Morphology-Based Three-Dimensional Interpolation

Tong-Yee Lee* and Wen-Hsiu Wang

***Abstract*—In many medical applications, the number of available two-dimensional (2-D) images is always insufficient. Therefore, the three-dimensional (3-D) reconstruction must be accomplished by appropriate interpolation methods to fill gaps between available image slices. In this paper, we propose a morphology-based algorithm to interpolate the missing data. The proposed algorithm consists of several steps. First, the object or hole contours are extracted using conventional image-processing techniques. Second, the object or hole matching issue is evaluated. Prior to interpolation, the centroids of the objects are aligned. Next, we employ a dilation operator to transform digital images into distance maps and we correct the distance maps if required. Finally, we utilize an erosion operator to accomplish the interpolation. Furthermore, if multiple objects or holes are interpolated, we blend them together to complete the algorithm. We experimentally evaluate the proposed method against various synthesized cases reported in the literature. Experimental results show that the proposed method is able to handle general object interpolation effectively.**

***Index Terms*—Blending, dilation and erosion, distance-maps, interpolation, morphology, object centralization.**

I. Introduction

TODAY, clinicians exploit computer graphics tools to enable them to visualize, manipulate, and quantitate the three-dimensional internal structures of patients. Major sources of data in these medical applications are gathered from two-dimensional (2-D) medical-imaging devices such as CT, MRI and PET. A three-dimensional image, formed by stacking a contiguous series of 2-D images, can be used to visualize complex structures in three-dimensional (3-D). However, generally, the number of image slices generated from these instruments are not adequate enough to produce high-quality 3-D images. Therefore, in such situations, interpolation is always required to estimate the missing slices before the subsequent visualization.

In this paper, we utilize dilation and erosion operators in morphology to perform interpolation. In the past, a variety of approaches have been proposed to reconstruct 3-D objects. Here, we only review the most related prior work. Among this work, the simplest method is to linearly interpolate the gray values in the slices to fill in the gray values in the missing slices [1]–[6]. With this scheme, an artifact always arises when the location of a boundary between two uniform regions shifts considerably between two adjacent slices. Keys *et al.* [7] attempted to exploit higher order functions to reduce artifacts. To handle branching situations, a dynamically elastic surface interpolation scheme was proposed in [8]. The key concept is to identify a force acting on one contour and try to distort it to be like the other contour. However, the resulting surface may be coarse if there is high dissimilarity among these contours [9]. To alleviate this problem, a hybrid approach combining elastic interpolation, spline theory and a surface consistency theorem was proposed in [9] for constructing a smooth 3-D object. The above two methods [8], [9] have very high computational complexity.

Raya *et al.* [10] and Herman *et al.* [11] exploited the concept of a distance transform to interpolate binary-valued 3-D images. This widely used method is termed shape-based interpolation. Compared to [8], [9], this type of algorithm is considerably simpler in practical implementation and is very inexpensive in computational complexity. However, this method fails to interpolate the slices when there is no overlapping area between the two objects. To overcome this drawback, Guo *et al.* [12] developed a morphology-based interpolation method. In this method, only the nonoverlapping regions are interpolated using a sequence of dilation and erosion operations. Our proposed scheme is similar to this approach. However, their approach computed normal vectors to control disk-like morphology operators. In contrast, our proposed scheme is simpler in computational complexity, but performs as well as [12]. In some cases, such as invagination and branching, the proposed method performs better than [12].

The remainder of this paper is organized as follows. We present the proposed methodology in Section II. The algorithms and implementation details are described in Section III. In Section IV, experimental results and discussion are given. Finally, the conclusion and future work is stated in Section V.

II. Morphology-Based Interpolation

In this section, we will present the idea of the proposed interpolation scheme. First, we will consider one-to-one object interpolation and we extend this one-to-one example to handle more complicated cases. The more algorithmic details will be given in Section II-A.

A. One-to-One Object Interpolation

Shape-based interpolation converts binary images into distance maps by distance transformation functions such as chamfer or city-block distance template [8], [10], [11]. These templates are used to efficiently approximate the *Euclidean shortest distance* between the pixel and the contour of the object. Unfortunately, if there is no *prior* alignment between two input images, shape-based interpolation cannot perform well. For example, in Fig. 1(a) and (b), there are two contours

Manuscript received November 9, 1999; revised May 15, 2000. This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 89-2213-E-006-067. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was J. Liang. *Asterisk indicates corresponding author*.

*T.-Y. Lee and W.-H. Wang are with the Visual System Lab., Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, R.O.C.

Publisher Item Identifier S 0278-0062(00)07326-2.

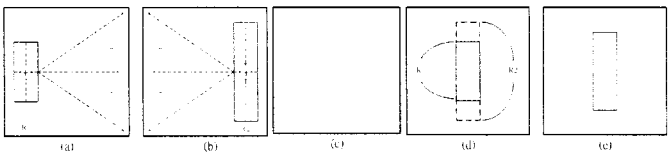

Fig. 1. Shape-based interpolation and object centralization.

denoted as R_1 and R_2 on two binary images. Without an appropriate alignment, the shape-based scheme creates a bad interpolation as illustrated in Fig. 1(c), where there is no contour in this interpolated image. In shape-based scheme, the distance stored on a pixel, say P, is defined by

$$\text{dist\_code}(P) = \begin{cases} 0, & \text{/* pixel on contour} \\ +distance(P, \partial X), & \text{/* pixel in object} \\ -distance(P, \partial X), & \text{/* pixel outside object} \end{cases} \quad (1)$$

where

X	represents the object;
∂X	represents the contour of the object X;
dist_code(P)	shortest distance from P to ∂X.

Observing Fig. 1(a) and (b), we know the interpolated distance codes all will be negative on the interpolated image in Fig. 1(c). Therefore, there is no object on it according to (1). On the other hand, if we perform an appropriate alignment so as to match the centroids of two objects *prior* to distance transformation [as shown in Fig. 1(d)], we can obtain a better interpolation as shown in Fig. 1(e). Therefore, we will perform this kind of alignment which is termed *object centralization* in this paper.

Morphology-based interpolation can be schematized as shown in Fig. 2. First, we align two corresponding objects X_0 and X_{n+1} using *object centralization*. After this alignment, there are three kinds of possible portions: region I, II, and III, respectively. Regions I and II represent the morphological difference between the two objects X_0 and X_{n+1}. Then, we apply a *dilation* operator to both regions I and II, respectively. The purpose of this step is to obtain the dilation-based distance from the pixel P (i.e., on region I or II) to the boundary of region III. After this, we can apply an *erosion* operator to interpolate the results. More details about these two operators are provided in Section III.

