence. Controlling shape transformations by altering initial conditions gives rise to modeling by trial-and-error in that the initial conditions are changed iteratively until a desirable transformation sequence is obtained. Such a modeling paradigm is non-interactive because, more often that not, the morphing algorithm is run off-line after the initial conditions are changed. A user must then examine the results and adjust the input conditions accordingly. We seek a method that reduces this input-output loop so that as the user rotates, translates, or scales the source and target shapes, these changes are reflected immediately in the transformation sequence.

0.2.3 Hardware-Accelerated Algorithms

Hardware-accelerated algorithms have become increasingly prevalent as the computational power of the graphics processing unit (GPU) increases faster than that of general processors. In particular, researchers have taken advantage of the GPU's ability to process data in parallel in the form of textures. Recently, hardware algorithms have branched out into two categories – real-time shading/rendering and real-time computations (geometric and non-geometric). Our approach to modeling shape transformations involves interactive 2D and 3D visualization and real-time geometric computations. In the area of visualization, speed-ups in volume rendering have been made possible through 3D texturing [1]. The interactive system we present was inspired by the work of Westermann et al. in [14] where they develop a method to render volumes interactively. They do so by rendering planes parallel to the image plane that are clipped against the 3D texture domain. The hardware interpolates the 3D texture coordinates and blends successive parallel planes that have been textured with the volumetric data. They also shade the iso-surfaces of volumes without extracting a polygonal representation. In our approach, we apply their framework to interactively visualize the intermediate shapes in a 3D transformation without explicitly extracting an iso-surface.

We use the interpolated distance field method of Payne and Toga to produce a smooth morph. Hence, computation of the distance field for an arbitrarily

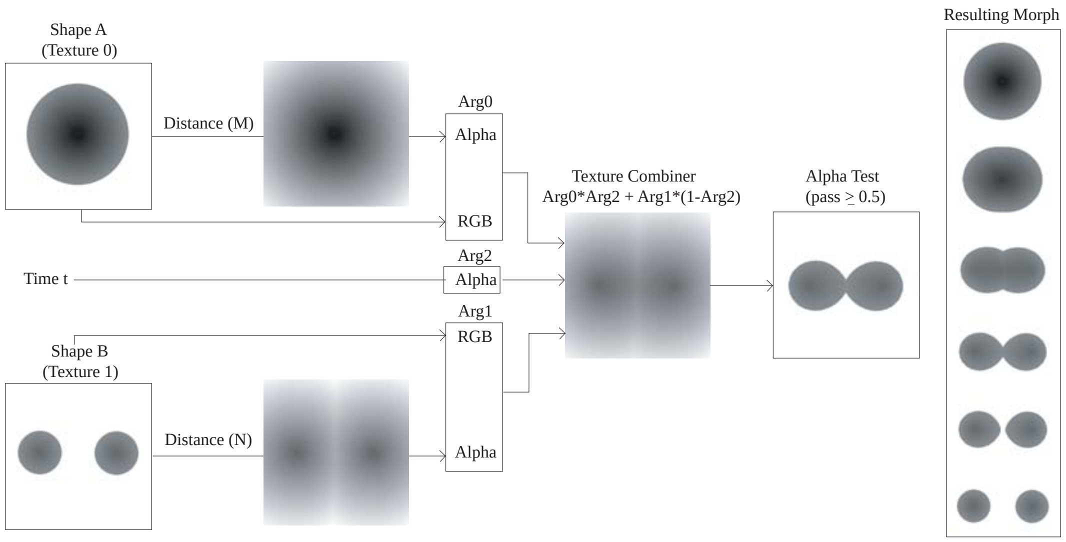

Figure 2: Generating one frame of a shape transformation in real-time using texture combiners and alpha testing (fragment shaders can be used instead). The interpolated distance field at each time interval is shown as the shapes' color. Note that the source shape undergoes a change of topology.

complex shape at interactive frame rates is desirable. Hoff et al. compute distance fields by rendering a 3D *distance* mesh that approximates the distance function [4]. For a point on the 2D plane, the distance on the plane away from the point can be described by an inverted cone whose apex is centered at the point. When the cone is rendered using parallel projection, values in the z-buffer form the distance field. Hoff et al. construct distance meshes for points, lines, and polygons in 2D, and generalize the approach to 3D. Calculation of a 3D distance field requires processing one slice at a time, and is not obtained at interactive rates.

