# Polymorph: Morphing Among Multiple Images

**Seungyong Lee**
*Postech, Korea*

**George Wolberg**
*City College of New York*

**Sung Yong Shin**
*KAIST, Korea*

**Polymorph extends conventional morphing to derive morphed images from more than two images at once, effectively integrating geometric manipulations and color blending.**

Image metamorphosis has proven to be a powerful visual effects tool. Many breathtaking examples now appear in film and television, depicting the fluid transformation of one digital image into another. This process, commonly known as *morphing*, couples image warping with color interpolation. Image warping applies 2D geometric transformations on images to align their features geometrically, while color interpolation blends their colors. Details of various image morphing techniques can be found in several recent papers.[1-4]

Traditional image morphing considers only two input images at a time—the source and target images. In that case, morphing among multiple images involves a series of transformations from one image to another. This limits any morphed image to the features and colors blended from just two input images. Given morphing's success using this paradigm, it seems reasonable to consider the benefits possible from a blend of more than two images at a time. For instance, consider generating a facial image with blended characteristics of eyes, nose, and mouth from several input faces. In this case, morphing among multiple images involves a blend of several images at once—a process we call *polymorphing*.

Rowland and Perrett considered a special case of polymorphing to obtain a prototype face from several tens of sample faces.[5] They superimposed feature points on input images to specify the different positions of features in sample faces. Averaging the specified feature positions determined the shape of a prototype face. A prototype face resulted from image warping each input image and then performing a cross-dissolve operation among the warped images. In performing predictive gender and age transformations, they used the shape and color differences between prototypes from different genders and ages to manipulate a facial image.

In this article, we present a general framework for polymorphing by extending the traditional image morphing paradigm that applies to two images. We formulate each input image as a vertex of an $(n-1)$-dimensional simplex, where $n$ equals the number of input images. Note that an $(n-1)$-dimensional simplex is a convex polyhedron having $n$ vertices in $(n-1)$-dimensional space, such as a triangle in 2D or a tetrahedron in 3D. An arbitrary in-between (morphed) image can be specified by a point in the simplex. The barycentric coordinates of that point determine the weights used to blend the input images into the in-between image. When considering only two images, the simplex degenerates into a line. Points along the line correspond to in-between images in a morph sequence. This case is identical to conventional image morphing. When considering more than two images, a path lying anywhere in the simplex constitutes the in-between images in a morph sequence.

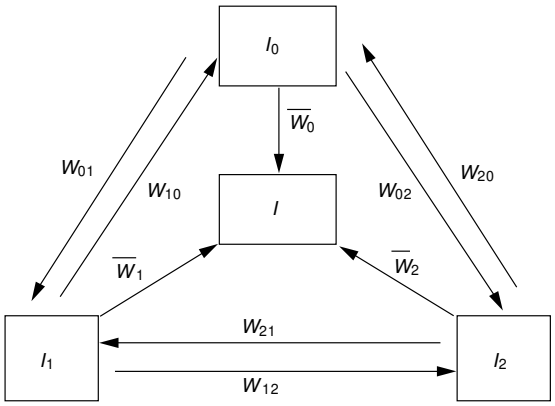
In morphing between two images, nonuniform blending was introduced to derive an in-between image in which blending rates differ across the image.[3,4] This lets us generate more interesting animations, such as a transformation of the source image to the target from top to bottom. Nonuniform blending was also considered in volume metamorphosis to control blending schedules.[6,7] In this article, the framework for polymorphing includes nonuniform blending of features in several input images. For instance, a facial image can be generated to have its eyes, nose, mouth, and ears derived from four different input faces.

Polymorph is ideally suited for image composition applications. It treats a composite image as a metamorphosis of selected regions in several input images. The


0272-1716/98/$10.00 © 1998 IEEE

regions seamlessly blend together with respect to geometry and color. The technique produces high-quality composites with considerably less effort than conventional image composition techniques. In this regard, polymorphing brings to image composition what image warping has brought to cross-dissolve in deriving morphing: a richer, more sophisticated class of visual effects achieved with intuitive and minimal user interaction.

First we'll look at the mathematical framework for polymorph, followed by warp function generation and propagation, blending function generation, and the implemented polymorph system. Metamorphosis examples demonstrate the use of polymorph for image composition.




**1 Warp functions for three input images.**

## Mathematical framework

This section presents the mathematical framework for polymorph. We extend the metamorphosis framework for two images[3,4] to generate an in-between image from several images. The framework is further optimized by introducing the notion of a central image. Finally, we introduce preprocessing and postprocessing steps to enhance the usefulness of the polymorphing technique.

### *Image representation*

Consider $n$ input images $I_1, I_2, \ldots, I_n$. We formulate each input image to be a vertex of an $(n-1)$-dimensional simplex. An in-between image is considered a point in the simplex. All points are given in barycentric coordinates in $R^{n-1}$ by $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, subject to the constraints $b_i \geq 0$ and $\sum_{i=1}^{n} b_i = 1$. Each input image $I_i$ corresponds to the $i$th vertex of the simplex, where only the $i$th barycentric coordinate is 1 and all the others are 0. An in-between image $I$ is specified by a point $\mathbf{b}$, where each coordinate $b_i$ determines the relative influence of input image $I_i$ on $I$.

In conventional morphing between two images, transition rates 0 and 1 imply the source and target images, respectively.[3,4] An in-between image is then represented by a real number between 0 and 1, which determines a point in 2D barycentric coordinates. The image representation in polymorph can be considered a generalization of that used for morphing between two images.

In the conventional approach, morphing among $n$ input images implies a sequence of animations between two images, for example, $I_0 \rightarrow I_1 \rightarrow \cdots \rightarrow I_n$. The animation sequence corresponds to a path visiting all vertices along the edges of the simplex. In contrast, polymorphing can generate an animation corresponding to an arbitrary path inside the simplex. The animation contains a sequence of in-between images that blends all $n$ input images at a time. In the following, we consider the procedure to generate the in-between image associated with a point along the path. The procedure can be readily applied to all other points along the path to generate an animation.

### *Basic metamorphosis framework*

Suppose that we want to generate an in-between image $I$ at point $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ from input images $I_1, I_2, \ldots, I_n$. Let $W_{ij}$ be the warp function from image $I_i$ to image $I_j$. $W_{ij}$ specifies the corresponding point in $I_j$ for each point in $I_i$. When applied to $I_i$, $W_{ij}$ generates a warped image where-by the features in $I_i$ coincide with their corresponding features in $I_j$. Note that $W_{ii}$ is the identity warp function, and $W_{ji}$ is the inverse function of $W_{ij}$.

To generate an in-between image $I$, we first derive a warp function $\overline{W}_i$ by linearly interpolating $W_{ij}$ for each $i$. Each image $I_i$ is then distorted by $\overline{W}_i$ to generate an intermediate image $\bar{I}_i$. Images $\bar{I}_i$ have the same in-between positions and shapes of corresponding features for all $i$. In-between image $I$ is finally obtained by linearly interpolating the pixel colors among $\bar{I}_i$.

Each coordinate $b_i$ of $I$ is used as the relative weight for $I_i$ in the linear interpolation of warps and colors. We call $\mathbf{b}$ a *blending vector*. It determines the blending of geometry and color among the input images to generate an in-between image. For simplicity, we treat the blending vectors for both geometry and color as identical, although they may differ in practice.