During interpolation, each pixel in both regions I and II will gradually move toward region III. The number of erosions is determined by an *erosion factor*, denoted as E. Assume we want to interpolate n slices between two objects, then the *erosion factor* for the kth slice is determined by

$$E_k^n = \frac{k}{n+1} \qquad 1 \le k \le n \quad (2)$$
$$E_k^{n^*} = 1 - \frac{k}{n+1}$$

where E_k^n is an *erosion factor* for region I and $E_k^{n^*}$ is the factor for region II. The interpolated object X_k for the kth slice is defined by

$$X_k = \text{Erosion}\,(X_0, E_k^n) + \text{Erosion}\left(X_{n+1}, E_k^{n^*}\right) = \text{Erosion}\,(\text{I}, E_k^n) + \text{Erosion}\left(\text{II}, E_k^{n^*}\right) + \text{III}. \quad (3)$$

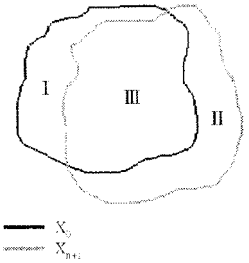

Fig. 2. Morphology difference between X_0 and X_{n+1} after object centralization.

The Erosion() performs exact interpolation and it is accomplished by an erosion operator. For the X_k, we apply Erosion() to both regions I and II with erosion factors E_k^n and $E_k^{n^*}$, respectively, and then we combine them with region III to obtain the object X_k.

Prior to employing our proposed interpolation scheme, several preprocessing steps must be performed such as contour extraction and hole identification. These topics are well researched and are beyond the scope of this paper. In the following discussion, we assume that the above preprocessing steps are finished and provide each corresponding object pair such as X_0 and X_{n+1}.

B. Pseudoobject Generation

Hollows or holes may occur in the image slices. We treat holes and objects separately. In other words, we will perform hole-to-hole interpolation the same way described in Section II-A for object-to-object interpolation, but in a separate step. We will treat holes as negative objects and nonhole objects as positive objects. For any hollow pair, there must exist a corresponding positive object pair. Sometimes we will have one hollow inside one object, but there is no corresponding hollow inside the other object. In such a situation, we need to produce a pseudohole; in order to create a hole pair. Similarly, if we have a positive object on a slice and there is no corresponding positive object on the other slice, we will create a pseudopositive object.

To generate a pseudopositive object is very straightforward. Assume we have a positive object and its center is at C_{centroid}. Then, on the other slice, we create a corresponding pseudopositive object at C_{centroid} with a size of one pixel. For a pseudonegative object, there is some extra work described as follows. In Fig. 3, there are two corresponding objects and their centroids are O_1 and O_2. The two objects are bounded by boxes $A_1B_1C_1D_1$ and $A_2B_2C_2D_2$. Each box is further divided into four subregions (A_iO_i, B_iO_i, C_iO_i, D_iO_i) based on O_i. Assume on the first object, we have a hole and its centroid is C_{centroid}. In Fig. 3(a), we suppose C_{centroid} is located in D_1O_1 (denoted as region$_1$). Then, we will create a pseudohole at D_2O_2 (denoted as region$_2$) in Fig. 3(b), too. The size of this pseudohole is one pixel and its centroid, C'_{centroid} can be computed in

$$\overrightarrow{V_{\text{oc2}}}.x = \overrightarrow{V_{\text{oc1}}}.x \cdot \frac{\text{length of region}_2}{\text{length of region}_1}$$
$$\overrightarrow{V_{\text{oc2}}}.y = \overrightarrow{V_{\text{oc1}}}.y \cdot \frac{\text{length of region}_2}{\text{length of region}_1} \quad (4)$$

$$C'_{\text{centroid}} = O_2 + \overrightarrow{V_{\text{oc2}}}\,. \quad (5)$$

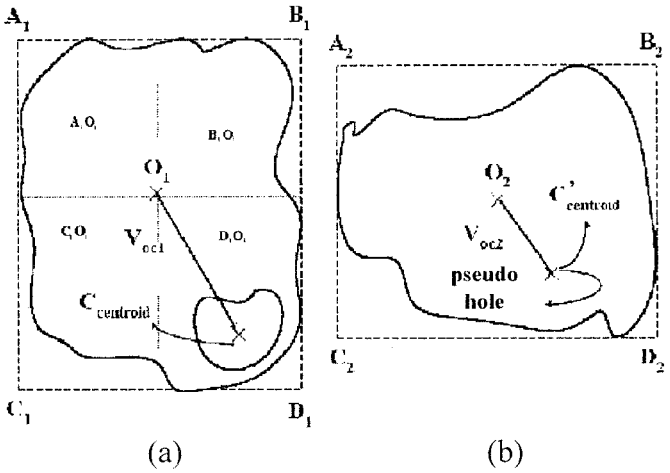

Fig. 3. Pseudohole generation.

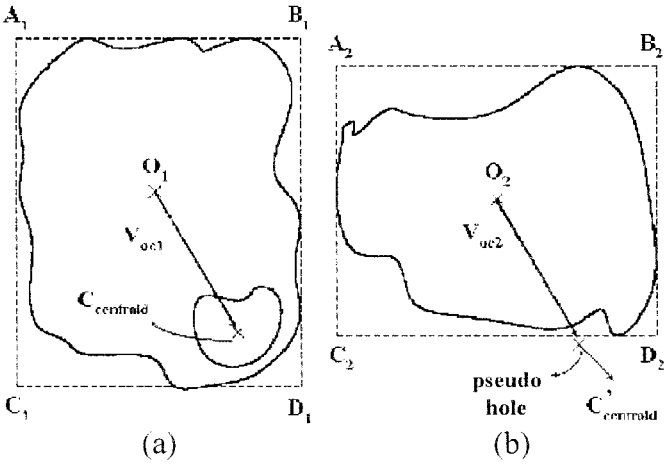

Fig. 4. Example of incorrect pseudohole generation.

In (4) and (5), V_{oc1} is a vector from O_1 to C_{centroid}. If, instead, we tried to compute C'_{centroid} directly by $O_2 + V_{\text{oc1}}$, it could incorrectly create a pseudohole outside the object (as shown in Fig. 4).

Finally, we should mention that in the pseudoobject generation, there would be problems if the holes or objects in a slice pair do not match. There could be a case where each slice has one hole. The proposed algorithm would join them, but these two holes could really be just axial concavities, one ending at slice 0 and one starting at slice $n+1$. However, for such special cases, there is nothing one can do about this aliasing problem. For these cases, the proposed matching algorithm could fail here to detect the mismatch.

C. Multiple Object Matching

Multiple objects may exist on two input slices. In this situation, we need to solve matching problem. In this paper, we do not concentrate on this issue, but we provide a simple rule for it described as follows. First, for each potential object pair (a, b), we will evaluate a score of matching given by (6). Then, if this score is higher than a threshold, we say this object pair is matched.

$$\text{score} = \text{overlap}(a, b) * w_1 + (\text{dist}_{\text{threshold}} - \text{dist}(a, b)) * w_2. \quad (6)$$

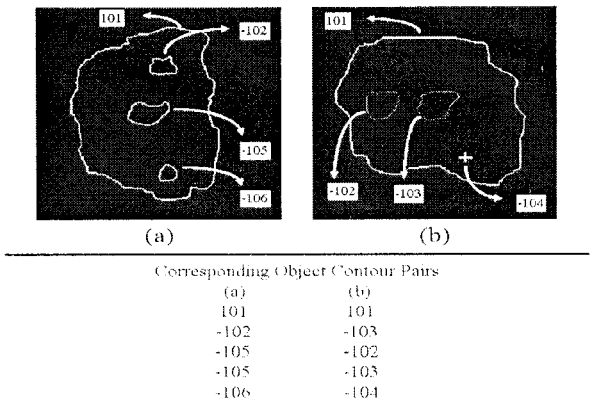

Fig. 5. Object matching.