More recently, work has appeared on computing distance fields using fragment shaders. Sigg et al. construct signed distances for triangle meshes within a small band around the surface [12]. Their algorithm samples the distance field on a regular Cartesian grid, but for each triangle, only grid points inside the Voronoi cell of the triangle is sampled. A fragment program performs the nonlinear interpolation to compute the Euclidean distance function and minimization of distance values when several Voronoi polyhedra overlap a sample point. Although their method generates precise Euclidean distances, a narrow band is insufficient for implicit morphing, especially if the surface of source and target shapes are far from each other. Lefohn et al. present a method to compute level sets on the GPU [7]. They evolve a 3D surface at interactive frame rates to allow users to guide the evolution. To obtain a distance field over the entire 2D or 3D domain, the level set must be propagated to the extents of the domain. Because the number of time steps required is data dependent, it is unclear that the distance field can be obtained in the same amount of time for binary shapes of arbitrary complexity. Our image-based method expends the same amount of time for a given resolution regardless of how complex the shape may be. In [13], the authors present a method to compute the distance transform by pre-computing an arbitrarily defined distance function from a point and storing that distance field in a texture. The distance field for a given shape is then generated by rendering the pre-computed texture centered at every boundary point with the blend mode set to GL_MIN. They have not extended their algorithm to 3D, though this may be possible. The complexity of this algorithm is also data-dependent, unlike our image-based method.

In Section 0.5, we describe our image-based method for generating distance fields for binary 2D and 3D shapes of arbitrary complexity. Unlike previously published work, the complexity of our algorithm is dependent only on the resolution of the shapes and not dependent on the complexity of the shapes themselves. We achieve interactive frame rates for calculating complete distance fields of high resolution 2D (512^2) and low resolution 3D (64^3) objects. In addition, our image-based method can quickly update the distance field when a user manipulates the source or target shape in a morph and uncovers regions of the distance field that have not been computed.

0.3 Real-time Implicit Shape Transformation

Our new approach to modeling implicit shape transformations is an interactive framework in which users can manipulate the source and target shapes and immediately see the change in the morph. This approach relies on the texturing hardware to apply and display the transformation. The premise behind our approach is that 2D and 3D binary and implicit shapes can be stored as 2D and 3D textures. We can then manipulate these textures through per-pixel operations and matrix transforms without extracting any explicit iso-surfaces, as inspired by Westermann and Ertl in [14].

In the following sections, we describe the key elements of our interactive system – (1) the real-time computation and visualization of transforming shapes, (2) the ability to immediately see changes reflected in a morph when source and target shapes are manipulated, and (3) the real-time computation of discrete distance fields for shape transformation.

In describing our algorithm, we will use hardware capabilities available in several generations of graphics cards, including the Nvidia GeForce 4, GeForce FX 5900, and the GeForce 6800. We show that our approach can be implemented using the hardware capbilities of the older generation of graphics cards

(GeForce 4) that do not have fragment shading capabilities. This is a benefit of our algorithm because this generation of cards is still quite prevalent on desktops. We use the following functionalities provided by the GeForce 4: *texture combiners*, *texture shaders*, and *register combiners*. Texture Combiners in conjunction with multi-texturing extensions enable textures to be combined through multiplication, signed and unsigned addition, subtraction, and interpolation. Texture shaders, also called *pixel shaders*, provide flexibility for mapping textures by interpreting the color values of one texture as the texture coordinates for another texture. Register combiners enable per-pixel operations such as addition, subtraction, component-wise multiplication, and dot products of color images. They can also perform multiplexing based on the high bit of one of the input alpha channels. In the later generations (GeForce 5900 and 6800), fragment shaders encompass these capabilities while enabling additional texture units, floating point *pbuffers* (non-visible frame buffer), and more GPU progammability in general.

0.3.1 Computing and Visualizing Transformations

A standard method for morphing between two shapes is to cross-dissolve the distance fields of the two shapes, resulting in a linear interpolation between source and target distance fields [11]:

$$d = (1 - t) * dist(A) + t * dist(B); \tag{1}$$

In the above equation, d is the interpolated distance, t is time, $dist(A)$ is the signed distance field for source shape A, and $dist(B)$ is the signed distance field for target shape B. As t is varied, intermediate shapes are extracted where the interpolated distance field evaluates to zero. At starting time, $t = 0$, the resulting shape is completely defined by the source shape A, and at the ending time, $t = 1$, the resulting shape is completely defined by the target shape B.

We have implemented this technique for implicit shape transformation by storing source and target distance fields in the alpha channel of two textures that are interpolated via a third alpha channel that acts as the interpolant. The source and target shapes' color values are stored in the RGB components of the textures. We use hardware-enabled texture combiners to perform the interpolation. The mapping of the source and target shapes and time t to the texture combiner arguments is shown in Figure 2. To display the transforming shape, the textures are mapped to a quadrilateral that is rendered in real-time. The parameter t is mapped to the quadrilateral's alpha value and can be varied in real-time on every rendering pass. Note that this pipeline can also be implemented using the multiplexing operator (OR operator) in register combiners, or using fragment shaders enabled in more recent grahics hardware (e.g. Nvidia GeForce FX). We use texture combiners in Figure 2 to show how our approach can implemented in older hardware that does not have fragment shading capabilities (e.g. Nvidia GeForce 4).