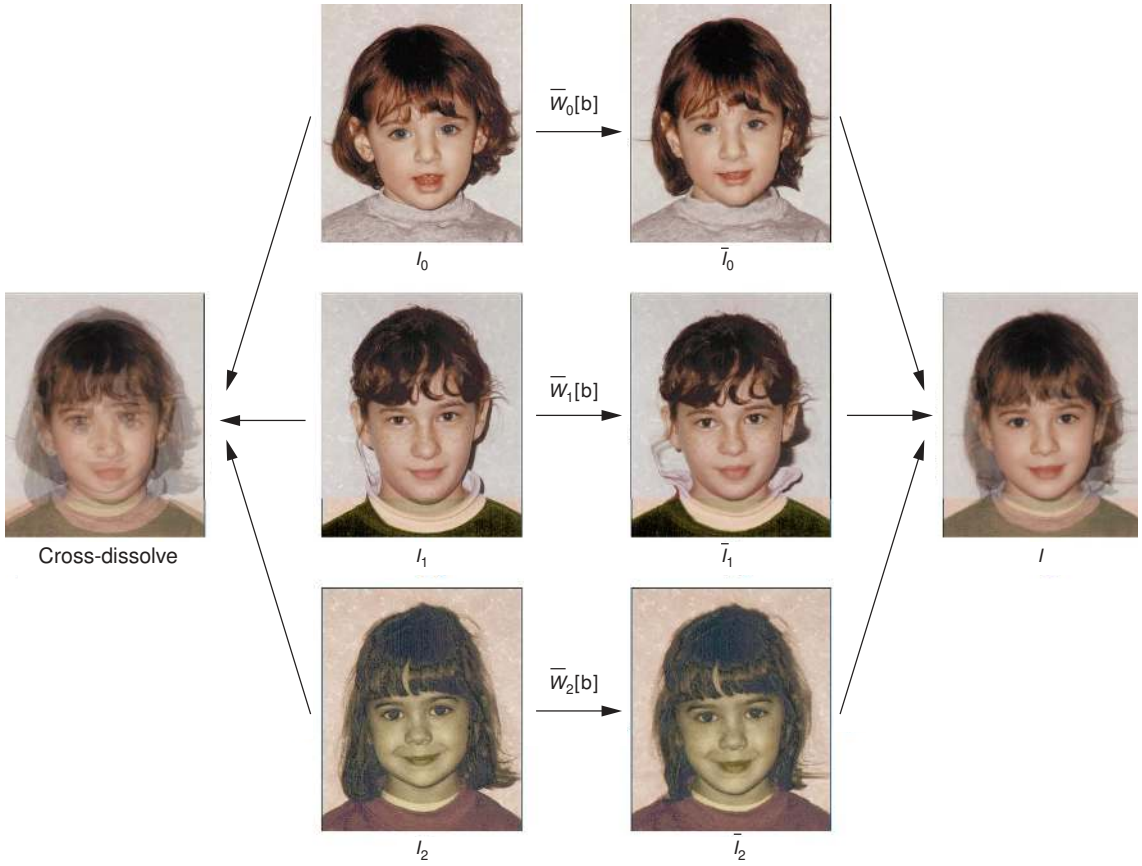
Figure 1 shows the warp functions used to generate an in-between image from three input images. Each warp function in the figure distorts one image toward the other so that the corresponding features coincide in their shapes and positions. Note that warp function $W_{ij}$ is independent of the specified blending vector $\mathbf{b}$, while $\overline{W}_i$ is determined by $\mathbf{b}$ and $W_{ij}$. Since no geometric distortions exist between intermediate image $\bar{I}_i$ and the final in-between image $I$, it is sufficient for Figure 1 to depict warps directly from $I_i$ to $I$, omitting any reference to $\bar{I}_i$. In this manner, the figure considers only the warp functions and neglects color blending.

Given images $I_i$ and warp functions $W_{ij}$, the following equations summarize the steps for generating an in-between image $I$ from a blending vector $\mathbf{b}$. $\overline{W}_i \bullet I_i$ denotes the application of warp function $\overline{W}_i$ to image $I_i$. $p$ and $r$ represent points in $I_i$ and $I$, respectively, related by $r = \overline{W}_i(p)$. Color interpolation is achieved by attenuating the pixel colors of the input images and adding the warped images.

$$\overline{W}_i(p) = \sum_{j=1}^{n} b_j W_{ij}(p)$$

$$\bar{I}_i(r) = \overline{W}_i(p) \bullet b_i I_i(p)$$

$$I(r) = \sum_{i=1}^{n} \bar{I}_i(r)$$




**2 A cross-dissolve and uniform in-between image from three input images.**

The image on the left in Figure 2 results from ordinary cross-dissolve of input images $I_i$. Notice that the image appears triple-exposed due to the blending of misaligned features. The images on the right of $I_i$ illustrate the process to generate an in-between image $I$ using the proposed framework. Although the intensities of intermediate images $\bar{I}_i$ should appear attenuated, we show the images in full intensity to clearly demonstrate the distortions. The blending vector used for the figure is $\mathbf{b} = (1/3, 1/3, 1/3)$. The resulting image $I$ equally blends the shapes, positions, and colors of the eyes, nose, and mouth of the input faces.

### *General metamorphosis framework*

The framework presented above generates a *uniform in-between image* on which we use the same blending vector across the image. We can obtain a more visually compelling in-between image by applying different blending vectors to its various parts. For example, consider a facial image that has its eyes, ears, nose, and mouth derived from four different input images. We introduce a *blending function* to facilitate a *nonuniform in-between image* that has different blending vectors over its points.

A blending function specifies a blending vector for each point in an image. Let $\overline{B}_i$ be a blending function defined on image $I_i$. For each point $p$ in $I_i$, $\overline{B}_i(p)$ is a blending vector that determines the weights used for linearly interpolating $W_{ij}(p)$ to derive $\overline{W}_i(p)$. Also, the $i$th coordinate of $\overline{B}_i(p)$ determines the color contribution of point $p$ to the corresponding point in in-between image $I$.

The metamorphosis characteristics of a nonuniform in-between image are fully specified by one blending function $\overline{B}_i$ defined on any input image $I_i$. This is analogous to using one blending vector to specify a uniform in-between image. From the correspondence between points in input images, the blending information specified by $\overline{B}_i$ can be shared among all input images. The blending functions $\overline{B}_j$ for the other images $I_j$ can be derived by composing $\overline{B}_i$ and warp functions $W_{ji}$. That is, $\overline{B}_j = \overline{B}_i \circ W_{ji}$, or equivalently, $\overline{B}_j(p) = \overline{B}_i(W_{ji}(p))$. For all corresponding points in the input images, the resulting blending functions specify the same blending vector.

Given images $I_i$, warp functions $W_{ij}$, and blending functions $\overline{B}_i$, the following equations summarize the steps for generating a nonuniform in-between image $I$. $b_i^j(p)$ denotes the $j$th coordinate in blending vector $\overline{B}_i(p)$.

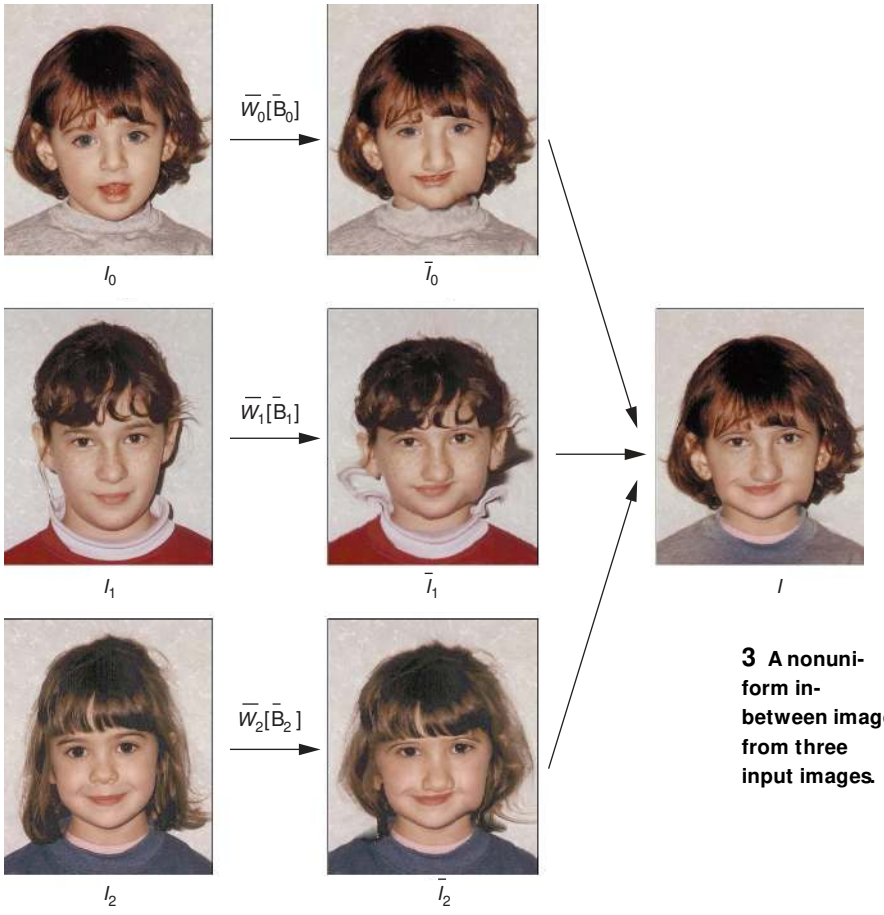
$$\overline{W}_i(p) = \sum_{j=1}^{n} b_i^j(p) W_{ij}(p)$$

$$\bar{I}_i(r) = \overline{W}_i(p) \bullet b_i^i(p) I_i(p)$$

$$I(r) = \sum_{i=1}^{n} \bar{I}_i(r)$$

Figure 3 illustrates the above framework. We have chosen blending functions $\overline{B}_i$ that make in-between image $I$ retain the hair, eyes and nose, and mouth and chin from input images $I_0$, $I_1$, and $I_2$, respectively. The $\overline{B}_i$

determine warp functions $\overline{W}_i$, which generate distortions of $I_i$ whereby the parts of interest remain intact. Here again intermediate images $\bar{I}_i$ appear in full intensity for clarity. In practice, input images $I_i$ are nonuniformly attenuated by applying $\bar{B}_i$ before they are distorted. The attenuation maintains those parts to be retained in $I$ at their full intensity. The strict requirement to retain specified features of the input images has produced an unnatural result around the mouth and chin in Figure 3. Additional processing might be necessary to address the artifacts inherent in the current process. First, we will consider how to reduce runtime computation and memory overhead in generating an in-between image.




