Multidimensional Image Morphs: Construction and User Interface

by

Matthew R. Peters

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
[February 2003]
Jan 2003

Author .....
Department of Electrical Engineering and Computer Science
Jan 20, 2003

Certified by .....
Leonard McMillan
Associate Professor
s Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Multidimensional Image Morphs: Construction and User Interface

by

Matthew R. Peters

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 20, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis, I design and implement a system that creates multidimensional image morphs. Multidimensional image morphs are images constructed by cross-fading an arbitrary number of input images after alignment. This alignment is specified by corresponding features. The feature correspondences used by my system are given as a collection of digital image morphs between pairs of images. These two-image morphs are converted to global feature alignment by chaining and merging morph pairs. The feature alignment in each image is converted into high-dimensional vectors, and I describe a process to produce novel images using combinations of these vectors. Specifications of the images produced by my system are given by increasing or decreasing user-intuitive traits. I apply my implementation of this system to three sample datasets and discuss the quality of the resultant output images.

Thesis Supervisor: Leonard McMillan
Title: Associate Professor

Acknowledgments

Thanks to Howard Chan for creating the datasets and helping to implement the user interface of the system.

Images of dogs in this work are courtesy of the American Kennel Club, and reproduction of these images in any subsequent publication or medium is not permitted without explicit permission of the AKC.

Contents

List of Figures

Chapter 1

Introduction

A developing problem in computer graphics is sample-based images: how to create new images based on a group of sample images. The output images must appear to be similar to the examples (i.e. if dogs are pictured, then the new images should be pictures of dogs). It is also important that the constructed images can show the full variability of the space of input images (showing pictures of poodles, Saint Bernards, and anywhere in between). Since the constructed images must be created according to artistic specification, user input controls must be available to interactively change the properties of the constructed image.

There are two major applications of sample-based images. The first is animations of transformations from one image to another. The frames illustrating a transformation must show a continuous reshaping to look realistic. Cross-fading two images or wiping from one to another does not give the same sense of transformation that growing, shrinking, or reshaping physical features does. In many transformations, the animation can be expressed as becoming more or less like one example image or another. Creating the in-between frames of such transformations is an application of sample-based images.

The second application is generating novel images. In games, movies, or other graphical simulations, it is often useful to be able to create new images of a certain class: hundreds of generic faces for an in-game database, or an illustration of a new species of fish, or a depiction of a fantasy car. Sample-based images can provide new images of the same general appearance as a group of input images. User input controls allow changing the constructed image to meet user requirements such as a larger nose on a face, more prominent scales on a fish, or an SUV body on an automobile.

A partial solution to sample-based images is well-known: digital image morphing. Digital image morphs require feature correspondences between two images. In-between images are created by interpolating the feature positions, warping each image to align features with the in-between positions, and then cross-fading the two aligned images. Images created in this manner can animate a smooth transformation sequence between the two input images, when the pictured objects have similar geometry. In the transformation, large-scale geometry appears to shift, or bulge in or out, and on the smaller scale, the texture fades from one image to another as the transformation progresses. In-between images also serve as novel images that resemble the two input

images. This solution is limited, however, since only two images can be combined at once.

Recently, a more general solution to the problem has been developed: multidimensional morphs. Multidimensional morphs specify corresponding features in an arbitrary number of input images. Multiple source images are warped to a common shape and cross-faded together to make an output image. The position of the features in the aligned image are interpolated from their positions in the input images. When the shapes of input images are similar (i.e. all are pictures of dogs taken from the same angle), multidimensional morphs create images that retain this same general shape. The combinations of input images chosen to dominate the shape and texture provide variability in the appearance of the output image.

In this thesis, I describe an algorithm for creating multidimensional morphs. My system differs from previous implementations of multidimensional morphs in several areas. Multidimensional morphs can be initialized from a set of morphs between pairs of images. I refer to this process as morph registration. My implementation of morph registration is a significant extension to previous works. I consider redundancies in the graph of input two-image morph pairs, and variability in the quality of the morphs. After morph registration, the shape of new images and the coefficients of the cross-fade of the warped input images must be specified. I discuss linear and non-linear techniques for this specification based on dimensionality reduction; this is an improvement over the process used in previous works. And finally, specification of new images has previously required manual adjustment of non-intuitive values. I describe a better system based on traits: the user can specify arbitrary properties of the output images to increase or decrease.

The layout of this thesis proceeds as follows. In Chapter 2 I discuss the background of morphing, and more recent work dealing with multidimensional morphs. In Chapter 3 I define terms used throughout the description of the algorithm, and introduce some useful mathematical tools for dimensionality reduction. The description of the algorithm is given in three parts, in chapter 4 through Chapter 6. Chapter 4 describes morph registration: how to construct feature alignment in all images by chaining and merging paths in a connected graph of standard image morphs. The chapter also describes how the feature alignment for each image may be represented as a vector for use in the later steps. Chapter 5 describes how dimensionality reduction is used to represent these vectors as points in an embedding space. Any point in the embedding space can then be reprojected to produce image vectors, which are used to specify a novel image. Chapter 6 discusses the user input tools, and most especially traits: how they are input into the system, the mathematical problem they represent, and how the problem may be solved. Chapter 7 gives the details of my implementation of the algorithm I describe, and the results of its application on sample datasets. Finally, Chapter 8 covers my conclusions, and describes possibilities for future work.

Chapter 2

Background and Related Work

In this chapter, I will first discuss techniques used for digital image morphing of two input images. I then cover some extensions of two-image morphing to handle changes in camera angle. Finally, I cover recent works describing methods to extend image morphing to multidimensional morphs, which combine more than two input images.

2.1 Digital Image Morphing

Digital image morphing was developed in the 1980's to blend images. A more naive technique is cross-fading: in-between pixels are averages of the red, blue and green values in each of the input images. Naively cross-fading images has the obvious flaw of *ghosting*: unaligned features fade in or out, creating an unrealistic blurry appearance. A cross-fade between two breeds of dog - a border terrier (left) and a weim (right) - is illustrated in Figure 2-1.

Morphing solves this problem by first aligning the features and then cross-fading. Features are aligned by *warping* each of the input images so that each member of a set of features is in a specified position. In Figure 2-2, the tail of the border terrier is slimmed and translated, the snout of the weim is shortened, and so on, so that the features of each dog are in the same position. These warped images are shown on the bottom left and bottom right, respectively. The cross-fade of these two images is

Figure 2-1: A cross-fade of two dogs. The images on the left and right are merged in a 50/50 cross-fade. Images courtesy of the American Kennel Club.

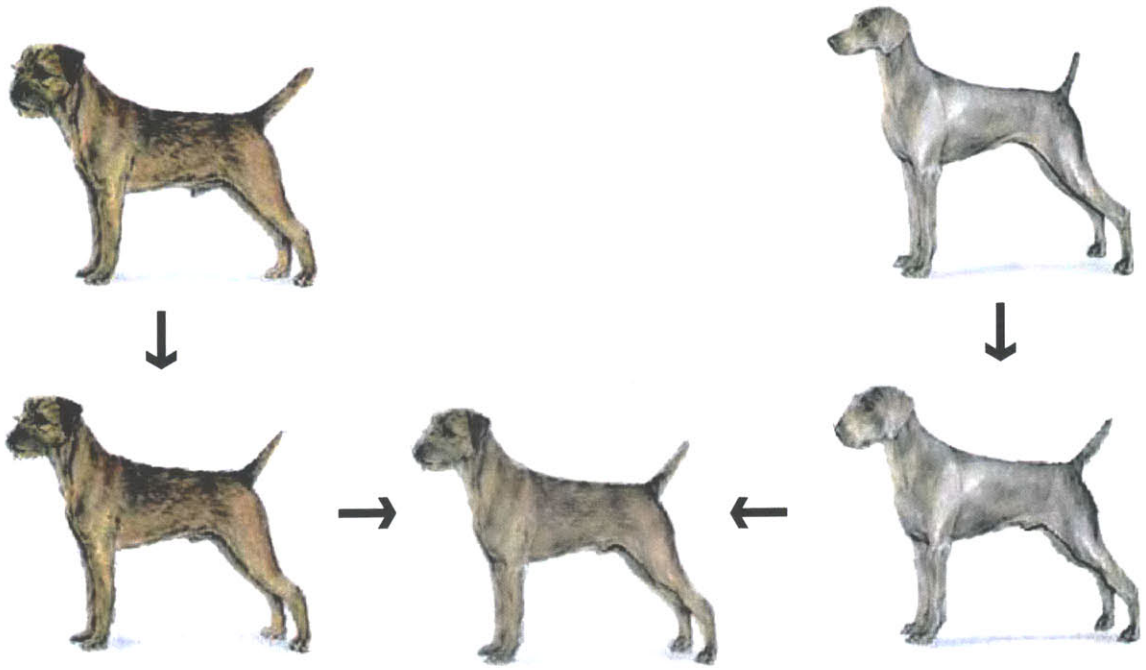

Figure 2-2: A morph of two dogs. The images on the top left and top right are first warped to align them with a common position, and then these warped images are merged in a 50/50 cross-fade. Images courtesy of the American Kennel Club.

shown in the bottom center.

Several types of image morphing have been developed and are in use. They differ primarily in two areas: the type of control features used to specify the warp (corresponding points, lines, and/or curves), and construction of the warp function from the control features (linear warps of mesh simplices, field warping, and scattered data interpolation). Otherwise, the morphing algorithms are nearly identical, solving the common problems of (1) creating the in-between control features (2) sampling the warped images and (3) cross-fading the warped images.

2.1.1 Canonical Morphing Subproblems

Creating In-between Control Features

To create an in-between morphed image, the two input images are warped to a common set of feature positions. Features may be corresponding points, lines, or curves. Since the in-between image should look like a combination of the two input images, the in-between features should be closely related to both sets of input image control features. In most implementations, the in-between feature positions are taken to be the average of the input positions. The in-between features are characterized by a variable α: $0 \leq \alpha \leq 1$. $\alpha = 0$ is the same set of features as in the first image, and $\alpha = 1$ is the same set of features as in the second image.

For a corresponding point feature, the in-between position $\mathbf{p}_{between}$ is given by

$$\mathbf{p}_{between} = (1 - \alpha) * \mathbf{p}_1 + \alpha * \mathbf{p}_2. \tag{2.1}$$

Corresponding line features are handled in one of two ways. In some cases, the segment endpoints are treated as point features, and the in-between point feature positions are connected to create the in-between line segment feature. In other cases, the segment midpoint is treated as a point feature. The segment length and angle are then averaged in the same way as point features:

$$\theta_{between} = (1 - \alpha) * \theta_1 + \alpha * \theta_2 \tag{2.2}$$

$$|l|_{between} = (1 - \alpha) * |l|_1 + \alpha * |l|_2 \tag{2.3}$$

$$\mathbf{p}_{mid,between} = (1 - \alpha) * \mathbf{p}_{mid,1} + \alpha * \mathbf{p}_{mid,2} \tag{2.4}$$

$$\mathbf{p}_{begin} = \mathbf{p}_{mid} + \frac{|l|}{2} * \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{2.5}$$

$$\mathbf{p}_{end} = \mathbf{p}_{mid} - \frac{|l|}{2} * \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{2.6}$$

Whether lines are treated as connected endpoints or as center, rotation, and length is an artistic choice.

The most general form of corresponding features are arbitrary corresponding curves (Bézier curves are a common example). In-between curve features are created by treating curves as a list of corresponding points. Each member of a corresponding pair of curves is sampled into points (the same number of points for each curve). These points are treated as point features, and the in-between positions of the point features are defined by Equation 2.1. The in-between curve feature is then interpolated from the in-between points.

The Sampling Problem

Most warping functions are continuous functions of a continuous domain (pixel correspondence warps are an exception.) Creating a set of pixels for a warped image requires solving the sampling problem - how should pixel colors in the destination image be calculated from the warp function and the pixels of the source image. The two common techniques for solving the sampling problem are forward mapping and reverse mapping.

In forward mapping, every pixel in the input image is treated as a quadrilateral, and the warping function determines a warped quadrilateral region inhabited by the pixel. The color of each destination pixel is the weighted sum of the colors of the warped source pixels which touch the destination pixel's square. The weight of each source pixel in the sum is the percent of the destination square occupied by the warped source pixel. The weights are normalized so that they sum to one. I discuss summing pixel colors when describing cross-fading below.

In reverse mapping, each pixel in the destination image is mapped to a point in the input image, using the inverse warp. If the inverse warp is not obtainable,

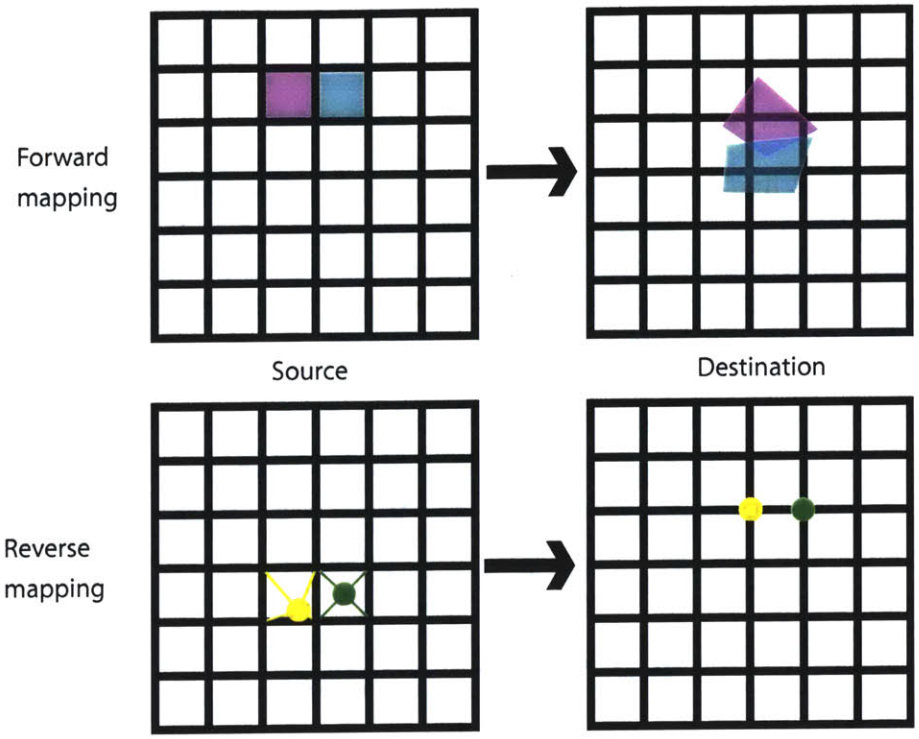

Figure 2-3: In forward mapping (top), a pixel in the source image is warped to a quadrilateral in the destination image and contributes to the color of every destination pixel the quadrilateral contains. In reverse mapping (bottom), a pixel in the destination image is warped to a point in the source image, and the neighboring pixels of that point contribute to its color.

reverse mapping is not possible. The color of the destination pixel is the average of the four source pixels surrounding the warped point. This average is weighted by the Manhattan distance of the warped point to the center of each pixel. This weighted sum is equivalent to bilinear interpolation.

The sampling problem - as well as anti-aliasing and sophisticated methods to average pixels - is discussed in *Digital Image Morphing* by George Wolberg[19].

Cross-fading

Image morphing produces two warped input images. These images are combined into one output image by cross-fading. The height and width of the images are set equal; pixels outside the bounds of the original images are set to black. Then the cross-faded pixel at (x, y) is the average of the (x, y) pixels at each image. In color images, each color component (red, blue, and green in RGB; hue, saturation and luminance in HSV) is averaged separately. The average of the source pixels is weighted by a variable β. The values of β are from 0 to 1 and have similar meanings to the values of the in-between feature positions variable α. When β is 0, the output pixels are determined entirely by the first image, and when β is 1, the pixels are determined

entirely by the second image. In general,

$$\begin{aligned} \text{red}_{crossfade} &= (1-\beta) * \text{red}_1 + \beta * \text{red}_2 \\ \text{green}_{crossfade} &= (1-\beta) * \text{green}_1 + \beta * \text{green}_2 \\ \text{blue}_{crossfade} &= (1-\beta) * \text{blue}_1 + \beta * \text{blue}_2. \\ & 0 \leq \beta \leq 1 \end{aligned} \tag{2.7}$$

A similar formula holds for hue, saturation, and luminance. When creating two-image morphs, β is often but not necessarily set equal to α.

Many operations used in two-image morphs use this same process for cross-fading pixels. The sampling problem described above is one example. Whenever pixels are "averaged", they are combined by cross-fading. When more than two pixels are combined, the generalized equivalent to the variable β is a vector $\vec{\beta}$. The elements of this vector must be non-negative and usually sum to one. (When using RGB values, if the elements do not sum to one then the luminance of the cross-faded pixel is multiplied by the sum of the elements. In some situations this is acceptable.)

2.1.2 Mesh Warping

Quadrilateral Mesh Warping

One of the first techniques for morphing was quadrilateral mesh warping. This type of morphing is attributed to George Wolberg. Industrial Light and Magic used this method to create a transformation sequence from animal to animal in the movie Willow in 1988 [20]. The user supplies a set of point features in one image and a corresponding set of point features in the other image. The point features in each image are referred to as mesh points. The user also supplies the connectivity of the mesh points in one of the images, creating a quadrilateral mesh. The same connectivity is used to create a mesh in the other image. All mesh points must be unique in both images, and no mesh quadrilaterals may overlap. An example of quadrilateral mesh input is given in Figure 2-4.

An in-between set of mesh points is created by averaging the mesh points of each image (the standard for point features). These points define a warp for each of the input images. The domain of the warp is constrained to lie within the quadrilateral mesh of the source image. (Note that this does not necessarily include all points within the convex hull of the mesh points.) The warped position of each mesh point is already specified by the in-between mesh points. All other points are in the interior or on the perimeter of a mesh quadrilateral. Let $\mathbf{p}$ be a point in the input image, and let Q_p be the quadrilateral containing $\mathbf{p}$. Let the vertices of Q_p be v_{00}, v_{01}, v_{11}, and v_{10}, labeled in counter-clockwise order. The choice of v_{00} is inconsequential. There is a unique two-dimensional projective transform that takes the vertices of Q_p onto the vertices of the unit square so that v_{00} goes to (0,0), v_{10} goes to (1,0), v_{11} goes to (1,1), and v_{01} goes to (0,1). Applying this transformation to $\mathbf{p}$, yields the normalized coordinate (x, y), $0 \leq x \leq 1$ and $0 \leq y \leq 1$. $\mathbf{p}^w$ - the warped position of $\mathbf{p}$ - is then

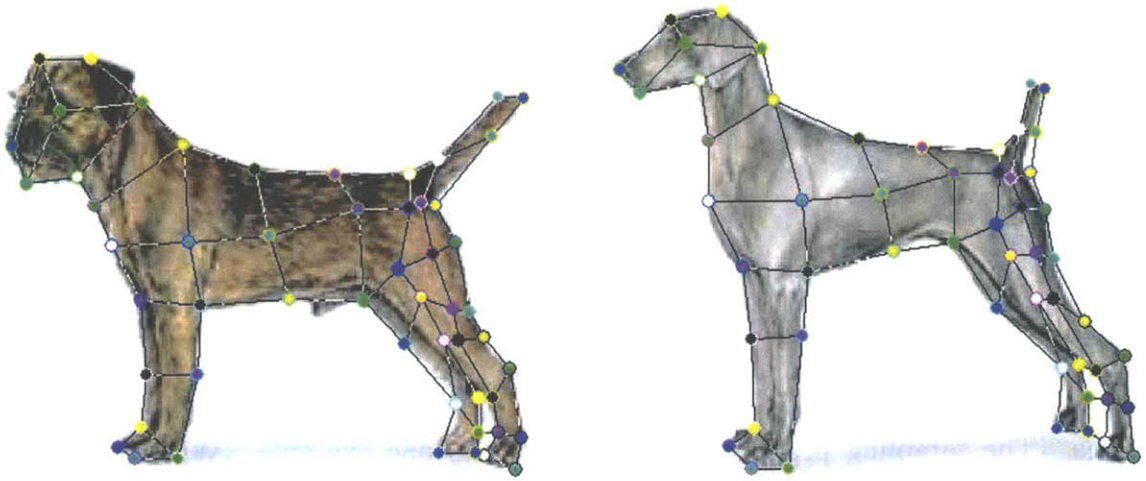

Figure 2-4: Corresponding quadrilateral meshes over two dogs. Images courtesy of the American Kennel Club.

given by bilinear interpolation of the warped position of the vertices:

$$\mathbf{p}^w = (1-y) * ((1-x) * \mathbf{p}^w_{00} + x * \mathbf{p}^w_{10}) + y * ((1-x) * \mathbf{p}^w_{01} + x * \mathbf{p}^w_{11}). \tag{2.8}$$

This method applies to points on a mesh boundary as well as to the interior. At a mesh boundary, the two possible warped positions for a point given by the two adjacent quadrilaterals are equal.

When solving the sampling problem with quadrilateral mesh warping, reverse mapping requires a warp from the in-between image to the input image, and hence an in-between mesh. This mesh is usually taken to have the same connectivity as the mesh in the input images. Careful attention by the artist is necessary to make sure that overlapping quadrilaterals in this mesh do not adversely affect the output image. Quadrilateral mesh warping is discussed further in *Digital Image Warping* by George Wolberg [19].

Triangle mesh warping

A technique similar to quadrilateral mesh warping is triangle mesh warping, which can be attributed to several sources, including A. Goshtasby in 1986 as one of the earliest [6]. Triangle mesh warping is identical to quadrilateral mesh warping, except that the mesh is composed of triangles instead of quadrilaterals. A major benefit of triangle mesh warping is that no mesh needs to be supplied by the user, since there are robust methods for finding a triangulation of a set of points. The triangulation used most often is the Delaunay triangulation, which avoids long, thin triangles.

The warped position of a sample point inside (or on the perimeter of) a triangle is defined by the barycentric coordinates of the point. The barycentric coordinates of a point in a triangle are defined such that the average of the vertices, weighted by the coordinates, equals the point. The barycentric coordinates must also sum to one, making them unique (assuming the triangle has non-zero area). In triangle mesh

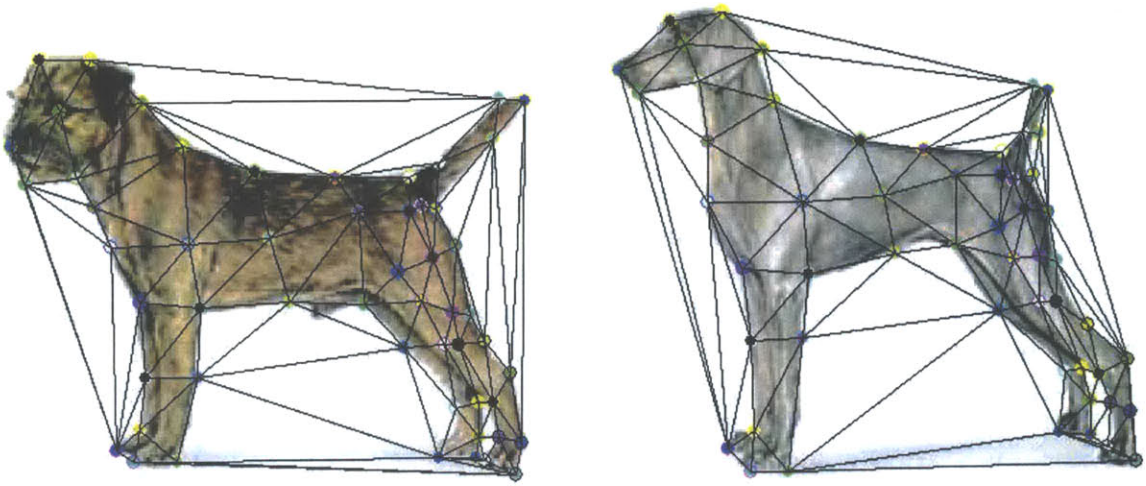

Figure 2-5: Delaunay triangulations of the mesh points used in the quadrilateral meshes of Figure 2-4. Images courtesy of the American Kennel Club.

warping, the warped position of a sample point is the average of the warped vertices of the triangle, weighted by the barycentric coordinates of the sample point in the unwarped triangle. This is equivalent to transforming each point by the simplitude transform uniquely defined by the warped triangle vertices.

When using reverse mapping, triangle mesh warps are equivalent to texture mapping each triangle in the destination image: the source image is the texture map, and the source triangle vertices are the texture coordinates. Thus triangle mesh warps can be easily accelerated by taking advantage of graphics hardware.

Triangle mesh warps can also support line features as input. Corresponding line segments are interpreted as a constraint that the points on the segment in one image correspond exactly to the points on the corresponding segment in the other image. This constraint can be supported by constrained Delaunay triangulations, which find a Delaunay triangulation in which the constrained segments are edges. (Constrained segments may need to be subdivided into smaller constrained segments to preserve the Delaunay property.)

Ruprecht and Muller discuss triangle mesh interpolation in *Image warping with scattered data interpolation* [13].

2.1.3 Pixel correspondence warps

Often warps need to be generated without the aid of an artist. Generation of these warps is done using optical flow. An optical flow algorithm compares two images and returns a best mapping of the pixels in one image to a nearby pixel in image 2 with the same color. The warps generated by optical flow are called pixel correspondence warps.

Pixel correspondence warps are fundamentally defined in the discrete domain, so that the sampling problem is entirely avoided. For theoretical purposes, however, the continuous warping function given by a pixel correspondence warp is well defined.

The warped position of (x, y) is given by adding to (x, y) the same translation given to $(\lfloor x \rfloor, \lfloor y \rfloor)$ - the pixel containing (x, y). That is,

$$\begin{bmatrix} x \\ y \end{bmatrix}^w = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \lfloor x \rfloor \\ \lfloor y \rfloor \end{bmatrix}^w - \begin{bmatrix} \lfloor x \rfloor \\ \lfloor y \rfloor \end{bmatrix} \tag{2.9}$$

where $(x, y)^w$ is the warped position of (x, y).

Pixel correspondence warps are accelerated because they do not have to solve the sampling problem: each pixel in the destination image corresponds to the entirety of exactly one source image pixel. However, when constructed from optical flow they are restricted in use: the images compared must be highly similar for optical flow to yield a reasonable morph. Pixel correspondence warps are sometimes constructed by resampling other warps: the corresponding pixel of a source pixel is the nearest destination pixel to the pixel's warped position. The resampled pixel correspondence warp is less computationally expensive, but is of lower quality, since it assumes a trivial (non-interpolating) solution to the sampling problem.

2.1.4 Beier-Neely warping

Beier-Neely warps, or field warps, define a continuous warp function over an entire plane. Beier-Neely warping was described in a 1992 Siggraph paper by Thaddeus Beier and Shawn Neely [1]. The Beier-Neely warp function is based on line features (a point feature can be represented as a degenerate case of a line feature).