TABLE I
THE MATCHING SCORE FOR EACH OBJECT PAIR IS SHOWN. THE POSITIVE AND NEGATIVE OBJECTS CANNOT BE MATCHED, SO WE SHOW "*" IN THIS TABLE

Object (a) \ Object (b)	+101	-102	-103
+101	459	*	*
-102	*	-39	31
-105	*	17	153
-106	*	-80	-10

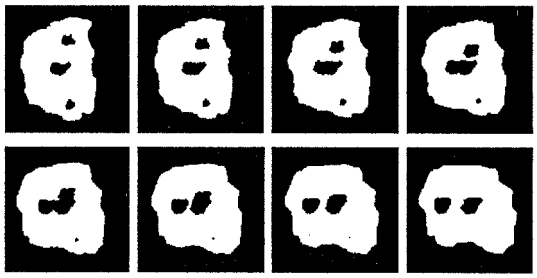

Fig. 6. An example of interpolation using our matching policy.

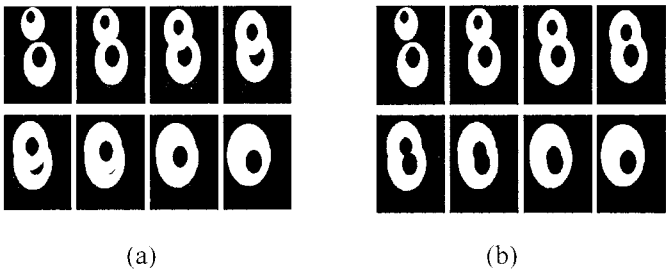

Fig. 7. (a) Bad interpolated results without following (8). (b) Better interpolation results, if we follow the *blending order* in (8).

In (6), we take two factors into account: 1) object pair (a, b) is overlapped or not (i.e., returns 1 or 0), and 2) the distance between two object centers is within a range or not. Both w_1 and w_2 are user-specified weights to compute (6). For a given object pair, we select the larger one and find the *width* and *length* of its bounding box. Then, we let $\text{dist}_{\text{threshold}}$ be the sum of *width* and *length*. In Fig. 5, we show two slices (a) and (b). Using this simple rule, we obtain the following matched object pairs. For clear illustration, we assign a positive identifier to a

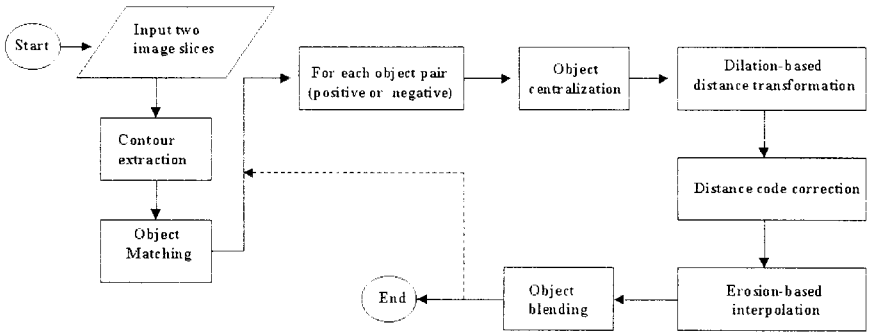

Fig. 8. The flowchart of the proposed scheme.

positive object and a negative one to a hole. In this experiment, $w_1 = 100$, $w_2 = 1$ and the score threshold is 0. Table I shows detailed matching scores under this configuration.

From Table I, the contour (−105) in (a) forms corresponding contour pairs with both (−102) and (−103) in (b). Therefore, the list in Fig. 5(a) has (−105) twice. Negative scores such as pairs in (−102, −102), (−106, −102) and (−106, −103) mean these pairs can not be matched using (6). In Fig. 5(b), a pseudohole is marked by "+." Since in (a) there is no matching for the hole (−106), we must create a pseudohole termed (−104) in (b).

Next, in Fig. 6, we show interpolated results for the above example using our simple policy. We see three holes (vertically) gradually become two holes (horizontally). For more information about interpolation, see Section II-D.

D. Object Blending

In Section II-A, we propose to interpolate objects after object centralization. Assume that we have two objects X_0 and X_{n+1} and their centroids are O_0 and O_{n+1}, respectively. After interpolation, we compensate for the effects of object centralization by translating the interpolated object X_k^n back to the correct position, computing the new centroid of X_k^n, O_k^n, according to

$$\mathrm{O}_k^n = \mathrm{O}_0 * E_k^{n^*} + \mathrm{O}_{n+1} * E_k^n \tag{7}$$

As mentioned earlier, we separately interpolate positive object pairs and negative object pairs. Afterwards, we combine them using

$$\begin{cases} X_R^+ = \bigcup X_i, & \text{if } X_i \in \text{positive object} \\ X_R^- = \bigcup X_i, & \text{if } X_i \in \text{negative object} \\ X_R = X_R^+ - X_R^-. \end{cases} \tag{8}$$

In (8), objects X_R^+ and X_R^- are blended results for the positive and negative objects, respectively. This equation defines the *blending order*: we blend all positive and negative objects separately and then subtract X_R^- from X_R^+. Fig. 7(a) and (b) show results without and with being guided by (8).

III. Algorithm Details

Fig. 8 shows the flowchart of the proposed method, and in this section, we will provide more algorithmic details about dilation, erosion, and distance code correction. In this figure, there is a dotted line connecting the "end" with the beginning. It means that each object pair will iterate the same tasks.

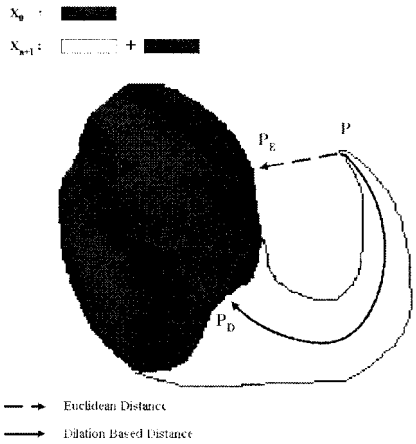

Fig. 9. Dilation Based Distance versus Euclidean Distance.

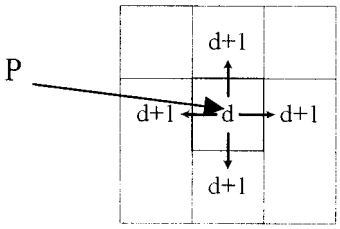

Fig. 10. Dilation cross-structure element.