**3 A nonuniform in-between image from three input images.**

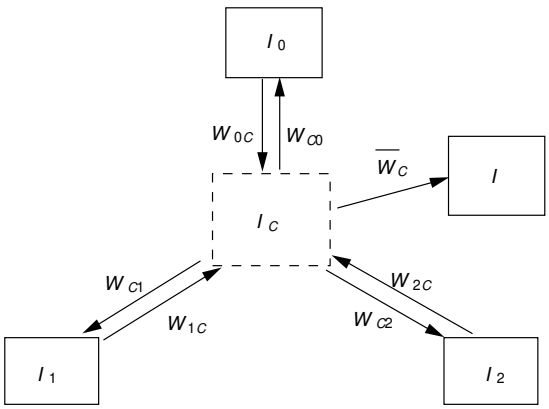
### *Optimization with a central image*

The evaluation of warp functions $W_{ij}$ is generally expensive. To reduce runtime computation, we compute $W_{ij}$ only once, when feature correspondences are established among input images $I_i$. We sample each $W_{ij}$ over all source pixels and store the resulting target coordinates in an array. The arrays for all $W_{ij}$ are then used to compute intermediate warp functions $\overline{W}_i$, which depend on blending functions $\bar{B}_i$. For large $n$, these $n^2$ warp functions require significant memory overhead, especially for large input images. To avoid this memory overhead, we define a *central image* and use it to reduce the number of stored warp functions.

A central image $I_C$ is a uniform in-between image corresponding to the centroid of the simplex that consists of input images. The blending vector $(1/n, 1/n, \ldots, 1/n)$ is associated with this image. Instead of keeping $n^2$ warp functions $W_{ij}$, we maintain $2n$ warp functions, $W_{iC}$ and $W_{Ci}$, and compute $\overline{W}_i$ from a blending function specified on $I_C$. $W_{iC}$ is a warp function from input image $I_i$ to central image $I_C$. Conversely, $W_{Ci}$ is a warp function from $I_C$ to $I_i$. $W_{Ci}$ is the inverse function of $W_{iC}$.

Let a blending function $\bar{B}_C$ be defined on central image $I_C$. $\bar{B}_C$ determines the metamorphosis characteristics of an in-between image $I$. $\bar{B}_C$ has the same role as $\bar{B}_i$ except that it operates on $I_C$ instead of $I_i$. Therefore, $\bar{B}_C(q)$ gives a blending vector that specifies the relative influences of corresponding points in $I_i$ onto $I$ for each point $q$ in $I_C$. The equivalent blending functions $\bar{B}_i$ for input images $I_i$ can be derived by function compositions $\bar{B}_C \circ W_{iC}$.

To obtain warp functions $\overline{W}_i$ from $I_i$ to $I$, we first generate a warp function $\overline{W}_C$ from $I_C$ to $I$. For each point in $I_C$, warp functions $W_{Ci}$ give a set of corresponding points in input images $I_i$. Hence, $\overline{W}_C$ can be derived by linearly interpolating $W_{Ci}$ with the weights of $\bar{B}_C$. The $W_i$ are then obtained by function compositions $\overline{W}_C \circ W_{iC}$. Figure 4 shows the relationship of warp functions $W_{iC}$, $W_{Ci}$, and $\overline{W}_C$ with images $I_i$, $I_C$, and $I$. Note that $W_{iC}$ and $W_{Ci}$ are independent of $\bar{B}_C$, whereas $\overline{W}_C$ is determined by $\bar{B}_C$ from $W_{Ci}$.




**4 Warp functions with a central image.**

Given images $I_i$ and warp functions $W_{iC}$ and $W_{Ci}$, the following equations summarize the steps for generating an in-between image $I$ from a blending function $\bar{B}_C$ defined on central image $I_C$. Let $p$, $q$, and $r$ represent points in $I_i$, $I_C$, and $I$, respectively. They are related by $q = W_{iC}(p)$ and $r = \overline{W}_C(q)$. $b_i^j(p)$ and $b_C^j(q)$ denote the $j$th coordinates in blending vectors $\bar{B}_i(p)$ and $\bar{B}_C(q)$, respectively.

$$\overline{W}_C(q) = \sum_{i=1}^{n} b_C^i(q) W_{Ci}(q)$$

$$\overline{W}_i(p) = (\overline{W}_C \circ W_{iC})(p) = \overline{W}_C(W_{iC}(p))$$

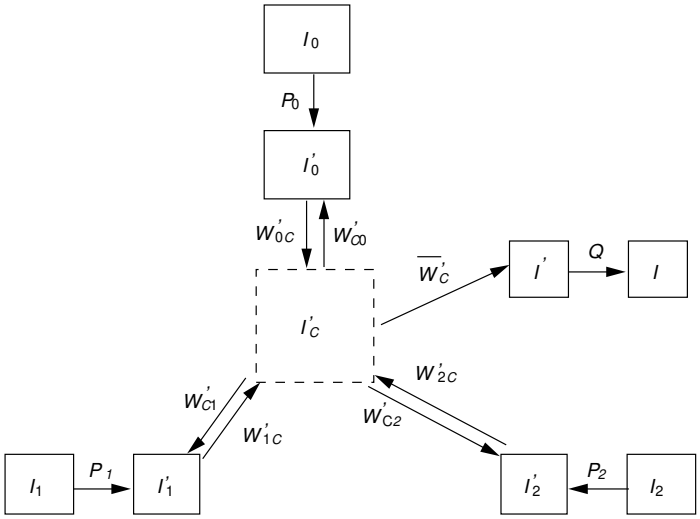
$$\overline{B}_i(p) = (\overline{B}_C \circ W_{iC})(p) = \overline{B}_C(W_{iC}(p))$$

$$\bar{I}_i(r) = \overline{W}_i(p) \bullet b_i^i(p) I_i(p)$$

$$I(r) = \sum_{i=1}^{n} \bar{I}_i(r)$$

Note that the • and ◦ operators denote forward and inverse mapping, respectively. For an image $I$ and warp function $W$, $W \bullet I$ maps all pixels in $I$ onto the distorted image $I'$, while $I \circ W^{-1}$ maps all pixels in $I'$ onto $I$, assuming that the inverse function $W^{-1}$ exists. Although the ◦ operator could have been applied to compute $\bar{I}_i$ above, we use the • operator because $\overline{W}_i^{-1}$ is not readily available.

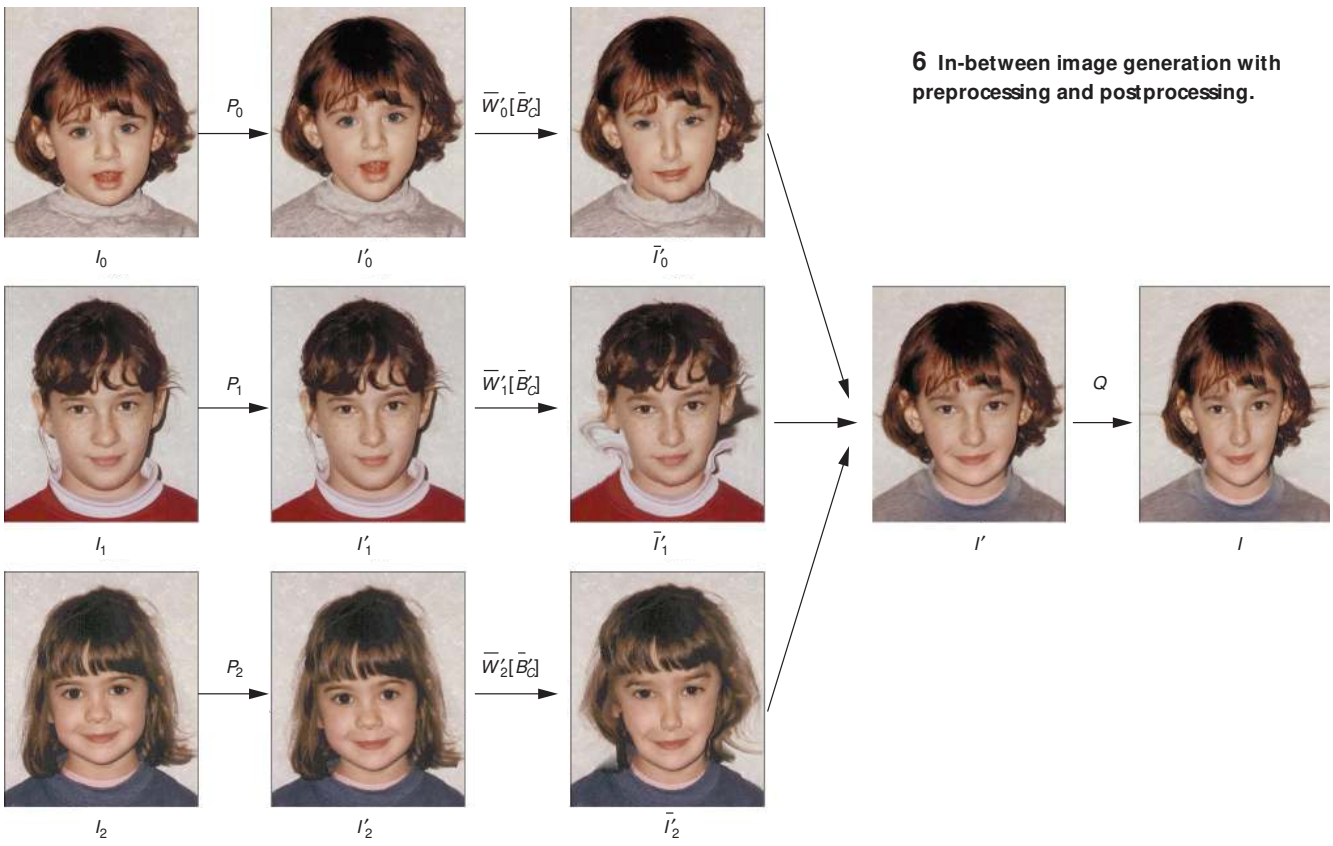
In Figure 4, central image $I_C$ has dashed borders because it is not actually constructed in the process of generating an in-between image. We introduced it to provide a conceptual intermediate step to derive the necessary warp and blending functions. Any image, including an input image, can be made to play the role of the central image. However, we have defined the central image to lie at the centroid of the simplex to establish symmetry in the metamorphosis framework. In most cases, a central image relates to a natural in-between image among the input images. It equally blends the features in the input images, such as a prototype face among input faces.