Each line correspondence specifies a warp for the entire plane in the following manner. Let the directed line segment l be defined by start point $\mathbf{o}$ and direction $\mathbf{v}$. Let $\hat{\mathbf{v}}$ be the normalized $\mathbf{v}$, and let $\hat{\mathbf{v}}_\perp = \text{Rot}(\hat{\mathbf{v}}, 90°)$, be the normalized perpendicular (arbitrarily chosen to be rotated by 90° instead of by −90°). A source point $\mathbf{p}$ is projected onto the coordinate system $(\mathbf{o}, \mathbf{v}, \hat{\mathbf{v}}_\perp)$ given by the line l:

$$\begin{aligned} x_l &= \tfrac{1}{|\mathbf{v}|} \; (\mathbf{p} - \mathbf{o}) \cdot \hat{\mathbf{v}}_l \\ x_\perp &= \qquad (\mathbf{p} - \mathbf{o}) \cdot \hat{\mathbf{v}}_\perp. \end{aligned} \tag{2.10}$$

These coordinates $(x_l, x_\perp)$ are then used as coordinates in the corresponding line's reference frame to specify the warped position $\mathbf{p}^w$ given by the line. Let $\mathbf{o}^w$ be the warped position of $\mathbf{o}$, and let $\mathbf{v}^w$ and $\hat{\mathbf{v}}_\perp^w$ be the warped line and normalized perpendicular vectors. Then

$$\mathbf{p}^w = \mathbf{o}^w + x_l \cdot |\mathbf{v}^w| \cdot \hat{\mathbf{v}}^w + x_\perp * \hat{\mathbf{v}}_\perp^w. \tag{2.11}$$

Since the axis parallel to the line is unnormalized and the axis perpendicular to the line is normalized, Beier-Neely line features stretch space in the parallel but not the perpendicular direction. Beier-Neely lines features rotate, translate, and scale points in the plane.

In the presence of multiple lines, the final warped position for $\mathbf{p}$ is an average of the warped positions ${\mathbf{p}^w}_l$ for each line l. The weight given to $\mathbf{p}_l^w$ in the average is some function inversely related to the distance of $\mathbf{p}$ from l. This function has several

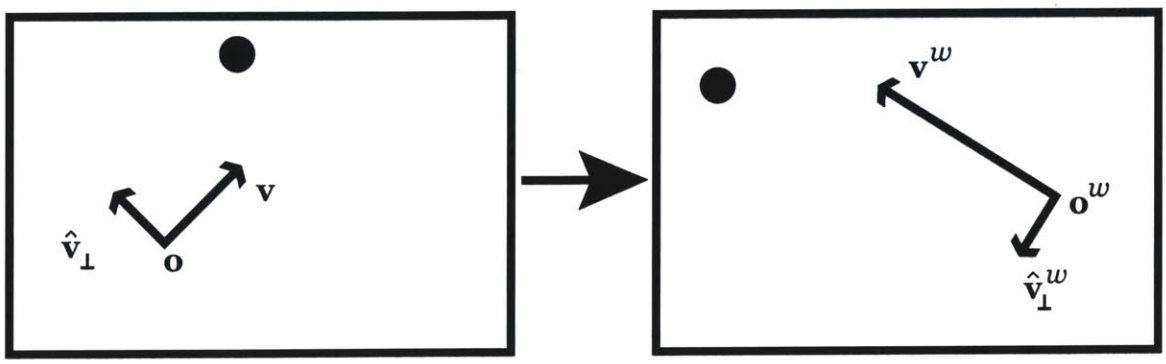

Figure 2-6: A diagram of the warp of a point given by a single Beier-Neely line correspondence. Note that the distance of the point from the line is not stretched (the absolute distance is unchanged), while distance along the line is stretched (the ratio of the distance to the line length is unchanged.

parameters that are hand-tuned. Because each line specifies a "field of influence", Beier and Neely refer to this warp style as "field morphing".

Beier-Neely warps are continuous, infinitely differentiable except at singularities along line features, and are infinite in extent. Thus, they are more robust than mesh warps or pixel correspondence warps. This comes at the cost of additional computational complexity.

2.1.5 Other methods

More sophisticated warps than mesh warps and Beier-Neely warps have been developed. Many of these techniques rely on scattered data interpolation. A set of point correspondences is viewed as two bivariate functions: $x_{dest}(x, y)$ and $y_{dest}(x, y)$. With this formulation, the tools of scattered data interpolation can be used to specify the warped position of $(x_{dest}(x, y), y_{dest}(x, y))$ of all source points. An overview of techniques using scattered point interpolation is given by Ruprecht and Miller [13]. George Wolberg gives an overview of warp functions implemented using radial basis functions, thin plate splines, and a method based on energy minimization [20]. In all of these techniques, control features are sampled into points and treated as a scattered data problem.

2.1.6 Problems for warps

Certain characteristics of warping functions can lead to visually poor results.

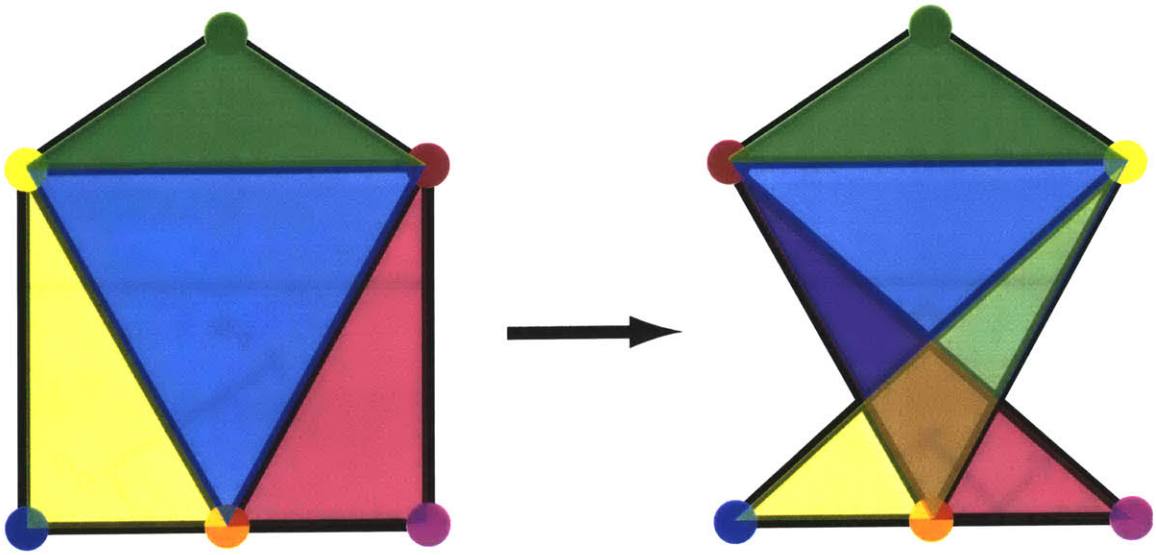

Figure 2-7: A simple example of foldover.

Areas of derivative discontinuity in the warping function, if large enough, can be seen with the naked eye. This problem is common in mesh warps: the boundary of a mesh simplex may be visible in the output. This problem can be hidden by increasing the density of the mesh (usually around 30 pixels per triangle in a triangle mesh warp is more than sufficiently dense.) It can also be solved algorithmically by using more sophisticated interpolation techniques such as cubic splines. Higher-degree interpolation polynomials have enough degrees of freedom to set continuous boundary derivatives. This line of thought leads to the warps based on scattered point interpolation.

Another warp problem is foldover, in which multiple source points are warped to the same destination point. In mesh warps, this occurs when mesh simplices in the output image overlap. In Beier-Neely warps, intersecting or near-intersecting line features can cause foldover. The visual effect of foldover is that a portion of the warped image appears to be occluded, as if part of the image were folded over another part. Except for certain types of scattered point interpolation warps based on energy minimization functions [20], it is up to the artist to avoid foldover. See Figure 2-7 for an illustration of foldover with a triangle mesh. Figure 2-8 shows the visual effect of foldover in an actual morph.

A third problem particular to mesh warps is holes in the output image. When using forward mapping, it is not guaranteed that every output pixel will be filled by a warped input pixel. In particular, only points inside the convex hull of the warped point positions will be filled. The restriction to the convex hull is also present when using reverse mapping. Holes in the interior of the image (when using forward mapping) can be fixed either by filling in with background pixels or by interpolating neighboring pixels. For missing pixels outside the convex hull, it is possible to define warped positions that vary smoothly to infinity, but these are often not the results the artist would expect. Most commonly, the artist providing the warp avoids input

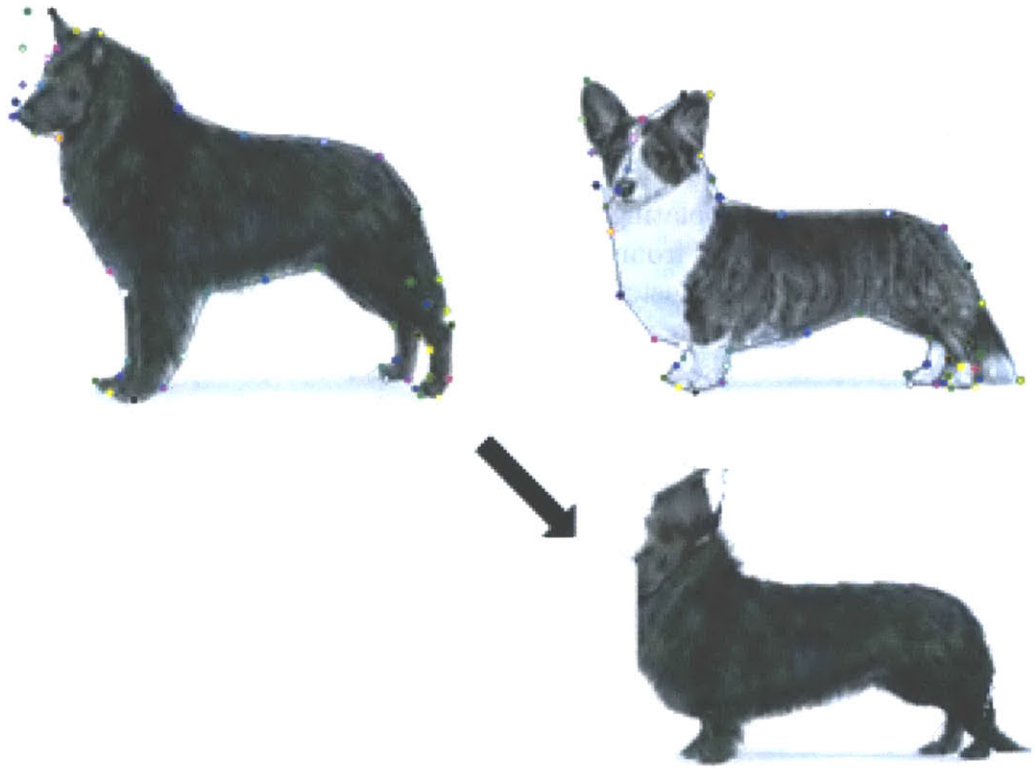

Figure 2-8: Foldover in a triangle mesh warp. In this rather severe example, the user has attempted to fix differences in the three-dimensional geometry of the dogs' poses by putting in contradictory corresponding points on both the head of the left dog and the tail of the right dog. When the left dog is warped to the feature positions of the right dog, foldover of the dog's tail and leg is visible. Images courtesy of the American Kennel Club.

which leads to holes.

2.2 View Morphing

View morphing is an extension to digital image morphing designed to handle changes in camera pose. Seitz and Dyer defined and demonstrated the problem raised by changes in camera pose in their 1996 paper *View Morphing* [14].

Consider two images of the same object taken from different camera positions. In a standard morph, features of the object (points, lines, or curves) are selected and their positions in both images are marked. An in-between image of the object is calculated by linearly interpolating the feature positions and using these in-between positions to construct a warp from each of the input images to the in-between image. A more principled way to construct the in-between image would be to reposition the camera into an in-between pose obtained by smoothly rotating, translating and zooming, and then to retake the picture (or rerender the scene). To demonstrate that these two methods are not, in general, equivalent, it suffices to show that the position of an in-between feature point calculated by a linear average of the positions of that point in the two input images is not the same position obtained by changing the camera and retaking the picture. This will show that standard image morphs do not correctly simulate a change in camera pose.

Let the camera pose for one of the input images - image 1 - be given by the 4×4 projective matrix H_1, and let the pose for image 2 be given by H_2. Let $(x, y, z, 1)$ be an object feature point in homogenous three-dimensional coordinates, let (x_i, y_i, z_i, w_i) be the projection of the point in image i, and let(x_{ip}, y_{ip}) be the screen position in image i. Then

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} = \mathrm{H}_1 * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad x_{1p} = \frac{x_1}{w_1}, y_{1p} = \frac{y_1}{w_1}. \tag{2.12}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = \mathrm{H}_2 * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad x_{2p} = \frac{x_2}{w_2}, y_{2p} = \frac{y_2}{w_2}. \tag{2.13}$$

Let the projected point for $(x, y, z, 1)$ in the in-between image be (x_p, y_p). In a

morph, this point is given by a linear interpolation of the image points:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1-\alpha) * \begin{bmatrix} x_{1p} \\ y_{1p} \end{bmatrix} + \alpha * \begin{bmatrix} x_{2p} \\ y_{2p} \end{bmatrix} \tag{2.14}$$

$$x_p = (1-\alpha) * \frac{x_1}{w_1} + \alpha * \frac{x_2}{w_2} \tag{2.15}$$

$$= (1-\alpha) * \frac{(a_1x + b_1y + c_1z + d_1)}{m_1x + n_1y + o_1z + 1} + \alpha * \frac{(a_2x + b_2y + c_2z + d_2)}{m_2x + n_2y + o_2z + 1} \tag{2.16}$$

$$y_p = (1-\alpha) * \frac{(e_1x + f_1y + g_1z + h_1)}{m_1x + n_1y + o_1z + 1} + \alpha * \frac{(e_2x + f_2y + g_2z + h_2)}{m_2x + n_2y + o_2z + 1} \tag{2.17}$$

$$= (1-\alpha) * \frac{y_1}{w_1} + \alpha * \frac{y_2}{w_2} \tag{2.18}$$

$$\tag{2.19}$$

($a_i, b_i, c_i, d_i, \ldots$ represent the elements of the projective matrices.) The equations for a warp due to a change of camera position is thus a sum of fractions, and in general will be a ratio of quadratics. Thus, the in-between images created by standard morphing can not, in general, be expressed as a projective transform of the original object (every projective transform can be expressed as a ratio of linear terms.) This will create non-realistic output images, in which the object appears to bend straight lines, for example.

Seitz and Dyer discuss an algorithm for handling pose changes. First they demonstrate that for view changes containing only a translation perpendicular to the camera's eye vector and a change in zoom, linear interpolation of the image points correctly models the 3D transformation. For transformations restricted to this form, a progression of in-between morph images appears to gradually translate and zoom the image.

In the general case, the input images are prewarped using image reprojection. Image reprojection is a projective transform that reprojects an image onto the image plane of a camera in a different pose position. Once the two input images are projected onto aligned image planes (image planes separated only by a translation), standard morphing can be done to create in-between images with the same orientation. Finally, the in-between image is postwarped to be aligned with an in-between image plane. (The in-between image plane is created by specifying a path for the camera during the transition.)

A disadvantage of view-morphing is that it requires knowledge of the camera pose (the H matrices used in each image) to compute the image reprojections.

2.3 Multidimensional Morphs

Image morphing can be generalized to merge more than two input images at once. Image morphing warps two input images into alignment using corresponding pairs of features in each image. The aligned images are then cross-faded. In the same way, feature correspondences can be specified in multiple images. Each of the images

are warped to a common shape, and then the set of aligned images are cross-faded. Images created in this way are known as multidimensional morphs.

2.3.1 Multidimensional Morphable Models

Jones and Poggio [7] discuss a method called Multidimensional Morphable Models (MMM) for creating multidimensional morphs. They select a reference image out of a group of images from a certain class (i.e. images of faces), and create pixel-wise correspondences between the reference image and every other image in the group. Manual feature specification is not required; instead, they use a bootstrapping algorithm based on their earlier work with Vetter in 1997 [18], which I discuss below. In their notation, each image has three parts: an affine transform, a *shape vector*, and a *texture vector*. The affine transform removes any overall scale, rotation, translation or shear that the image contains with respect to the reference image. The shape vector is the pixel correspondence of each pixel in the reference image to a point in the destination image. The texture vector for an image is the pixels of the unwarped image. These vectors define a pixel correspondence warp using reverse mapping for each image to the reference image. The affine transform is applied after the pixel correspondence. Let the affine transformation be represented by the matrix A, in homogenous two dimensional coordinates, and let the warped position of the pixel at (x, y) given by the pixel correspondence warp be $(x_{\text{pixel}}, y_{\text{pixel}})$. Then the final texture coordinate for a destination pixel (x, y) is

$$\begin{bmatrix} x_{\text{texture}} \\ y_{\text{texture}} \\ 1 \end{bmatrix} = \text{A} * \begin{bmatrix} x_{\text{warp}}(x, y) \\ y_{\text{warp}}(x, y) \\ 1 \end{bmatrix} \tag{2.20}$$

New images may be created in the model by specifying a new shape vector, which defines a warp from the reference image to the new image. This warp is *chained* with the warps from each of the input images to the reference image to create a warp from each input image to the new image. Chaining warps means taking the composition of the warp functions to find the effect of warping by one of the functions and then the other. This is discussed further in Chapter 3. The warped input images are cross-faded together; the barycentric coordinates used as weights in the cross-fade are part of the specification of the new image. The texture weights must be barycentric (non-negative, sum to one) to prevent changes in luminosity or other pixel addition artifacts. This is the generalization of using α and $1 - \alpha$ as the texture coefficients in two-image warps. An affine transform may optionally be applied to the constructed image, in analogy with the transforms used for the warps of the input images.

Jones and Poggio use a bootstrapping algorithm to generate the input warps used in their algorithm. It works incrementally: the system is initialized with a few images and warps of these images to the reference image. A new input image can be matched to an image created by the system automatically using the following technique. A given shape vector, set of texture coefficients, and affine transform can be used by the MMM system to construct an image. The L2 pixel distance of this constructed image from the image to be matched is a function of the elements of the shape vector,

texture coefficients, and affine transform. This function can be minimized by solving a least squares, gradient descent problem. The variables obtaining the minimum will specify a constructed image as close as possible to the new image.

The shape vector found in the minimization problem defines coarse correspondences from the reference image to the new input image. It gives a corresponding position in the new image for each of the point features that define the shape vector in the reference image. The match will not in general be perfect, but can be improved by finding the optical flow between the constructed image and the new input image. Chaining the pixel correspondence warp returned from the optical flow algorithm with the warp given by the corresponding shape vectors will produce the final shape vector for the new input image. This process can be iteratively applied for each new input image.

2.3.2 Polymorph

Lee, Wolberg, and Shin independently developed another multimorph method in the same year called Polymorph [10]. Their work does not not require manual (or automated) correspondences between the same set of features in every image. Instead, they accept as input a graph of standard two-image morphs. Each vertex of the graph is an image, and each edge is a morph between two images. A path in the graph - a list of consecutive morphs - may be chained together to produce a morph between the first and last images on the path. An incremental all-pairs shortest paths algorithm is used to create morphs between every image and every other image. When multiple shortest paths are present, the morphs given by each path are merged.

After a morph is calculated for each pair of images, Lee, Wolberg and Shin introduce two possible methods for specifying novel images. A set of barycentric coordinates identifies an image; each element of the coordinates corresponds to one of the input images. Let these barycentric coordinates be $(\alpha_1, \alpha_2, \ldots, \alpha_n)$. The first of their two suggested techniques to generate an image from the coordinates is more robust, but has greater computational complexity. A warp from input image I_i to the new image is created by averaging the outgoing warps from I_i. If the warped position of (x, y) in the warp from image i to image j is (x_{ij}, y_{ij}), then the warped position of (x, y) in the warp from image i to the new image is

$$\begin{bmatrix} x_{\text{warped}} \\ y_{\text{warped}} \end{bmatrix} = \alpha_1 * \begin{bmatrix} x_{i1} \\ y_{i1} \end{bmatrix} + \alpha_2 * \begin{bmatrix} x_{i2} \\ y_{i2} \end{bmatrix} + \cdots + \alpha_n * \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{2.21}$$

In this way, each of the input images can be warped to the common shape specified. The warped images are cross-faded together using another set of barycentric coordinates (or possibly the same set), as is done in the work by Jones and Poggio.

The second, more efficient technique, specifies a reference image. This image may be either one of the input images or some interpolated image. Warps must be specified from the reference image to every one of the input images. In the implementation by Lee, Wolberg and Shin, the reference image is taken to be the average image: an image created using the first method (described above) with the coordinates $(1/n, 1/n, 1/n, \ldots 1/n)$. Once this image is calculated, only warps from this

image to every input image need to be preserved, along with their inverse warps from input images to the reference image. This saves memory and processing time, as there are now $2n$ warps instead of $n^2 - n$.

A warp from the reference image to a new image is specified by averaging the warps from the reference image to every other image. This average is characterized by a set of barycentric coordinates α_i, and is constructed by the same procedure described above for constructing the warps from each input image. A warp from the new image to each input image is created by chaining the warp from the new image to the reference image with the warp from the reference image to each input image. The warped images are cross-faded together as in the first method. This technique strongly resembles the method used by Jones and Poggio, varying only in the method used to generate the correspondences from the reference image to every input image. Also similar to Jones and Poggio, affine transforms can be specified as a preprocess on each image to correct for translations, rotations, scale, and shear.

Lee, Wolberg and Shin also discuss non-uniform blending of the input images. This technique can be defined with standard two-image morphs. In a two-image morph, the two images are warped to a common shape created by interpolating the position of the features in each image. When using a blending function, instead of using a single coordinate α to define the in-between shape, a function is defined over one of the input images giving the degree to which the geometry of that point should be warped to create the geometry of the in-between image. If the blending function $B(x, y)$ is defined over image 1, and (x_1, y_1) in image 1 corresponds to (x_2, y_2) in image 2, then the in-between point corresponding to both of these points is given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = (1 - B(x_1, y_1)) * \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + B(x_1, y_1) * \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \qquad 0 \leq B(x, y) \leq 1 \tag{2.22}$$

Note that it suffices to specify the blending function over one image. The same blending function (or a separate function) can be used to specify the texture coefficients used at each point in the cross-fade. This method can be generalized to multidimensional morphs: a blending function is defined over one image specifying a vector of barycentric coordinates at each point. These barycentric coordinates give the weight of each input image in specifying the geometry (and texture) at that point.

2.3.3 Synthesis of 3D faces

In *A morphable model for the synthesis of 3D faces*, Blanz and Vetter introduce a similar concept to multidimensional morphs that does not explicitly rely on image warps [2]. Input to their system is a set of 3D faces (a 3D mesh, along with the color at each vertex) in full correspondence. Full correspondence means that vertex i in face 1 is a corresponding feature to vertex i in face 2, and vertex i in face 3, and so on. The output of the system is a parameterized face mesh simulated from the input meshes. The simulated faces are created in the same way as in multidimensional morphs: a new 3D mesh is specified by taking a linear combination of the input meshes, and new vertex colors are created by a (barycentric) linear combination of the input mesh vertex colors. This is a powerful extension to image morphs because

it creates actual 3D geometry. Blanz and Vetter obtained their facial mesh input from a $Cyberware^{TM}$ volume scanner. Full correspondence between the input images is not necessary; correspondence can be established using the same bootstrapping algorithm refined with optical flow described by Vetter, Jones, and Poggio [18].

Blanz and Vetter also introduce the subject of facial attributes. A facial attribute, or in general an image attribute, specifies a value for each image representing the degree to which that image possesses the attribute. In my work, I refer to image attributes as *traits*. Traits are an important user-interface tool for creating multidimensional morphs. Finding the parameters in a multidimensional morph system to create an image possessing a particular attribute is not a user-friendly problem. The parameters of the system are unwieldy vectors of feature positions with no simple intuitive meaning. Traits are defined by the user and can be used to find the parameters which produce an image possessing more or less of the defined trait.

To work with traits, images are represented by vectors. How this is done is dependent upon the multidimensional morphing system used; my implementation is discussed in Chapter 5. Blanz and Vetter use the 6-dimensional mesh coordinates (x, y, z and color as (red, green, blue)) to assign $6n$-dimensional vectors to each face. To refine the vector space, they first use PCA (principle component analysis) on the set of input image vectors. PCA returns the vectors of greatest variance in the input, and the projections along those vectors of each input image's deviation from the mean. Linear combinations of the vectors of greatest variance are more numerically stable than linear combinations of the input vectors. Blanz and Vetter also use this tool to judge the plausibility of synthetic faces; the likelihood that an output image is a face is proportional to the product of its distance from the mean along each eigenvector (normalized by variance of the input along that eigenvector.) PCA and other techniques for *dimensionality reduction* are covered in Chapter 3.

Once the vector space of input images has been defined, and the values of a trait for each image have been assigned, Blanz and Vetter show how to use these values to create a trait vector - a direction in the high dimensional space representing that trait. This assumes the trait is a linear function in the vector space (a hyperplane). Their technique to compute the vector is to take the average of the image vectors, weighted by the value of the trait for the images. My approach to traits, including a discussion of Blanz and Vetter's technique and further solutions for both linear and non-linear traits, is discussed in Chapter 6.

My research extends the state of the art in morphing, by generalizing the concept of morphing to more than two images. My work differs from previous multimorphing techniques in several respects. It accepts traditional two-image morphs as a starting point, and in the presence of cyclic graphs it merges the redundant user input. It uses non-linear methods to specify the shape and texture of new images. And it allows user specification of output images through defined attributes, which are also modeled using non-linear methods. In the next chapter, I discuss terms and definitions used in the rest of this work.

Chapter 3

Terms and Definitions

This chapter covers terms and mathematical tools used in the rest of the thesis. I first discuss the terminology I use for standard image morphs, which I treat as a black box, subject to several requirements. I then describe techniques for dimensionality reduction, which I use in Chapter 5 to manage morph vectors. Next I describe the problem of *dimensionality reprojection*, the inverse problem of dimensionality reduction. And finally I introduce the idea of *traits* - representing objects in a vector space and finding a direction in the space which represents an increase in a selected attribute.

3.1 Notation

Labeled structures and algorithms are written in *italicized text* when defined, but then referred to afterwards using plain text. Mathematical symbols are written in *mathitalics*. Vectors are written in **boldface**, or with a vector symbol: $\vec{v}$. A vector of unit length is written with a hat symbol: $\hat{v}$. Matrices are written in UPPERCASE ROMAN. When writing a a set of p points in a d dimensional space, as a matrix, each point is a column of the matrix, and the matrix is $d \times p$. I refer to row i of of a matrix M as M_i, to column j as M^j, and to the element in row i, column j as M_{ij}. The notation $\{\mathbf{v}_i\}$ represents the set of vectors $\mathbf{v}_i$ where i and the range of i is determined by context.

3.2 Image Morphs

In my discussion of multidimensional morphs, I treat two-image morphs as a black box. In principle, any morphing algorithm may be used to implement multidimensional morphs. In the following, I define two-image morphs and describe my requirements on their input, output, and supported operations.

3.2.1 Definition of Two-image Morphs

A *two-image morph* is a function to produce an output image from two source images, two image warps, and two real-valued in-between parameters. I refer to two-image morphs as *morphs* when the term is unambiguous. In the following paragraphs, I define the structures that compose a two-image morph, and then describe the definition of the two-image morph in more detail.