A. Distance-Based Distance Transformation

In [13], [14], a variety of distance transformations for digital images are discussed. In general, most approaches concentrate on using different convolution masks such as city block or chamfer template to approximate *Euclidean shortest distance* efficiently. Considering an example in Fig. 9 based on Fig. 2, assume there are two objects X_0 and X_{n+1}, and X_0 is fully enclosed by X_{n+1}. In this case, a pixel P belonging to X_{n+1} will correspond to P_E on X_0 using *Euclidean shortest distance* transformation. This result seems awkward, since a better result would have the intermediate pixel moving from P to P_D. To achieve this, we propose to use a dilation-based distance instead of a *Euclidean shortest distance*. For this example, we first will employ a dilation operator to calculate the morphology difference area, and then apply an erosion operator to this region. We attempt to ensure that each P is contracting gradually toward X_0

```
Algorithm Dilation Based Distance Transformation
begin
  /* a. initialization */
  index ← 0
  initialize all elements of Array A0 and An+1 to be -1
  set elements A0(x,y) to be 0, if the pixel (x,y) ∈ X0
  set elements An+1(x,y) to be 0, if the pixel (x,y) ∈ Xn+1
  for each pixel (x,y) of ∂X0
    do insert a new node M(x,y,0,n+1,0)
       into the active dilation contour list LD
  for each pixel (x,y) of ∂Xn+1
    do  insert a new node M(x,y,n+1,0,0)
       into the active dilation contour list LD

  /* b. entering dilation process */
  while LD is not empty
    do retrieve the first node of the list LD and denote as N(x,y,a,b,d)
       /* c. the same dilation layer */
       if N.d = index
         /* d. check if 4-neighbors could be updated
               as the next dilation layer            */
         then for each four-neighbor P(x',y') of (N.x,N.y)
                /* e. check if inside another object */
                do if P∈Xb
                     then if Aa(P) ≥ 0 and Aa(P) ≤ index+1
                            then no update on Aa(P) and LD
                            else insert point P into the tail of the list
                                 LD with (x',y', a, b, index+1 )
                                 update Aa(P) ← index + 1
         /* f. next dilation layer */
         else index ←index + 1
              insert node N back to the first position of the list LD
end
```

Fig. 11. Dilation-based transformation algorithm.

and its intermediate results are all within the morphology-difference area.

Our *dilation* operator is a 3×3 cross-structure element as shown in Fig. 10. Assume this structure element is working on a pixel P and its distance code is d. After dilation, the distance of each neighboring pixel will be updated by $d+1$ if the distance code stored on the neighboring pixel is more than $d+1$. Otherwise, there is no update. The latter case implies there is a shorter path from another point to this neighboring pixel. In Fig. 11, we show the algorithm to perform distance transformation for two digital image slices. In this algorithm, we require a first-in–first-out queue (FIFO) linked list and on this list each node is a structure including five fields, namely,

1) x: pixel's X coordinate;
2) y: pixel's Y coordinate;
3) distance map identifier a;
4) distance map identifier b;
5) distance code d stored at pixel (X, Y).

In the above algorithm, we use two arrays A_0 and A_{n+1} to store distance codes for region II (i.e., $X_{n+1} - (X_0 \cap X_{n+1})$) and region I (i.e., $X_0 - (X_0 \cap X_{n+1})$), respectively. Initially [part (a) in algorithm], each pixel of the distance maps (A_0 or A_{n+1}) is set to be -1 (i.e., outside X_0 or X_{n+1}) or 0 (i.e., inside X_0 or X_{n+1}). Then, we insert all pixels of both ∂X_0 and ∂X_{n+1} into a list called the active dilation list L_D. The distance code of these contour pixels is all zero. Fig. 12 shows an example for this initialization. In this example, we assume a smaller object X_0 is fully enclosed by a larger object X_{n+1}. We start from each node at active linked list L_D to perform the dilation [part (b)]. In other words, we start the dilation from layer zero, since

the distance code of all nodes at L_D is zero (i.e., contour pixels at both ∂X_0 and ∂X_{n+1}). In the course of dilation, there will be many dilation layers created in a monotonically increasing order (distance code value) and each layer has an equal distance code [part (c)]. Furthermore, once a layer is totally completed, then we start another layer [part (f)]. While performing dilation, a node will insert a new node belonging to the next layer into the tail of L_D. Therefore, this newly added node will not start dilation until those nodes (with smaller distance codes) in front of it finish dilation. The whole dilation will be repeated until there is no pixel with -1 within the morphology difference region [part (e)]. Fig. 13 shows dilation results for Fig. 12. In Fig. 13(b), because the morphology difference $(X_0 - (X_0 \cap X_{n+1}))$ is a null set, there is no change on A_{n+1} after completing dilation. To the contrary, on distance map A_0, all pixels have positive distance code within area $X_{n+1} - (X_0 \cap X_{n+1})$

B. Distance Code Correction

After dilation, we obtain many contour-like layers with positive codes for the morphology-difference area. Furthermore, we will classify those pixels located on the outermost layer into two categories: namely, *terminal* and *transient* pixels. The distance code of each *transient* pixel will be corrected in the proposed scheme. We illustrate the idea of this correction by using Fig. 14. Assume we have two rectangular objects: the larger one is called X_{n+1} and the smaller one is called X_0. In this figure, the morphology-difference is the nonshaded region. After accomplishing dilation, the distance codes for this nonshaded area are all positive. In Fig. 14(a), we only show the distance codes for those pixels located on the outermost layer. In this example, let us consider three pixels on the outermost layer, labeled A, B, and C. Their distance codes are 0, 3, and 6, respectively. From A to C, we have distance codes 0, 1, 2, 3, 4, 5, and 6. Suppose, we directly interpolate an intermediate object at $E_k^n = 0.5$.

Each outermost pixel will erode $0.5 * dist$ steps toward the shaded region where $dist$ is the distance code stored at each outermost pixel. In this manner, for those pixels from A to C, they will erode 0, 1, 1, 2, 2, 3, and 3 steps in a linear manner. These erosions lead to an awkward result as shown in Fig. 14(b). A better interpolation would generate a rectangular object. To achieve this goal, we propose to modify these distance codes as shown in Fig. 14(c) *prior* to erosions. In this paper, those pixels that are not to be corrected are called *terminal* pixels. Otherwise, they are called *transient* pixels. From our point of view, these *transient* pixels are in the course of interpolation (erosion path) of the *terminal* pixels. Therefore, we cannot erode starting from them.

The algorithm of distance-code correction is illustrated in Fig. 15. For a pixel on the outermost layer, we will search for a special pattern in its eight neighboring pixels (left and right, bottom and top, and two diagonals) to correct its code as shown in Fig. 16. Note that for the diagonal cases, the special pattern consists of three consecutive codes in ascending order (such as 2, 4, and 6).