**5** Warp functions with preprocessing and postprocessing.

### *Preprocessing and postprocessing*

Polymorphing proves useful in feature-based image composition, where selected features from input




**6** In-between image generation with preprocessing and postprocessing.

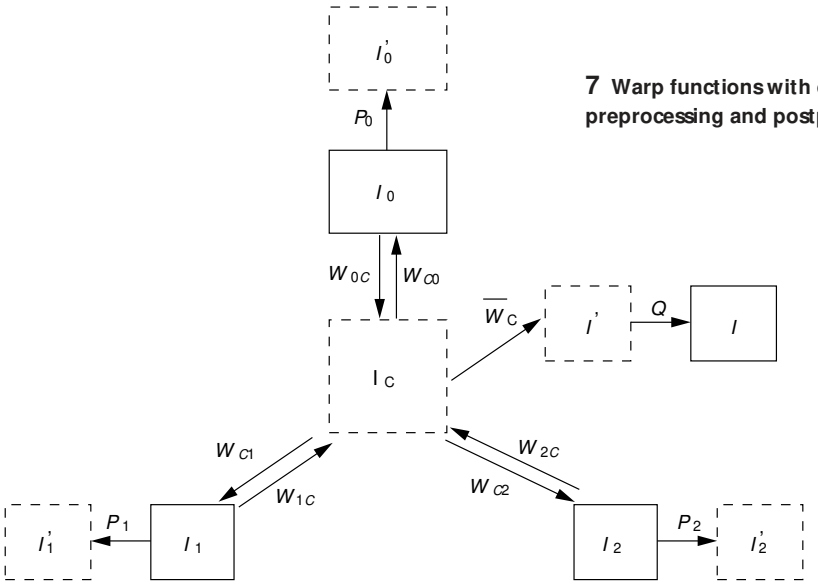
images blend seamlessly in an in-between image. In that case, if the shapes and positions of the selected features do not match among the input images, the in-between image might not have features in appropriate shapes and positions. For example, the in-between face in Figure 3 retains hair, eyes and nose, and mouth and chin features from the input faces, yet it appears unnatural. This results from the rigid placement of selected input regions into a patchwork of inappropriately scaled and positioned elements—akin to a cut-and-paste operation where smooth blending was limited to areas between these regions.

We can overcome this problem by adding a preprocessing step to the metamorphosis framework. Before using the framework on input images $I_i$, we can apply warp functions $P_i$ to $I_i$ to generate distorted input images $I_i'$. The distortions change the shapes and positions of the selected features in $I_i$ so that an in-between image from $I_i'$ has appropriate feature shapes and positions. In that case, we apply the framework to $I_i'$ instead of $I_i$, treating $I_i'$ as the input images.

After deriving an in-between image through the framework, we sometimes need image postprocessing to enhance the result, even though the preprocessing step has already been applied. For example, we might want to reduce the size of the in-between face in Figure 3—not readily done by preprocessing input images. To postprocess an in-between image $I'$, we apply a warp function $Q$ to $I'$ and generate the final image $I$. The postprocessing step is useful for local refinement and global manipulation of an in-between image.

Figure 5 shows the warp functions used in the metamorphosis framework including the preprocessing and postprocessing steps. Warp functions $P_i$ distort input images $I_i$ toward $I_i'$, from which an in-between image $I'$ is generated. Applying warp function $Q$ to $I'$ derives the final image $I$. Figure 6 illustrates the process with intermediate images. In preprocessing, the hair of input image $I_0$ and the mouth and chin of $I_2$ move upwards and to the lower right, respectively. The nose in $I_1$ narrows slightly. The face in in-between image $I'$ now appears more natural than that in Figure 3. In postprocessing, the face in $I'$ is scaled down horizontally to generate the final image $I$.

When adding the preprocessing step to the metamorphosis framework, distorted input images $I_i'$ determine the central image $I_C'$ and warp functions $W_{iC}'$ and $W_{Ci}'$. However, in that case, whenever we apply different warp functions $P_i$ to input images $I_i$, we must recompute $W_{iC}'$ and $W_{Ci}'$ to apply the framework to $I_i'$. This can become very cumbersome, especially since several preprocessing iterations might be necessary to derive a satisfactory in-between image. To overcome this drawback, we reconfigure Figure 5 to Figure 7 so that $I_C$, $W_{iC}$, and $W_{Ci}$ depend only on $I_i$, regardless of $P_i$. In the new configuration, we can derive the corresponding points in the $I_i'$ for each point in $I_C$ by function compositions $P_i \circ W_{Ci}$. Given a blending function $\overline{B}_C$ defined on $I_C$, warp function $\overline{W}_C$ is computed by linearly interpolating $P_i \circ W_{Ci}$ with the weights of $\overline{B}_C$. The resulting $\overline{W}_C$ is equivalent to $\overline{W}_C'$ used in Figure 5. Then, the warp functions from $I_i$ to $I'$ can be obtained by $\overline{W}_C \circ W_{iC}$, properly reflecting the effects of preprocessing. Warp functions $\overline{W}_i$ from $I_i$ to the final in-between image $I$, including the postprocessing step, can be derived by $Q \circ \overline{W}_C \circ W_{iC}$.




**7 Warp functions with optimal preprocessing and postprocessing.**

Consider input images $I_i$, warp functions $W_{iC}$ and $W_{Ci}$, and a blending function $\overline{B}_C$ defined on central image $I_C$. The following equations summarize the process to generate an in-between image $I$ from $\overline{B}_C$ and warp functions $P_i$ and $Q$, which specify the preprocessing and postprocessing steps. Let $p$, $q$, and $r$ represent points in $I_i$, $I_C$, and $I$, respectively. They are related by $q = W_{iC}(p)$ and $r = (Q \circ \overline{W}_C)(q)$.

$$\overline{W}_C(q) = \sum_{i=1}^{n} b_C^i(q)(P_i \circ W_{Ci})(q) = \sum_{i=1}^{n} b_C^i(q) P_i(W_{Ci}(q))$$

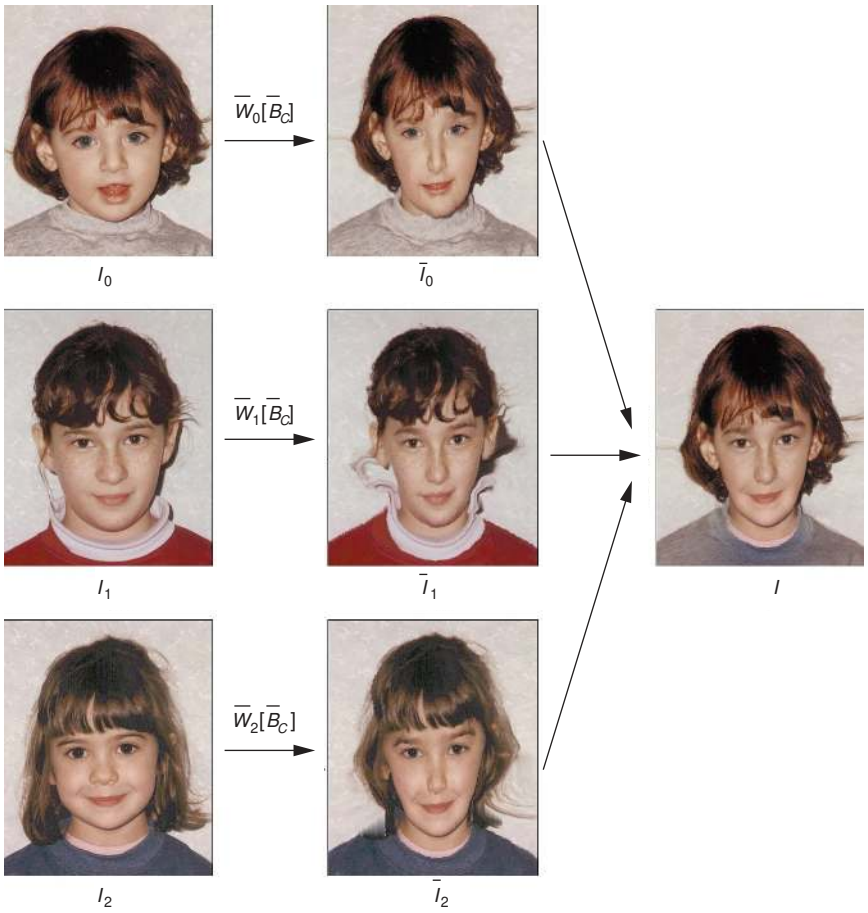
$$\overline{W}_i(p) = (Q \circ \overline{W}_C \circ W_{iC})(p) = Q(\overline{W}_C(W_{iC}(p)))$$