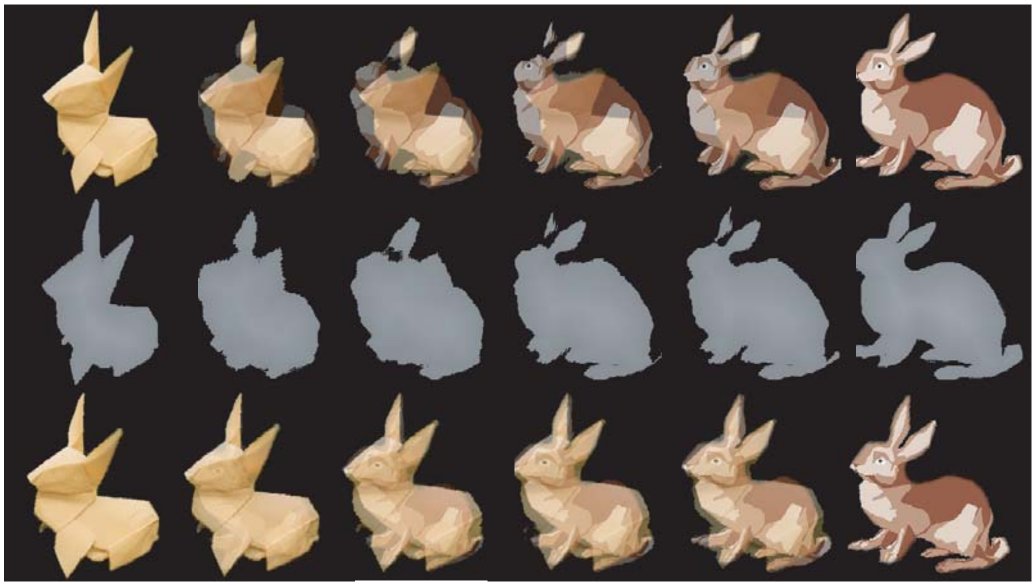

Figure 3: Top: 2D morph between a paper rabbit and a drawn rabbit without warping. Center: Corresponding distance field during the morph shows that undesirable topology changes occur. Bottom: Warping the source and target shapes eliminates the topology changes.

Although recent graphics hardware supports 32-bit floating point storage and computation using pbuffers, the displayed frame buffer is constrained to operate on values in the range [0,1]. Hence in order to display the intermediate slices in our implicit morph, RGB and alpha values are clamped to $[0.0, 1.0]$. We retarget the signed distance field such that negative distances are mapped to alpha values between $[0.0, 0.5)$, and positive distances are mapped to alpha values between $(0.5, 1.0]$. Recall that the intermediate shape is where the interpolated distance field evaluates to zero, which is now mapped to an alpha value of 0.5. The intermediate shape is extracted by alpha testing, passing all pixels with an alpha value equal to 0.5 for a zero level-set. In Figure 2, we show the interior of the shape, not just the contour, by passing all pixels with an alpha value greater than or equal to 0.5. The distance field is used as the RGB component for the source and target textures for illustrative purposes. Normally, the source and target shapes' color values are stored in the RGB components of the textures. Color is thus automatically interpolated as the shape transforms. Figures 1 and 3 are complex 2D examples, showing color changing during a shape transformation.

0.3.2 Extending to 3D

In 3D, the iso-surfaces displayed during the morphing are generated using the approach introduced by Westermann and Ertl [14]. Their technique interactively displays iso-surfaces without explicitly extracting an iso-surface. This is done using 3D texture mapping and pixel transfer operations to render the iso-surface on a per-pixel basis. Thus, only the pixels that are closer to the viewer (i.e., that pass the z-buffer test) and whose values are above the iso-value are rendered.

The two 3D textures used during the morphing process store the distance fields in their alpha component and the normals (computed using the distance fields) in their RGB components. Register combiners or fragment shaders are used to compute the dot products between each voxel normal and the lighting directions (for both textures), followed by the interpolation of the two 3D textures using using the method we previously described for 2D shapes. Both the normals and distance fields are interpolated. After interpolation, a multiplex operation is performed to pass only voxels whose interpolated values are greater than 0.5. Figure 4 shows a 3D shape transformation from a zebra to an elephant Intermediate shapes were not precomputed in these visualizations.

0.4 Manipulating the Shape Transformation

The intermediate shapes in an implicit shape transformation may be unsatisfactory. For example, undesirable topology changes may occur resulting in extra components forming and disappearing. In the top and center rows of Figure 3 one ear of the paper rabbit disconnects and then reconnects to form an ear of the drawn rabbit. The second ear of the drawn rabbit is an extra component that forms and connects to the rabbit's head. A 3D example is shown in Figure 4 where the legs of the zebra disconnect into separate components during the transition to the elephant.

When the intermediate shapes are unsatisfactory, the transformation can be modified by changing the initial conditions of the source and target shapes. Initial conditions include the size, position and orientation of the shapes, their relative proximity to each other, and any non-linear warps that are applied to the source and target shapes for better alignment. In our framework, a user may modify these initial conditions and immediately visualize the changes reflected in the morph. In this section, we describe affine and non-linear transformations that may be applied to source and target shapes in our interactive framework.