In the above algorithm, we use an extra 2-D array B to store distance codes. Recall that A_0 and A_{n+1} store distance codes for the morphology difference area. This extra B array is used to store the number of erosions for the contour of the morphology

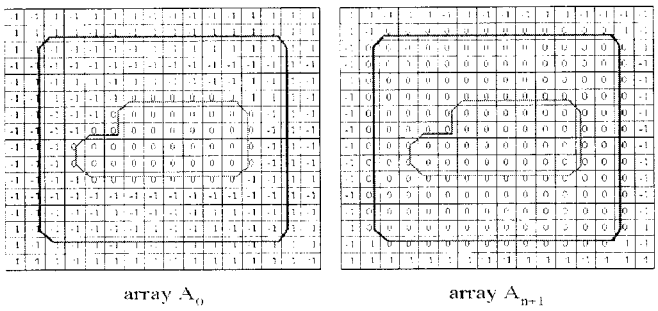

Fig. 12. Initialization for two distance maps: A_0 and A_{n+1}.

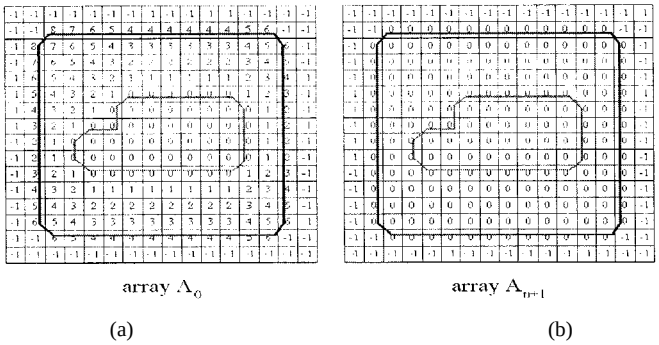

Fig. 13. (a) and (b) are dilation results on distance maps A_0 and A_{n+1}.

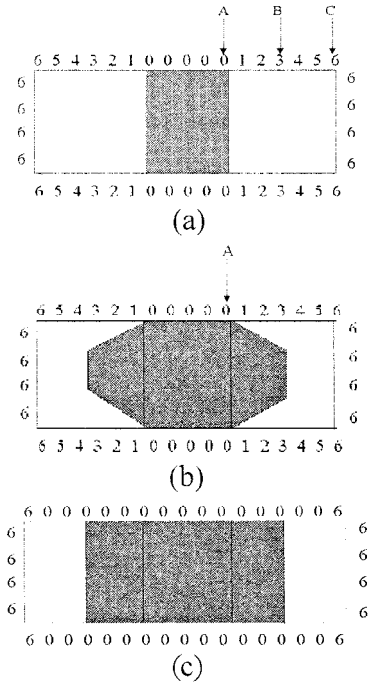

Fig. 14. Distance-code correction.

difference area. In Section III-C, we will show how to use array B to interpolate objects with the guidance of A_0 and A_{n+1}. Similarly, we also use an FIFO-linked list structure called L_C in this algorithm. This structure is exactly the same as L_D used in the previous section. First, the initialization consists of three consecutive steps:

1) initialize 2-D B array with "∞" (i.e., means a very large number);

```
begin
 /* a. initialize */
 initialize all elements of Array B to be "∞"
/* corresponds to region I illustrated in Figure 2.
 set B(x,y) ←-1, for each pixel (x,y) ∈ X0 - ( X0 ∩ Xn+1 )
 /* corresponds to region II illustrated in Figure 2.
 set B(x,y) ←-1, for each pixel (x,y) ∈ Xn+1 - ( X0 ∩ Xn+1 )
 set B(x,y) ← A0(x,y), for each pixel (x,y) of ∂Xn+1
 set B(x,y) ← An+1(x,y), for each pixel (x,y) of ∂X0
 /* b. insert working list */
 for each pixel (x,y) of ∂X0
  do we insert a new node M(x,y,n+1,0, An+1(x,y))
     into the correction contour list Lc
 for each pixel (x,y) of ∂Xn+1
  do we insert a new node M(x,y,0,n+1, A0(x,y))
     into the correction contour list Lc
 /* c. sort the list */
 sort all nodes in Lc in an increasing order
 with respect to node's element d

 /* d. correction process */
 while Lc is not empty
  do retrieve the first node of the list Lc and denote as N(x,y,a,b,d)
    flag_up ← false
    flag_down ← false
    min       ← "∞"
    /* e. could it be correction ? */
    for each eight-neighbor P(x',y') of (N.x,N.y)
      do if P ∈ Xb
           then if P is 4-neighbor of (N.x,N.y)
                  then if Aa(P) = N.d + 1
                         then flag_down ← true
                       if Aa(P) = N.d - 1
                         then flag_up   ← true
                              if B(P) < min
                                then set P' coordinate the same as P
                                     min ← B(P)
                  else if Aa(P) = N.d + 2
                         then flag_down ← true
                       if Aa(P) = N.d - 2
                         then flag_up   ← true
                              if B(P) < min
                                then set P' coordinate the same as P
                                     min ← B(P)
    if flag_up = true and flag_down = true
      then update B(N.x,N.y) ← B(P')
end
```

Fig. 15. Algorithm distance-code correction of distance maps A_0 and A_1 generated by algorithm in Fig. 11.

2) store -1 on each pixel of B for the morphology-difference area defined by $X_{n+1} - (X_0 \cap X_{n+1})$ and $X_0 - (X_0 \cap X_{n+1})$;

3) store the distance code of a pixel (x, y) that belongs to both $\partial X_0$and ∂X_{n+1} on $A_0(x, y)$ and $A_{n+1}(x, y)$ in $B(x, y)$.

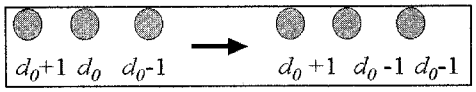

Fig. 16. When three successive distance codes increase monotonically, the distance code of the middle pixel is assigned the lower of the two adjacent distance codes.

Before performing distance code correction, we have finished dilation-based distance transformation. All pixels belonging to ∂X_0 and ∂X_{n+1} are with nonzero distance codes. Then, we insert all pixels belonging to ∂X_0 and ∂X_{n+1} into L_C and we sort nodes in an ascending order on L_C based on distance code d [algorithm part (a)–(c)]. The purpose of this sorting is to facilitate the distance-code correction. To find *transient* pixels, we search corrected patterns starting from node with the smallest d on L_C. For each checked node, we will check its eight-neighboring pixels in two phases: the first phase is to check four-neighboring nodes on left and right, and bottom and top, and the second phase is to check neighbors on two diagonals. We use two flags, **flag_up** and **flag_down** to indicate if we find a corrected pattern in the first and the second phases. For a node (x, y), if its both **flag_up** and **flag_down** are true, we conclude this node is a *transient* pixel. Then, we will correct its distance code on array $B(x, y)$. Therefore, in Fig. 14(c), except pixels A and C, the other pixels are *transient* pixels from A to C. We correct these *transient* pixels with the distance code of the pixel A. Fig. 17 shows the result for Fig. 13 after this correction. In this figure, the corrected pixel is marked using a *bold square*.