$$\overline{B}_i(p) = (\overline{B}_C \circ W_{iC})(p) = \overline{B}_C(W_{iC}(p))$$

$$\overline{I}_i(r) = \overline{W}_i(p) \bullet b_i^i(p) I_i(p)$$

$$I(r) = \sum_{i=1}^{n} \overline{I}_i(r)$$

The differences between this framework and that in the section "Optimization with a central image" lie only in the computation of warp functions $\overline{W}_C$ and $\overline{W}_i$. We included additional function compositions to incorporate the preprocessing and postprocessing effects. In Figure 7, images $I_i'$ and $I'$ appear with dashed borders because they are not actually constructed in generating $I$. As in the




**8 In-between image generation with optimal preprocessing and postprocessing.**

case of central image $I_C$, images $I_i'$ and $I'$ provide conceptual intermediate steps to derive $\overline{W}_C$ and $\overline{W}_i$.

Figure 8 shows an example of in-between image generation with the framework. Intermediate images $\overline{I}_i$ are the same as the distorted images generated if we apply warp function $Q$ to images $\overline{I}_i$ in Figure 6. In other words, $\overline{I}_i$ reflects the preprocessing and postprocessing effects in addition to the distortions defined by blending function $\overline{B}_C$. Hence, only three intermediate images are needed to obtain the final in-between image $I$, as in Figure 3, rather than seven as in Figure 6. Notice that $I$ is the same as in Figure 6.

Given $n$ input images, generating an in-between image through the framework requires a solution to each of the following three problems:

- how to find $2n$ warp functions $W_{iC}$ and $W_{Ci}$,
- how to specify a blending function $\overline{B}_C$ on $I_C$, and
- how to specify warp functions $P_i$ and $Q$ for preprocessing and postprocessing.

## Warp function generation

This section addresses the problem of deriving warp functions $W_{iC}$ and $W_{Ci}$, and $P_i$ and $Q$. A conventional image morphing technique computes warp functions for $(n-1)$ pairs of input images. We propagate these warp functions to obtain $W_{ij}$ for all pairs of input images. Averaging $W_{ij}$ for each $i$ produces $W_{iC}$. We compute $W_{Ci}$ as the inverse function of $W_{iC}$ by using a warp generation algorithm. $P_i$ and $Q$ are derived from the user input specified by primitives such as line segments overlaid onto images.

### *Warp functions between two images*

We can derive the warp functions between two input images using a conventional image morphing technique. Traditionally, image morphing between two images begins with establishing their correspondence with pairs of feature primitives such as mesh nodes, line segments, curves, or points. Each primitive specifies an image feature, or landmark. An algorithm then computes a warp function that interpolates the feature correspondence across the input images.

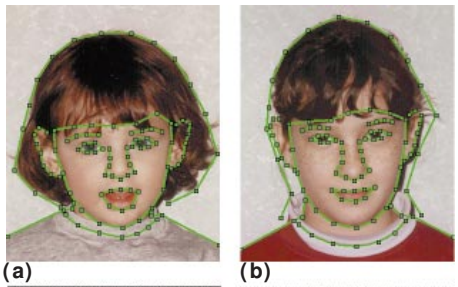
The several image morphing algorithms in common use differ in the manner in which features are specified and warp functions are generated. In mesh warping,[1] bicubic spline interpolation computes a warp function from the correspondence of mesh points. In field morphing,[2] pairs of line segments specify feature correspondences, and weighted averaging determines a warp function. More recently, thin-plate splines[4,8] and multilevel free-form deformations[3] have been used to compute warp functions from selected point-to-point correspondences.

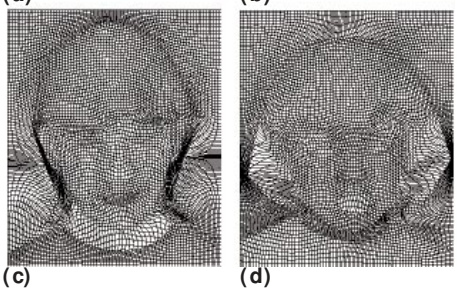
In this article, we use the multilevel free-form deformation algorithm[3] to generate warp functions between two input images. We selected this algorithm because it efficiently generates $C^2$-continuous and one-to-one

warp functions. The one-to-one property guarantees that the distorted image does not fold back upon itself.

A warp function represents a mapping between all points in two images. In this work, we save the warp functions in binary files to reduce runtime computation. A warp function for an $M \times N$ image is stored as an $M \times N$ array of target $x$- and $y$-coordinates for all source pixels. Once feature correspondence between two images $I_i$ and $I_j$ is established, we can derive warp functions in both directions, $W_{ij}$ and $W_{ji}$. Therefore, we store two $M \times N$ arrays of target coordinates for each pair of input images for which feature correspondence is established.

Figure 9 depicts the warp generation process. In Figures 9a and 9b, the specified features are overlaid on input images $I_0$ and $I_1$. Figures 9c and 9d illustrate warp functions $W_{01}$ and $W_{10}$ generated from the features by the multilevel free-form deformation.




**9** Specified features and the resulting warp functions.

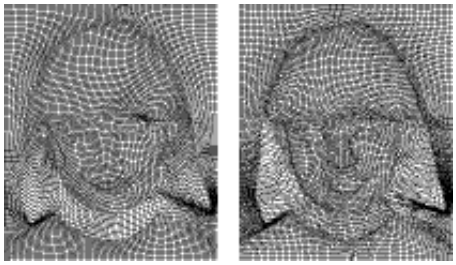
### Warp function propagation

We can derive the warp functions between two images by specifying their feature correspondence. Multiple images, however, require determining warp functions for each image to every other image. This exacerbates the already tedious and cumbersome operation of specifying feature correspondence. For instance, $n$ input images require establishing feature correspondence among $n(n-1)/2$ pairs of images. We now address the problem of minimizing this feature specification effort.

Consider a directed graph $G$ with $n$ vertices. Each vertex $v_i$ corresponds to input image $I_i$, and an edge $e_{ij}$ connects $v_i$ to $v_j$ if warp function $W_{ij}$ from $I_i$ to $I_j$ has been derived. To minimize the feature specification effort, we first select $(n-1)$ pairs of images so that the associated edges constitute a connected graph $G$. We specify the feature correspondence between the selected image pairs to obtain warp functions between them. These warp functions can then be propagated to derive the remaining warp functions for all other image pairs. The propagation occurs in the same manner as that used for computing the transitive closure of graph $G$. The connectivity of $G$ guarantees that warp functions are determined for all pairs of images after propagation.

To determine an unknown warp function $W_{ij}$, we traverse $G$ to find any vertex $v_k$ shared by existing edges $e_{ik}$ and $e_{kj}$. If we can find such a vertex $v_k$, we update $G$ to include edge $e_{ij}$ and define $W_{ij}$ as the composite function $W_{kj} \circ W_{ik}$. When there exist several such $v_k$, the composed warp functions through those $v_k$ are computed and averaged. If no $v_k$ connects $v_i$ and $v_j$, $W_{ij}$ remains unknown and $e_{ij}$ is not added to $G$. This procedure iterates for all unknown warp functions, and the iteration repeats until all warp functions are determined. If $W_{ij}$ remains unknown in an iteration, it will be determined in a following iteration as $G$ gets updated. From the connectivity of $G$, at most $(n-2)$ iterations are required to resolve every unknown warp function.

With the warp propagation approach, the user must specify feature correspondences for $(n-1)$ pairs of images. This is far less effort than considering all $n(n-1)/2$ pairs. Figure 10 shows an example of warp propagation. Figure 10a illustrates warp function $W_{02}$, which was derived by specifying feature correspondence between input images $I_0$ and $I_2$. To determine warp function $W_{12}$ from $I_1$ to $I_2$, we compose $W_{10}$ and $W_{02}$, shown in Figure 9d and Figure 10a, respectively. Figure 10b illustrates the resulting $W_{12} = W_{02} \circ W_{10}$. Notice that the left and right sides of the hair have widened in $W_{12}$ and narrowed in $W_{02}$.