An *image* is a continuous real-valued function defined over the plane, written as $I(x, y)$. In practice, this function is represented by a bitmap: it is discretely sampled, has a finite, rectangular domain, and has a multidimensional, integral, bounded range. Unless otherwise mentioned, I will ignore this detail and treat images as continuous and unbounded one-dimensional functions.

An *image warp* is an arbitrary warping of the plane. It is a real-valued two-dimensional function of two real variables: $(x_{\text{src}}, y_{\text{src}}) \rightarrow (x_{\text{dest}}, y_{\text{dest}})$. It is written as $\mathbf{W}(x, y)$. A warp of an image defines a *warped image*. The warped image given by image I_i and the image warp $\mathbf{W}(x, y)$ is labeled I_i^W. It is defined by

$$I_i^W(\mathbf{W}(x, y)) = I_i(x, y) \tag{3.1}$$

$$I_i^W(\mathbf{p}) = 0, \quad \forall \mathbf{p} \notin \{\mathbf{W}(x, y)\} \tag{3.2}$$

In practice, when images are not continuous functions, the definition of a warped image is left to the morphing algorithm. Therefore, an appropriate reconstruction filter must be used when sampling an image.

An *in-between parameter* is a real value $\alpha : 0 \leq \alpha \leq 1$. It specifies a progression from one element to another, such that $\alpha = 0$ corresponds to one element, $\alpha = 1$ corresponds to the other, and intermediate values are a combination of the two endpoint elements.

Two-image morphs are denoted by the symbol M. The output of a two-image morph is an image, referred to as an *in-between image*. The image, a function of x and y as defined above, is itself a function of two in-between parameters. In addition to the in-between parameters, a two-image morph is dependent upon several structured parameters. The morph is said to *connect* two images; the definitions of these images are part of the specification of the morph. One of the images is labeled the *first image* of the morph, and the other image is labeled the *second image* of the morph. A two-image morph also contains two image warps: one warp, referred to as the *forward image warp* of the morph, is applied to the first image of the morph, and warps that image into alignment with the second image. The other warp - the *reverse image warp* - is applied to the second image of the morph, and warps that image into alignment with the first image. The in-between parameters of the morph are labeled α and β. α specifies the weight the second image receives when determining the feature positions of the output image; $(1 - \alpha)$ is the weight of the first image. Similarly, β is the weight of the second image when determining the texture-fading coefficient of the morph, and $(1 - \beta)$ is the weight of the first image. The schematic of a two-image morph is shown in Figure 3-1.

In general, a two-image morph is notated by all six of its arguments: a two-image morph connecting image I_i to image I_j, with warp $\mathbf{W}_1$ from I_i to I_j, $\mathbf{W}_2$ from I_j to

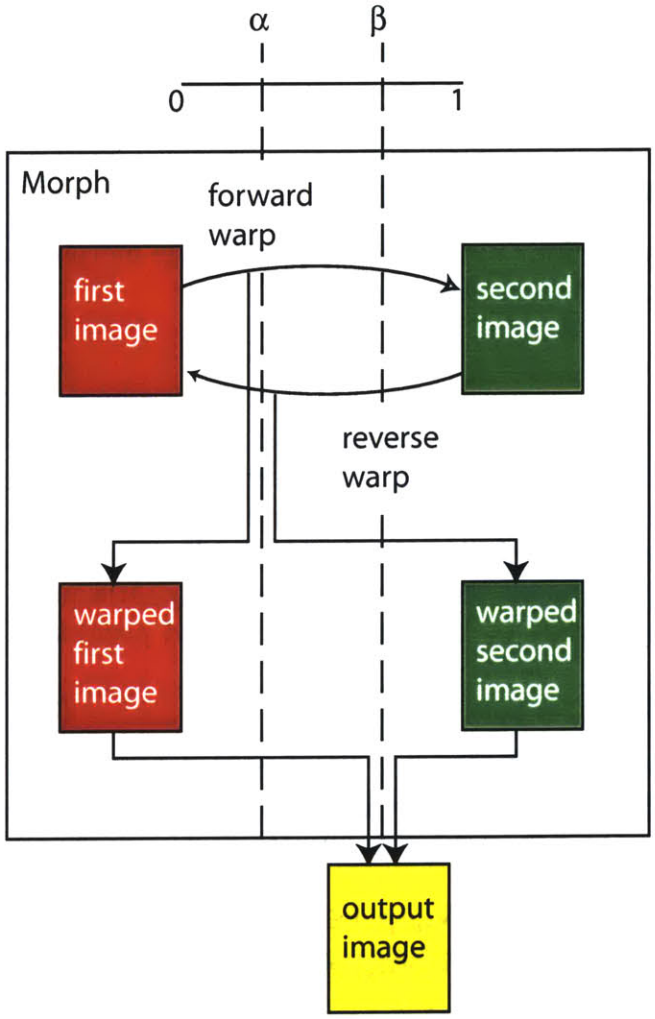

Figure 3-1: α and β characterize the output image of a morph. The first image is warped by the forward warp and the second image is warped by the reverse warp to create warped images. The distance of the in-between shape from the shape of the first image is set by α. The two warped images are combined (i.e. cross-faded) to produce the output image. The distance of the final image from the first warped image is set by β.

I_i, and intermediate values α and β, would be written as $M_{i,j,\mathbf{W}_1,\mathbf{W}_2,\alpha,\beta}$. For ease of notation, I write a morph connecting I_i to I_j with in-between values α, β as $M_{ij}(\alpha,\beta)$. M_{ij} written without the in-between values refers to the family of output images given by the morphs $M_{ij}(\alpha,\beta)$, $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$. The forward warp of M_{ij} (from I_i to I_j) is notated as W_{ij}, and the reverse warp is notated as W_{ji}. Note that if M_{ij} exists, then M_{ji} also exists: the first and second images and the forward and reverse warps are swapped, and in-between values α,β for M_{ij} are equivalent to $(1-\alpha),(1-\beta)$ for M_{ji}. In the rest of this document, I refer to the M_{ij} family of two-image morphs as the two-image morph from i to j.

3.2.2 Specializations of Two-image Morphs

The definitions above are the extent of my formal definitions of two-image morphs. I now give some common specializations of two-image morphs.

Feature-based Warps

A subset of image warps is *feature-based warps*. A feature-based warp $\mathbf{W}_{sd}$ is a member of a family of warps defined by a source image I_s, a destination image I_d, and a set of corresponding features in the two images. A *feature* of an image is a point, or set of points (i.e. line, curve), in the domain of the image. Corresponding features of images I_s and I_d put restrictions on the warp function $\mathbf{W}_{sd}$. If a point feature $\mathbf{p}_s$ in I_s corresponds to a point feature $\mathbf{p}_d$ in I_d, then the warped position of $\mathbf{p}_s$ must be $\mathbf{p}_d$:

$$\mathbf{W}_{sd}(\mathbf{p}_s) = \mathbf{p}_d. \tag{3.3}$$

If a set of points in I_s corresponds to a set of points in I_d, then there must be a one-to-one correspondence between the points in the set in I_s and the points in the set in I_d. So if a set of points P_s corresponds to a set of points P_d, then

$$\mathbf{p}_s \in P_s \;\rightarrow\; \mathbf{W}_{sd}(\mathbf{p}_s) \in P_d \tag{3.4}$$

$$\mathbf{p}_s \in P_s \;\&\; \mathbf{p}_t \in P_s \;\&\; \mathbf{p}_s \neq \mathbf{p}_t \;\rightarrow\; \mathbf{W}_{sd}(\mathbf{p}_s) \neq \mathbf{W}_{sd}(\mathbf{p}_t). \tag{3.5}$$

A set of corresponding features between image I_s and image I_d defines both a warp $\mathbf{W}_{sd}$ of I_s and a warp $\mathbf{W}_{ds}$ of I_d. In general, it is not a formal requirement that these be inverse functions. So, in general, $\mathbf{W}_{sd}^{-1}(x,y) \neq \mathbf{W}_{ds}(x,y)$.

Cross-fading Two-Image Morphs

Another subset of two-image morphs is *cross-fading two-image morphs*. A cross-fading two-image morph is a morph for which the in-between image is created by cross-fading (averaging pixel intensities) a warped image of the first image with a warped image of the second image. It is defined by two parameterized warp functions corresponding to the two warp functions of the morph: $\mathbf{W}_{ij}^p(x,y,\alpha)$ and $\mathbf{W}_{ji}^p(x,y,\alpha)$, $0 \leq \alpha \leq 1$. $\mathbf{W}_{ij}^p(x,y,\alpha)$ is a *partial warp* of $\mathbf{W}_{ij}(x,y)$. A partial warp of a warp function $\mathbf{W}$ is a warp resembling a partially completed $\mathbf{W}$ warp. It must satisfy

$\mathbf{W}^p(x,y,0) = \begin{bmatrix} x \\ y \end{bmatrix}$, and $\mathbf{W}^p(x,y,1) = \mathbf{W}(x,y)$. With these partial warps defined, the value of a pixel (x,y) in the image produced by the cross-fading morph is

$$(M_{ij}(\alpha,\beta))(x,y) = (1-\beta) * I^p_{ij}(x,y,\alpha) + \beta * I^p_{ji}(x,y,1-\alpha) \tag{3.6}$$

$$I^p_{ij}(\mathbf{W}^p_{ij}(x,y,\alpha),\alpha) := I_i(x,y) \tag{3.7}$$
$$I^p_{ji}(\mathbf{W}^p_{ji}(x,y,\alpha),\alpha) := I_j(x,y) \tag{3.8}$$

$I^p_{ij}(x,y,\alpha)$ is the warped image of I_i by $W^p_{ij}(x,y,\alpha)$, and similarly for $I^p_{ji}(x,y,\alpha)$.

Standard Two-image Morphs

Finally, the most specific - and most commonly used - subset of two-image morphs is *standard two-image morphs.* Standard two-image morphs are cross-fading two-image morphs which use feature-based warps, and which use the following method to create partial warps.

The feature positions of the endpoint images are averaged to create a third set of corresponding features. So if the corresponding features in I_i and I_j are P_i and P_j, respectively, then $\mathbf{W}^p_{ij}(x,y,\alpha)$ is created from the corresponding features given by P_i and $P_{ij}(\alpha)$:

$$\mathbf{p}_i \in P_i \;\;\&\;\; \mathbf{p}_j \in P_j \quad \rightarrow \quad \mathbf{p}_i * (1-\alpha) + \mathbf{p}_j * \alpha \in P_{ij}(\alpha). \tag{3.9}$$

The partial warp from $W^p_{ij}(x,y,\alpha)$ is set to be a feature-based warp based on corresponding features P_i and $P_{ij}(\alpha)$. Similarly, $W^p_{ji}(x,y,\alpha)$ is based on P_j and $P_{ij}(1-\alpha)$.

Optionally, the feature positions of each endpoint image may be prewarped before being averaged. Linear combinations between two sets of points do not correctly simulate rotations. That is, if a set of points P_1 is rotated by θ to create a corresponding set of points P_2, then the in-between set of points from linear averaging - $P_1 * (1-\alpha) + P_2 * \alpha$ - is not equal to the expected in-between rotation of P_1 by $\theta * \alpha$. For a single point (x_1, y_1) with $\theta = 90°$:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.10}$$

$$\begin{bmatrix} x_{\text{between}} \\ y_{\text{between}} \end{bmatrix} = (1-\alpha) \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \alpha \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \tag{3.11}$$

$$= \begin{bmatrix} (1-\alpha) & -\alpha \\ (1-\alpha) & \alpha \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.12}$$

$$\neq \begin{bmatrix} \cos(\alpha * 90°) & -\sin(\alpha * 90°) \\ \sin(\alpha * 90°) & \cos(\alpha * 90°) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.13}$$

$$\tag{3.14}$$

This argument is derived from two sets of points separated only by a rotation, but it applies as well to a more general morph between two different shapes. If the morph is

between two faces, for example, and one face is rotated with respect to the other face, then linear combinations of the feature points will not correctly simulate reshaping the face while rotating.

The solution to this problem is to prewarp the points. The best-fit Euclidean or affine transform from the second image to the first image is computed. This transform may be restricted to a rotation, or a rotation and translation, or a rotation, translation, and scale, or it may be a general affine transform. Translation, scale, and shear, however, are extraneous, since linear combinations correctly simulate gradual changes in these values. The inverse transformation gives the transformation from the first image to the second. These transformations may be scaled (multiplying θ by γ, translation by γ, etc.) so that they transform to some in-between configuration determined by $\gamma, 0 \leq \gamma \leq 1$. Setting $\gamma = \alpha$ simulates a gradual change in rotation. Setting $\gamma =$ constant chooses an orientation and eliminates the rotation.

The transformation may be computed in one of two ways. The first method uses least squares. The Euclidean distances between the points from the first image and the transformed points from the second image are minimized by non-linear least squares over θ and the translation, scale, and shear variables. The second method is due to Lasenby et. al. [9], who points out that this least squares method incorrectly gives a stronger weight to rotations of features farther away from the mean of the input features. The same difference in rotation at farther distances from the mean produces larger Euclidean distances, so the weight in the minimization of squared distances is greater at points farther from the mean. Lasenby et. al. give a more sophisticated method which aligns the vectors of greatest variance in the two point sets to find θ.

All of the morphing methods discussed in Chapter 2 are standard two-image morphs.

3.2.3 Operations Defined on Two-image Morphs

Create an In-between Image

The basic operation of a two-image morph is to create an in-between image. This is given by the output image of $M_{ij}(\alpha, \beta)$ for a particular α, β.

Chaining

Another operation on two-image morphs is to *chain* two morphs. Chaining is a binary operation of two morphs which produces another morph. It is written as Chain(M_1, M_2). Given a morph M_{ij} from I_i to I_j, and a morph M_{jk} from I_j to I_k, Chain(M_{ij}, M_{jk}) produces a morph M_{ik} from i to k. Chain is undefined when the second image of the first argument is not identical to the first image of the second argument. The output images of the chained morph should resemble a combination of the endpoint images i and k after being warped to alignment. The exact definition is dependent upon the morph algorithm.

As an example, feature-based warps can implement chaining by *propagating points.* The corresponding feature points used to create the M_{ij} morph are warped using

the warp $\mathbf{W}_{jk}$ from M_{jk}, creating a set of feature points in I_k which correspond to the feature points in I_i. The same is done with the feature points from M_{jk} warped by $\mathbf{W}_{ji}$, and the two sets of feature points are merged. To be specific, let the corresponding features from M_{ij} be P_{1i} in image I_i, and P_{1j} in image I_j. Let the corresponding features from M_{jk} be P_{2j} in image I_j, and P_{2k} in image I_k. Let the corresponding features used to create M_{ik} be called P_{3i} in image I_i and P_{3k} in image I_j. P_{3i} and P_{3k} are given by

$$P_{3i} = P_{1i} \cup \mathbf{W}_{ji}(P_{2j}) \tag{3.15}$$

$$P_{3k} = P_{2k} \cup \mathbf{W}_{jk}(P_{1j}) \tag{3.16}$$

Further details on propagating points are discussed in my implementation in Chapter 7.

Merging

Another operation on two-image morphs is to *merge* a set of morphs. Merging operates on a set of n morphs between the same two images I_i and I_j: $\{M_{ij}^k : 1 \leq k \leq n\}$. It also takes a set of weights to give to each morph: $\{w_k : 1 \leq k \leq n\}$. It is written as Merge$(\{M_{ij}^k\}, \{w_k\})$. The output of a merge is another morph M_{ij}^{merged}. Merging is undefined if the input morphs do not all share the same first and second images. The output images of the merged morph should resemble a feature-aligned average of the output images produced by the input morphs. How this is accomplished is left to the morph specification.

As an example, feature-based warps can implement merging by *merging points*. The control features from each input morph are combined into a set of features in the first image and a set of features in the second image. The features are sampled into point features, and each morph operates on the point features, producing a set of warped features for each image. The position of the merged features is taken to be the weighted average of the features produced by each morph. The two sets of corresponding features (one warped from I_i and one warped from I_j) are unioned, and the resulting set of corresponding features defines the merged morph. Further details on merging points are discussed in my implementation in Chapter 7.

3.3 Dimensionality Reduction

Dimensionality reduction is a method to reduce the information needed to specify a set of points. It takes as input p points in a high-dimensional space, of dimension n_h. In my discussion, I will assume these points are given as the columns of an $[n_h \times p]$ matrix. Dimensionality reduction assumes these points are samples on a low-dimensional manifold in the high-dimensional space. Letting the dimensionality of the manifold be n_l, the task is to specify a minimal n_l and provide embedding coordinates for each high-dimensional point, creating p points in n_l-dimensional space.

The embedding points are the mapping of the high-dimensional points onto a linear sub-space. In linear dimensionality reduction, the high-dimensional points are

assumed to lie on (or near) a *linear manifold*: a linear subspace which has been offset from the origin (i.e. a line, or a plane, or a higher-dimensional equivalent.) The embedding coordinates of a point on the linear manifold are the projection of that point along an orthogonal coordinate system of the subspace. In general, however, the manifold is non-linear, and so the mapping is more complicated than a simple Euclidean transformation. The mapping is selected so that the embedding points preserve certain properties of the high-dimensional points. Different algorithms for non-linear dimensionality projection preserve different properties of the points.

When morphing within a set of images there are often a small number of natural parameters, such as dog size, body type, etc. While the number of natural parameters may be in in the tens or hundreds, it is considerably smaller than the feature vectors of the morphs. Non-linear analysis attempts to discover this lower-dimensional parameter space by finding an approximation to the manifold containing the set of high-dimensional feature vectors.

3.3.1 Linear Dimensionality Reduction: Principle Component Analysis

The method for linear dimensionality reduction is universally accepted to be *principle component analysis* (PCA), or the equivalent method of *multi-dimensional scaling* (MDS). PCA assumes that the input points lie on a linear manifold, and finds an orthogonal coordinate system which spans the manifold and is composed of the vectors of greatest variance of the input. To find these vectors, the mean of the input points is first moved to the origin. Then the *singular value decomposition* (SVD) of the translated points will give the vectors of greatest variance. If there are p points, then the dimension of this subspace will be at most $p - 1$; vectors with 0 or negligible variance can be removed to reduce the dimensionality further. I now give a short derivation of PCA and MDS.

Let the matrix of high-dimensional points be called X (each point is a column of X). The SVD of X is the matrices U,S,V such that

$$\mathrm{USV}^T = \mathrm{X} \tag{3.17}$$

$$\mathrm{UU}^T = \mathrm{I} \tag{3.18}$$

$$\mathrm{VV}^T = \mathrm{I} \tag{3.19}$$

If X is $n_h \times p$, then U is an orthogonal $n_h \times n_h$ matrix, S is a rectangular diagonal $n_h \times p$ matrix, and V is an orthogonal $p \times p$ matrix. The columns of U are the eigenvectors of XX^T, and the columns of V are the eigenvectors of $\mathrm{X}^T\mathrm{X}$. The eigenvalues of XX^T and $\mathrm{X}^T\mathrm{X}$ are the same set of non-negative values; the non-negative square roots of these eigenvalues are on the diagonal of S and are known as the singular values. The ith column of U and the ith column of V correspond to the singular value at S_{ii}. By convention, the singular values are sorted in descending order, so that $\mathrm{S}_{1,1}$ is the largest.

I now show that when the points are centered around the origin, the columns of U are also the vectors of greatest variance: the vectors of greatest variance of a set

of input points are given by the eigenvectors of the covariance matrix of the input dimensions. The covariance matrix Σ of the rows of X (each row is a variable, and each column is an observation) is given by

$$\Sigma_{ij} = \frac{1}{p-1}\sum_{k=1}^{p}(\mathrm{X}_{ik}-\mu_i)*(\mathrm{X}_{jk}-\mu_j) \tag{3.20}$$

$$\mu_i = \frac{1}{p}\sum_{k=1}^{p}\mathrm{X}_{ik}. \tag{3.21}$$

μ is the mean of the input points. Since the points were recentered, $\mu = \vec{0}$, so that $\mu_i = \mu_j = 0$.

$$\Sigma_{ij} = \frac{1}{p-1}\sum_{k=1}^{p}(\mathrm{X}_{ik}-0)*(\mathrm{X}_{jk}-0) \tag{3.22}$$

$$= \frac{1}{p-1}\sum_{k=1}^{p}\mathrm{X}_{ik}*\mathrm{X}_{jk} \tag{3.23}$$

$$= \frac{1}{p-1}*(\mathrm{XX})^T_{i,j} \tag{3.24}$$

$$\Sigma = \frac{1}{p-1}*\mathrm{XX}^T \tag{3.25}$$

Therefore, the eigenvectors of the covariance matrix for X are given by the eigenvectors of XX^T, and the eigenvalues (the variances) are directly proportional to the eigenvalues of XX^T. The columns of U, which are the eigenvectors of XX^T sorted by descending order of eigenvalue, are the sorted vectors of greatest variance in the input.

The embedding given by PCA is the projection of each high dimensional point, with mean subtracted, along these vectors. Looking at the SVD equation $\mathrm{X} = \mathrm{USV}^T$ and considering a column of X at a time, the projection of the ith column of X along the jth column of U is given by $(\mathrm{SV}^T)_{ji}$. (This is true because $\mathrm{X}^i = \mathrm{U}(\mathrm{SV}^T)^i$, and the columns of U are orthonormal.)

The final step of PCA is deciding the dimensionality of the hyperplane. The *truncated SVD* of X to rank r is the first r columns of U, the upper-left $r \times r$ submatrix of S, and the first r columns of V. Let the rank of the input matrix X be r_x. Since the mean point was subtracted from each point (column) in X, the rank is less than or equal to $p-1$ (number of columns minus one), and less than or equal to n_h (number of rows). The truncated SVD of X to rank r_x reconstructs X exactly; the singular values $i > r_x$ are zero. If the dimensionality of the manifold is known, PCA uses the truncated SVD of X to rank n_l for its embedding space. When the dimensionality is not known a priori, it is given by the last non-zero singular value. For numerical issues, the L2 distance between the truncated SVD matrix $\mathrm{X}_r = \mathrm{S}_r\mathrm{U}_r\mathrm{V}_r^T$ and the original matrix X is quadratic in the discarded singular values. Thus, small but non-zero singular values can robustly be set to zero. (This is a standard technique when using the SVD to compute the pseudo-inverse of a matrix.)

PCA returns n_l orthogonal vectors along which the set of input points has non-zero variance. It also returns an embedding: the projection of each input point's deviation from the mean along these vectors. When all that is needed is the embedding, a related technique called multi-dimensional scaling (MDS) can be used. MDS requires only a distance matrix of the input points, and generates the same embedding coordinates as PCA. The coordinates are given by the eigenvectors of $\mathrm{X}^T\mathrm{X}$, which are recovered from the distance matrix using the following derivation. Since distance matrices are invariant to a global translation, it is assumed that the mean of the input points is the origin. Let the distance matrix be D, and the $\mathrm{X}^T\mathrm{X}$ matrix be P (both matrices are $p \times p$). By definition, D_{ij} is the distance (squared) of X^i from X^j:

$$\begin{aligned}
\mathrm{D}_{ij} &= (\mathrm{X}^i - \mathrm{X}^j)^T(\mathrm{X}^i - \mathrm{X}^j) && (3.26)\\
&= \sum_{k=1}^{n_h} \left((\mathrm{X}_{ki} - \mathrm{X}_{kj})^2\right) && (3.27)\\
&= \sum_{k=1}^{n_h} \left(\mathrm{X}_{ki}^2 + \mathrm{X}_{kj}^2 - 2\mathrm{X}_{ki}\mathrm{X}_{kj}\right) && (3.28)\\
&= \sum_{k=1}^{n_h} \mathrm{X}_{ki}^2 + \sum_{k=1}^{n_h} \mathrm{X}_{kj}^2 - 2\sum_{k=1}^{n_h} \mathrm{X}_{ki}\mathrm{X}_{kj} && (3.29)\\
&= (\mathrm{X}^i)^T(\mathrm{X}^i) + (\mathrm{X}^j)^T(\mathrm{X}^j) - 2(\mathrm{X}^i)^T(\mathrm{X}^j) && (3.30)\\
\mathrm{D}_{ij} &= \mathrm{P}_{ii} + \mathrm{P}_{jj} - 2 * \mathrm{P}_{ij} && (3.31)
\end{aligned}$$

P_{ij} is expressed in terms of D by *recentering* D. The average of each row i and column j are subtracted from D_{ij}, and the average of the entire matrix is added. Let the mean value of row i be $\bar{\mathrm{D}}_i$, the mean value of column j be $\bar{\mathrm{D}}^j$, and the mean value of D be $\bar{\mathrm{D}}$.

$$\begin{aligned}
\bar{\mathrm{D}}_i = \frac{1}{p}\sum_{k=1}^{p} \mathrm{D}_{ik} &= \frac{1}{p}\sum_{k=1}^{p} (\mathrm{P}_{ii} + \mathrm{P}_{kk} - 2 * \mathrm{P}_{ik}) && (3.32)\\
&= \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{ii} + \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{kk} - \frac{2}{p}\sum_{k=1}^{p} \mathrm{P}_{ik} && (3.33)\\
&= \mathrm{P}_{ii} + \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{kk} - \frac{2}{p}\sum_{k=1}^{p}\sum_{m=1}^{n_h} (\mathrm{X}_{mi}\mathrm{X}_{mk}) && (3.34)\\
&= \mathrm{P}_{ii} + \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{kk} - 2\sum_{m=1}^{n_h} \left(\mathrm{X}_{mi} * \frac{1}{p}\sum_{k=1}^{p} \mathrm{X}_{mk}\right) && (3.35)\\
&= \mathrm{P}_{ii} + \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{kk} - 2\sum_{m=1}^{n_h} (\mathrm{X}_{mi} * \mu_m) && (3.36)\\
&= \mathrm{P}_{ii} + \frac{1}{p}\sum_{k=1}^{p} \mathrm{P}_{kk} \qquad \text{(mean is zero)} && (3.37)
\end{aligned}$$

$$\bar{D}^j = \frac{1}{p}\sum_{k=1}^{p} D_{kj} = \frac{1}{p}\sum_{k=1}^{p} D_{jk} \quad \text{(D is symmetric)} \tag{3.38}$$

$$= P_{jj} + \frac{1}{p}\sum_{k=1}^{p} P_{kk} \tag{3.39}$$

$$\bar{D} = \frac{1}{p^2}\sum_{i=1}^{p}\sum_{j=1}^{p} D_{ij} = \frac{1}{p}\sum_{i=1}^{p}\left(\frac{1}{p}\sum_{j=1}^{p} D_{ij}\right) \tag{3.40}$$

$$= \frac{1}{p}\sum_{i=1}^{p} \bar{D}_i \tag{3.41}$$

$$= \frac{1}{p}\sum_{i=1}^{p}\left(P_{ii} + \frac{1}{p}\sum_{k=1}^{p} P_{kk}\right) \tag{3.42}$$

$$= \frac{1}{p}\sum_{i=1}^{p} P_{ii} + p * \frac{1}{p^2}\sum_{k=1}^{p} P_{kk} \tag{3.43}$$

$$= \frac{2}{p}\sum_{k=1}^{p} P_{kk} \tag{3.44}$$

$$D_{ij} - \bar{D}^i - \bar{D}_j + \bar{D} = P_{ii} + P_{jj} - 2P_{ij} - P_{ii} - \frac{1}{p}\sum_{k=1}^{p} P_{kk} - P_{jj} - \frac{1}{p}\sum_{k=1}^{p} P_{kk} + \frac{2}{p}\sum_{k=1}^{p} P_{kk} \tag{3.45}$$

$$P_{ij} = -\frac{1}{2}\left(D_{ij} - \bar{D}^i - \bar{D}_j + \bar{D}\right) \tag{3.46}$$

The eigenvectors of $P = X^T X$, multiplied by the square root of each vector's eigenvalue, give the embedding coordinates used in PCA. (Note that each eigenvector gives the coordinate of each input point in one dimension.) MDS is used when the actual points are unavailable, or when only the distance between points is defined (as in subjective judgements of similarity between objects.)