Many implicit methods for shape transformation include a user-defined step for alignment of the source and target shapes through translation, rotation, and scaling. In Figure 1, we show the effects of the shape transformation as the target shape is scaled, translated, and rotated in real-time. Figure 4 shows a 3D shape transformation from a zebra to an elephant with and without spatial alignment. An optimal alignment to line-up the legs of the source and target

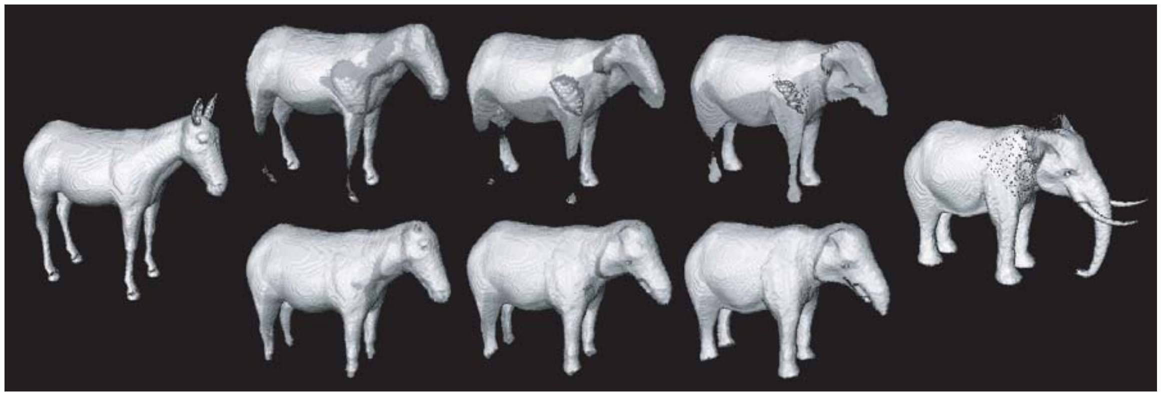

Figure 4: 3D implicit morph computed in real-time. Source and target volumes are 256^3. Top: Original misalignment of zebra and elephant (see Figure 5) causes extra components to form during the morph. Bottom: Better alignment of source and target eliminates unwanted topology change during the morph. The speckle present on the target elephant is due to undersampling of the flat ears when the volumetric model was generated and is not a side-effect of our implicit morph.

shapes was made possible by showing both shapes in one visualization, with one shape in a different color (see Figure 5). As in the 2D example, the intermediate slices of the morph change in realtime as the alignment is modified by the user.

Techniques for non-linear warping [2] can be applied to source and target shapes in our framework via pixel (or dependent) textures in a similar manner as in [15, 5]. The shape and distance field textures are dependent upon another texture in that they are texture mapped according to the coordinates stored in the third texture. The texture coordinates stored in a pixel texture need not be linearly related and can instead store non-linear warps. In our system, users can define a thin-plate warping function by specifying corresponding features points between source and target shapes, as described in [2]. Pixel textures describing the warps are generated by evaluating the warping function for each pixel, or voxel, and storing the warped texture coordinates in the pixel texture. When the shape textures and corresponding distance fields are mapped to polygons for rendering the morph, they are first warped by the pixel texture. Warped texture coordinates that index locations outside of where the distance field exists are clamped to the boundaries. If regions are uncovered where the distance field does not exist, the warped distance field can be updated in one pass using the image-based approach we describe next. An example of 2D warping is shown in Figure 3. Examples of warped 3D shapes is shown in Figure 6.

An even more direct mechanism for manipulating the implicit shape transformation is to edit source and target shapes using standard techniques found

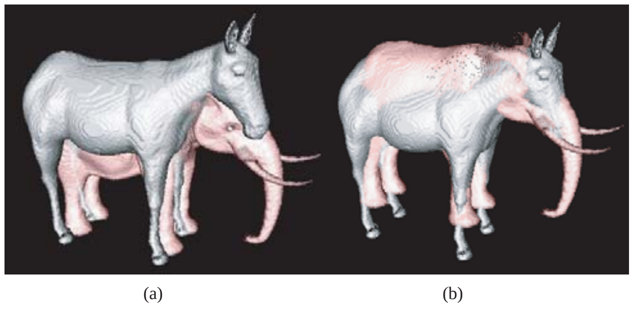

Figure 5: (a) Original misalignment of zebra and elephant (pink). (b) Better alignment of source and target.

in 2D paint and 3D sculpting programs. Our framework is capable of incorporating these changes into the shape transformation through updates on the distance field. Inclusion of such operators in our framework has been left to future work.