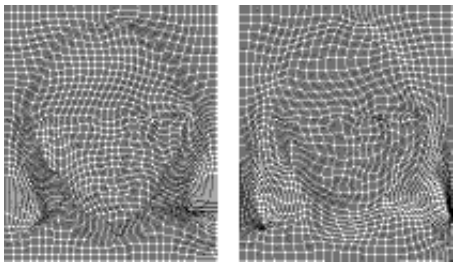

**10** Warp propagation example: (a) warp $W_{02}$ and (b) warp $W_{12} = W_{02} \circ W_{10}$.

### Warp functions for central image

We now consider how to derive the warp functions among the central image $I_C$ and all input images $I_i$. $I_C$ is the uniform in-between image corresponding to a blending vector $(1/n, 1/n, \ldots, 1/n)$. Hence, from the basic metamorphosis framework, warp functions $W_{iC}$ from $I_i$ to $I_C$ are straightforward to compute. That is, $W_{iC}(p) = \sum_{j=1}^{n} W_{ij}(p)/n$ for each point $p$ in $I_i$. Computing warp function $W_{Ci}$ from $I_C$ back to $I_i$, however, is not straightforward.

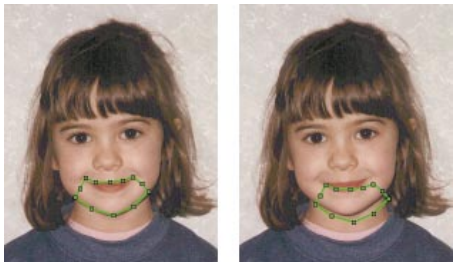
We determine $W_{Ci}$ as the inverse function of $W_{iC}$. Each point $p$ in $I_i$ is mapped to a point $q$ in $I_C$ by $W_{iC}$. The inverse of $W_{iC}$ should map each $q$ back to $p$. Hence, the corresponding points $q$ for pixels $p$ in $I_i$ provide $W_{Ci}$ with scattered positional constraints, $W_{Ci}(q) = p$. The multilevel free-form deformation technique, used to derive warp functions, can also be applied to the constraints to compute $W_{Ci}$.

When warp functions $W_{ij}$ are one-to-one, it is not mathematically clear that their average function $W_{iC}$ is also one-to-one. Conceptually, though, we can expect $W_{iC}$ to be one-to-one because it is the warp function that might be generated by moving the features in $I_i$ to the



(a) (b)

**11** Warp functions between $I_0$ and $I_C$: (a) warp $W_{0C}$ and (b) warp $W_{C0}$.



(a) (b)

**12** Specified primitives for preprocessing: (a) original positions and (b) new positions.

(a) (b)

**13** Specified primitives for postprocessing: (a) original positions and (b) new positions.

averaged positions. In the worst case that $W_{iC}$ is not one-to-one, the positional constraints, $W_{Ci}(q) = p$, may have two different positions for a point $q$. In that case, we ignore one of the two positions and apply the multilevel free-form deformation to the remaining constraints.

Figure 11 shows examples of warp functions between input images and a center image. Figure 11a illustrates warp function $W_{0C}$ from $I_0$ to $I_C$, which is the average of the identity function, $W_{01}$ in Figure 9c, and $W_{02}$ in Figure 10a. In Figure 11b, $W_{C0}$ has been derived as the inverse function of $W_{0C}$ by the multilevel free-form deformation technique.

### *Warp functions for preprocessing and postprocessing*

Warp functions $P_i$ for preprocessing are derived in a similar manner to those between two images—we overlay primitives such as line segments and curves onto image $I_i$. However, in this case, the primitives select the parts in $I_i$ to distort, instead of specifying feature correspondence with another image. $P_i$ is defined by moving the primitives to the desired distorted positions of the selected parts and computed by applying the multilevel free-form deformation technique to the displacements. Figure 12 shows the primitive and its movement specified on input image $I_2$ to define warp function $P_2$ in Figure 6. The primitive has been moved to the lower right to change the positions of the mouth and chin.

We can derive warp function $Q$ for postprocessing in the same manner as $P_i$. In this case, primitives overlaid on in-between image $I'$ in Figure 7 select parts to distort toward the final in-between image $I$. The overlaid primitives are moved to define $Q$, and multilevel free-form deformation computes $Q$ from the movements. We construct $I'$ only to allow for the specification of features and their movements, not for the process of in-between image generation. The postprocessing operation illustrated in Figure 13 horizontally scales down the primitive specified on $I'$ to make the face narrower.

## Blending function generation

This section addresses the problem of generating a blending function $\overline{B}_C$ defined on the central image $I_C$. A blending vector suffices to determine a $\overline{B}_C$ that generates a uniform in-between image. A nonuniform in-between image, however, can be specified by assigning different blending rates to selected parts of various input images. We derive the corresponding $\overline{B}_C$ by gathering all the blending rates onto $I_C$ and applying scattered data interpolation to them.

### *Uniform in-between image*

To generate a uniform in-between image from $n$ input images, a user must specify a blending vector $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, subject to the constraints $b_i \geq 0$ and $\sum_{i=0}^{n} b_i = 1$. If these constraints are violated, we enforce them by clipping negative values to zero and dividing each $b_i$ by $\sum_{i=0}^{n} b_i$. The blending function $\overline{B}_C$ is then a constant function having the resulting blending vector as its value at every point in $I_C$.

### *Nonuniform in-between image*

To generate a nonuniform in-between image $I$, a user assigns a real value $b_i \in [0,1]$ to a selected region $R_i$ of input image $I_i$ for some $i$. The value $b_i$ assigned to $R_i$ determines the influence of $I_i$ onto the corresponding part $R$ of in-between image $I$. When $b_i$ approaches 1, the colors and shapes of the features in $R_i$ dominate those in $R$. Conversely, when $b_i$ approaches 0, the influence of $R_i$ on $R$ diminishes. Figures 14a, 14b, and 14c show the polygons used to select regions in $I_0$, $I_1$, and $I_2$ for generating $I$ in Figure 3. All points inside the polygons in Figures 14a, 14b, and 14c have been assigned the value 1.0.

We generate a blending function $\overline{B}_C$ by first projecting the values $b_i$ onto $I_C$. We can do this by mapping points in $R_i$ onto $I_C$ using warp functions $W_{iC}$. Figure 14d shows the projection of the selected parts in Figures 14a, 14b, and 14c onto $I_C$. Let $(b_1, b_2, \ldots, b_n)$ be an $n$-tuple representing the projected values of $b_i$ onto a point in $I_C$. This $n$-tuple is defined only in the projection of $R_i$ on $I_C$. Since the user does not have to specify $b_i$ for all $I_i$, some of the $b_i$ may be undefined for the $n$-tuple.

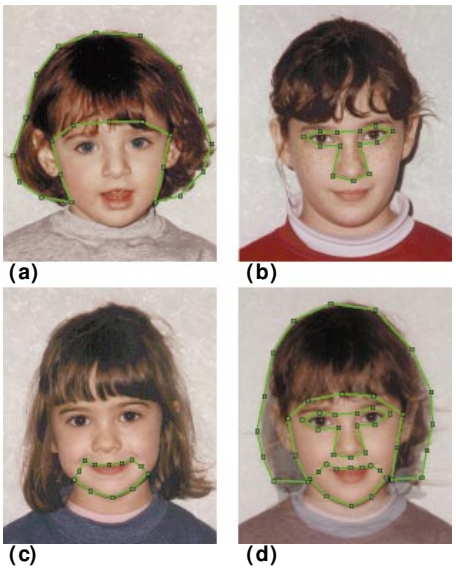
Let $D$ and $U$ denote the sets of defined and undefined elements $b_i$ in the $n$-tuple, respectively. Further, let $s$ be the sum of the defined values in $D$. There are three cases to consider: $s > 1$, $s \leq 1$ and $U$ is empty, and $s \leq 1$ and $U$ is not empty. If $s > 1$, then we assign zero to the undefined values and scale down the elements in $D$ to satisfy $s = 1$.

If $s \leq 1$ and $U$ is empty, then we scale up the elements in $D$ to satisfy $s = 1$. Otherwise, we assign $(1–s)/k$ to each element in $U$, where $k$ is the number of elements in $U$.

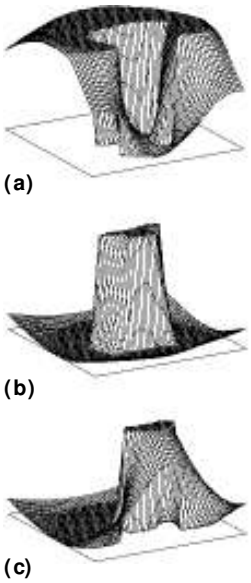
Normalizing the $n$-tuples of the projected values lets us obtain blending vectors for the points in $I_C$ that correspond to the selected parts $R_i$ in $I_i$. These blending vectors can then be propagated to all points in $I_C$ by scattered data interpolation. We construct an interpolating surface through the $i$th coordinates of these vectors to determine $b_i$ of a blending vector **b** at all points in $I_C$. For the scattered data interpolation, we use multilevel B-splines,[9] a fast technique for generating a $C^2$-continuous surface.