C. Erosion-Based Interpolation

In [12], Guo *et al.* used a circular disk-like structure element to perform erosion. This approach requires complicated computation to find the normal vector of boundary pixels indicating erosion direction. To apply this approach to Fig. 14, they will potentially create bad results like Fig. 14(b). In contrast, we use a simple cross-structure element instead to perform erosion. It is very simple in computational complexity and easy in practical implementation. In our implementation, there are three 2-D arrays: A_0, A_{n+1}, and B to assist in accomplishing interpolation. From our point of view, the first two arrays serve as guiding maps that show how to erode (contract) in the course of interpolation. The latter array B stores the number of erosion steps for each contour pixel of the morphology-difference area. In our implementation, B array is not a floating point array but it is rounded integer array. Fig. 18 shows the algorithm for erosion-based interpolation.

In the above algorithm, the part (a) will initialize an active erosion list L_E. The data structure of L_E is exactly the same as L_D and L_C. Then, in part (b), we calculate exact number of erosion steps D_{erosion} using (9) for each node $P(x, y)$ in L_E and update 2-D array $B[P]$ by D_{erosion}

$$D_{\text{erosion}}(P) = \begin{cases} B[P] * E_k^n, & \text{if } P \in X_0 \\ B[P] * E_k^{n^*}, & \text{if } P \in X_{n+1} \end{cases}. \quad (9)$$

In the above equation, E_k^n is an *erosion factor* for the region $X_0 - (X_0 \cap X_{n+1})$ and $E_k^{n^*}$ is used for the region $X_{n+1} - (X_0 \cap X_{n+1})$. In part (c), assume that we are now working at pixel $(N \cdot x, N \cdot y)$, and $B(N \cdot x, N \cdot y)$ indicates how many steps are still left to erode (contract). The pixel $(N \cdot x, N \cdot y)$ will potentially contract toward each four-neighboring pixel, say $P(x', y')$. Similar to dilation, there are two rules to guarantee correct erosion described as follows.

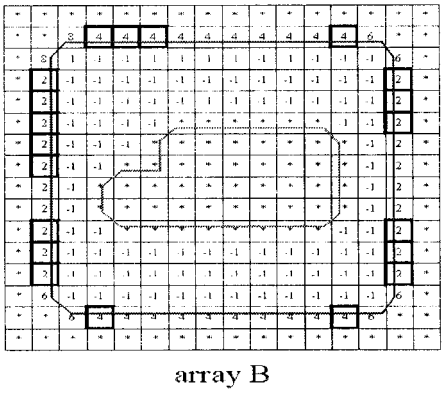

Fig. 17. Corrected distance map. Note that "*" means ∞.

1) Normal case: if $A_a(P)+1$ is equal to $A_a(N \cdot x, N \cdot y)$, and $N \cdot d - 1$ is greater than $B(P)$, then there is a dilated path from $P(x', y')$ to $(N \cdot x, N \cdot y)$ generated in Section III-A. Therefore, $(N \cdot x, N \cdot y)$ is allowed to contract toward P during erosion. Additionally, to finish erosion, P still needs to erode $N \cdot d - 1$ steps.
2) Special case: if $A_a(P)$ is equal to $A_a(N \cdot x, N \cdot y)$, we need to further check if all four-neighboring pixels, say Q_i, $(i = 1 \cdots 4)$, are all satisfied $A_a(Q_i) \leq A_a(P)$ or not. If yes, we say $(N \cdot x, N \cdot y)$ can shrink to P. However, to finish erosion, P still requires $N \cdot d$ steps instead of $N \cdot d - 1$.

If the above two conditions are satisfied, the pixel $(N \cdot x, N \cdot y)$ is allowed to shrink to P. Then, we need to determine if we will insert P in L_E described as follows. If $(N \cdot d - 1) > B(P)$, we will insert it in L_E and also update $B(P)$ by $(N \cdot d - 1)$. Otherwise, we do not need to insert P in L_E, since the consecutive erosions from P are totally included in the other longer and consecutive erosions. Therefore, we do not need to insert P in L_E, since we do not need to repeat erosions that will be included by the other node. Fig. 19(a) and (b) shows interpolated results with and without correcting code. Note that in both figures, the contour of interpolated object is composed of zero code pixels. Fig. 19(c) and (d) shows 3-D reconstructed results if we interpolate 100 slices between two input slices.

The experimental results show we can get better interpolation after distance code correction. In Fig. 19(c) and (d), we circle two places to remark their difference in interpolation. In this example, these should be linear edges in these places, but in Fig. 19(c) these edges appear as nonlinear edges.

IV. Experimental Results and Discussions

To evaluate the performance of the proposed scheme, we used many synthesized images that were used or similar to those used in previous studies. Both Figs. 20 and 21 were also tested in [8]. This approach [8] employed a very computationally intensive method to distort one contour to be like another one. Using the simpler proposed scheme, we see the shapes of intermediate contour change smoothly between two different shape contours. The next two examples were tested in [12]. Fig. 22

```
begin
  /* a. initialize */
  for each pixel (x,y) of ∂X_0
    do  insert a new node M(x,y,n+1,0,0)
        into the erosion contour list L_E
  for each pixel (x,y) of ∂X_{n+1}
    do  insert a new node M(x,y,0,n+1,0)
        into the erosion contour list L_E
  /* b. calculate the exact distance to be eroded
        and insert it into working list          */
  for each node M in the list L_E
    do D_erosion ← B(x,y) * E_k^n    if b = 0   or
                   B(x,y) * E_k^n'   if b = n+1
       update the value d of M by D_erosion
       update B(x,y) by D_erosion

  /* c. erosion process */
  while L_E is not empty
    do retrieve the first node of the list L_E and denote as N(x,y,a,b,d)
    /* d. if required to be eroded */
    if ( N.d > 0 and N.d ≥ B(N.x,N.y)
      then for each four-neighbor P(x',y') of (N.x,N.y)
             /* e. normal erosion case */
             do if A_a(N.x,N.y) = A_a(P)+1
                  then if N.d-1 > B(P)
                         then
                              update B(P) by N.d-1
                              insert point P into the tail of the list
                              L_E with (x',y', a, b, N.d-1 )
             /* f. special erosion case */
                 if A_a(N.x,N.y) = A_a(P)
                   then if all the four-neighbor Q of P, A_a(P) ≥ A_a(Q)
                         then
                              update B(P) by N.d
                              insert point P into the tail of the list
                              L_E with (x',y', a, b, N.d )
  /* g. get result */
  Finally, the newly interpolated contour consists
  of all pixels (x,y) with B(x,y) = 0.
End
```

Fig. 18. Erosion-based interpolation algorithm.

is a set of ring-like objects with no overlapping area. Guo *et al.* [12] reported that this tested data cannot be handled well by the shape-based scheme. However, our results show the proposed scheme yields very satisfactory results. The left side of Fig. 23 demonstrates the interpolation between a hollow object and a solid object. In [12], Guo *et al.*showed that unlike our method, shape-based [11] and dynamic elastic methods both fail to deal with this kind of deformation. Shape-based method simply interpolates distance code for the whole image. The right side of Fig. 23 shows 3-D rendered results as we interpolate 100 slices between the two input slices. In this case, we created a pseudonegative hole, and we separately interpolated positive and negative object pairs and then blended them.