0.5 Computing Distance Fields on the GPU

In order to immediately see the effects an edit imposes on the shape transformation, distance fields used to construct the implicit morphs must be calculated at interactive frame rates. In the following section, we show how this calculation can be performd at interactive frame rates for 2D shapes of up 512^2 and and 3D shapes of 64^3 using a simpler form of the distance field. To enable interactivity when manipulating larger 2D and 3D shapes, we use an image-based method that enables updates to be made to the distance field in less time than recalculating the entire distance field in software. For large 2D and 3D shapes, our system calculates an initial distance field when the source and target shapes are loaded and then achieves interactive frame rates while the source or target shape is manipulated by updating the distance field rather than recalculating it.

Distance fields have primarily been computed in software. Recent work in calculating distance fields using graphics hardware have been for polygonal models [4, 12], are data-dependent methods [7, 13], or generate only a narrow-band of distance values around the surface [12]. For these reasons, these methods cannot be applied to arbitrarily complex implicit shapes. We describe a new method to handle discrete shapes that are represented by a binary image or volume. We use the convention that 0 indicates existence of the object, and 1 is free space. Note that this algorithm can be applied to any implicit shape as long as the shape is first sampled into a discrete binary representation. Once we have a discrete representation for the shape, it can be loaded as a texture

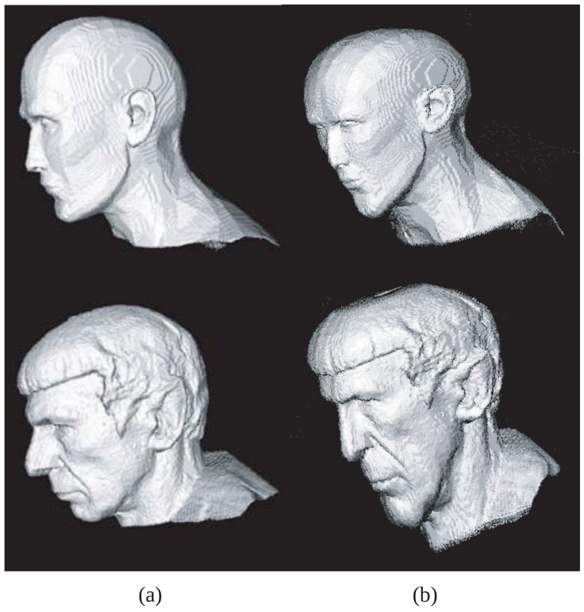
Figure 6: Warping of source and target volumes (256^3). The forehead of the generic head (top) compresses to match Spock's head, while Spock's jaw (bottom) elongates to match the generic head.

into the graphics hardware.

Standard methods for computing a discrete distance field for an object represented as a binary image are scan algorithms, such as Danielsson's Four Point Sequential Euclidean Distance (4SED) algorithm [3]. A good review of scan algorithms can be found in [10]. Scan algorithms operate by propagating distance values or vectors across the binary image in raster-scan order or by ordered propagation from shape contours. For example, the 4SED algorithm computes the distance transform for each pixel $p_{i,j}$ of the image by adding "1" distance unit to the pixels to the right $p_{i+1,j}$, the left $p_{i-1,j}$, above $p_{i,j+1}$ and below $p_{i,j-1}$ in two passes. The Euclidean distance $d_{i,j}$ from pixel $p_{i,j}$ to the shape boundary is the minimum of the distances $d_{i+1,j}+1$, $d_{i-1,j}+1$, $d_{i,j+1}+1$, and $d_{i,j-1}+1$. Complete forward and complete backward passes along the entire width and height of the image are necessary. In order to obtain a signed distance, the binary shape is inverted, and the same passes are applied to the inverted shape.

The first hardware-enabled approach we describe is a multi-pass algorithm that also traverses the image in several passes in a similar manner to scan algorithms for computing the distance field. Instead of propagating single values from one pixel to the next in raster-scan order, however, an entire image can be propagated at once by shifting the texture storing the binary shape by one pixel in the scan direction (left, right, up, or down). In each pass of this initial algorithm, the binary shape image is shifted one pixel in one of the four directions; each pixel of the shifted image is incremented by a distance value of "1"; and the results are compared to the image from the last pass. For each pixel, the minimum value between the two images (current pass and last pass) is kept and stored for the next pass. As with Danielsson's algorithm, complete forward and complete backward passes along the entire width and height of the image are required, and the same number of scans must be applied to the inverted binary image in order to obtain a signed distance. Figure 7 is a diagram of one pass of the algorithm in which the shape image is shifted to the right. The algorithm can be implemented using pixel textures and register combiners in older graphics cards, or using fragment shaders in more recent cards (e.g. Nvidia GeForce FX). With fragment shading, we increment and compare each pixel with its left, right, top, and bottom neighbors in a fragment program that is executed for every pixel in the frame buffer. Because we are shifting and incrementing whole images, our hardware-enabled method for calculating the distance field is image-based. Later, in Section 0.5.1, we further exploit the parallelism of the graphics hardware to make our image-based algorithm more efficient.