3.3.2 Non-linear Dimensionality Reduction

The assumption in PCA that the points are sampled from a linear manifold is quite restrictive. Two techniques for non-linear dimensionality reduction were developed in 2000. The first is *isomap*, developed by Tenenbaum, Silva, and Langform [15]. The second is *locally linear embeddings* (LLE), developed by Roweis and Saul [12]. Both rely on the assumption that the manifold is densely sampled. The points must be dense enough so that they are *locally linear*, as is explicitly mentioned in LLE. Local linearity means that, for each neighborhood of points in the sample, all points in the neighborhood lie on a linear manifold, to within some tolerance. A *neighborhood* of points is a set of the k closest sample points to a point in space, or the set of sample points less than ϵ Euclidean distance away from a point. The parameter k or ϵ, and the tolerance (the allowable least squares distance of each point from the linear manifold) are parameters of the strictness of the local linearity.

Isomap uses the assumption of local linearity to require that convex combinations of neighboring points are also on the manifold. (On a linear manifold, convex combinations of any subset of points are on the manifold.) Isomap constructs a distance matrix for the set of points, where the distance between two points is approximately the *manifold distance*. This is the shortest distance between two points when the path is restricted to lie on the manifold. Isomap approximates this by constructing a graph of the points, with edges drawn between neighbors. The neighbors of a point are defined either by taking all points within a distance of ϵ, or by taking the k closest points. (This is the same definition of neighborhood as in the definition of local linearity.) The values for ϵ and k are tuned parameters of the algorithm. If the assumption of local linearity is valid, then paths between neighbors will lie approximately on the manifold. Manifold distance between points may then be approximated by finding the shortest graph distance between two points. Once the distance matrix is constructed, isomap uses a straight-forward application of MDS (as discussed above, MDS only requires the distance matrix) to define the embedding coordinates. Isomap preserves the manifold distance of the high-dimensional points.

LLE uses the assumption of local linearity to require that each point be a linear combination of its neighbors. LLE reasons that a smooth manifold in n_h dimensional space can be flattened out into a linear manifold in the space by rotating and translating local regions of the manifold so that they are in alignment in a linear subspace. Consider a 3D example: a half-sphere. The hemisphere can be nearly flattened into a plane by cutting it into patches, translating each patch's center to $z = 0$, and then rotating the patch so that its perpendicular is parallel to the z-axis. Since each patch is a curved section of a sphere, points on the patches will rise out of the x-y plane, but as the patch size becomes smaller, this deviation becomes arbitrarily small. The embedding of the points on the hemisphere are then the x and y coordinates of the flattened points on the plane. LLE preserves the linear combinations of each high dimensional point in terms of its neighbors.

Each point is represented as a linear combination of its neighbors, a combination invariant to rotations and translations. All linear combinations are invariant to linear transforms, which includes rotations:

$$\text{If} \quad \sum_{i=1}^{k} \alpha_i * \mathbf{x}_i = \mathbf{y} \tag{3.47}$$

$$\text{Then} \quad \sum_{i=1}^{k} \alpha_i * \mathrm{A}\mathbf{x}_i = \mathrm{A}\sum_{i=1}^{k} \alpha_i * \mathbf{x}_i \tag{3.48}$$

$$= \mathrm{A} * \mathbf{y}. \tag{3.49}$$

For a linear combination to be invariant to translations, the sum of the coefficients

must be 1:

$$\text{If} \quad \sum_{i=1}^{k} \alpha_i * \mathbf{x}_i = \mathbf{y} \tag{3.50}$$

$$\text{Then} \quad \sum_{i=1}^{k} \alpha_i * (\mathbf{x}_i + \mathbf{x}_t) = \sum_{i=1}^{k} \alpha_i * \mathbf{x}_i + \sum_{i=1}^{k} \alpha_i * \mathbf{x}_t \tag{3.51}$$

$$= \mathbf{y} + \left(\sum_{i=1}^{k} \alpha_i \right) * \mathbf{x}_t \tag{3.52}$$

$$\tag{3.53}$$

The least-squares solution for these translation-invariant combinations can be found using Lagrange multipliers for the extra constraint. Note that for an n_l-dimensional manifold, the number of neighbors needed to reconstruct a point is at least $n_l + 1$. To insure that the neighbors span the linear manifold at the given point, it is better to choose k (number of neighbors) to be larger than $n_l + 1$, and if necessary to solve the under-constrained least-squares problem by using a regularizer such as the pseudo-inverse.

Once the linear combinations have been found, the n_p-dimensional vectors best reconstructed by the combinations give the embedding vectors. These vectors correspond to the dimensions of the flattened linear manifold. The vectors are given by the closest solutions to $Wy = y$, where W is the matrix of linear combinations found in the first step. The only non-zero entries on each row of W are the weights of the neighbors used to reconstruct the point corresponding to that row. The vectors y satisfying $Wy = y$ are given by the near-zero eigenvectors of $(W - I)^T(W - I)$. Each eigenvector gives a coordinate for each point along one dimension. Because the weights were constructed to be translationally invariant, the vector $(1, 1, 1, \ldots, 1)$ (all ones) will be one of the eigenvectors with eigenvalue 0, and must be discarded when specifying the embedding.

3.4 Dimensionality Reprojection

In PCA the embedding coordinates directly specify high-dimensional coordinates. Each embedding coordinate is the coefficient of projection along one of the high-dimensional eigenvectors found by SVD. This property is not shared by MDS or the non-linear methods. In these methods, all that is found are the high-dimensional input points and their low-dimensional embedding coordinates. Finding the high-dimensional point corresponding to a new embedding point in these methods is what I call *dimensionality reprojection.*

Both LLE and Isomap embeddings preserve the combination of a point in terms of its neighbors: LLE does this explicitly, and isomap does this because, assuming local linearity, preserving distance to more than n_l neighbors preserves linear combinations in terms of those neighbors as well. Since the high-dimensional points are assumed to be locally linear, convex combinations of nearby points lie on the manifold. Using

these two properties, dimensionality reprojection recovers a high-dimensional point by finding the barycentric coordinates of an embedding point on its neighbors, and solving a least squares problem to find the high-dimensional point preserving those coordinates. Given high-dimensional input points, their n_l-dimensional embedding points, and a new low-dimensional point, my algorithm for dimensionality reprojection is:

1. Find the $n_l + 1$ nearest neighbors.
2. Calculate the barycentric coordinates of the point over the neighbors.
3. Project these coordinates on the high-dimensional coordinates of the neighbors.

3.5 Traits

In Chapter 6, I introduce the problem of traits for a collection of vectors. Each vector (representing an image in my work) is given a value for a user-defined trait. The values given to vectors may be either unbounded real values, or binary values - 0 or 1. (Intermediate restrictions such as integers are not considered.) The purpose of these traits is, starting from a given point, to increase the amount of the trait by moving in the direction represented by the trait.

As demonstrated by Blanz and Vetter [2], a linear solution to traits can be given trivially. Let the vectors and trait values be $\mathbf{v}_i$ and t_i respectively, $1 \leq i \leq p$. The trait vector $\mathbf{v}_t$ is given by the weighted sum of the deviation of each $\mathbf{v}_i$ from the mean, $\bar{\mathbf{v}}$, with the weights given by each t_i's deviation from the mean. To express this algebraically,

$$\mathbf{v}_t = \sum_{i=1}^{p} (t_i - \bar{t})(\mathbf{v}_i - \bar{\mathbf{v}}) \tag{3.54}$$

I discuss this solution for traits and some alternatives in Chapter 6.

Chapter 4

Morph Registration

In this chapter, I describe the first part of my algorithm for creating multimorph images, which I refer to as *morph registration.* This first step takes user input in the form of two-image morphs, and creates morphs from every image to a common *reference image.* I call these morphs *reference morphs.* They are created by chaining and merging the input morphs. Afterwards, the morphs are resampled and represented as vectors. Vector operations can then be used to create new images; this process is the subject of Chapter 5.

The chapter is organized into five parts. The first part describes the input to the algorithm. The second part discusses the fundamental elements of the algorithm - chaining and merging morphs. Part three describes selection of the reference image. Part four describes selecting the paths in the graph that represent the reference morphs. And part five describes how the reference morphs are used to create a multimorph image, and how they can be represented as vectors.

4.1 Input

The input to morph registration is a set of n_v images $\{I_i\}$, and a set of image morphs between them. The input image morph between I_i and I_j is labeled M_{ij}. In principle, any two-image morphing algorithm satisfying the requirements listed in Chapter 3 can be used for the two-image morphs.

I consider the input as a graph, with the images as vertices and the morphs as bi-directional edges (each warp of a morph is a directed edge). There are input restrictions on this graph: it must be connected, there must be no edges from a vertex to itself, and there must be no duplicate edges. Note that the third restriction explicitly disallows having multiple input morphs between two images. Solving redundancies of that type is not covered by my work.

4.2 Chaining and Merging Morphs

Chaining and merging morphs are operations I defined broadly in Chapter 3. Implementations may differ, depending especially on the nature of the two-image morphing

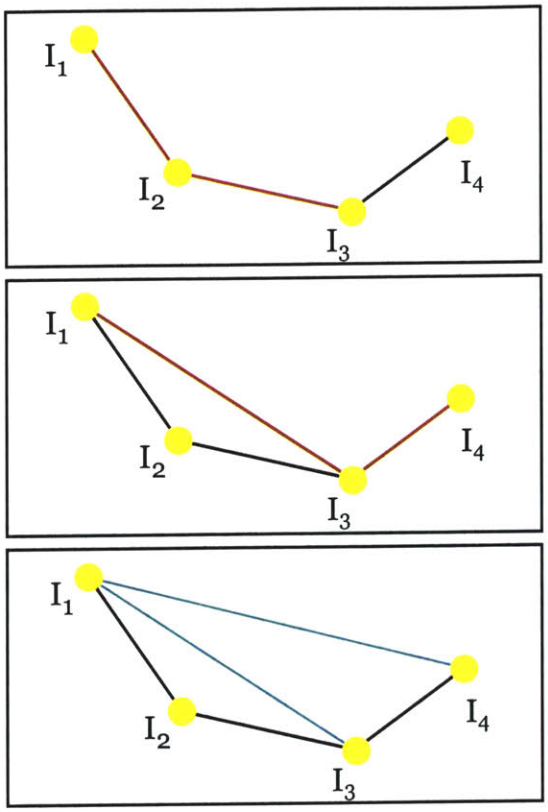

Figure 4-1: Chaining three image morphs. In the first frame, the morph from I_1 to I_2 is chained with the morph from I_2 to I_3 to produce the morph from I_1 to I_3 in the second frame. This morph is then chained with the morph from I_3 to I_4 to produce the morph from I_1 to I_4 in the third frame.

algorithm. I expect, however, that chaining and merging morphs will usually be done by a process fundamentally equivalent to warp composition and warp averaging, respectively. These are the most natural definitions. In this section, I describe the characteristics of chaining and merging when using these "typical" methods.

4.2.1 Chaining Morphs

Morphs from the input images to the reference image are constructed by chaining series of two-image morphs together. The quality of a chained morph has a significant impact on the quality of the in-between images it produces. In this section, I discuss several factors affecting chained morph quality.

The first factor affecting chained morph quality is the quality of the morphs contributing to the chain. As mentioned in Chapter 2, problems for warps include derivative discontinuities, foldover, and holes in the output image. Typically, morph chaining is implemented by composing the warp functions, such that $\mathbf{W}_{ik}(x, y) = \mathbf{W}_{jk}(\mathbf{W}_{ij}(x, y))$. Derivative discontinuities, foldover, and holes are all preserved by composition of warp functions. Thus a chained morph contains the union of the problems that occur in the morphs in the chain.

Another factor affecting the quality of a chained morph is the length of the chain. Two-image morphs are created by careful user alignment of features. When a morph from M_{ij} is chained with a morph M_{jk}, features in the M_{ij} morph may not correspond to the control features used to create the M_{jk} morph. The correct positioning of these features is dependent upon the power of the interpolation used in the algorithm creating the M_{jk} morph. In most cases, the interpolation is not perfect and small misalignments of features occur; the visual effect is a blurring in the in-between images created by the chained morph. Assuming a normal error distribution, the error in successive chainings adds as the square root: when making a chain from I_i to I_j to I_k to I_l, $\sigma_{ijkl} = \sqrt{\sigma_{ijk}^2 + \sigma_{jkl}^2}$. Thus error in the final chain increases as the square root of the length of the chain.

Another factor affecting chained morph quality is the compression and stretching of space. In a two-image morph a large area in one image may be warped to a small area in another image. When an area is compressed in one morph, and then stretched in a following morph in the chain, large visual artifacts are created: small errors in position are magnified.

Minimizing these errors in chained morphs is important when selecting the reference image and when creating the path from each input image to the reference image.

4.2.2 Merging Morphs

In a cyclic graph, multiple paths exist from some input images to the reference image. One of the possible options in this case is to merge the morphs corresponding to these paths to create a single morph from the input image to the reference image. Merged morphs are typically implemented by averaging the warp functions, such that

$$\begin{aligned} \mathbf{W}_{ab}^{\text{merge}}(x,y) &= \sum_{i=1}^{k} \alpha_i \mathbf{W}_{ab}^{i}(x,y) \qquad (4.1) \\ \sum_{i=1}^{k} \alpha_i &= 1. \end{aligned}$$

α_i is the weight given to morph i in the merge; it is a measure of the confidence of that morph. Averaging warp functions should be done when the correct warp is unknown, and each of a group of possible warps have equal confidence (or have an array of confidence values). In my morph registration algorithm, estimates of morph quality are available for every morph; but higher values imply poorer quality. Inverting these values and then normalizing them so that the values of the morphs being merged sum to one, gives a set of confidence values $\{\alpha_i\}$.

4.3 Selecting the Reference Image

The selection of the reference image used in morph registration has a large effect on the quality of the results. Multimorph images are cross-fades between warped images

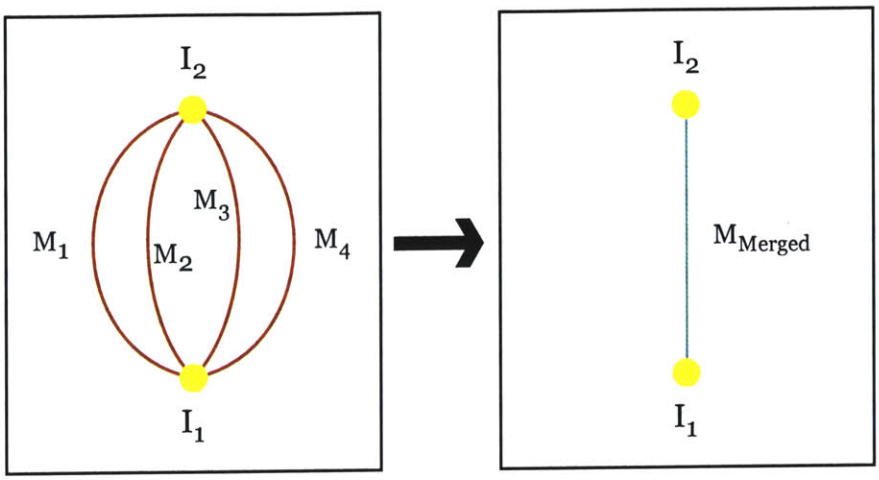

Figure 4-2: Merging four image morphs. The morphs M_1, M_2, M_3, and M_4 morph between image I_1 and I_2. They are merged into a single morph M_{Merged}.

from each input image. These warped images are constructed by chaining morphs between the input images and the reference image with a morph from the reference image to the shape of the multimorph image. If the warps from the input images to the reference image are of poor quality, then the warped images and hence the multimorph images will be of poor quality.

As discussed below, an estimate of morph quality is available for morph in the input graph. This estimate is a real value, with higher values implying lower quality. Further, the quality of a chain of morphs can be approximated as a function only of the quality estimate for each morph in the chain. The natural choice for the reference image, then, is to minimize the weighted distance of the reference image from every other image. This choice is subject to heuristics, however. One heuristic is to eliminate images considerably smaller than the average image size from consideration. A small reference image would cause large compression of space in most reference morphs. Another is to detect images for which all connected morphs are of poor quality. Selecting such an image may minimize distance to all other images, but at the cost of no better than a mediocre morph to any image. Tests like these, or more complicated ones, can be used to veto choices for the reference image, when otherwise deciding only on the basis of minimum path length to all vertices.

Minimizing the distance of the reference image from every other image can be done by solving an all-pairs shortest weighted path problem (run-time: $O(n_v^3 \log n_v)$).) The image (vertex) with the smallest mean or max distance to every other vertex is selected as the center vertex. The weights on each edge indicate the quality of the morph; a higher weight means a lower quality. The assignment of these weights is discussed below in Section 4.4.3.

Selecting the vertex with minimum distance is subject to the pruning discussed above: simple tests of morph quality of the edges leaving a vertex may remove possible vertices from consideration. When choosing from the remainder, one question that arises is whether the smallest mean or the smallest maximum distance should be

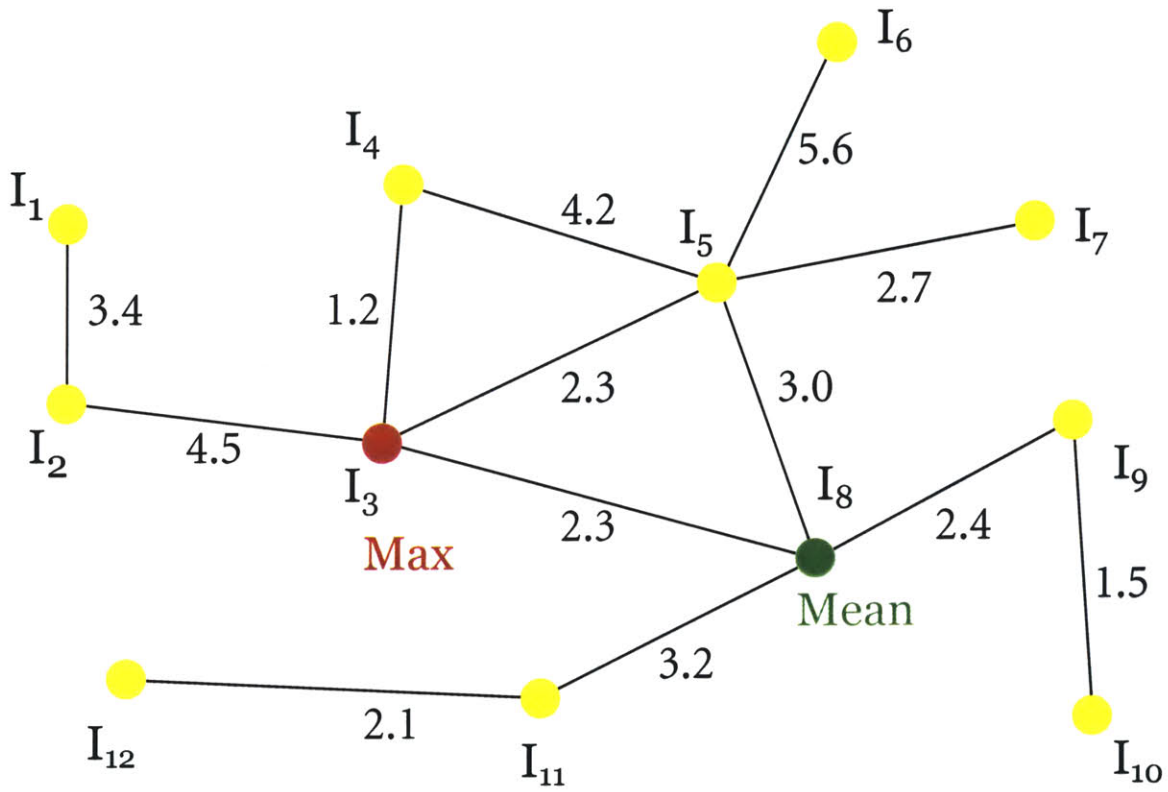

Figure 4-3: An illustration of selecting the reference image. The numbers along each edge are the weight assigned to the morph represented by that edge. In this example, the vertex with minimum mean distance to every other vertex is I_8, while the vertex with minimum maximum distance is I_3. Selecting I_8 for the center vertex is the best choice; it sacrifices a poor morph to I_1 for better morphs to I_9 through I_{12}.

used as the deciding factor. For a well-formed small graph, and for most large graphs ($n_v > 20$), the choice won't matter: the vertex with the smallest maximum distance from every other vertex will have a small mean distance as well. In a graph where it does make a difference, using the smallest maximum distance best prevents outliers in the data: vertices which are poorly represented because of their large distance from the center. Smallest mean distance improves the overall quality of the images produced from the system, at the cost of a higher degree of misalignment in the worst images. I'm assuming in my work that each input image is to some extent expendable: they are all equally important and some redundancies occur. In this case, the best choice is to use the smallest mean distance; the worst-aligned images can be given smaller weights when cross-fading, or removed entirely.

Since this method for selecting the reference vertex is heuristics-based, it may sometimes lead to a poor choice. A vertex just barely satisfying the image size and viable edges heuristics may be have a significantly higher mean distance than a vertex just barely not satisfying the heuristics. In such pathological cases, it is possible that manually checking the heuristics may yield a better choice of the reference image.

4.4 Choosing and Combining Paths

4.4.1 Objective

After the reference image is selected, morphs must be created from each input image to the reference image. These morphs are created by chaining the input morphs given as edges in the graph. A path of edges in the input graph corresponds to a series of morphs, which are chained to create a single *path morph*: a morph connecting the first and last images (vertices) of the path. Creation of the reference morph for an input image thus reduces to finding a path, or paths, between that image and the reference image.

When chaining morphs, loops in the path are counter-productive: revisiting a vertex implies that the chained morph consists of an initial morph chained with a morph from an image to itself. Since the objective of morph registration is feature alignment (as opposed to changing the shape of an image), the identity morph is the exact solution for a morph from an image to itself. The *identity morph* is just a morph M_{ij} with *identity warps* $\mathbf{W}_{ij}$ and $\mathbf{W}_{ji}$:

$$\mathbf{W}_{ij}(x, y) = \mathbf{W}_{ji}(x, y) = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{4.2}$$

Replacing any morph from an image to itself in a chain with the identity morph is equivalent to removing that morph from the chain. Since the identity morph is the exact solution, this replacement will improve or not affect the quality of the chained morph. Therefore no looping paths need be considered in morph registration.

4.4.2 Trivial Case: Acyclic Graph

In an undirected, acyclic graph, there is at most one non-looping path between any two vertices. Since the input to morph registration is a connected graph, this path exists, and the reference morphs are uniquely specified by choice of the reference image. A depth-first search from the reference node suffices to find the chain of morph edges for each reference morph.

4.4.3 General Case: Cyclic Graph

In a cyclic graph, some of the input images will have multiple paths to the reference image. I allow the case where it is important to consider more than the single best path. In general, the number of possible non-looping paths between two vertices is exponential in the number of vertices of the graph. The branching factor is proportional to the number of edges per vertex, so that in sparse graphs it may be possible to enumerate each path. In the following, I describe methods to deal with multiple paths when creating reference morphs, trading speed for thoroughness, if necessary. I begin by discussing my criterion for comparing the quality of morphs given by different paths.

Comparing Chained Morphs

As discussed above in Section 4.2.1, there are three factors that degrade the quality of a chained morph: morph quality along the chain, the length of the chain, and the compression and stretching of warped space along the chain.

The simplest option for comparing chained morph quality is to compare only length of the chain. This somewhat naive option is a reasonable approach when all input morphs and edges are of similar quality. Chain error is then dominated by the error introduced in each chaining, and so the shortest path is the best path.

On the other end of the spectrum is explicit evaluation of the quality of the morph function for every constructed path. This is a difficult problem, since morph quality is to some extent subjective. An approximate measurement is possible, however. As discussed in Chapter 2, there are several quantifiable problems which occur in warps: derivative discontinuities, foldover, and holes in the warped image. The extent to which the forward and reverse warps of a morph possess these problems can be measured. The total area in which the derivative of a warp is discontinuous is approximated by straightforward sampling and numerical integration. Foldover and holes in the warped image are detected by dividing the domain and range of a warp into a fine-meshed grid and counting the number of times each point in the destination grid is touched by a warped source grid square. (This measurement makes the implicit assumption that the warp can be approximated by a dense enough mesh warp.) If a destination point is touched by more than one warped source simplex, foldover occurs at that point. If a destination point is not touched by any source simplex, there is a hole at that point.

Explicit evaluation of a morph requires a large amount of computation. If many chained morphs need to be compared (as, for example, the numerous paths between

input vertex and reference vertex that occur in even moderately sparse graphs) then this method may be computationally infeasible.

A hybrid of these two methods is to give each morph edge a weight, and to base comparisons of chained morphs on the length of the weighted path of the morphs in the chain. Weights for each edge are calculated using the heuristics discussed above for evaluating the quality of a morph, or may be given as input. The weights are restricted to be non-negative, and larger weights imply lower morph quality. I allow the exact formula for the weight of a chain of morphs given the weights of each morph in the chain to be implementation specific, with the constraint that it must be a monotonically increasing function (discussed below). The standard solution is just to take the sum of the weights in the chain. A shortest path algorithm, or the best k paths algorithm described below, is then used to find the best paths from reference image to input images.

The hybrid solution is not computationally expensive, since it requires only linear time per input morph for the edge weights and $O(n_v^2)$ for the single source shortest weighted path problem (or a thresholded time for the best k paths). And it approximates ranking paths based on morph quality. This is a reasonable balance between complexity and accuracy.

Reference Morph Creation Method: Best Path Only

Given one of the methods above for comparing the quality of morphs, there are two choices for how the reference morph should be created. In the simplest method, the single highest-ranking morph out of all paths between an input image and the reference image is set as the reference morph for that input image. This algorithm has the advantage that, when using the weighted or unweighted path lengths to evaluate morphs, it reduces to a single-source shortest path problem, $O(n_v^2)$. When using the explicit evaluation method for comparison, it is not so simple, since long-chained paths may need to be considered. If the explicit evaluation is guaranteed to be monotonically increasing (that is, the quality of a chained morph is worse than the quality of either of its two input morphs), then it may be solved with the algorithm given below for finding the best k paths. There are two disadvantages to picking only the best path: it is fragile in the presence of failure of the best path heuristic, and it does not consider all user input.

Fragile to heuristic failure means that the algorithms described above for measuring chained morph quality are heuristics-based, and may err when ranking morphs. By selecting only one morph path to contribute to the reference morph, it is more likely that a heuristic failure will severely, negatively affect the reference morph.