The next example is shown in Fig. 24. In this figure, the source object contains more connected regions than the target object (i.e., branching case). Similar examples have been widely tested [8], [9], [12], [15]. The proposed scheme yields very satisfactory results. Furthermore, our results seem better than those of most

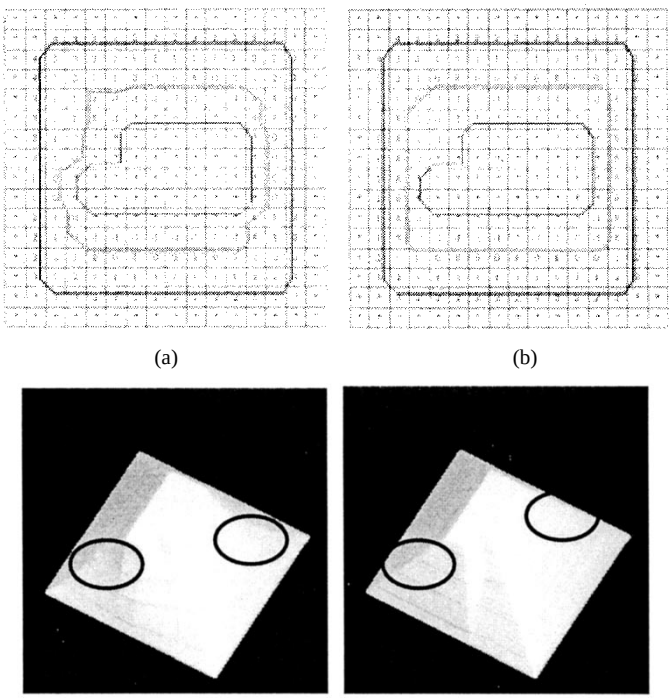

Fig. 19. Interpolated result with and without correcting distance code. (a) Array B without distance correction. (b) Array B with distance correction. (c) Without correction. (d) With correction.

Fig. 20. Interpolation between two synthetic contours that was tested in [8].

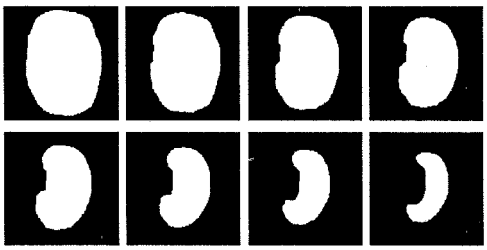

Fig. 21. Interpolation for concave case [8].

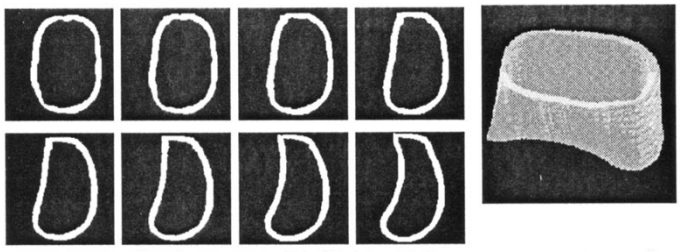

Fig. 22. Interpolation for a set of ring-like objects [12].

approaches. In this case, the proposed algorithm will first independently interpolate three positive object pairs. Then, we unite these three interpolated results together. The final two examples

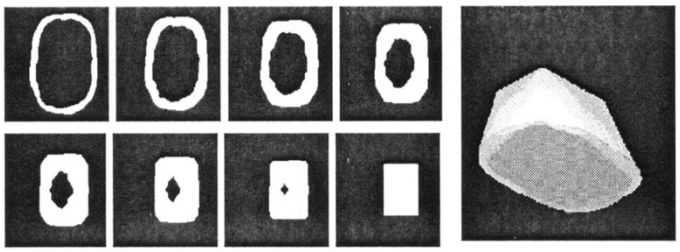

Fig. 23. Interpolation between a hollow object and a solid object [12].

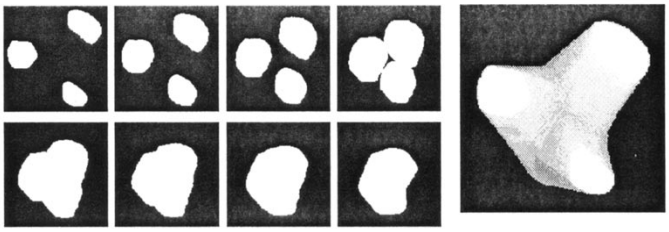

Fig. 24. Branching case [8], [9], [12], [15].

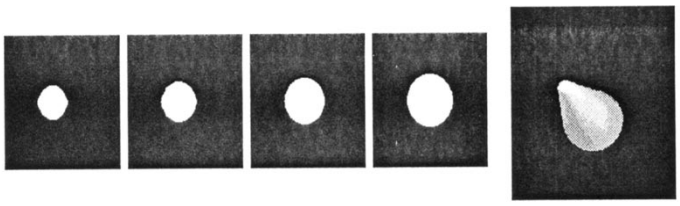

Fig. 25. The ring case [15].

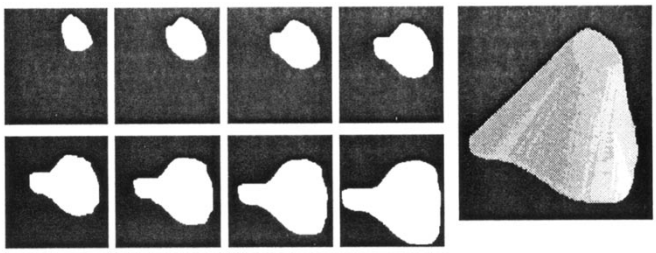

Fig. 26. The invagination case (abrupt change in shape) [15].

are shown in Figs. 25 and 26. In Fig. 25, the source image contains a small ring, while the target image contains a large one with large offset. Fig. 26 is called heavy invagination case (i.e., abrupt change in shape) in [15]. In [15], both cases were not handled well by another morphology-based scheme [12]. However, from the experimental results, it is clear that the proposed scheme can handle both cases well. Additionally, [12] exploited the beginning and ending points of morphology-difference vectors as cue to interpolate object. This approach seems to be more complex than the proposed scheme. Therefore, the proposed dilation and erosion can obtain better efficiency in computation. In summary, we have evaluated the proposed scheme using a variety of examples that were used in the previous work. From the above examples with synthesized objects, the proposed method handles different situations effectively.