After constructing $n$ surfaces, we have an $n$-tuple at each point in $I_C$. Since each surface is generated independently, the sum of the coordinates in the $n$-tuple does not necessarily equal one. In that case, we scale them to force their sum to one. The resulting $n$-tuples at all points in $I_C$ define a blending function $\bar{B}_C$ that satisfies the user-specified blending constraints.

Figure 15 illustrates the constructed surfaces to determine $\bar{B}_C$ used in Figure 3. The heights of the surfaces in Figures 15a, 15b, and 15c represent $b_0$, $b_1$, and $b_2$ of blending vector **b** at the points in $I_C$, respectively. In Figure 14b, $b_1$ is 1.0 at the points corresponding to the eyes and nose in $I_C$, satisfying the requirement specified in Figure 14b. They are 0.0 around the hair, mouth, and chin due to the value 1.0 assigned to those parts in Figures 14a and 14c. Figures 15a and 15c also reflect the requirements for $b_0$ and $b_2$ specified in Figure 14.




**14** Selected parts on input images and their projections onto the central image.




**15** Constructed surfaces for a blending function.

## Implementation

This section describes the implementation of the polymorph framework. We also present the implemented system's performance.

### *Polymorph system*

The polymorph system consists of three modules. The first module is an image morphing system that considers two input images at a time. It requires the user to establish feature correspondence among two input images and generates warp functions between them. We adopted the morph system presented by Lee et al.[3] because it facilitates flexible point-to-point correspondences and produces one-to-one warp functions that avoid undesirable foldovers. The first module is applied to $(n-1)$ pairs of input images, which correspond to the edges selected to constitute a connected graph $G$. The generated warp functions are sampled at each pixel and stored in binary files.

The second module is the warp propagation system. It first reads the binary files of the warp functions associated with the selected edges in graph $G$. Those warp functions are then propagated to derive $n(n–1)$ warp functions $W_{ij}$ among all $n$ input images $I_i$. Finally, the second module computes $2n$ warp functions in both directions between the central image $I_C$ and all $I_i$. The resulting warp functions, $W_{iC}$ and $W_{Ci}$, are stored in binary files and used as the input of the third module, together with all $I_i$.

The third module is the in-between image generation system. It lets the user control the blending characteristics and the preprocessing and postprocessing effects in an in-between image. To determine the blending characteristics of a uniform in-between image, the user must provide a blending vector. For a nonuniform in-between image, the user selects regions in $I_i$ and assigns them blending values. If preprocessing and postprocessing are desired, the user must specify primitives on images $I_i$ and $I'$, respectively, and move them to new positions. Once the user input is given, the third module first computes blending function $\bar{B}_C$ and warp functions $P_i$ and $Q$. The module then generates an in-between image by applying the polymorph framework to $I_i$, $W_{iC}$, and $W_{Ci}$.

The polymorph system modules are independent of each other and communicate by way of binary files storing warp functions. Any image morphing system can serve as the first module if it can save a derived warp function to a binary file. The second module does not need input images and manipulates only the binary files passed from the first module. The first and second modules together serve to compute warp functions $W_{iC}$ and $W_{Ci}$. Given input images, these modules run only once,

**16** Input images (top row) and combinations of the hair, eyes and nose, and mouth and chin regions (middle and bottom rows).

**17** Effects of varying blending values with the same selected regions.

and the derived $W_{iC}$ and $W_{Ci}$ pass to the third module. The user then runs the third module repeatedly to generate several in-between images by specifying different $\bar{B}_C$, $P_i$, and $Q$.

### *Performance*

We now discuss the performance of the polymorph system in terms of the examples already shown. The input images' resolution in Figure 2 is $300 \times 360$, and we measured the runtime on a Sun Sparc10 workstation. The first module, when applied to input image pairs $(I_0, I_1)$ and $(I_0, I_2)$, derived warp functions $W_{01}$, $W_{10}$, $W_{02}$, and $W_{20}$. The second module, which runs without user interaction, computed $W_{iC}$ and $W_{Ci}$ in 59 seconds.

The third module took seven seconds to obtain the uniform in-between image in Figure 2 from blending vector $\mathbf{b} = (1/3, 1/3, 1/3)$. A nonuniform in-between image requires more computation than a uniform in-between image because three surfaces must be constructed to determine blending function $\bar{B}_C$. It took 38 seconds to derive the in-between image in Figure 3 from the user input shown in Figure 14. To use preprocessing and postprocessing, we must compute warp functions $P_i$ and $Q$. It took 43 seconds to derive the in-between image in Figure 8, which includes the preprocessing and postprocessing effects defined by Figure 12 and Figure 13.

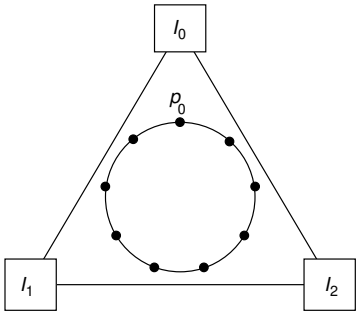
## Metamorphosis examples

The top row of Figure 16 shows the input images, $I_0$, $I_1$, and $I_2$. We selected three groups of features in these images and assigned them blending value $b_i = 1$ to generate in-between images. The feature groups consisted of the hair, eyes and nose, and mouth and chin. Each feature group was selected in a different input image. For instance, Figure 14 shows the feature groups selected to generate the leftmost in-between image in the middle row of Figure 16. Notice that the in-between image is the same as $I'$ in Figure 6. The middle and bottom rows of Figure 16 show the in-between images resulting from all possible combinations in selecting those feature groups from the input images.

Figure 17 shows the changes in in-between images when we vary the blending values assigned to selected feature regions in the input images. For example, consider the middle image in the bottom row of Figure 16. We derived that image by selecting the mouth and chin, hair, and eyes and nose from input images $I_0$, $I_1$, and $I_2$, respectively. Blending value $b_i = 1$ was assigned to each selected feature group. In Figure 17, we generated in-between images by changing $b_i$ to 0.75, 0.5, 0.25, and 0.0, from left to right and top to bottom. Note that decreasing an assigned blending value diminishes the influence of the selected feature in the in-between image. For instance, with $b_i = 0$, the selected features in all the input images vanish in the lower right in-between image in Figure 17.

In polymorph, an in-between image is represented by a point in the simplex whose vertices correspond to input images. We can generate an animation among the input images by deriving in-between images at points that constitute a path in the simplex. Figure 18 shows a metamorphosis sequence among the input images in Figure 16. We obtained the in-between images at the sample

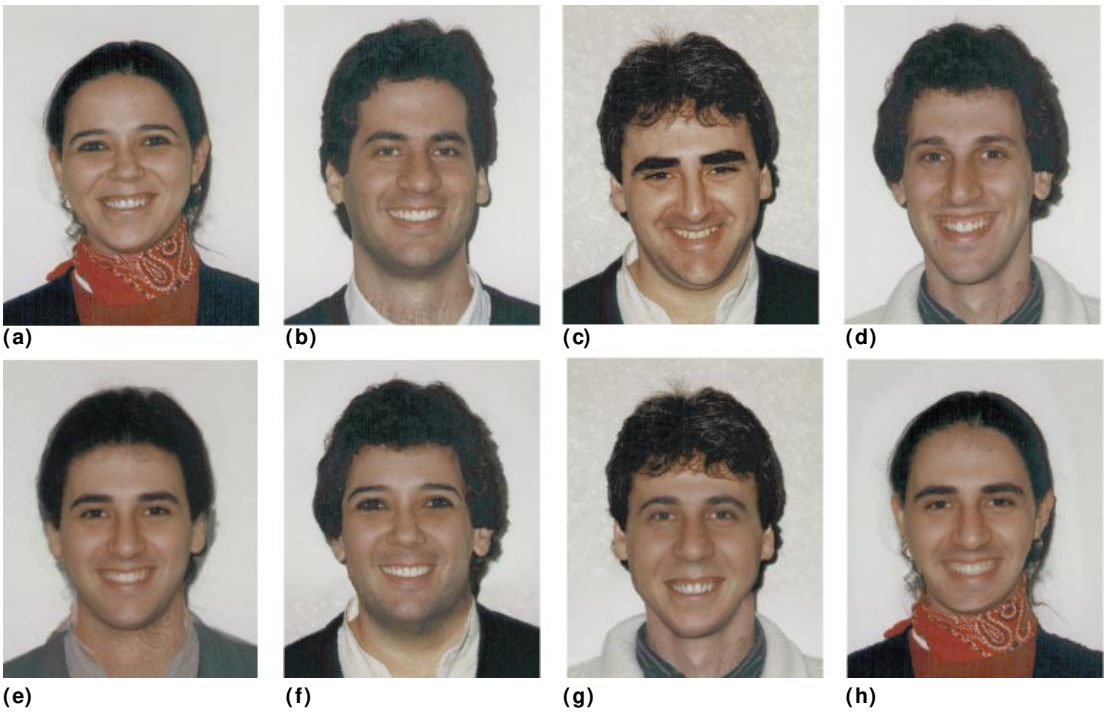
**18** A metamorphosis sequence among three images.