Note that the distance field that we generate is not Euclidean (L_2 norm), but rather, it is the *min-norm* defined as:

$$dist(p^1, p^2) = min_i |p_i^1 - p_i^2| \qquad i = 1, 2, ..., n \tag{2}$$

In the above equation $dist(p^1, p^2)$ is the distance between two points p^1 and p^2; i goes from 1 to dimension n; $n = 2$ for 2D images; and $n = 3$ for 3D volumes. We believe that the min-norm is sufficiently accurate for visualizing shape transformations.

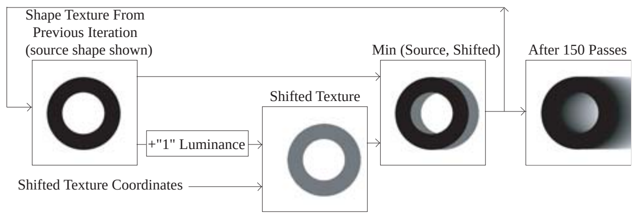

Figure 7: One pass of the distance field calculation in hardware. The texture shift (implemented using pixel textures or fragment shaders) and added luminance have been exagerated to show the difference between the input and shifted images. Far right: distance field generated after 150 passes.

A single pass of our hardware implementation is a shift of the entire shape image to the right, left, up, or down of one pixel. The total number of passes in our initial approach is $2 * width + 2 * height$ for an unsigned distance (double that

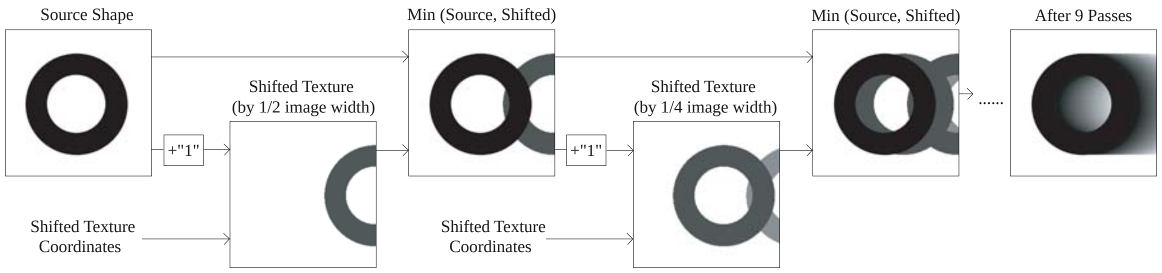
Source Shape
Min (Source, Shifted)
Min (Source, Shifted)
After 9 Passes
Shifted Texture (by 1/2 image width)
Shifted Texture (by 1/4 image width)
+"1"
+"1"
......
Shifted Texture Coordinates
Shifted Texture Coordinates

Figure 8: Two passes of our improved method to calculate distance fields using graphics hardware. After the second pass, the computed distance field contains more distance values than two passes of our earlier algorithm. Only 9 shifts are needed to generate the forward horizontal pass of the distance field.

for a signed distance). In order to obtain a signed distance, the binary shape image is inverted, and the same number of passes is applied to the inverted image. For a 256^2 image, a max of 2048 passes are required to generate a signed distance function. Using either pixel textures in older graphics cards or fragment shaders in newer cards, left and right horizontal scans can be combined into one scan (and both vertical scans can be combined into one), reducing the number of passes by half to 1024.

In order to compare the image of the current pass with the image from the last pass, the last pass must be stored in a texture, requiring a copy (or rendering) to texture memory. Rendering to a texture remains expensive in current graphics hardware, and is the bottleneck in our initial hardware-enabled approach. An algorithm requiring 1024 renders to texture, such as ours, cannot achieve interactivity. We have found the average time to be 1.0 seconds for images of 256^2 resolution, and 3.9 seconds for images of 512^2 resolution. Our solution is to reduce the number passes (and thus, the number of renders to texture) as described next.

0.5.1 Achieving Real-time Distance Field Updates

We can reduce the number of passes by further exploiting the parallelism of the graphics hardware in processing fragments. In a distance field, the distance stored in a pixel that is half of an image width away from the shape boundary should be equal to half the maximum distance ($w/2$, where w is the image width). In our implementation, the maximum distance is equal to the image width or height. Similarly, if a pixel is half of an image width away from another pixel that has a distance value of d, then our pixel of interest should take on a value of $d + w/2$. Hence, for each pixel p_i that is $w/2$ away from some other pixel with a distance value of d_i, the distance value for p_i is:

$$dist(p_i) = d_i + w/2 \tag{3}$$

If we have half of the distance field already computed up to a distance of $w/2$, we can shift the distance field by $w/2$, increment the distance values by $w/2$, and combine this with our original, unshifted, unmodified distance field. The shifted and modified distance field spans from $w/2$ to w. Combining the two half-width distance fields would result in a complete distance field. The key difference is that we have now done only $w/2 + 1$ shifts rather than w shifts. Since each shift is a single pass in our hardware-enabled approach, this is a significant saving.