With respect to computational complexity, the proposed algorithm is in proportion to the number of object pairs. For each object pair, we only need to interpolate their morphology-difference area rather than the whole image. In final, we blend all interpolated

TABLE II
WE SHOW THE EXECUTION TIMING FOR EACH EXPERIMENT AND WE INTERPOLATE 100 SLICES BETWEEN TWO INPUT SLICES IN EACH EXPERIMENT

Figure No.	Fig. 20	Fig. 21	Fig. 22	Fig. 23	Fig. 24	Fig.25	Fig. 26
Timings (sec.)	36	69	137	107	88	24	63

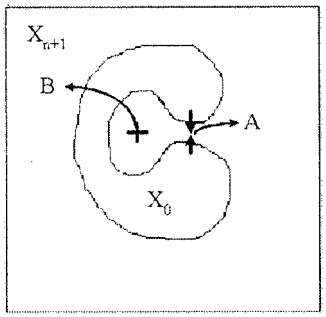

Fig. 27. An example cannot be handled well by the proposed method.

objects. On the other hand, the shape-based method interpolates distance on whole images. In this respect, if there are fewer object pairs or large overlapping regions, the proposed scheme's computational complexity is lower than shape-based methods. But, in the reverse case, the shape-based method seems faster, but it has many drawbacks as pointed out in previous work. Similarly, the complexity of [12] is also linear in regard to the number of object pairs. This approach requires expensive cost in determining the correct erosion vector. In this respect, our proposed scheme seems more practical than this approach. Furthermore, the experimental results show the proposed scheme can handle more general cases than [12]. As for the other higher complexity algorithms such as [8], [9], they are specialized for branching or shape with dissimilarity case. The proposed scheme also can handle well their examples but at much lower computational cost. We conclude the proposed scheme is very simple at both computational complexity and practical implementation, but very effective at handling different cases. Finally, Table II shows execution timing for each experiment. Our experiments were performed on the Intel Pentium II, 233 MHz personal computer with 256 MB main memory.

Although the proposed method can solve many drawbacks reported in previous studies, in some situation our method cannot handle it well. In Fig. 27, the region III ($X_0 \cap X_{n+1}$) is equal to X_0. There is a very narrow concavity (i.e., marked by A) in the object X_0. In the course of dilation, the region near to A will be filled up earlier than the region near to B. The distance codes of the region near to B will be larger than those of the region near to A. In this situation, unfortunately, the erosion cannot contract into the region near to B. Therefore, we cannot obtain an appropriate interpolation.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a morphology-based interpolation scheme. The proposed scheme has been experimentally shown to successfully resolve complex interpolation problems that cannot be well handled by previous approaches. Without complicated computation such as elastic force, the proposed scheme can handle branching problems. By combining object centralization and pseudoobject generation, the proposed scheme can handle interpolating objects with large offsets and objects with holes. Additionally, to create smooth results, the proposed scheme corrects distance code *prior* to interpolation. In implementation, we use three 2-D arrays to store distance maps (A_0 and A_{n+1}) and the erosion map (B). The former two arrays serve as interpolating maps to guide how to interpolate objects. These two maps are unchanged for a given object pair. The array B is a temporary array to indicate how many steps are left to interpolate. Compared to other schemes, the proposed scheme is simpler in computational complexity, but from the experimental results of test examples, our method is proven to handle general object interpolation, including branching, hollow case, and invagination. All these examples can be automatically interpolated and properly handled in our scheme. In near future, we plan to apply the proposed scheme to handle multidimensional objects. In particular, we are interested in exploiting this scheme to visualize dynamic movement of organs in four dimensions. Additionally, we will study how to solve the problem as shown in Fig. 27.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their helpful comments and criticisms in improving the earlier versions of this paper.

REFERENCES

[1] G. Herman and C. Coin, "The use of 3D computer display in the study of disk disease," *J. Comput. Assist. Tomogr.*, vol. 4, no. 4, pp. 564–567, Aug. 1980.
[2] C. C. Liang, C. T. Chen, and W.C. Lin, "Intensity interpolation for reconstructing 3-D medical images from serial cross-sections," in *Proc. IEEE Eng. Med. Bio. Soc. 10th Int. Conf.*, New Orleans, LA, Nov. 1988, CH2566-8, pp. 1389–1390.
[3] ——, "Intensity interpolation for branching in reconstructing three-dimensional objects from serial cross-sections," in *Proc. SPIE Conf. Med. Imaging V: Image Processing*, vol. 1445, San Jose, CA, Feb.–Mar. 1991.
[4] G. J. Grevera and J. K. Udupa, "Shape-based interpolation of multidimensional grey-level images," *IEEE Trans. Med. Imag.*, vol. 15, pp. 881–892, 1996.
[5] G. J. Grevera, J. K. Udupa, and Y. Miki, "A task-specific evaluation of three-dimensional image interpolation techniques," *IEEE Trans. Med. Imag.*, vol. 18, pp. 137–143, Feb. 1999.
[6] K. Chuang, C. Chen, L. Yuan, and C. Yeh, "Shape-based grey-level image interpolation," *Phys. Med. Biol.*, vol. 44, pp. 1565–1577, 1999.
[7] R. G. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 1153–1160, Dec. 1981.
[8] W.-C. Lin, C.-C. Liang, and C.-T. Chen, "Dynamic elastic interpolation for 3-D medical image reconstruction from cross sections," *IEEE Trans. Med. Imag.*, vol. 7, pp. 225–232, Sept. 1988.
[9] S.-Y. Chen and W.-C. Lin, "Automated surface interpolation technique for 3-D object reconstruction from serial cross sections," *Comput. Med. Imag. Graph.*, vol. 15, no. 4, pp. 265–276, 1991.
[10] S. P. Raya and J. K. Udupa, "Shape-based interpolation of multi-dimensional objects," *IEEE Trans. Med. Imag.*, vol. 9, pp. 32–42, Mar. 1990.
[11] G. T. Herman, J. Zheng, and C. A. Bucholtz, "Shaped-based interpolation," *IEEE Comput. Graph. Applications*, pp. 69–79, May 1992.
[12] J.-F. Guo, Y.-L. Cai, and Y.-P. Wang, "Morphology-based interpolation for 3D medical image reconstruction," *Comput. Med. Imag. Graph.*, vol. 19, no. 3, pp. 267–279, 1995.
[13] G. Borgefors, "Distance transformations in arbitrary dimensions," *Comput. Vision, Graph. Image Processing*, vol. 27, no. 3, pp. 321–345, Sept. 1984.
[14] ——, "Distance transformations in digital images," *Comput. Vision, Graph. Image Processing*, vol. 34, no. 3, pp. 344–371, Sept. 1986.
[15] Y.-H. Liu, Y.-N. Sun, C.-W. Mao, and C.-J. Lin, "Edge-shrinking interpolation for medical images," *Comput. Vision, Graph. Image Processing*, vol. 21, no. 2, pp. 91–101, 1997.