Not considering user input means that some of the input morph edges do not contribute to any of the reference morphs. This is an obvious corollary of selecting all shortest paths from a vertex to every other vertex: it will not, in general, include all edges of the graph. This is not necessarily a difficulty, however. Depending on the application, it may be unimportant for a particular user-created morph to be part of the system.

Reference Morph Creation Method: Best k Paths

The second method for creating the morph from an input image to the reference image is to combine the morphs from multiple paths into one. Merging morphs reduces the chance that a single poor-quality path morph will corrupt the reference morph due to a failure of the heuristics for chained morph comparison. It is in general a more robust algorithm than selecting a single morph: this is the same principle as averaging several samples of a physical measurement instead of using a single measurement. Since merging morphs of good quality with morphs of poor quality is likely to result in a morph of poor quality, it is necessary to merge only the best morphs. The following algorithm describes how to approximately find the best k paths from a given vertex (the reference image) to every other vertex (all input images). Best in this case means the paths of minimum weight (weight is inversely proportional to morph quality).

This algorithm requires only that path values be monotonically increasing: if a path from I_a to I_b has value w, and there is an edge from I_b to I_c, then the same path from I_a to $_b$ followed by the edge from I_b to I_c has value greater than or equal to w. The algorithm therefore can be used together with any of the three morph comparison methods, assuming that explicit evaluations of chained morphs can be shown to be monotonically increasing. The computational cost of this algorithm is proportional to the computation needed to calculate the morph quality of a path when extended by one edge.

BestKPaths$(k, G(V, E, W), v_0, max\_size, max\_ratio)$
finds the paths of minimum weight in an undirected, non-negative weighted graph G. For each vertex in the graph, it finds the k paths of minimum weight from the vertex to a center vertex v_0. If less than k paths can be found, it returns all discovered paths. The paths for all vertices are calculated in one search of the graph. A path starting at v_0 is inserted into a search queue. Then, until the queue is empty, the top (lowest weighted) path on the queue is extended into several new paths: each edge departing the path's terminal vertex is appended to create a new path, so long as this does not introduce a loop. The top path is removed from the queue and the extended paths are added. The removed path is added to a list of paths to its terminal vertex; a maximum of the k best paths to that vertex are kept in the list.

The search is bounded by removing high-weight paths from the queue of paths being extended. When the search queue has size greater than *max_size*, the highest weighted paths on the queue are removed. Additionally, as a preprocess, shortest weighted paths to each vertex are computed. Any path being considered that has weight greater than *max_ratio* times the weight of the maximum shortest weighted path is not added to the search queue.

1. (Initialize the data structures.) The list of paths $Q[v]$ to each vertex v is set empty. The Queue of search paths S is set to contain a path starting and ending at v_0.

   (a) For each v in V, set $Q[v] \leftarrow$ Empty.

   (b) Set $S \leftarrow$ Empty.

   (c) Insert Path($\{v_0\}$) into S.

2. (Calculate shortest weighted path.) Find upper bounds on the weights of paths by finding the shortest weighted path to any vertex.

   (a) Set $shortestPaths \leftarrow$ SHORTEST_WEIGHTED_PATHS(G, v_0).

   (b) Set $minSearchWeight \leftarrow \max_{v \in V}$ weight$(shortestPaths[v])$.

3. (Run all-paths search.)
   While(S not empty) Do
   (a) Set $path \leftarrow$ top(S).
   pop(S).
   Set $v_a =$ terminal_vertex($path$).

   (b) For each $v_b \in \{v_b : (v_a, v_b) \in E\}$
   If not v_b in $path$ Then
   $extendedpath =$ concat$(path, v_b)$
   If weight$(extendedpath) \leq max_ratio * minSearchWeight$ Then
   Insert $extendedPath$ into S.
   End If
   End For

   (c) Insert $path$ into $Q[v_a]$.

   (d) If size$(Q[v_a]) > k$ Then
   popHighest$(Q[v_a])$
   End If

   (e) If size(S) > max_size Then
   removeHighestN(S, size(S) - max_size)
   End If

   End While

4. (Collect the paths for each vertex.) For each $v \in V$, the best paths to v that were discovered are now the lowest-weighted elements of $Q[v]$. Return the k lowest weighted elements out of each $Q[v]$, or the entire queue if size$(Q[v]) < k$. If $Q[v]$ is empty, return $shortestPath[v]$.

This algorithm approximates a brute-force search of all paths. If the thresholds used in pruning the queues are set to infinity (pruning is not done) then this algorithm

is exactly equivalent to listing all paths and taking the k lowest-valued paths to each vertex. The pruning of the queues by maximum size prevents all paths in a dense graph from being searched, and so prevents the run time from being exponential (Run time is bounded by $n_v * max\_size$.) This is a well-behaved modification: since higher-valued paths values are pruned before lower-valued paths, and since path values are monotonically increasing, then if the thresholds are high compared to k, the k best paths will likely be discovered. The pruning of the queue by value is done for efficiency; it does not effect the accuracy of the approximation, if *max_ratio* is set appropriately. (For example, $max\_ratio \leftarrow 10$ implies that morphs with weight more than 10 times the maximum distance from the reference image should not be considered.)

Once the k best paths have been found for each vertex, they are merged to create the reference morph for each input image.

4.5 Representing Multimorph Images as a Vector

4.5.1 Creation of Multimorph Images

The reference morphs constructed above provide global feature alignment between all input images. Multimorph images are created by warping each input image into alignment and then cross-fading the warped images. The default alignment is the reference image, since each reference morph warps an input image to the reference image. The feature positions are changed by specifying a further warp of the reference image. I refer to this warp as the *geometry warp*. Chaining the geometry warp with the morph from an input image to the reference image produces a warp from the input image to the new feature alignment.

The domain of multimorph images consists of a geometry warp to specify feature positions and an array of weights for each image in a multi-image cross-fade. There are two problems with using this interface directly. First, it can produce images that do not resemble the input images and are not pictures of the same category of objects. Since I've placed no restrictions on the geometry warp function, there are essentially no restrictions on the space of output images. Second, using arbitrary geometry warps to specify output images requires intensive input. In the design of my system, I expect that the only extensive input required is given in initialization: the two-image morphs between input images. Specifying the geometry warp by creating a two-image morph at run time is possible but not desirable. The solution to both of these problems is the same: I restrict the domain of allowable geometry warps to warps related to the input image feature positions, and this domain consists of (relatively) few parameters. I create this restriction by representing geometry warps as vectors, which I describe below.

4.5.2 Creation of Multimorph Images from Vectors

A morph based on feature points may be represented as two vectors. The points in each image of the morph are serialized such that each point occupies two elements of a

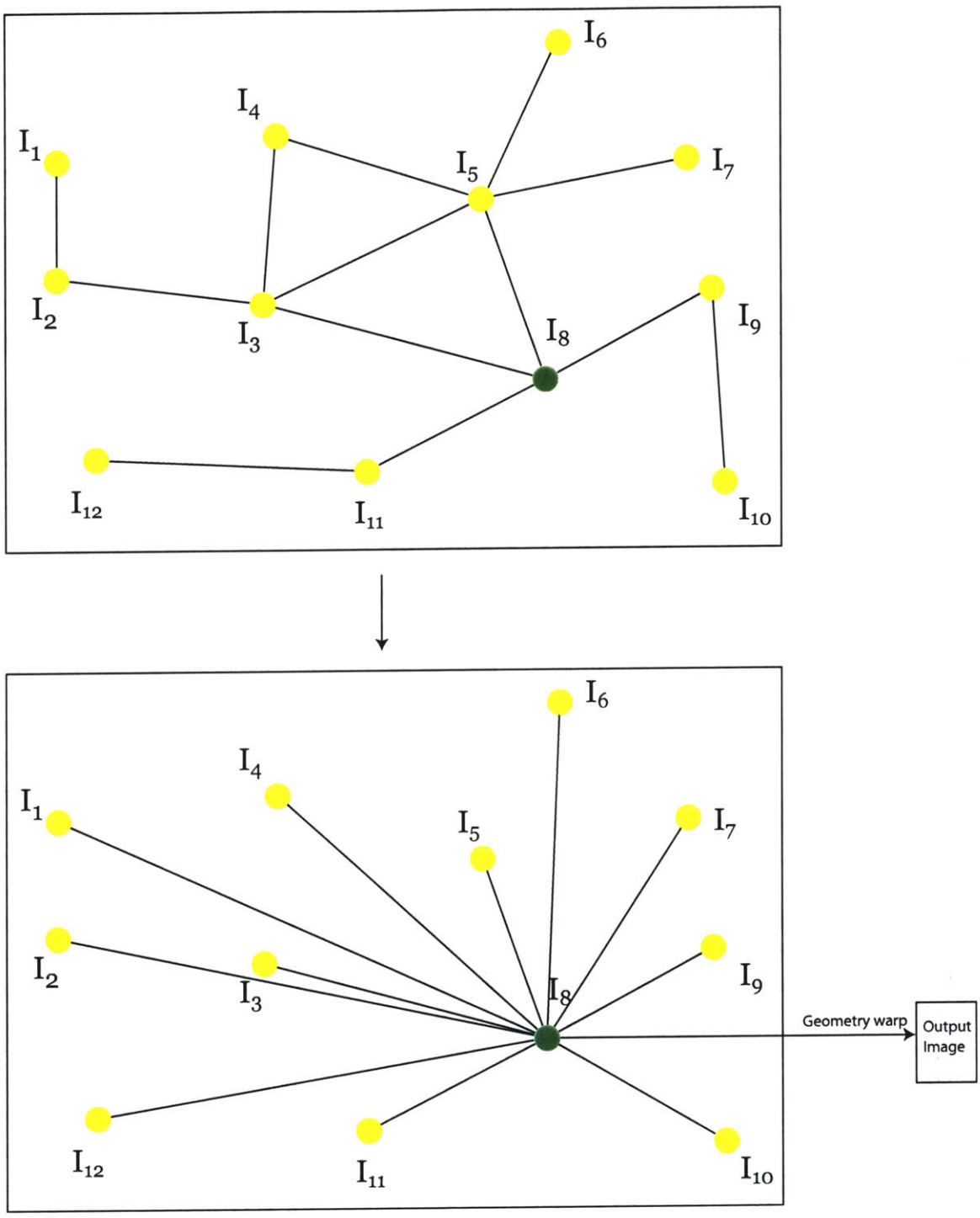

Figure 4-4: After creating the reference morphs, the original graph of two-image morphs has been converted into a new structure with a morph from every image to the reference image and no other. Multimorph images are created by chaining the warp from each image to the reference image with a warp of the reference image to the a desired shape. Each input image is warped to this shape, and the images are cross-faded.

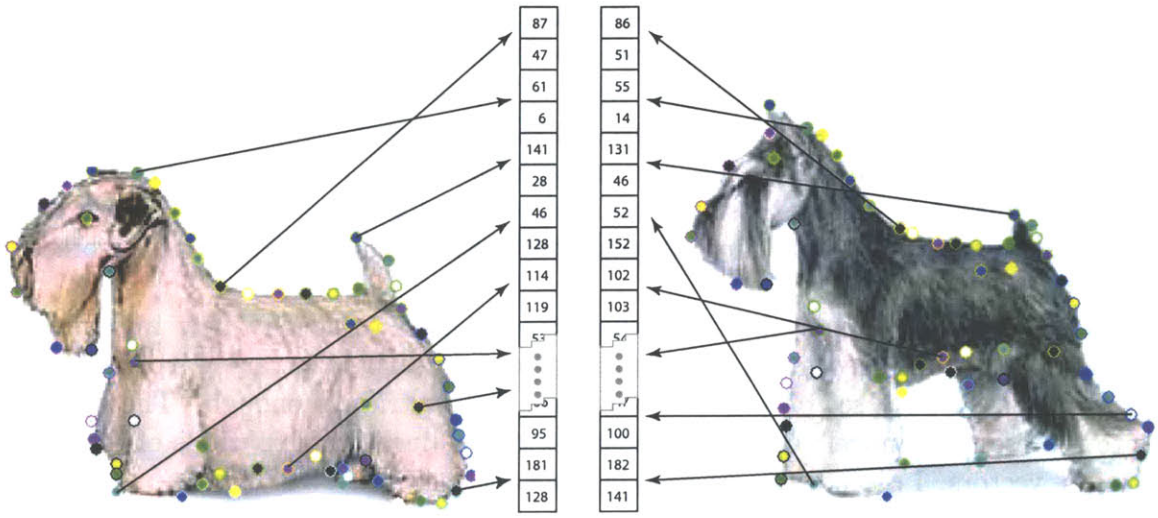

Figure 4-5: A morph based on point features can be represented as two vectors. Each feature point in each image occupies two positions in a vector. Images courtesy of the American Kennel Club.

vector. Using zero-based vectors (the 0th dimension is the first dimension), dimension $(2i)$ of the vector is the x coordinate of the ith feature point, and dimension $(2i+1)$ is the y coordinate. A point feature based morph with n_p points can therefore be represented as $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$, where, for $i = 1$ To n_p, $\begin{bmatrix} \mathbf{v}_1[2i] \\ \mathbf{v}_1[2i+1] \end{bmatrix}$ in the first image corresponds to $\begin{bmatrix} \mathbf{v}_2[2i] \\ \mathbf{v}_2[2i+1] \end{bmatrix}$ in the second image.

For multimorphs, I create a vector for each image in the same way. First, however, the reference morphs must be *resampled* onto a grid of points. Resampling an image morph means using the morph to find the warped position of features that are then used to represent the morph in a different way. I create a grid of points in the reference image, and resample all of the reference morphs onto this grid. Warping the grid by a warp from the reference image to an input image returns the points in the input image corresponding to the grid points. These point correspondences then define a point feature based warp. (Note that I am now adding a restriction to the two-image morphing algorithm I previously treated as a black box: the morphing algorithm must be based on point features.)

I set the density of the grid in the reference image to pixel-density. This suffices in most cases, since the details of the morph on a finer grain than pixels is unimportant. In pathological data, areas containing many pixels in some images may be warped into a single pixel in the reference image, and information about these areas is lost when the grid is resampled. To handle this case would require further calculations to determine the (perhaps adaptive) density of the grid. I did not consider these cases in my work, however. For computational reasons, pixel-density may be too dense of a grid. In this case the grid should be made as dense as possible, with morph quality being traded for speed.

The vector of grid points in the reference image and the vectors of warped grid points in the input images now define morphs from every input image to the reference image. Further, since all the vectors refer to the same point features, they are *all* in correspondence, and any two vectors represent a morph between their respective images. And in general, any vector of the same cardinality represents a warp of all of the input images. I refer to these vectors as the *geometry morph vectors*. Determining the feature positions of a multimorph image now reduces to specifying a new geometry morph vector.

Not just any geometry morph vector can be used to specify new multimorph images. As I mentioned above, arbitrary warps (or nearly arbitrary warps now that they are based on a set of feature points) are not restrictive enough. The solution is to represent the novel geometry morph vectors as linear combinations of the input images' vectors. I discuss this process in the next chapter.

In this chapter I discussed the process for generating morphs to a common reference image. I gave an algorithm for evaluating two-image morphs, producing weights that are used to select the reference image and compare possible paths (series of chained warps) between one image and another. I described how to use these weights to chose the reference image. I gave an algorithm for merging the paths from an input image to the reference image to create a reference morph. And I described how to use the reference morphs to create geometry morph vectors that represent the shape of each image. In the next chapter, I discuss how to represent the texture (pixels) of each image as a set of aligned vectors, and then describe how the geometry and texture vectors may be interpolated to produce new images.

Chapter 5

Dimensionality Reduction

I now describe my method for constructing new morph vectors. In this chapter, I first describe the use of the geometry morph vectors developed in Chapter 4 as the high-dimensional input to dimensionality reduction. I then introduce texture vectors, which are an alternate way of expressing the coefficients used to cross-fade warped input images. And finally, I describe how to create new geometry and texture vectors from the input vectors. Geometry and texture are created separately, but using similar techniques. I give three methods for creating the new vectors: linear combinations of the input vectors, linear combinations of the singular vectors, and non-linear dimensionality reduction and reprojection.

5.1 Dimensionality Reduction Input

5.1.1 Geometry Vectors and Cross-fade Coefficients

Morph registration creates a geometry morph vector for each image; from here on I refer to these vectors as the geometry vectors. Each geometry vector is a list of the positions of a set of corresponding point features in its image. I interpret any new geometry vector as a new shape for this set of points, such that it describes a point feature based warp of every input image. This is a change in the way in which I have been viewing morphs: previously when describing a morph I put emphasis on the warps of each image; the control features were just a means to that end. With geometry vectors, it makes more sense to consider each vector of point features as the *shape* of an image, and the warps are just a means of changing the current shape of each image.

Any new geometry vector specifies a new shape for the input images. Most geometry vectors will produce a shape that does not resemble the input image shapes. The 0-vector, for example, is a valid vector that specifies a shape in which the entire image is compressed into a single point. Avoiding shapes such as this is one of the main reasons for this chapter.

For each geometry vector, n_v new images are available - the warped input images. These images are cross-faded to produce the final multimorph image, weighted by a coefficient for each image. The selection of these weights is just as important as the

selection of the geometry vector. They essentially select the texture that is painted onto the shape of the geometry vector. Using only cross-fade coefficients, it is difficult to select a texture with desired properties. This information is hidden inside each warped image. Exposing this information can be done in a number of ways. I discuss my method - texture vectors - in the next section.

A new geometry vector and a set of cross-fade coefficients (or a texture vector) describe a new image. Creating geometry vectors and cross-fade coefficients based on a reduced set of parameters is the objective of this chapter.

5.1.2 Warped Geometry Vectors

As discussed in Section 3.2.2, linear combinations of corresponding points in two images do not correctly handle the overall rotations of the images. Since linear combinations of geometry morph vectors are used to create new geometry morph vectors, rotations will not be accounted for in multimorph images unless the geometry morph vectors are prewarped.

To solve this problem, I create *warped geometry morph vectors.* A desired orientation is specified by an affine transform of the reference image. (This transform may be restricted to disallow some degrees of freedom such as shear or change in aspect ratio.) This transform is then applied to the geometry morph vector of the reference image to create the *reference warped geometry vector.*

For each geometry morph vector, the average transform from that vector to the reference warped geometry vector is calculated. The geometry morph vector is transformed by this average transform to create a warped geometry morph vector. The warped geometry morph vectors are used in linear combinations to create the new geometry vectors; they are all in the same orientation, so rotations will be correctly handled. The unwarped geometry morph vectors are preserved; they are still needed to specify the morph from each input image to the shape of the new geometry morph vectors.

5.1.3 Geometry and Texture Notation

When referring to morph vectors, I use d_g for the dimensionality of the geometry morph vectors, $\mathbf{g}_i$ for the warped geometry morph vector of image i (i from 1 to n_v), and $\mathbf{g}$ for an arbitrary geometry vector. Similarly, for the texture vectors introduced below, the dimensionality is d_t, the texture vector of image i is $\mathbf{t}_i$, and an arbitrary texture vector is $\mathbf{t}$. When discussing the vector of coefficients used in the cross-fade to produce the multimorph image, I use the term $\mathbf{c}$: c_i is the coefficient of the ith warped image.

5.2 Texture Vectors

Specifying cross-fade coefficients directly is undesirable for two reasons. First, coefficients used in a cross-fade have a restricted domain. As mentioned in Section

2.1.1, they should sum to one and be non-negative to prevent changes in luminosity or other artifacts. Requiring coordinates to sum to one projects space onto a hyperplane. This introduces an additional layer of complexity when creating new vectors in the space: entire sets of vectors are mapped to the same point before being used in the actual cross-fade. This must be taken into account when moving through the space of vectors in search of particular characteristics. The second disadvantage of direct cross-fade coefficients is that they carry no information about the images they create. The value of each element is a weight given to another vector (the pixels of a warped image) and does not, for example, contain the value of any particular pixel. This makes data-mining techniques such as dimensionality reduction less effective.

5.2.1 Definition of Texture Vectors

The alternative to direct specification of cross-fade coefficients is *texture morph vectors.* Each input image defines a texture morph vector. The texture morph vector, or texture vector, is fundamentally a vector of the pixels of an image. To be more specific, however, the texture vector is the pixels of the input image after being warped to a common alignment. The input images are of different sizes, and if they were extended to be the same size, the pixels would still be out of alignment - this is the same problem as in cross-fading. So the images are warped to a common shape, using the geometry morph vectors. The geometry vector specifying the common shape is the reference warped geometry vector. In this context, I refer to the geometry vector used for the shape of the texture vectors as the *texture vector shape.*

5.2.2 Implementing Texture Vectors as Bitmaps

To create texture vectors, the warped images must be represented as bitmaps (i.e. finite in extent). The extent of the warped images varies depending on the morph algorithm used and (for some morph algorithms) depending on the input image. However, most morph algorithms produce poor results outside of the convex hull of the feature points. Further, the feature points represented by the morph geometry vectors were selected in Chapter 4 to cover the entirety of the reference image, and it is therefore reasonable to assume that they will also cover the entirety of any warped shape. I therefore use the bounding box of the feature points in a geometry vector as the size of the multimorph image for that vector. Because some morph algorithms do draw pixels outside of the convex hull, I extend the hull by some small percentage (say 5% on each side). This typically provides enough space for all warped images. Unwritten pixels are set to a blank color (i.e. black).

The pixels of an input image after being warped to the texture reference shape are the texture morph vector for that image. Since the vectors will be added and multiplied, each pixel primitive should occupy a vector element. This means, for example, that an RGB pixel takes up three elements of the vector instead of one. The dimensionality of the texture vectors is the size of the warped images, multiplied by the number of elements per pixel: $d_t = 3 * \text{width} * \text{height}$ when using RGB. (This size is the same for each warped image.)

5.2.3 Warping Texture Vectors versus Converting to Cross-fade Coefficients

An important point when dealing with texture vectors is whether they should be used in warps directly, or converted back to cross-fade coefficients.

Warping Texture Vectors

Dimensionality reduction creates a low dimensional embedding space from the high-dimensional texture vectors, and the user provides a low-dimensional point to specify new samples (discussed in Chapter 6). From this embedding space, the primary purpose of dimensionality *reprojection* is to create a high-dimensional sample point corresponding to the user-provided low-dimensional point. When warping texture vectors directly, the reprojected high-dimensional vector lists the pixels of an image in the shape of the texture reference shape. This image is then warped to the shape of the new geometry morph vector to create a multimorph image (no further cross-fading is required).

One disadvantage to this technique is that the reprojected high-dimensional vectors may not fulfill all requirements of texture vectors. When using RGB, each element of a texture vector must be between 0 and 255 (or between some other minimum and maximum value). Convex linear combinations preserve this property, but it is not guaranteed to hold in all forms of dimensionality reprojection. If this requirement is not satisfied, the reprojected texture vector must be further altered by clamping or rescaling the vector.

A second problem with direct warping of texture vectors is that the shape of a reprojected texture vector is the texture reference shape. It must be warped to the shape given by the reprojected geometry vector to construct a multimorph image. The reprojected texture vector, however, is derived from a set of images which have already been warped (the texture vectors of each input image). Warping the reprojected texture vector adds an extra sampling step, which degrades image quality.

Converting to Cross-fade Coefficients

The alternative to warping texture vectors is to reconfigure dimensionality reprojection to reproject coefficients of the input vectors. Dimensionality reprojection typically produces a new high dimensional point. It may be modified, however, to produce instead a set of coefficients of the input vectors that best correspond to the embedding coordinates. When using linear combinations of input vectors or non-linear dimensionality reduction, these coefficients are actually the default: the high-dimensional reprojected vector is the weighted sum of input vectors using these coefficients. When using linear combinations of singular vectors, these coefficients are implied - they are given exactly by multiplying the embedding coordinate by the V matrix of the singular value decomposition.

Once the input vector coefficients are calculated, they can be used as cross-fade coefficients. Each of the input images is warped to the shape given by the reprojected

geometry vector. The multimorph image is then the cross-fade of these warped images weighted by the reprojected cross-fade coefficients, exactly as if the coefficients had been directly specified without referring to the texture vectors.

Since the coefficients are used in cross-fading, they must be nonnegative and sum to one. For embedding points inside the convex hull of input points, these constraints are always satisfied. For points outside the convex hull, however, the nonnegativity constraint may be impossible to satisfy. For these points, either the nonnegativity constraint must be relaxed or the points must be projected onto the convex hull. This is the major disadvantage of converting back to cross-fade coefficients.

5.3 Creating New Morph Vectors

Each input image has a geometry vector and a texture vector that describe the image. To create new images, new geometry and texture vectors are constructed; the geometry vectors and the texture vectors are in general unrelated. To create images which appear to be similar to the input images, the fabricated vectors must be related to combinations of the input vectors. In the following, I describe three possible methods to create new morph vectors from the input morph vectors. Any of these three methods may be used with either the geometry morph vectors or the texture morph vectors. Although I mention both geometry vectors and texture vectors in the discussion of each method, this is just the point out the details that arise from using texture vectors to reconstruct cross-fade coefficients. The method to reconstruct geometry need not be the same as that used to reconstruct texture.

5.3.1 Linear Combinations of Input Vectors

The simplest method to create new morph vectors is to take linear combinations of the input vectors:

$$\mathbf{g} = \sum_{i=1}^{n_v} \alpha_i * \mathbf{g}_i \tag{5.1}$$

or

$$\begin{aligned} \mathbf{t} &= \sum_{i=1}^{n_v} \beta_i * \mathbf{t}_i \\ \mathbf{c} &= \beta. \end{aligned} \tag{5.2}$$

I've used α_i as the coefficients of the geometry vectors, and β_i as the coefficients of the texture vectors, as is standard in two-image morphing. In general, α and β are unrelated. Note that if texture vectors are converted back to cross-fade coefficients, this dimensionality reduction is equivalent to specifying the cross-fade coefficients directly. The values for β are nonnegative and sum to one.

An immediate improvement that can be made is to recenter the new morph vectors

about the mean of the input images:

$$\mathbf{g} = \bar{\mathbf{g}} + \sum_{i=1}^{n_v} \alpha_i * (\mathbf{g}_i - \bar{\mathbf{g}}) \tag{5.3}$$

or

$$\begin{aligned} \mathbf{t} &= \bar{\mathbf{t}} + \sum_{i=1}^{n_v} \beta_i * (\mathbf{t}_i - \bar{\mathbf{t}}) \\ \mathbf{c} &= \beta. \end{aligned} \tag{5.4}$$

This recentering is an improvement for two reasons. First, all vectors given by α and β after recentering are restricted to lie on the linear manifold spanned by the input vectors. To put it another way, the output vectors can vary from the mean only along the directions laid out by the input vectors. This restricts the space of the output vectors, making them more likely to retain the appearance of the class of input images.

Second, values of zero for α and β now specify the mean of the input vectors. (Hence the term "recentering".) Points inside and around the convex hull of the input vectors can be specified with values of α and β around the space $-1 \leq (\alpha_i, \beta_i) \leq 1$. As the magnitudes of the α and β coordinates grow greater than 1, the constructed vectors move farther outside the convex hull of the input vectors and become less similar to the input images. For small extensions (say $\{|\alpha_i|, |\beta_i|\} \leq 5$) the results remain similar to the input images. The input images are scattered samples of a class of images, and it is unlikely they are on the very extreme ends of the class. (And since most image classes are fuzzy definitions, the region of valid-image producing vectors has thick boundaries.) But as the vectors become significantly far away from the convex hull of the input vectors, the results tend to degrade, looking less like members of the class represented by the input images.