**19** The path and sample points for the metamorphosis sequence.

points on the circle inside the triangle shown in Figure 19. The top left image in Figure 18 corresponds to point $p_0$. The other images, read from left to right and top to bottom, correspond to the points on the circle in counterclockwise order. Every in-between image blends the characteristics of all the input images at once. The blending is determined by the position of the corresponding point relative to the vertices in the triangle. For example, input image $I_0$ dominates the features in the top left in-between image in Figure 18 because point $p_0$ lies close to the vertex of the triangle corresponding to $I_0$.

Figure 20 shows metamorphosis examples from four input images. The input images appear in the top row. Figure 20e is the central image, a uniform in-between




**20** Input images (top row) and in-between images from them (bottom row).

image generated by blending vector (1/4, 1/4, 1/4, 1/4). In Figure 20f, the hair, eyes and nose, mouth and chin, and clothing were derived from Figures 20d, 20a, 20b, and 20c, respectively. We rotated and enlarged the eyes and nose in Figure 20a in the preprocessing step to match them with the rest of the face in Figure 20f. The eyes and nose in Figure 20g resulted from selecting those in Figures 20b and 20d and assigning them blending value $b_i = 0.5$. The hair and clothing in Figure 20g were derived from Figure 20c and 20d, respectively. In Figure 20h, we retained the hair and clothing from Figure 20a. The eyes, nose, and mouth were blended from Figures 20b, 20c, and 20d with $b_i = 1/3$. The resulting image resembles a man with a woman's hairstyle and clothing.

## Discussion

In this section, we discuss the application of polymorph in feature-based image composition and extensions of the implemented system.

### *Application*

Polymorph ideally suits image composition applications that seamlessly blend elements from two or more images. The traditional view of image composition is essentially one of generalized cut-and-paste. That is, we cut out a region of interest in a foreground image and identify where to paste it in the background image. Composition theory permits several variations for blending, particularly for removing hard edges. Although image composition is widely used to embellish images, current methods are limited in several respects. First, composition is generally a binary operation restricted to only two images at a time—the foreground and background elements. In addition, geometric manipulation of the elements is not effectively integrated. Instead, it is generally handled independently of the blending operations.

Our examples demonstrated the use of polymorphing for image composition. We extended the traditional cut-and-paste approach to effectively integrate geometric manipulation and blending. A composite image is treated as a metamorphosis between selected regions of several input images. For example, consider the regions selected in Figure 14. Those regions seamlessly blend together with respect to geometry and color to produce the in-between image in Figure 6. That image would otherwise be considerably more cumbersome to produce using conventional image composition techniques.

### *Extensions*

In the polymorph system, we use warp propagation to obtain warp functions $W_{ij}$ for all pairs of input images. To enable the propagation, we need to derive warp functions associated with the edges selected to make graph $G$ connected. The selected edges in $G$ are independent of each other in computing the associated warp functions. We can specify different feature sets for different pairs of input images to apply different warp generation algorithms. This permits the reuse of feature correspondence previously established for a different application, such as morphing between two images. Also, simple transformations like an affine mapping may serve to represent warp functions between an image pair when appropriate.

We derive warp functions $W_{ij}$ between input images to compute warp functions $W_{iC}$ and $W_{Ci}$. Suppose that we specified the same set of features for all input images. For example, given face images, we can specify the eyes, nose, mouth, ears, and profile of each input face $I_i$ as its feature set $F_i$. In this case, $W_{iC}$ and $W_{Ci}$ can be computed directly without deriving $W_{ij}$. That is, we compute the central feature set $F_C$ by averaging the positions of feature primitives in $F_i$. We can then derive $W_{iC}$ and $W_{Ci}$ by applying a warp generation algorithm to the correspondence between $F_i$ and $F_C$ in both directions.

With the same set of features on input images, we can derive warp functions $\overline{W}_i$ for a uniform in-between image in the same manner as $W_{iC}$. Given a blending vector, we derive an in-between feature set $F$ by weighted averaging feature sets $F_i$ on input images. $\overline{W}_i$ can then be computed by a warp generation algorithm applied to $F_i$ and $F$. With this approach, we do not need to maintain $W_{iC}$ and $W_{Ci}$ to compute $\overline{W}_i$ for a uniform in-between image. However, this approach requires a warp generation algorithm to run $n$ times whenever an in-between image is generated. This takes more time than the approach we described using $W_{iC}$ and $W_{Ci}$. In the polymorph system, once we have derived $W_{iC}$ and $W_{Ci}$, we can quickly compute $\overline{W}_i$ by linearly interpolating warp functions and applying function compositions.

## Conclusions

Polymorph provides a general framework for morphing among multiple images. We extended conventional morphing to derive in-between images from more than two images at once. This paradigm requires feature specification among only $(n-1)$ pairs of input images, a significant savings over all $n(n-1)/2$ pairs. The use of preprocessing and postprocessing stages accommodates fine control over the scaling and positioning of selected input regions. In this manner we resolve conflicting positions of selected features in input images when they are blended to generate a nonuniform in-between image.

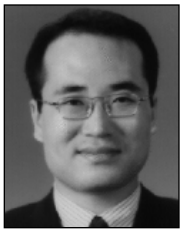
Polymorph is ideally suited for image composition applications where elements from multiple images are blended seamlessly. A composite image is treated as a metamorphosis between selected regions of input images. Future work remains in simplifying the feature specification process through the use of snakes[3] and intelligent scissors.[10] ■

## Acknowledgments

Seungyong Lee performed work on this article while he was a visiting scholar in the Department of Computer Science at the City College of New York. This work was supported in part by NSF Presidential Young Investigator award IRI-9157260, PSC-CUNY grant RF-667351, and Pohang University of Science and Technology grant 1RB9700601.

## References

1. G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, Calif., 1990.
2. T. Beier and S. Neely, "Feature-Based Image Metamorphosis," *Computer Graphics* (Proc. Siggraph 92), Vol. 26, No. 2, 1992, pp. 35-42.
3. S.-Y. Lee et al., "Image Metamorphosis Using Snakes and Free-Form Deformations," *Proc. Siggraph 95*, ACM Press, New York, 1995, pp. 439-448.
4. S. Lee et al., "Image Morphing Using Deformation Techniques," *J. Visualization and Computer Animation*, Vol. 7, No. 1, 1996, pp. 3-23.
5. D.A. Rowland and D.I. Perrett, "Manipulating Facial Appearance through Shape and Color," *IEEE Computer Graphics and Applications*, Vol. 15, No. 5, 1995, pp. 70-76.
6. J.F. Hughes, "Scheduled Fourier Volume Morphing," *Computer Graphics* (Proc. Siggraph 92), Vol. 26, No. 2, 1992, pp. 43-46.
7. A. Lerios, C.D. Garfinkle, and M. Levoy, "Feature-Based Volume Metamorphosis," *Proc. Siggraph 95*, ACM Press, New York, 1995, pp. 449-456.
8. P. Litwinowicz and L. Williams, "Animating Images with Drawings," *Proc. Siggraph 94*, ACM Press, New York, 1994, pp. 409-412.
9. S. Lee, G. Wolberg, and S.Y. Shin, "Scattered Data Interpolation with Multilevel B-splines," *IEEE Trans. Visualization and Computer Graphics*, Vol. 3, No. 3, 1997, pp. 228-244.
10. E.N. Mortensen and W.A. Barrett, "Intelligent Scissors for Image Composition," *Proc. Siggraph 95*, ACM Press, New York, 1995, pp. 191-198.


***Seungyong Lee*** *is an assistant professor in the Department of Computer Science and Engineering at Pohang University of Science and Technology (Postech), Korea. He received his BS in computer science and statistics from Seoul National University, Korea, in 1988, and his MS and PhD in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1990 and 1995, respectively. His research interests include computer graphics, computer animation, and image processing.*



***George Wolberg*** *is an associate professor in the Department of Computer Science at the City College of New York. He received his BS and MS in electrical engineering from Cooper Union in 1985, and his PhD in computer science from Columbia University in 1990. His research interests include image processing, computer graphics, and computer vision. Wolberg received a 1991 NSF Presidential Young Investigator Award and the 1997 CCNY Outstanding Teaching Award. He is the author of* Digital Image Warping *(IEEE Computer Society Press, 1990).*



***Sung Yong Shin*** *is a full professor in the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea. He received his BS in industrial engineering in 1970 from Hanyang University, Seoul, Korea, and his MS and PhD in industrial and operations engineering from the University of Michigan, Ann Arbor, USA, in 1983 and 1986, respectively. His research interests include computer graphics and computational geometry.*

*Contact Lee at the Department of Computer Science and Engineering, Postech, Pohang, 790-784, S. Korea, e-mail leesy@postech.ac.kr.*

*Contact Wolberg at the Department of Computer Science, City College of New York, 138th St. at Convent Ave., New York, NY 10031, e-mail wolberg@cs-mail.engr.ccny.cuny.edu.*

*Contact Shin at the Dept. of Computer Science, KAIST, Taejon, 305-701, S. Korea, e-mail syshin@jupiter.kaist.ac.kr.*