This shift, increment, and combine procedure can be applied to distance fields that have values up to $w/4$ to generate a distance field with values up to $w/2$. The procedure can also be applied to distance fields that have values up to $w/8$ to generate a distance field with values up to $w/4$. We can continue to apply this technique to halved distance fields until we are generating distances that are just one pixel away from the shape boundary and take on a value of $1/w$. By successively combining smaller distance fields to form larger ones, we can complete one full horizontal pass (spanning the entire width) in $log_2 w$ shifts as opposed to w shifts as required by our first approach. We still need to complete one horizontal pass (that combines left and right shifts) and one vertical pass (that combines up and down shifts) to generate an unsigned distance field. The total number of shifts is $log_2 w + log_2 h$ as opposed to $w + h$. For a 256^2 image, only 16 passes are needed with our new technique for the unsigned distance (32 passes for the signed distance). Our new hardware-enabled distance field calculation can now be achieved at interactive frame rates due to the savings in the number of required passes and textures to render. We have found the average time to calculate a signed distance field to be 0.1 seconds (10 Hz) for images of 256^2 resolution, and 0.35 seconds (2.9 Hz) for images of 512^2 resolution (with 9 shifts per direction). The frame rates we obtain were on the Nvidia GeForce FX 5900 graphics card. Figure 8 is a diagram of this more efficient multi-pass algorithm.

The primary advantage of our image-based approach to computing the min-norm is not the speed in computing a full distance field, however, because a software implementation of the min-norm can compute the full distance field in less time (see Table 1). The key advantage of our image-based approach is that it can quickly update a partially computed distance field. When an affine transformation or non-linear warp is applied to the source or target shapes, regions may be uncovered where the distance field does not exist, as shown in Figure 9. The distance field is then updated by shifting and incrementing the current distance field by some power of 2 such that the updated distances propagate across the uncovered region. We have found that only a single shift and increment pass is needed to fill in uncovered regions as a user translates or rotates the source or target shapes. A single pass of our image-based approach to updating the distance field is faster than recomputing the entire distance field in software. Since the software algorithm cannot determine which parts of the distance field needs to be updated without first traversing those regions, complete raster-scans are required in the software implementation even though only part of the distance field needs to be computed. Note that shifting and incrementing the *signed* distance field will not produce correct results because distance values need to be incremented in regions outside of the shape and decremented in regions inside the shape. Hence, it is necessary to store and update the two regions of the unsigned distance field separately prior to combining them.

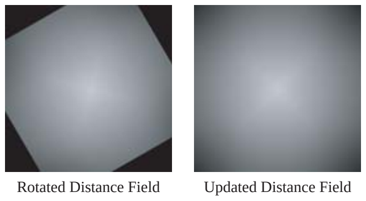

Figure 9: Left: Rotating the shape after the distance field has been computed uncovers regions where the distance field does not exist. Right: The distance field is updated after an affine transformation using our image-based approach.

Updating the discrete distance field in hardware is essential to our interactive shape transformation system because this allows the source and target shape images and corresponding distance fields to remain in texture memory as they are being modified or calculated. A software distance field implementation would require updating the distance field in main memory every time a user modifies the shape image. The updated distance field would then need to be loaded back into texture memory.

0.5.2 Extending to 3D Distance Fields

Extending our above algorithm to 3D shapes requires calculating the 2D distance function for each slice along the depth axis in the volume. Several slices cannot be computed at once in the same manner that several shifts to the left/right/above/below were computed in the 2D distance field. The number of passes required along the depth axis is equal to the depth d, not $log_2 d$, because rendering intermediate results to textures can only be applied to 2D textures (*glCopyTexSubImage* applies only to 2D, not 3D, textures). Simply applying the 2D distance field calculation to each slice in the volume would not give an accurate distance field along the depth axis. Instead, we calculate the 3D distance field by first rotating the volume about the vertical axis by 90 degrees so that the width is now equal to d. Distance values are then computed horizontally for each slice of the volume as previously described, requiring $log_2 d$ passes for each slice. There are now w slices due to the 90 degree rotation. The resulting distance values, when rotated back by -90 degrees, are distances along the depth axis. The unsigned 3D distance field is generated by computing distance values along the width and height of the volume for each unrotated slice along the depth axis as described in the previous section. The full pipeline is shown in Figure 10. The total number of rendering passes required for computing the unsigned 3D distance field is: $d * (log_2 w + log_2 h) + w * (log_2 d)$. For a signed distance field, the shape is inverted, the same number of passes are applied, and negative and positive distance fields are combined. For a volume of 64^3, this amounts to 1152 passes. We have found the average time to calculate a signed distance field to be 0.5 seconds (2 Hz) for volumes of 64^3 resolution

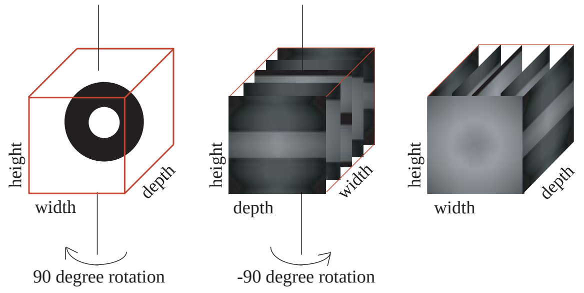

Figure 10: The 3D shape is rotated for calculating the distance along the depth, and rotated back for distance along the width and height.