5.3.2 Linear Combinations of Singular Vectors

An alternative to direct linear combinations is to use the methods of Principle Component Analysis (PCA). As described in Chapter 3, PCA takes a set of n_v high dimensional vectors and finds a set of high-dimensional vectors which span the same space about the mean as the input vectors, using the singular value decomposition. These vectors (the singular vectors) are orthogonal and are sorted by increasing order of variance in the input vectors. The projection of the input vectors on the singular vectors gives a low-dimensional embedding.

Linear combinations of the singular vectors can be used to specify constructed high-dimensional vectors in the same way as linear combinations of the input vectors: α_i is now a coefficient of the ith singular vector, instead of the ith input vector. Let the singular vectors for the geometry input vectors be $\{\mathbf{gs}_i\}$, and let the singular vectors for the texture vectors be $\{\mathbf{ts}_i\}$. The input vector coefficients required by texture vectors are given exactly (i.e. not least squares) by the projections of each

singular vector on the input vectors; let the projection of the jth singular vector on the ith input vector be P_{ij}. These three sets of vectors are given by

$$\begin{aligned} \{\mathbf{gs}_i\} &= \text{SingularVectors}(\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{n_v}) && (5.5) \\ \{\mathbf{ts}_i\} &= \text{SingularVectors}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{n_v}) && (5.6) \\ \mathrm{P} &= \text{SingularVectorProjections}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{n_v}). && (5.7) \end{aligned}$$

SingularVectors returns the US matrix of the singular value decomposition of its arguments, and SingularVectorProjections returns the V matrix. I multiply U by S instead of V by pseudoInverse(S) so that the domain of embedding coordinates corresponding to the full range of the input points is 0 to 1 instead of 0 to the singular value of each singular vector. The number of singular vectors with non-zero variance is at most $n_v - 1$, so the constructed image vectors are given by

$$\mathbf{g} = \bar{\mathbf{g}} + \sum_{i=1}^{n_v-1} \alpha_i * \mathbf{gs}_i \tag{5.8}$$

or

$$\begin{aligned} \mathbf{t} &= \bar{\mathbf{t}} + \sum_{i=1}^{n_v-1} \beta_i * \mathbf{ts}_i && (5.9) \\ \mathbf{c} &= \text{Projection}(\mathrm{P}\vec{\beta}) \end{aligned}$$

To preserve the sum-to-one constraint on the cross-fade coefficients, $\mathbf{c}$ is projected onto the nearest point in the region defined by $\{\sum_{i=1}^{n_v} c_i = 1, c_i \geq 0\}$.

This method has some advantages over linear combinations of the input vectors. Since the vectors are sorted by variance, new images can be specified in a coarse-to-fine manner: the dominant features of a geometry or texture vector are selected first, and then refined as needed. This is useful for user input and for numerical stability. As far as the cross-fade coefficients are concerned, this method is only a slight improvement over linear combinations of input vectors: more information is available to describe the texture produced by an embedding coordinate, but the embedding space is warped by projection onto the sum-to-one hyperplane. Note that $\vec{\beta} = \vec{0}$ is not a valid point, but all other values are, since P is orthonormal. (This assumes that the length of $\vec{\beta}$ is the number of non-zero singular values. If β is longer, then the non-zero requirement applies to elements of β corresponding to the non-zero singular values.)

PCA finds a more convenient set of basis vectors for the linear manifold described spanned by the input vectors, but otherwise does not change the space of high-dimensional vectors produced.

5.3.3 Non-Linear Dimensionality Reduction

Linear combinations of the input vectors (including PCA) are global linear combinations. *Global linear combinations* are combinations of all input vectors: each vector

is multiplied by a coefficient and added to the sum. A coefficient may be zero, removing the vector from the sum, but in general it is not. Global combinations do not always produce good results. For a complicated class of images, the manifold of geometry and texture vectors representing the class is not guaranteed to be a linear manifold. Thus linear combinations of the input vectors' deviation from the mean are not guaranteed to remain on the manifold. For morph vectors, leaving the manifold by definition means that the image produced by the morph vectors is not a member of the class of images given by the input images. The farther away from the manifold the morph vectors lie, the less the constructed image will look like a member of the class of input images.

There are two types of solutions to this problem. The first solution is to use non-linear combinations of the input vectors. If the manifold is non-linear, then a non-linear function of the input points must be needed to reconstruct the points. An example of this method is finding the best quadratic fit to the input points, and using this function to interpolate output points. This is an improvement over simple linear combinations, since the space of quadratic manifolds is a proper superset of the space of linear manifolds. The drawback to this method is that the actual manifold is not guaranteed to be quadratic, either. More and more complicated interpolation functions can be used, but for high-dimensional problems, the non-linearity of the manifold is usually more complicated than can be matched by a global fit.

The second solution is to approximate the non-linear manifold using locally linear patches. As described in Chapter 3, isomap and LLE are methods of dimensionality reduction that find an embedding on a non-linear manifold, assuming that the manifold is densely sampled enough to be locally linear. Unfortunately, this assumption of local linearity is not likely to be true in the types of problems presented by morph vectors, because the dimensionality of the morph vectors is significantly greater than the number of morph vectors. Morphs are by nature expensive to create (requiring large amounts of artistic input) and so it is reasonable to expect that multimorph input has less than one thousand images, or even less than one hundred. The dimensionality of the geometry and texture vectors is on the order of ten thousand or more. This means that the space is sparsely sampled.

Using non-linear dimensionality reduction and reprojection is still an improvement, however, because of the restriction to linear combinations of only the closest vectors. The possible deviation from the manifold of linear combinations of input vectors decreases as the distance between the combined input vectors decreases. (This assumes that the manifold distance metric is proportional to the Euclidean distance from the manifold.) The effect of using non-linear dimensionality reduction and reprojection is to create a very coarse grid of simplices to represent the manifold. This is enough to capture large-scale non-linearity.

Of these two solutions, non-linear dimensionality reduction is the most feasible. I apply Isomap, LLE, or a similar technique to the high-dimensional input points. Isomap and LLE produce an embedding space of n_v dimensions. It is possible that the full range of data is captured by an embedding space of fewer dimensions, in which case only the embedding dimensions of 1 to n_l are used. The choice of n_l is based on the measurement of approximation accuracy provided by LLE and Isomap, and is

left to the user. Dimensionality reprojection projects the n_l-dimensional coordinates onto a high dimensional vector, or onto a vector of input vector coefficients.

Let the dimensionality reduction used (LLE or isomap) on a set of vectors $\{\mathbf{v}_i\}$ create an embedding matrix $\mathrm{E}(\{\mathbf{v}_i\})$ (E^i is the embedding coordinate of $\mathbf{v}_i$). Let dimensionality reprojection be represented by the function $\mathbf{r}(\mathbf{E}, \mathbf{v})$, which takes an embedding E and a low-dimensional point $\mathbf{v}$ as input, and let $\mathbf{rc}(\mathbf{E}, \mathbf{v})$ be dimensionality reprojection onto input vector coefficients. Let G be the reduction of the geometry vectors, and T be the reduction of the texture vectors:

$$\begin{aligned} \mathrm{G} &= \mathrm{E}(\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{n_v}) \\ \mathrm{T} &= \mathrm{E}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{n_v}). \end{aligned} \tag{5.10}$$

Let the user provide $\vec{\alpha}$ and $\vec{\beta}$ variables to the reprojection functions. Then the morph vectors constructed using non-linear dimensionality reduction and reprojection are

$$\mathbf{g} = \mathbf{r}(\mathrm{G}, \alpha) \tag{5.11}$$

or

$$\begin{aligned} \mathbf{t} &= \mathbf{r}(\mathrm{T}, \beta) \\ \mathbf{c} &= \mathbf{rc}(\mathrm{T}, \beta). \end{aligned} \tag{5.12}$$

5.4 Summary

Any of the three techniques described above produce a geometry vector and either a texture vector or a set of cross-fade coefficients for a given pair of embedding points $\vec{\alpha}, \vec{\beta}$. These constructed structures are used to create multimorph images. In the next chapter, I describe how to select the embedding points α and β that produce a desired image.

Chapter 6

Specifying New Images: Input Combinations and Traits

After morph registration, new multimorph images are represented by geometry morph vectors and cross-fade coefficients. Dimensionality reduction creates two low-dimensional embedding spaces. Points in the geometry embedding space specify geometry vectors, and points in the texture embedding space specify texture vectors or cross-fade coefficients. A pair of embedding coordinates $(\vec{\alpha}, \vec{\beta})$ in the geometry and texture embeddings spaces, respectively, specifies a new image. In this chapter, I discuss my tools for selecting pairs of embedding coordinates based on desired output image characteristics.

Since the dimensionality reduction and reprojection for geometry and texture vectors are similar, when describing operations that apply to both of them I express the operation in terms of the geometry vectors, using α for embedding coordinates and $\mathbf{g}$ for high dimensional vectors. When discussing operations relevant only to cross-fade coefficients, I use the texture terms: β for the embedding coordinates, $\mathbf{t}$ for high-dimensional vectors, and $\mathbf{c}$ for the vector of cross-fade coefficients.

6.1 Reconstructing Input Images

6.1.1 Reconstructing a Single Image

A simple test of a multidimensional morph system is how it reconstructs its input images. In my system, the embedding coordinates of each input image are given explicitly by the output of dimensionality reduction. Converting one of these pairs of coordinates to an image reconstructs its input image.

With linear combinations of input vectors, the embedding coordinates of I_i are $\mathbf{e}^i$: the elementary vector with $\mathbf{e}^i_i = 1$ and $j \neq i \rightarrow \mathbf{e}^i_j = 0$. With linear combinations of singular vectors, the coordinates are given by the embedding output of principle component analysis: the V^T matrix of the singular value decomposition, when the singular vectors are taken to be UV as in my system. With non-linear dimensionality reduction and reprojection, the coordinates are given by the output of either isomap

or LLE.

Using dimensionality reprojection on these points returns the original high-dimensional morph vector exactly, subject to numerical error. With linear combinations of singular vectors, this is trivially true. With linear combinations of singular vectors, the reprojection of an input coordinate reduces to $\mathbf{g} = \mathrm{US} * \vec{\alpha}(\mathbf{g}_i) = \mathrm{US} * (\mathrm{V}^T)^i = \mathbf{g}_i$. With isomap or LLE, dimensionality reprojection finds the least squares projection of $\vec{\alpha}$ on the nearest neighboring input embedding coordinates. When $\vec{\alpha}$ equals one of these coordinates exactly, the weight of that input coordinate is 1, and all other weights are 0, so that the reprojected high-dimensional vector is the input vector.

The same reasoning applies for the input vector weights for the texture cross-fade coefficients: the cross-fade vector returned has a weight of 1 for the input image and a 0 elsewhere.

Reprojection of an input image coordinate therefore returns the warped geometry morph vector and texture morph vector (or set of cross-fade coefficients) of that image. (Recall that the input vectors used to create the geometry embedding space are the warped geometry vectors. Multimorph images, however, are created by using the morph defined by the new (warped) geometry vector and the input images' (unwarped) geometry vectors.)

If cross-fade coefficients are used, the new multimorph image is created by a morph from the geometry morph vector of an image to its warped geometry morph vector. This is just an affine transform of the input image, if rotations are handled correctly by the morph algorithm. If texture vectors are warped directly, then creating the multimorph image corresponding to the embedding coordinates of an input image can be seen as a two stage process. First the input image is warped to the texture reference shape and sampled, creating the texture vector. And second, this warped image is warped to an affine transformation of the input image (the new geometry vector), and sampled, creating the multimorph image. This two-stage sampling introduces artifacts into the image, but it is otherwise identical to an affine transform of the input image.

6.1.2 Constructing Image Combinations

The logical extension to reconstructing input images is to construct combinations of input images, as is done in two-image morphs. A multidimensional morphing system can imitate a two-image morph between any of its input images, using the geometry vectors of the images to define the two-image morph. A larger extension, however, is the ability to combine three or more images at once. In general, this amounts to global linear combinations of the input images' shape and texture.

In my system, global linear combinations of the input images are computed in embedding space: the input images' embedding coordinates are combined to produce a new embedding coordinate. When using non-linear dimensionality reduction and reprojection, this allows global linear combinations of images to be expressed as linear combinations of only a local neighborhood of high-dimensional geometry or texture vectors.

The process for creating the multimorph image of a linear combination of input images is as follows. The embedding coordinates for each input image are given by dimensionality reduction as discussed above. The user supplies the desired global linear combination of images: a vector of coefficients for combining the geometry - **a** - and a vector of coefficients for combining the texture - **b**. As in two-image morphs, often **a** = **b** in practice. These vectors are used as the weights in a linear combination of the input image embedding coordinates. Let $\vec{\alpha}_i$ be the geometry embedding coordinates for input image i, and let $\vec{\beta}_i$ be the texture embedding coordinates for input image i. Then the embedding coordinates for the multimorph image are

$$\vec{\alpha} = \sum_{i=1}^{n_v} a_i * \vec{\alpha}_i \tag{6.1}$$
$$\vec{\beta} = \sum_{i=1}^{n_v} b_i * \vec{\beta}_i.$$

These coordinates are then used to create the geometry vector and texture vector of the multimorph image as described in Chapter 5.

6.2 Creating Traits

For many applications of a multimorph system, the desired output images are not easily expressed as combinations of the input images. Often the presence or absence of several traits are desired. Traits, also known as attributes or characteristics, are properties possessed to different extents by the input images.

A trait is user-defined: the user assigns a value to each image, based on physical measurements or subjective properties. The larger the value, the higher the degree to which the image possesses the trait. I consider two kinds of traits: real-valued traits, in which each image is assigned any real number, and binary traits, in which each image is assigned one of two values.

Traits are based on physical qualities of the images (shape or texture). The ultimate goal of the trait calculations is to isolate and increase or decrease those qualities.

Binary traits are often distilled from real-valued traits. They are cases in which it is only important, or known, whether an image's value is greater or less than some threshold. The canonical example is gender of faces. The physical qualities which make a facial image appear masculine or feminine have been studied and can be used to grade a face's gender. For many purposes, however, it is only important to know which gender a face is, and the degree of femininity or masculinity of the face is unimportant.

A problem with user-defined traits is user error, or fuzzily-defined traits. It is sometimes difficult, when designing a trait, to consistently rank the images. Traits with fuzzy definitions, such as the prominence of scales on fish, are a prime example. The degree of difficulty this problem presents for trait calculations is dependent upon how many of the images are ranked inconsistently. If the percentage of incorrectly

ranked images is small, say ten percent or less, then the trait is not ambiguous, and the calculations can be expected to succeed. If the percentage is greater than fifty percent, then good results are unlikely. The trait calculations will change the images, but it is not likely that the physical qualities they effect will be those envisioned by the user, or that any physical qualities will be consistently changed at all.

Assuming a user-defined trait has been given, the user selects a starting embedding coordinate. This may be initialized as a linear combination of the input vectors, or it may result from previous trait calculations. They then select a trait to increase, and give a delta (a real number) for how much of the trait to add. For real-valued traits, the scale of delta is set by the assigned trait values. A positive delta increases the presence of the trait, a negative delta decreases it.

For binary traits, each value of the trait can be thought of as a region in space; the two regions cover all of space. A *neutral image* is an image near the border between the two regions. Let one of the values for the trait be the *positive* value, and the other be the *negative* value. The scale of delta is defined so that if the positive region is bounded, then a delta of 1 changes a neutral image so that is in the center of the positive region. If the positive region is unbounded, a delta of 1 changes a neutral image so that it is the same distance from the boundary as the farthest of the positive input images. The same scale holds for negative deltas and for the negative value of the trait.

The trait can be added independently to either the geometry or texture embedding coordinates, producing a new set of coordinates. The multimorph image corresponding to these images possesses more of the trait, assuming the trait calculation succeeded. The trait calculation may fail due to inconsistent data, or due to local extrema in the trait function. Traits can be added repeatedly, with the same trait or with a mixed set of traits, creating a path of images illustrating a transformation by the various traits.

In the next section, I discuss how to calculate the new embedding coordinate given by increasing the value of a trait at an initial embedding coordinate.

6.3 Trait Calculations

The mathematical problem presented by traits is the following. The values of a function are given at n points in a d-dimensional embedding space. The objective of a trait is to start at a given point and move in the direction of the gradient of the trait function until the function's value has increased or decreased by some given delta. A negative delta requires moving in the direction of the negative gradient. Note that traits do not handle the problem of local extrema in the trait function; if a local maximum is found when increasing the trait value, then the point of maximum is returned, and no increases of the trait function at that point will be successful.

The problem of traits thus reduces to finding the gradient of the trait function at a given point. Arbitrary increases in the trait function (excepting the presence of local maxima) are then created by the following numerical procedure. Let the requested amount to increase be $s * \delta$, $\delta > 0, s = \pm 1$. Let the numerical step size be $h > 0$.

Let the current point be $\mathbf{v}$. Let the (unknown) trait function be T. Then the trait algorithm is:

While $\delta > 0$
  Calculate $\mathbf{d} \leftarrow \nabla T|_{\mathbf{v}}$
  If $|\mathbf{d}| = 0$ then
    Exit while
  End If
  If $h|\mathbf{d}| > \delta$ then
    $\mathbf{v} \leftarrow \mathbf{v} + s\delta|\mathbf{d}|^{-1}\hat{\mathbf{d}}$
    $\delta \leftarrow 0$
  Else
    $\mathbf{v} \leftarrow \mathbf{v} + sh\hat{\mathbf{d}}$
    $\delta \leftarrow \delta - h|\mathbf{d}|$
  End If
End While
Return $\mathbf{v}$

I now cover some methods for calculating the gradient of the trait function from a set of sample points.

6.3.1 Linear Traits

A simplifying assumption that can be made when solving traits is to assume the trait function is linear. In this case the gradient is constant over the entire space. Let T be the trait function, and the d-dimensional vector $\mathbf{v}$ be its argument. The trait function can be written as

$$T(\mathbf{v}_i) = t_i \tag{6.2}$$
$$T(\mathbf{v}) = \mathbf{v_T} \cdot \mathbf{v} + t_0 + \delta(\mathbf{v}) \tag{6.3}$$
$$\equiv T_l(\mathbf{v}) + \delta(\mathbf{v}) \tag{6.4}$$

The linear function T_l has value t_0 at the origin, and gradient $\mathbf{v_T}$. $\delta(\mathbf{v})$ is the deviation of the trait function from linearity at $\mathbf{v}$. If the assumption of linearity is correct, then $\delta(\mathbf{v}) \equiv 0$ and $T \equiv T_l$. Otherwise, $\mathbf{v_T}$ and t_0 are defined such that the sum $\sum_{i=1}^{n} ||\delta(\mathbf{v}_i)||^2$ is minimized, where $\mathbf{v}_i$ is the ith sample point (i.e. T_l is the linear regression of the set of sample points).

Finding the vector $\mathbf{v}_T$ solves the problem, for linear traits. t_0 can be found by back-substitution, but is not necessary for the purposes of traits. If $\sum_{i=1}^{n_v} |\delta(\mathbf{v}_i)|^2$ is non-negligible when compared to the variance of t_i, then a linear function does not accurately describe the trait.

Of the following, Blanz and Vetter Traits, Linear Least Squares Traits, and Optimal Partitioning Hyperplane Traits make the assumption of linearity. When discussing these methods, I use the symbol $\mathbf{v}_m$ as the gradient vector the method discovers.

6.3.2 Blanz and Vetter Traits

As discussed in Chapter 3, Blanz and Vetter [2] discuss a method for solving linear traits. Their formula for $\mathbf{v}_m$ is

$$\mathbf{v}_m = \sum_{i=1}^{n} (t_i - \bar{t})(\mathbf{v}_i - \bar{\mathbf{v}}). \tag{6.5}$$

As Blanz and Vetter point out, this is the *variance-normalized* $\mathbf{v}_T$. To show this, replace the values for t_i and $\bar{t}$ with the expansions of $T(\mathbf{v}_i)$ and $T(\bar{\mathbf{v}})$ using equation 6.3:

$$\begin{aligned}
\mathbf{v}_m &= \sum_{i=1}^{n} (t_i - \bar{t}) * (\mathbf{v}_i - \bar{\mathbf{v}}) \\
&= \sum_{i=1}^{n} (\mathbf{v}_i \cdot \mathbf{v}_T + t_0 + \delta(\mathbf{v}_i) - (\bar{\mathbf{v}} \cdot \mathbf{v}_T + t_0 + \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{v}_i))) * (\mathbf{v}_i - \bar{\mathbf{v}}) \\
&\quad \text{Note} \quad \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{v}_i) = 0, \quad \text{since } \mathbf{v}_T \text{ is defined by least squares.} \\
&= \sum_{i=1}^{n} ((\mathbf{v}_i - \bar{\mathbf{v}}) \cdot \mathbf{v}_T + \delta(\mathbf{v}_i)) * (\mathbf{v}_i - \bar{\mathbf{v}}) \\
&= \sum_{i=1}^{n} (\mathbf{v}_i - \bar{\mathbf{v}})^T \mathbf{v}_T (\mathbf{v}_i - \bar{\mathbf{v}}) + \sum_{i=1}^{n} \delta(\mathbf{v}_i) * (\mathbf{v}_i - \bar{\mathbf{v}}) \\
&\quad \text{Let} \quad \boldsymbol{\Delta}_{\text{NL}} = \sum_{i=1}^{n} \delta(\mathbf{v}_i) * (\mathbf{v}_i - \bar{\mathbf{v}}) \qquad (6.6) \\
\mathbf{v}_m &= \sum_{i=1}^{n} (\mathbf{v}_i - \bar{\mathbf{v}})(\mathbf{v}_i - \bar{\mathbf{v}})^T \mathbf{v}_T + \boldsymbol{\Delta}_{\text{NL}} \\
&= \left(\sum_{i=1}^{n} (\mathbf{v}_i - \bar{\mathbf{v}})(\mathbf{v}_i - \bar{\mathbf{v}})^T \right) \mathbf{v}_T + \boldsymbol{\Delta}_{\text{NL}} \\
&= (n-1) * \mathrm{C}_v \mathbf{v}_T + \boldsymbol{\Delta}_{\text{NL}} \qquad (6.7)
\end{aligned}$$

Where C_v is the covariance matrix of the vectors $\{\mathbf{v}_i\}$. Which shows that (assuming the trait function is linear and so $\boldsymbol{\Delta}_{\text{NL}} = 0$) $\mathbf{v}_m$ is the variance-normalized $\mathbf{v}_T$ vector.

This definition of $\mathbf{v}_m$ is useful when it is added to the variance-normalized $\mathbf{v}_i$ vectors (as was done in the work of Blanz and Vetter). This does not hold in my work, since all trait calculations are done in the same embedding space. I therefore use the following method to calculate $\mathbf{v}_m$ by least squares.

6.3.3 Linear Least Squares Traits

I now derive a more straight-forward approach which solves for $\mathbf{v}_T$ exactly, using least squares in the case of non-zero $\delta(\mathbf{v})$. Evaluating equation (6.3) at each of the input

points gives a matrix equation:

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T & 1 \\ \mathbf{v}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{v}_n^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_T \\ t_0 \end{bmatrix} + \begin{bmatrix} \delta(\mathbf{v}_1) \\ \delta(\mathbf{v}_2) \\ \vdots \\ \delta(\mathbf{v}_n) \end{bmatrix} \tag{6.8}$$

Solving the regression problem for $\mathbf{v}_T$ is equivalent to assuming the unknowns $\delta(\mathbf{v}_i)$ are 0, since these are by definition the least squares residues.

$$\begin{bmatrix} \mathbf{v}_T \\ t_0 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T & 1 \\ \mathbf{v}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{v}_n^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \tag{6.9}$$

I use $\mathbf{v}_m = \mathbf{v}_T$ as the trait gradient vector when assuming a linear trait.

6.3.4 Non-Linear Least Squares Traits

Non-linear least squares functions can be found in the same manner as linear least squares. I express the trait function as the sum of a multi-variable polynomial of degree k, plus an additional error term showing deviation of the function from a polynomial of this degree. For $k = 2$, for example, the trait function is written as

$$T_2(\mathbf{v}) = t_0 + \mathbf{v}_T \cdot \mathbf{v} + \mathbf{v}^T Q_T \mathbf{v} + \delta(\mathbf{v}). \tag{6.10}$$

Here the matrix Q_T lists all coefficients of the quadratic terms in the function $T(\mathbf{v})$. This formulation can extend to arbitrary k. The compact notation above requires tensors to represent the polynomials of higher dimensions: a tensor of degree three to represent every cubic cross-term, degree four for quartics, and so on.

Regardless of the value for k, the least squares technique is the same. A matrix of equations in terms of the polynomial coefficients is derived from the input points. Each input point creates a row (equation) of the matrix. Each column (variable) of the matrix is the value of the polynomial represented by that column evaluated at the input point for each row. In brief, the terms t_0, $\mathbf{v}_T$, Q_T, and so on in equation 6.10 are left as variables, and the equation is written once for each $\mathbf{v}_i$.

Let the least squares polynomial of degree k be written as $T_k(\mathbf{v})$. The gradient of the trait function at a particular embedding point $\mathbf{v}$ is given by $\nabla T_k(\mathbf{v})$, for whatever value of k is chosen to represent the trait.

The method of non-linear least squares, however, has limited usefulness in morphing, because the number of sample points is usually small. For datasets of less than 100 images, fitting quadratic functions in a 10-dimensional embedding space is under-constrained. In most cases, it is sufficient to use only linear least squares.

6.3.5 Optimal Partitioning Hypersurface Traits

For binary traits, the trait function evaluation is treated as a classification rather than a real-valued function. In this case, it makes sense to apply *support vector machines* to find the *optimal partitioning hypersurface* of the set of points. I first define the *optimal partitioning hyperplane* for a set of points, and then generalize to the non-linear case.

Definition of the Optimal Partitioning Hyperplane

The optimal partitioning hyperplane is defined for a set of n points in d-dimensional space, over which a binary classification has been given. Let the input points be labeled $\mathbf{x}_i$, and let the classification y_i be given to each point, such that $y_i \in \{-1, 1\}$. The optimal partitioning hyperplane is a hyperplane in d-dimensional space; let it be notated by the normal vector $\mathbf{w}$ and the origin offset b such that all points $\mathbf{x}$ on the plane satisfy

$$\mathbf{w} \cdot \mathbf{x} = b. \tag{6.11}$$

The objective of the optimal partitioning hyperplane is to find $\mathbf{w}$ and b such that, for $i = 1$ to n, if $y_i = 1$ then $\mathbf{w} \cdot \mathbf{x}_i \geq b + 1$, and if $y_i = -1$ then $\mathbf{w} \cdot \mathbf{x} \leq b - 1$. This can be written as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \qquad i = 1, \ldots, n \tag{6.12}$$

The purpose of requiring positive $\mathbf{x}_i$ to be greater than $b + 1$, and negative $\mathbf{x}_i$ to be less than $b - 1$ is to set a scale for the otherwise homogenous equation. The distance from the hyperplane of the positive $\mathbf{x}_i$ closest to the plane, added to the distance from the hyperplane of the negative $\mathbf{x}_i$ closest to the plane, is equal to $\frac{2}{||\mathbf{w}||}$. This sum is know as the *margin* of the hyperplane. The data points lying on the planes $\mathbf{w} \cdot \mathbf{x}_i = b + 1$ and $\mathbf{w} \cdot \mathbf{x}_i = b - 1$ realize this margin, and they are known as the *support vectors* of the dataset.