0.6 Results

Table 1 shows timing results for different resolutions of 2D and 3D shapes using the Nvidia GeForce FX 5900. We have recorded the time for computing and rendering morphs using texturing hardware which achieves real-time frame rates for all resolutions. As expected, the timing shows that updating the distance field takes less time than the initial calculation. Dependent textures are created for warping. Note that the time recorded for creating dependent textures is the evaluation time for a warp function consisting of one pair of corresponding points between the shapes. The actual time is highly dependent on the type of warp (thin-plate or Gaussian) and number of corresponding points between the source and target shapes.

0.7 Conclusions

We have presented an interactive technique to construct implicit shape transformations using texturing hardware. We have also discussed the implementation

Dim.	Software Distance	Hardware Distance	Compute & Render Morph	Distance Update	Dependent Texture
256^2	0.016	0.078	< 0.01	0.016	0.18
512^2	0.062	0.36	< 0.01	0.062	0.60
64^3	0.172	0.51	< 0.01	0.164	0.03
128^3	1.89	3.46	< 0.01	1.046	0.22
256^3	17.45	29.21	0.02	7.75	1.90

Table 1: Timing Results (in secs.)

of our method on older and newer generations of graphics cards with differing capabilities. Our approach allows users to interactively manipulate source and target shapes as the shape transformation is computed and displayed. The method relies on a new image-based technique to update discrete distance fields that is faster than a software implementation. As with many image-based approaches, ours can handle complex shapes of arbitrary complexity. Because our framework creates an implicit shape transformation, it is capable of generating plausible transitions between shapes of differing topology. The immediate feedback helps a user to visualize and control how the shape morphs and how topology changes in a transformation sequence.

Bibliography

[1] Cabral, B., N. Cam, and J. Foran. "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware", *Proceedings of ACM Symposium On Volume Visualization*, 1994, pp.91–98.

[2] Cohen-Or, Daniel, David Levin and Amira Solomovici. "Three Dimensional Distance Field Metamorphosis", *ACM Transactions on Graphics*, 1998, Vol.17, pp.116–141.

[3] Danielsson, P. "Euclidean Distance Mapping", *Computer Graphics and Image Processing*, 1980, Vol 14, pp.227-248.

[4] Hoff, K.E., T. Culver, J. Keyser, M. Lin, and D. Manocha. "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware", *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 99), August 1999, pp277–285.

[5] Kniss, J., S. Premoze, C. Hansen, and D. Ebert. "Interactive Translucent Volume Rendering and Procedural Modeling", *Proceedings of IEEE Visualization*, October 2002, pp.109–116.

[6] Lazarus, F. and A. Verroust, "Three-dimensional Metamorphosis: a Survey," *The Visual Computer*, Vol.14(8/9), December 1998, pp. 373–389.

[7] Lefohn, A.E., J.M. Kniss, C.D. Hansen, and R.T. Whitaker. "Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware" *Proceedings of IEEE Visualization*, October 2003, pp.75–82.

[8] Lerios, A., C.D. Garfinkle, and M. Levoy. "Feature-based Volume Metamorphosis", *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 95), July 1995, pp. 449–456.

[9] Lorensen, W.E. and H.E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm", *Computer Graphics Proceedings of SIGGRAPH 87*, July 1987, pp.163–169.

[10] Maurer, C.R. "A Linear Time Algorithm for Computing Exact Euclidean Distance Transform of Binary Images in Arbitrary Dimensions", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003, Vol.25(2), pp.265–270.

[11] Payne, B.A. and A.W. Toga. "Distance Field Manipulation of Surface Models", *IEEE Computer Graphics and Applications*, January 1992, Vol.12(1), pp.65–71.

[12] Sigg, C., R. Peikert, and M. Gross. "Signed Distance Transform Using Graphics Hardware", *Proceedings of Visualization '03*, 2003.

[13] Strzodka, T. "Generalized Distance Transforms and Skeletons in Graphics Hardware", *Proceedings of Symposium on Visualization*, 2004.

[14] Westermann, R. and T. Ertl. "Efficiently Using Graphics Hardware in Volume Rendering Applications", *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 98) July 1998, pp.169–177.

[15] Westermann, R. and Rezk-Salama. "Real-Time Volume Deformations", *Proceedings of Eurographics 2001*, 2001.