Such a division does not always exist. In one dimension ($d = 1$), for example, the input $x_1 = 0, x_2 = 1, x_3 = 2$ and $y_1 = -1, y_2 = 1, y_3 = -1$, can not be partitioned by any 1-dimensional hyperplane $w * x = b$. When a linear division exists, the data is said to be *separable*; when a linear division does not exist, the data is said to be *unseparable*.

When the data is separable, the optimal partitioning hyperplane is defined by the $\mathbf{w}$ and b which satisfy equation 6.12 and which minimize $||\mathbf{w}||$. This definition finds the hyperplane with the biggest margin between the positive and negative data points.

When the data is unseparable, then the optimal partitioning hyperplane is characterized by a parameter C which gives the relative importance of misclassifications when compared to maximizing the margin between positive and negative data points. Equation 6.12 is reexpressed with the *slack variables* ϵ_i:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \epsilon_i \tag{6.13}$$

$$\epsilon_i \geq 0 \tag{6.14}$$

$\mathbf{w}$ and b are then defined to be the values which satisfy equation 6.13 and minimize $||\mathbf{w}|| + C\sum_{i=1}^{n} \epsilon_i$.

Definition of the Optimal Partitioning Hypersurface

The definition for the optimal partitioning hypersurface is similar to the definition for the optimal partitioning hyperplane. Instead of a hyperplane separating the spaces, it is now an arbitrary surface. For a quadratic polynomial hypersurface, equations 6.12 and 6.13 become

$$y_i(\mathbf{x}_i^T \mathrm{Q} \mathbf{x}_i + \mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \qquad i = 1, \ldots, n \tag{6.15}$$

$$\begin{aligned} y_i(\mathbf{x}_i^T \mathrm{Q} \mathbf{x}_i + \mathbf{w} \cdot \mathbf{x}_i - b) &\geq 1 - \epsilon_i & (6.16) \\ \epsilon_i &\geq 0 & (6.17) \end{aligned}$$

As in non-linear least squares traits, this can be extended to higher polynomials of $\mathbf{x}_i$. It is also common to use non-linear functions other than polynomials. A sum of the first k radial basis functions may be used, for example, instead of the first k x^k polynomial basis functions.

Support Vector Machines

Vladimir Vapnik [17] defined the problem of optimal partitioning hypersurfaces and introduced support vector machines to solve it in the late 1970s. Fundamentally, the optimal partitioning hypersurface problem requires solving a convex program (i.e. a linear program, save that the constraint function is in general non-linear, but convex).

Support vector machines solve the convex program by finding the support vectors of a given dataset. Finding the support vectors from the input data points is known as the *learning stage* of support vector machines. New points are then classified in the *evaluation stage*, by taking their inner products with a linear combination of the support vectors and comparing the result to the offset b. Non-linear hypersurfaces are found by using a non-linear *kernel*. The kernel used defines the inner-products: the inner product with a non-linear kernel represents evaluation of a non-linear function.

For a more throughout explanation of the optimal partitioning hypersurface problem and support vector machines, see *A Tutorial on Support Vector Machines for Pattern Recognition* by Christopher Burges [3].

Gradient Vectors from Optimal Partitioning Hypersurfaces

Support vector machines find a hyperplane separating the data points. The gradient vector of the trait function is then given by the normal to this hyperplane. When finding the optimal partitioning hyper*surface*, support vector machines represent the separation as a level set of a linear combination of basis functions (i.e. the polynomial basis functions x^k or the radial basis functions). The gradient in this case is found by differentiation of the basis functions.

6.4 Conclusion

Using trait calculations (either least squares or optimal partitioning hypersurfaces), embedding coordinates can be specified by the degree to which they possess user-defined traits. Coordinates are initialized by some linear combination of the input images. The embedding coordinates thus defined directly specify output images. With these two tools, it is possible to create a wide variety of images that are constrained to resemble the input images. This user interface is the end-result of my multidimensional morph system.

In the next chapter, I give the details of my implementation, and a discussion of my results.

Chapter 7

Implementation and Results

7.1 Implementation

I implemented the multimorph algorithm in a multistage system. The first stage is user input: two-image morphs, options, and traits. The second stage is morph registration. The third stage is setting the output image orientation and calculation of texture vectors. The fourth stage is dimensionality reduction. And the fifth stage is the input of embedding coordinates by input image combinations and traits. In this section, I discuss my implementation of each of these stages. I begin by my implementation of triangle mesh warps, which I used to create the two-image morphs required by the multimorph algorithm.

7.1.1 Triangle Mesh Warps

I used triangle mesh warps in my system because they are computationally fast and are easy to implement.

A set of corresponding point and line features across two images defines a triangle mesh morph. A corresponding point feature is stored as an array of four real values: (x_1, y_1) in the first image and (x_2, y_2) in the other image. A corresponding line feature is stored as an array of two indices: (i_1, i_2). These values are the indices of the two corresponding point features that define the line feature.

In-between images are created using the method of standard two-image morphs discussed in Section 3.2.2. In-between point features are the average (weighted by α) of the point features in the two source images. In-between line features are defined by connecting their endpoint in-between point features (i.e. line feature indices do not change).

Two options are left to user specification when creating the in-between point features. The first option is prewarping. If prewarping is selected, then the points are prewarped to remove the average transformation as discussed in Section 3.2.2. I implemented the average transformation using least squares. The second option is elimination of rotation. If prewarping is selected, and elimination of rotation is not selected, then the rotation is gradually changed depending on α: $\gamma \leftarrow \alpha$. If elimination of rotation is selected, then the rotation of the in-between image is eliminated by

setting its rotation to the average of both images ($\gamma \leftarrow .5$). When using prewarping, I allow rotation, translation, and scale in the transformations, but I do not allow shear or a change in aspect ratio.

To warp points from a source image to an in-between image, I calculate a Delaunay triangulation of the feature points and lines in the source image. (Note that this will be a different triangulation for each of the source images). A line feature is interpreted as a constrained edge in the Delaunay triangulation. Source points are then warped to their in-between positions by finding the source triangle in which they lie. The barycentric coordinates in the source triangle are calculated from a straight-forward least squares problem (using the pseudo inverse when the source triangle has zero area.) These coordinates are then used as weights in an average of the corresponding three points in the in-between image to create the destination point for the source point.

One caveat to using this method is that points outside the convex hull of the source points have no clear destination point. To solve this problem, I find the three hull points nearest to an outlying point, and use the barycentric coordinates of their triangle. This is equivalent to letting the edge triangles determine the warping of their adjacent space. Using this method does not always yield good results, and so I override it wherever possible in two-image morphs by setting the corners of the image in correspondence.

To create an in-between image, I use reverse mapping of the in-between points. The Delaunay triangulation of these points give a source image point for each destination pixel (pixels are treated as points). The color of a destination pixel is set by bilinear interpolation and cross-fading of the four surrounding source image pixels.

To implement chaining of triangle mesh morphs, the points in the common image in one of the morphs are warped to the unique image of the other morph using the triangle mesh warp for that other morph. (Line connections are preserved.) Note that this is not symmetric, so that chaining edge morph A-B with edge morph B-C is not necessarily the same as C-B chained with B-A. In future work, this should be changed to calculate both directions and blend the two morphs. For most well-crafted morphs (i.e. not pathological input) the triangulations of the two sets of points for an image morph are similar, and so lack of symmetries such as this are not so important. For more pathological data, it is not trivial to determine what the "correct" chaining of the morphs should be.

I implement merging of triangle morphs by creating a grid of points in both images. The grid for each image is warped to the other image once for each morph being merged (so there are $2n$ warped grids if there are n morphs being merged.) The point positions of the warped grids are then averaged, using the weights for each morph in the merger as the weights of the point averages. This creates two grids of corresponding points, one for each image, and these two sets are combined to create the final set of correspondence points. Corresponding lines are not preserved when merging morphs. As with chaining, the performance of this algorithm on pathological data could be improved, but for well-formed data it performs acceptably.

7.1.2 Two-image Morph, Options, and Traits Input

Two-image Morph Input

When using my system, the user first sets the list of input images used to create the multimorph images. This list is given as the vertices of a graph. Each vertex has an associated image, and so I call these elements *imagevertices*. This is illustrated in Figure 7-1.

The second step is creation of two-image morphs. In this implementation, I allow only two-image morphs based on point features and line features. This restriction accommodates most types of two-image morph algorithms, and is a simple interface to implement.

To create a morph, the user selects two imagevertices to connect with a direct edge (the imagevertices must not already be connected). This creates a *morphedge* between the two vertices. The user then specifies corresponding points and lines in the two images assigned to the selected imagevertices. The images are displayed side by side; adding a point in one image creates a new point at that mouse position, as well as its corresponding point in the other image. The position of the corresponding point is set intelligently: the offset from the last point added is set equal in both images. Afterwards, position corrections are made by dragging the points. The color of added points cycles through a set of border and fill colors to show which points are in correspondence. Corresponding lines are added by selecting two previously placed point features to connect. (Note adding a corresponding line does not affect the list of the point features.) This is illustrated in Figure 7-2.

When creating morphs in my system, I allow the creation of standard two-image morph movies in each morphedge. The user selects the number of frames (in-between images) to be computed, and the starting and ending values for α (feature position in-between value) and β (cross-fade in-between value). A linear progression of the start and end points gives a value of α and β for each frame. The two-image morph output images corresponding to these values are concatenated into a movie. The two-image morph algorithm used to create the images is a black box, save that it must be point and line-feature based.

Options Input

Some features of the multimorph algorithm have clear but hard to define effects on the results. I leave these options to user control.

When making two-image morph movies, there are two choices for how the movie should be made: whether prewarping should be used, and if so, whether rotations should be eliminated. These options are discussed in my implementation of triangle mesh warps, above.

Morph registration has several options. The first group of options is for the evaluation of the input morphedges. My implementation of morph registration uses the best k weighted paths. The weight of each input morphedge is based on derivative discontinuities, foldover, and holes. The relative weight given to each of these three values, as well as a base weight for each edge, are set by the user. These weights

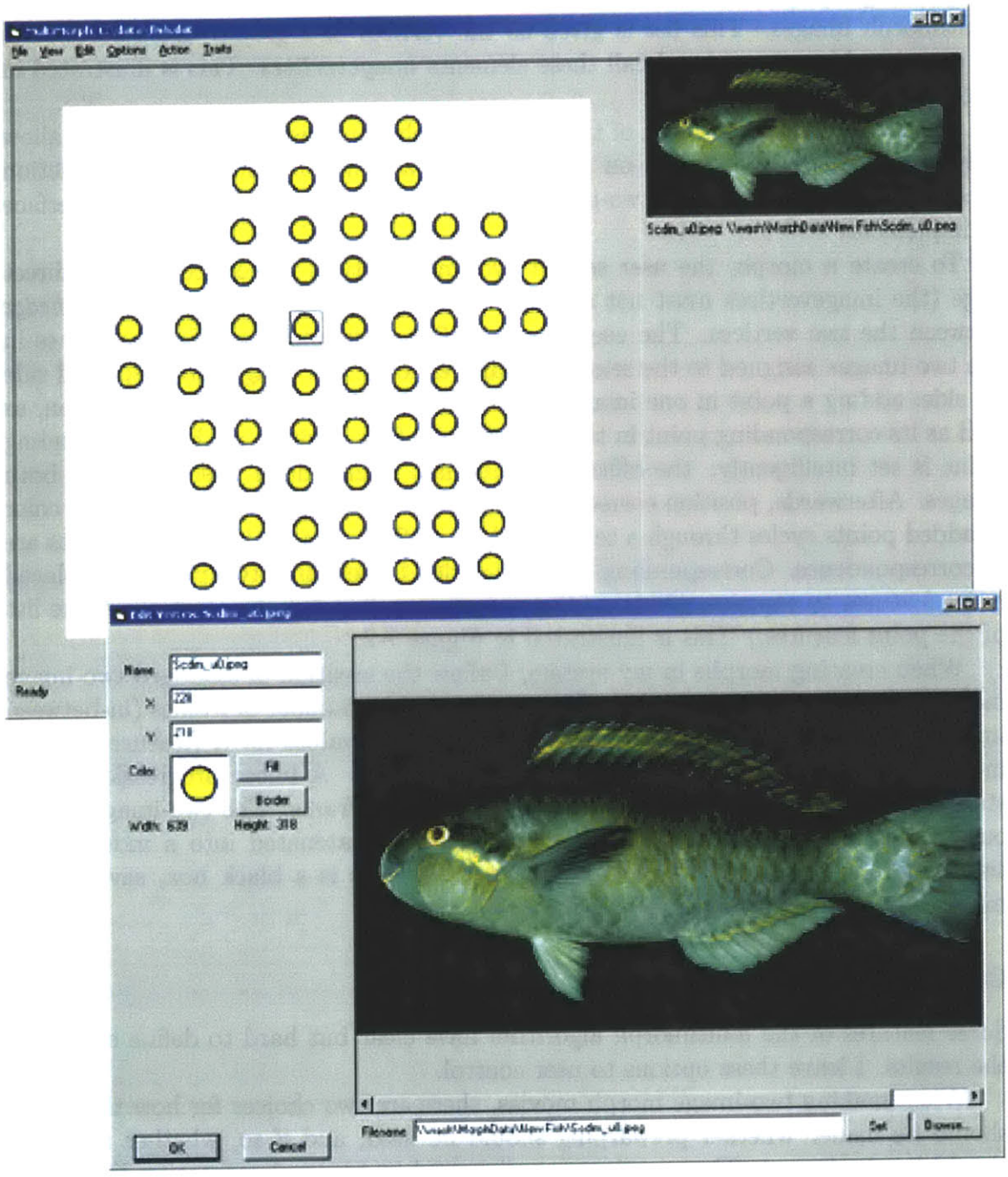

Figure 7-1: Creating imagevertices.

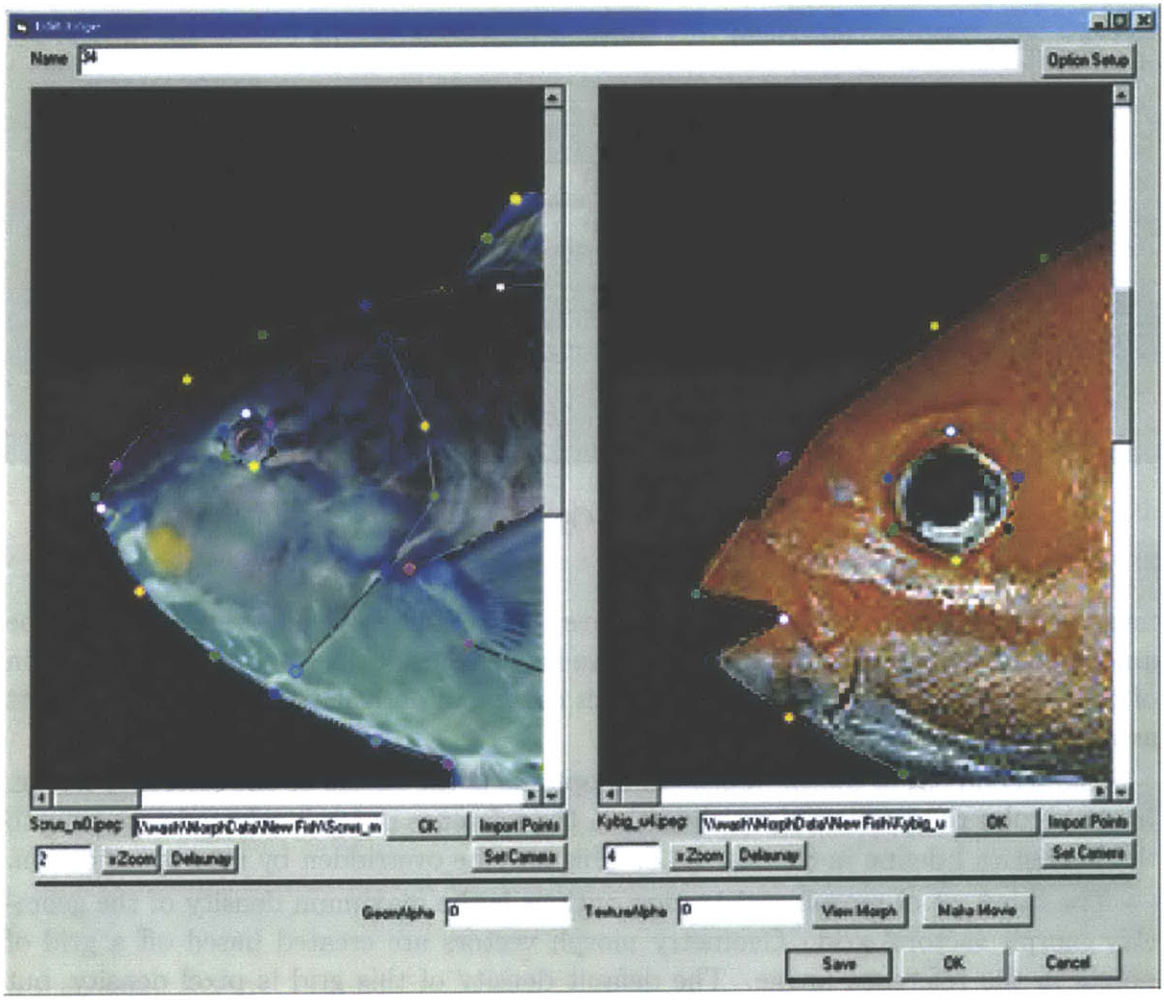

Figure 7-2: Creating morphedges.

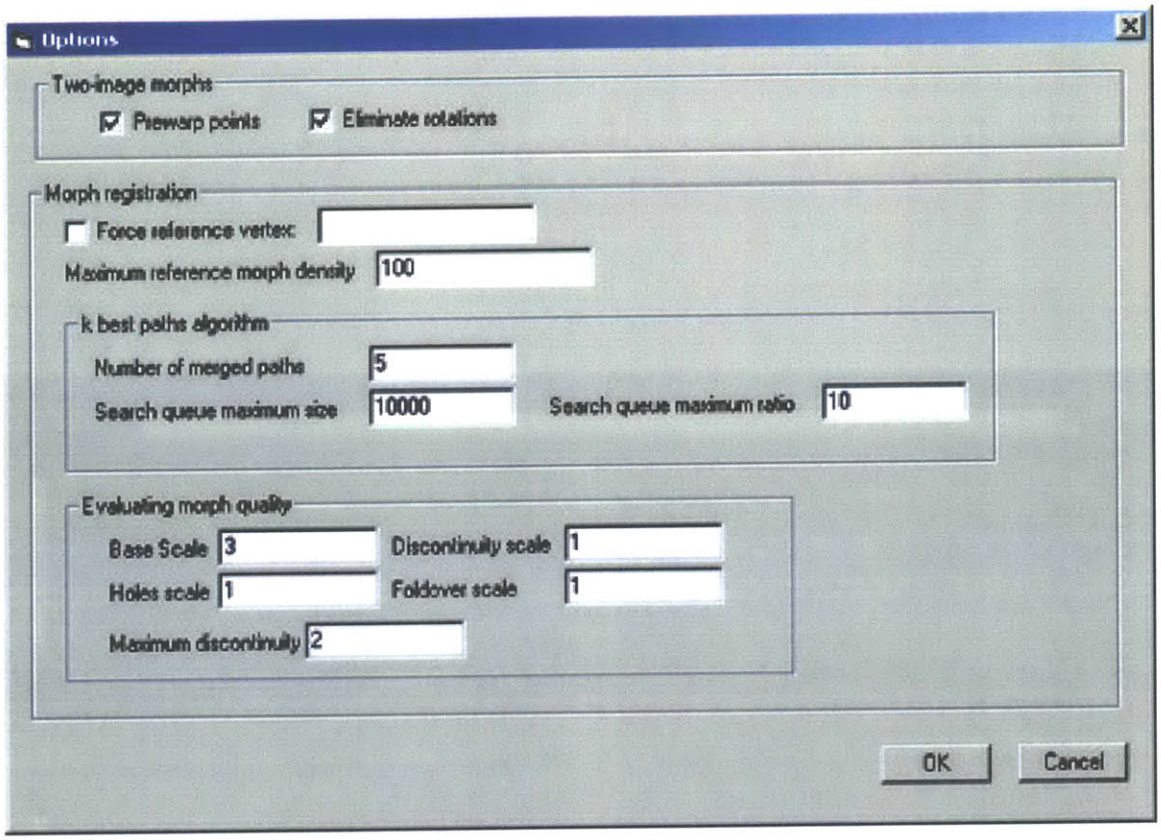

Figure 7-3: Option setup.

must be non-negative, and are homogenous (multiplying all four values by the same amount does not change the input). There is a further option for the approximation of derivative discontinuities: the minimum change in derivative per unit that counts as a discontinuity.

The second set of morph registration options is the choice of the reference image. In the regular progression of my algorithm, the reference image is selected according to the heuristics I define in Section 4.4.3. This may be overridden by user specification.

The third set of morph registration options is the maximum density of the geometry morph vectors' grid. Geometry morph vectors are created based off a grid of points in the reference image. The default density of this grid is pixel density, but the user may select a maximum density for speed of computation. The units of the entered value are the maximum number of x or y grid lines.

The final set of morph registration options are the parameters of the best k paths algorithms. As described in Section 4.4.3, the best k paths algorithm has several parameters which determine the tradeoff between being thorough and being fast. The user selects the maximum number of merged paths (k), the threshold for maximum size of the search queue, and the maximum ratio of the weight of a path weight in the search queue to the weight of the longest shortest weighted path from reference image to any other image.

An illustration of the dialog box to set these options is given in Figure 7-3.

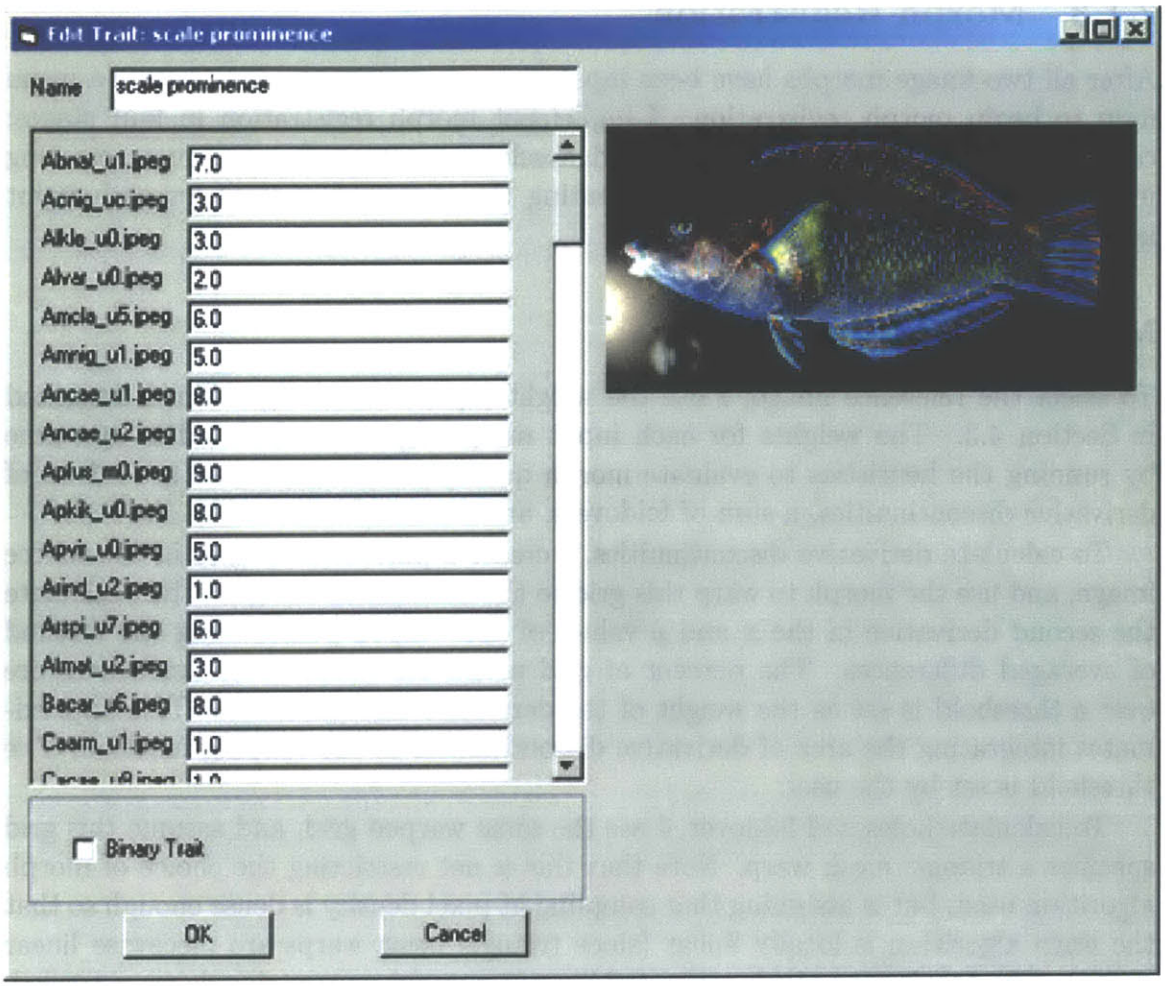

Figure 7-4: Defining traits.

Options for dimensionality reduction, output orientation, and texture morph vectors are set by the user in later stages of my process. I discuss these options below in the discussion of those stages.

Traits Input

The user inputs traits by setting a value for each imagevertex. This is done in a dialog for each trait: the name of each imagevertex is listed next to an input field for that imagevertex's value. This dialog is illustrated in Figure 7-4. Clicking on a imagevertex's name displays the image. In my experience, entering a trait requires 5-10 seconds per image. The user also selects whether the trait is binary; if so, then it is only important whether each vertex's value is greater or less than the average of the minimum and maximum values assigned to imagevertices in the trait.

7.1.3 Morph Registration

After all two-image morphs have been input into my system, the user selects a menu item to begin morph registration. I implement morph registration in four stages: calculation of the weights of each morphedge, selection of the reference image, creating morphs to the reference image, and creating the geometry vectors for each input image.

Morph Weights

To select the reference image, I use the weighted-path based algorithm I described in Section 4.3. The weights for each input morphedge are calculated at run-time by running the heuristics to evaluate morph quality. These heuristics are a sum of derivative discontinuities, a sum of foldovers, and a sum of holes.

To calculate derivative discontinuities, I create a pixel-density grid in the source image, and use the morph to warp this grid to the destination image. I then calculate the second derivative in the x and y values of the warped points using the method of averaged differences. The percent of grid points for which these derivatives are over a threshold is set as the weight of the derivative discontinuities. This approximates integrating the area of derivative discontinuities in the warping function. The threshold is set by the user.

To calculate holes and foldover, I use the same warped grid, and assume this grid specifies a triangle mesh warp. Note that this is not restricting the choice of morph algorithm used, but is assuming that sampling at pixel density is dense enough so that the warp algorithm is locally linear (since triangle mesh warps are piecewise linear warps). The destination pixels contacted by each warped source triangle are recorded. Destination pixels not touched by any source triangle count as holes, and each time a source triangle touches a destination pixel which has already been touched counts as foldover. The number of holes and foldovers are divided by the number of destination pixels to create the weights for holes and foldovers, respectively.

The final weight of the morph is the weighted sum of foldover, holes, derivative discontinuities, and a base weight. The sum is weighted by values provided by the user. These user-input weights are interpreted as relative weights, so that identical values for all four weights should result in the morph weight being equally dependent on all four values. Since the measure for holes, foldover, and derivative discontinuities are not in general the same, each of the sub weights must be rescaled to have the same variance. I therefore divide all of the derivative discontinuity morph weights by the mean derivative discontinuity weight, and so on, before adding the four values (base weight, derivative discontinuity weight, holes weight, and foldover weight). Since the forward warp of a morph is not guaranteed to be the inverse of the reverse warp, the warp weights are calculated for both of these warps and then averaged to create the morph weight.

Valid weights range from 0 to positive infinity (or the maximum extent of a double). 0 is a perfect morph (e.g. the identity morph) and higher weights indicate degrading quality. When chaining morphs, the chained morph weight is the square

root of the sum of the squares of the morphs along the path (i.e. chained weight grows like a standard deviation.) When merging morphs, the weight each point is given in the average of point positions is the inverse weight of the point's morph, divided by the sum of the inverse weights for all of the merged morphs. (Since morph weights may be zero, I put a restriction on the inverse weight, so that it may be no greater than 10^{-3}.) When calculating the weight of merged morphs, the weights are averaged, weighted by these same inverse weights. This is equivalent to adding resistances: the inverse of the merged weight is the sum of the inverses of the edge weights.

Selecting the Reference Image

After a weight is assigned to each morphedge, I use an all-pairs shortest path algorithm ($O(n_v^3 \log n_v)$ running time) to find the distance of the shortest weighted path from each image to every other imagevertex. I then take the reference image to be the imagevertex with the smallest mean distance to every imagevertex.

Creation of the Reference Morphs

After selecting the reference image, I run the best k paths algorithm I described in Section 4.4.3, using the calculated edge weights. I merge the best k paths from each imagevertex to the reference image to create the reference morph for that vertex; the value for k is set by the user. During the execution of the all-paths search, the queue of search paths is pruned, for both a maximum size and for a maximum ratio of path weight to the maximum shortest path weight. The thresholds for these values are set by the user. The paths are chained and merged by the currently selected morph algorithm (i.e. triangle mesh morphs in my implementation.)

Creation of the Morph Geometry Vectors

After the reference morphs have been created, they are converted into morph geometry vectors. I lay out a grid in the reference image. The grid has the density of a node at every pixel, but is shifted by a half pixel, such that no grid node is on the border of the image. If the grid density is too high (so that the number of divisions is greater than the maximum which was set by the user) then the grid is made less dense such that it has the maximum number of divisions, still equidistant, and still spanning the entire image. x and y thresholds are handled separately. Each grid node is converted into a point, and this list of points is the prototype geometry vector.

I warp the prototype geometry vector to each input image using the reference morphs, and the resulting list of points for each image is the geometry morph vector for that image. Note that this forces the morph algorithms used when creating multimorph images to be strictly based on point features. (Eliminating, for example, Beier-Neely morphs.) The geometry vectors are usually large: When calculating results, it is common to use 100x100 or denser for the maximum grid density. The geometry vectors are therefore stored on disk instead of in memory.

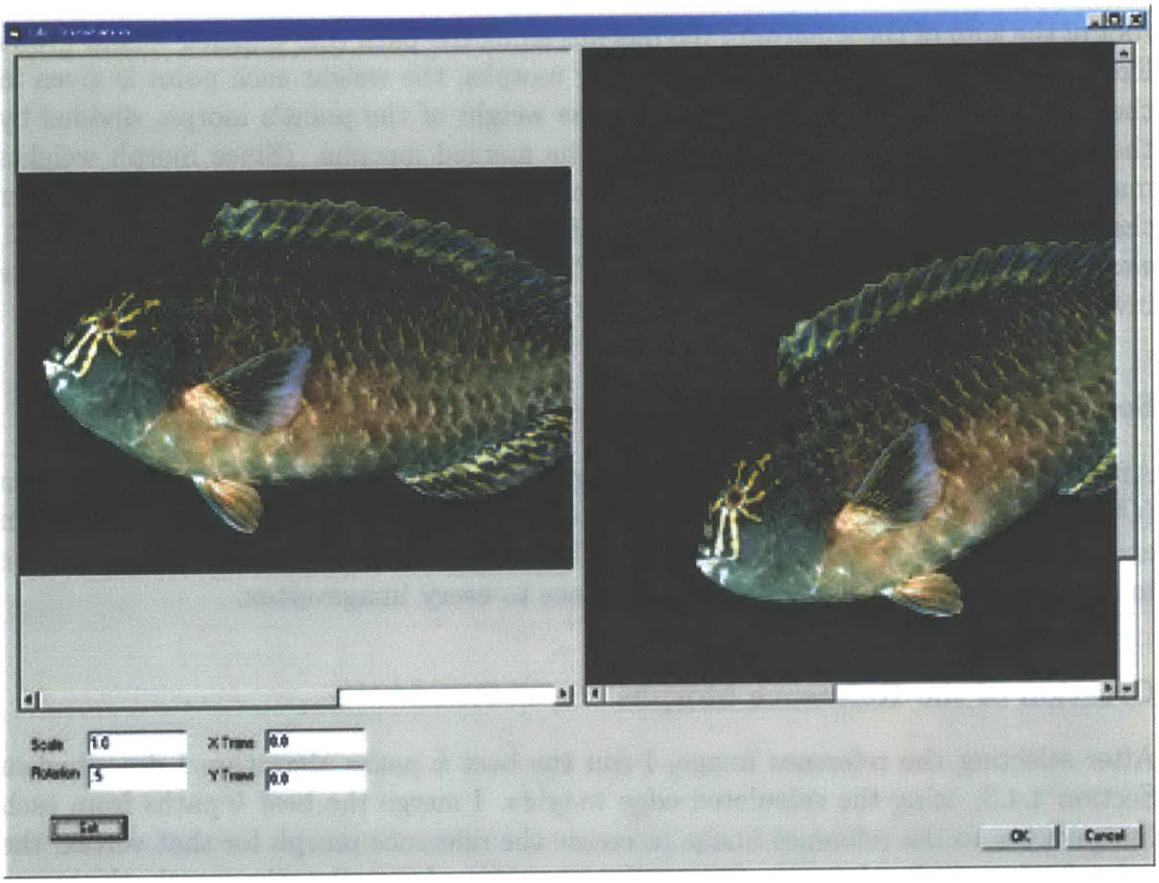

Figure 7-5: Specifying output orientation.

7.1.4 Output Orientation and Texture Vectors

Output Orientation: Warped Geometry Vectors

After morph registration has been completed, geometry morph vectors specify the shape of each image, by describing the warp location of a grid of points in the reference image. To create new shapes that are (linear) combinations of all or some of the input image shapes, it is first necessary to eliminate average rotation from each input image. This was discussed in Section 5.1.2. The user specifies an orientation for the output image by giving a rotation, translation, and scale of the reference image. Selecting a menu item then applies this transform to the geometry morph vector for the reference image (which is the original grid of points), producing the reference warped geometry vector. Then, for each input image, the general translation, rotation, and scale to best align that image's morph vector to the output orientation vector is calculated, as described for two-image morphs. The morph vector is transformed by these values to create the warped geometry vector for that input image. The warped geometry vectors can be directly combined in linear combinations, and will correctly eliminate overall rotations from one image to another. Note that the warped geometry vectors are not used to specify the warp from an input image; the original untransformed geometry morph vectors are still used for that purpose. The warped geometry vectors

are only used to specify new geometry morph vectors.

Texture Vectors

Once the output orientation has been selected, texture vectors can be calculated. Selecting a menu item starts the calculation of the texture vectors. As described in Section 5.2, texture vectors are the pixels of each input image after being warped to the texture reference shape (the same as the reference warped geometry vector). Together with the texture reference shape, each geometry morph vector specifies a triangle mesh warp, and these warps are used to create a warped image for each input image.

Since the warped images are warped to the same shape, they are all in alignment and of the same size. As I discussed in Section 5.2.2, the image size of a multimorph image (of which the texture vectors are a special case) is determined by the bounding box of the geometry morph vector for that image. The pixels of the warped input images are treated as vectors of dimensionality (image width)·(image height) and are saved to files, creating the texture vectors.

7.1.5 Dimensionality Reduction

The user sets dimensionality reduction options and then selects a menu option to calculate the embedding. This is done independently for geometry and texture vectors. The reduction of the geometry vectors uses the warped geometry vectors as the high-dimensional vectors.

The dimensionality reduction options include most importantly the type of reduction to use. The options are the same as I mentioned in Chapter 5: linear combinations of input vectors, linear combinations of singular vectors, isomap, and LLE. When setting options for texture vectors, the user also selects whether texture vectors should be used to directly create multimorph images, or whether texture vectors should be converted back to cross-fade coefficients. If the texture vectors are converted to cross-fade coefficients, then the output of dimensionality reprojection is barycentric coordinates rather than high-dimensional vectors, as I described in Section 5.2.

Linear combinations of input vectors or of singular vectors are common techniques with no significant tuned parameters. Isomap and LLE, however, both use a parameter k for the number of points in a local neighborhood. By default, I use 12 for the number of neighbors. This parameter may be overridden by the user.

After running the dimensionality reduction, the user chooses the dimensionality of the embedding space to use. For linear combinations of input vectors, only a single value - the number of input vectors - is allowed. For linear combinations of singular vectors, isomap, and LLE, the maximum allowed value is one less than the number of input images. (In theory, this maximum could be lower, if some of the morph vectors are linearly dependent. In practice, however, since the dimensionality of the morph vectors is much higher than the number of morph vectors, linearly dependent morph vectors never occur.)

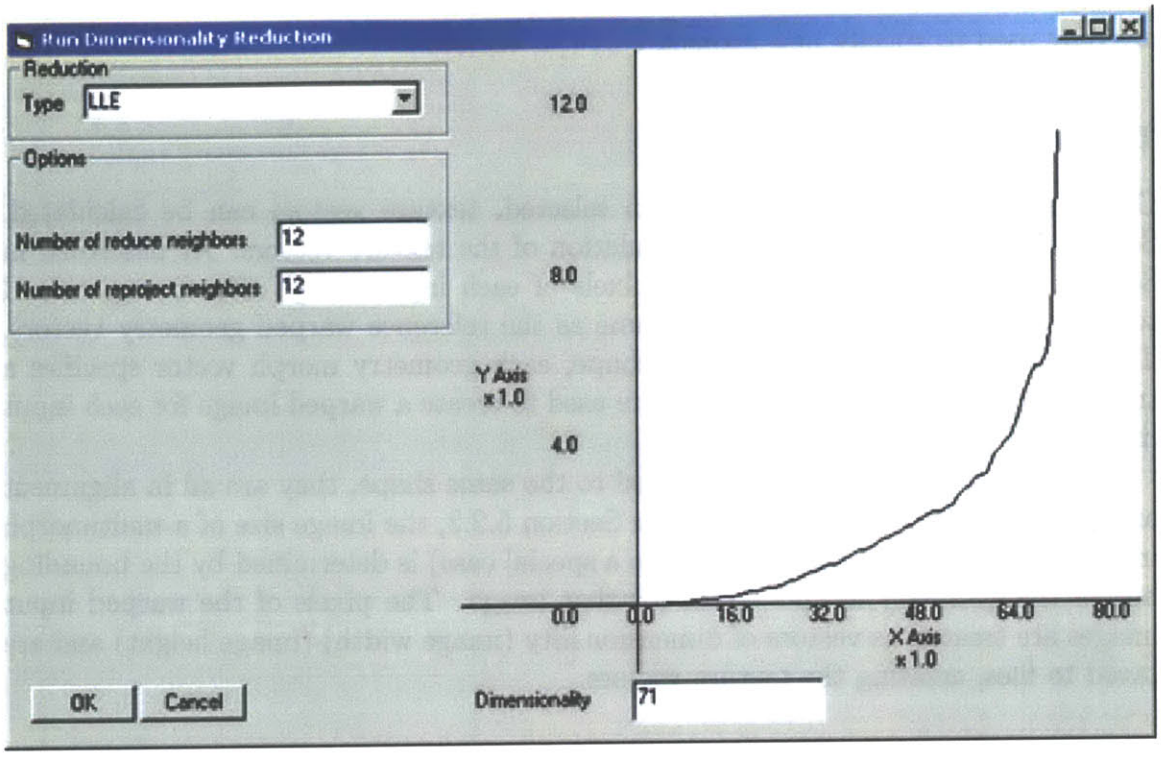

Figure 7-6: Setting dimensionality reduction options for geometry morph vectors. LLE is selected; the graph of LLE residues is displayed at the right.

Each of these three techniques plots a measure of how well each dimensionality - from 1 to the maximum - represents all of the data in the original high-dimensional space. For singular vectors and isomap, this measure is the size of the singular value for each dimension (dimensions with negligible values may be safely neglected.) For LLE, the measure is more complicated. LLE calculates a weight matrix, W, by least squares, such that $WX \sim X$, where X is the data matrix. Then, the near-one Y eigenvectors of W are calculated, such that $WY \sim Y$. Each column of Y is an embedding dimension. The LLE measure of each dimension is the L2-distance of the dimension with the matrix that results when it is multiplied by the weight matrix (these should be equal). These values increase with each dimension; dimensions with extremely large values can be safely neglected. After examining the plot for the dimensionality reduction chosen, the user selects the embedding dimensionality to use.

A final option, when using isomap or LLE, is set by the user in the dimensionality reduction options. Dimensionality reprojection using these techniques is done by calculating the least squares barycentric coordinates of a new embedding point in terms of its k nearest neighbors, and using these coordinates as weights on the high-dimensional vectors of those neighbors to construct the new high-dimensional point. The value of k defaults to 12, the same as the default value for the number of neighbors in isomap or LLE. This value may be overridden by the user.

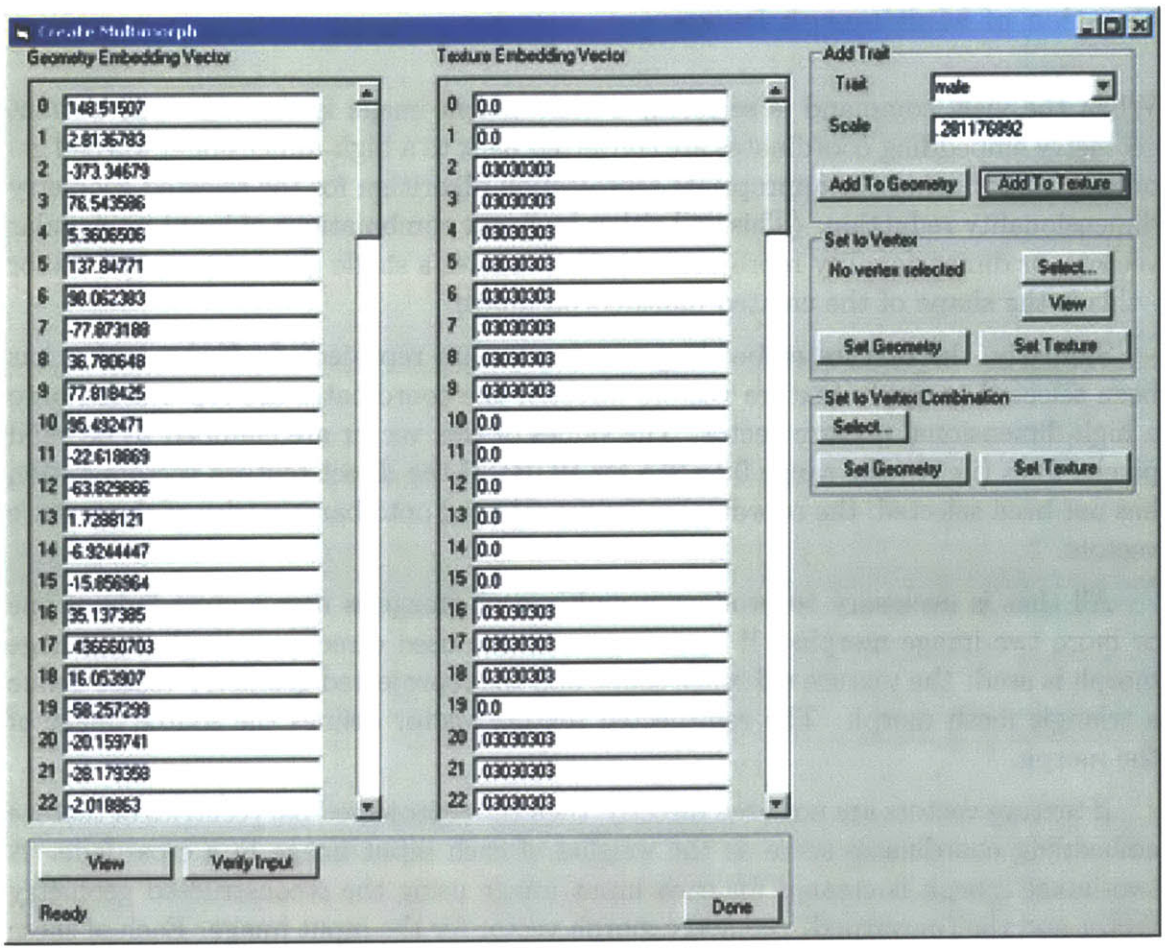

Figure 7-7: Specifying a multimorph image.

7.1.6 Specification and Creation of Multimorph Images

User Input

The user selects a menu item to launch a multimorph image creation dialog. The dialog displays a list of fields for the selected embedding space of both the geometry and texture vectors. These values can be manually adjusted, set to the embedding coordinates for a selected input image, or set to the embedding coordinates for a linear combination of input images. Clicking the view command creates a multimorph image for the current embedding coordinates.

A more powerful method of specifying user input, however, is to use traits. The user selects a trait to apply from the list of traits they created earlier. They then select a scale - the distance they wish to move along the direction of the trait. And finally, the user clicks a button to add the trait to either the current geometry embedding coordinates, or the current texture coordinates. Clicking this button moves the current geometry or texture coordinates, respectively.

Creation of Multimorph Image

When the view command is selected, a multimorph image is created. The current geometry embedding coordinates are converted back to a high-dimensional warped geometry vector using the appropriate reprojection algorithm for the selected geometry dimensionality reduction. (This will either be linear combinations of input or singular vectors, or dimensionality reprojection.) This creates a single geometry morph vector - this is the shape of the created multimorph image.

Similarly, the texture embedding coordinates are reprojected. If the option has been selected to apply texture vectors directly, the coordinates are reprojected onto a high-dimensional texture vector. The values of this vector are clamped to be valid pixel values (i.e. in the range 0 to 255 for RGB). If the direct texture vectors option has not been selected, the coordinates are reprojected onto barycentrics of the texture vectors.

All that is necessary to produce a multimorph image is now a cross-fade of one or more two-image morphs. If texture vectors are used directly, a single two-image morph is used: the texture reference shape and the reprojected geometry vector define a triangle mesh morph. The reprojected texture vector defines the source image of the morph.

If texture vectors are not used directly, then the reprojected barycentrics of texture embedding coordinates serve as the weights of each input image in a cross-fade. A two-image morph is created for each input image using the reconstructed geometry vector and the (unwarped) geometry morph vector for the input image. Each of these morphs is used to create a warped image, with $\alpha = 1$ (the geometry is the geometry of the reconstructed geometry vector) and $\beta = 0$ (the cross-fade weight of the input image is 1). These warped images are then cross-faded together according to the cross-fade coefficients of the reconstructed texture vectors.

Traits

Due to time constraints, I did not implement support vector machines for binary traits. All traits, real-valued or binary, are computed with non-linear least squares. I determine the degree of the polynomial that is fit by setting the number of polynomial parameters to less than or equal to the number of equations (points). If a quadratic polynomial has more parameters than the number of points, then I use a linear fit. For a dataset with 100 images, for example, and an embedding space of 8 parameters (i.e. LLE or isomap with a reduced embedding space), I use a quadratic polynomial, since a quadratic polynomial has $1+8+64 = 73$ parameters, and a cubic polynomial has $1+8+64+512 = 585$ parameters. For the stepsize used in successive evaluations of the trait gradient, I use $h = \frac{min\_dist}{100}$, where min_dist is the minimum distance between the embedding coordinates of any two input images.

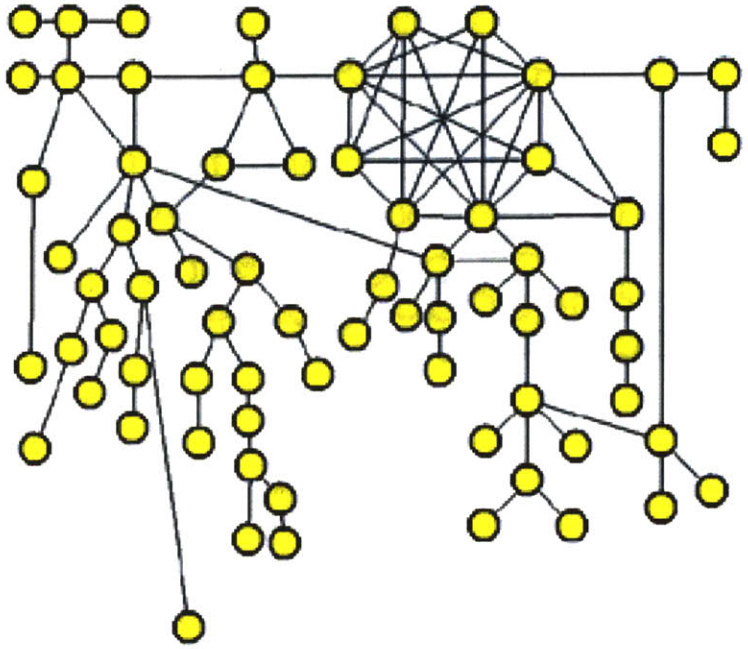

Figure 7-8: Layout of imagevertices and morphedges in the Faces dataset. The graph is cyclical.

7.2 Results

7.2.1 Datasets

I tested my multimorph system on three datasets. Each dataset consists of 50-100 images. The input system I described in Section 7.1.2 was used to create each dataset.

Faces Dataset

The Faces dataset is composed of 72 images of human faces. The images were collected from on-line pictures of celebrities and others. The spread of face types includes both genders and various ethnicities. The images are in varying aspect ratios and resolutions. The Faces dataset has redundancies: the input graph is cyclical. I use this dataset when testing the performance of morph registration.

Dogs Dataset

The Dogs dataset is composed of 100 images of dogs. The images were obtained from an on-line directory hosted by the American Kennel Club [4]. A large variety of different breeds and sizes of dogs were used. The dogs are in similar poses, but have significantly varying three-dimensional geometry. The images are in similar aspect ratios and resolutions. The input graph of the Dogs dataset is acyclic.

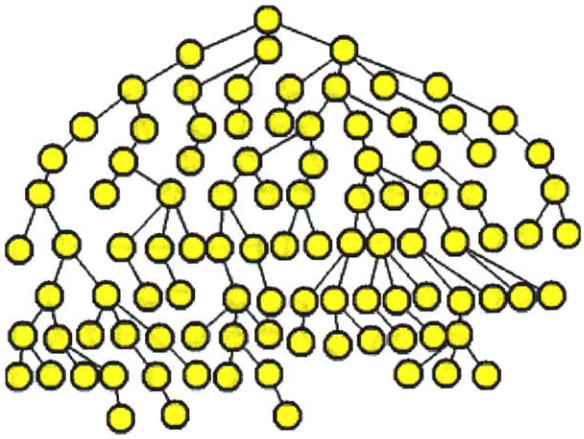

Figure 7-9: Layout of imagevertices and morphedges in the Dogs dataset. The graph is acyclic.

Fish Dataset

The Fish dataset is composed of 65 images of fish. The images were obtained from an on-line directory hosted by `fishbase.org` [5]. A large variety of fish species were used. The fish are all in the same pose, but the images are in varying aspect ratios and resolutions. The input graph of the Fish dataset is acyclic.

7.2.2 Overall Results

In the following, I show general results for the Faces, Dogs, and Fish datasets. In each set, I show first the *average image*: the linear combination of input images with the weight of each image set to $1/n_v$. (The linear combination is done in embedding space, as described in Section 6.1.2.) I then show a sequence of images produced by increasing and decreasing two traits.

The results for the Faces dataset are given in figures 7-11 and 7-12. The results for the Dogs dataset are given in figures 7-13 and 7-14. And the results for the Fish dataset are given in figures in 7-15 and 7-16.

Each of these results were created using isomap with 20 neighbors for geometry dimensionality reduction, and linear combinations of basis vectors for texture dimensionality reduction. Texture vectors were used directly to specify multimorph images (cross-fade coefficients were not used.) In the Faces dataset, the morph evaluation parameters were set to chain length weight = 3, derivative discontinuity weight = 1, foldover weight = 1, and holes weight = 1. The number of merged paths was $k = 5$, and the search-path thresholds were a maximum size of 10,000, and a maximum ratio of 10. When altering traits, polynomial regression was used to calculate the gradient, and the embedding spaces had large enough dimensionality that only

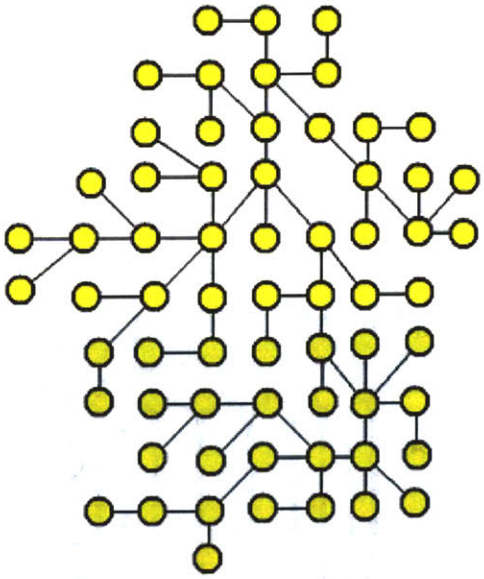

Figure 7-10: Layout of imagevertices and morphedges in the Fish dataset. The graph is acyclic.

linear polynomials were needed.

Results in the Faces Dataset

The Faces dataset has the best alignment results, but only within the bounds of the face. The morphedges in the input graph did not specify correspondence in the region outside the face, and so in this area the images are unaligned, producing gray when many images are cross-faded, and producing obvious errors and ghosting effects when only a few images are cross-faded. The performance of this dataset points out the need for a background identifier, which I discuss in Chapter 8

Within the face, the algorithm has good results. The hair-line is misaligned, both on the forehead and to the left and right of the face, but this is a problem of conflicting three-dimensional geometry that was present in even the input morphedges. Otherwise, the features of the face are well aligned, and the average image looks authentic. It does not strongly resemble any of the input images. A surprising result is that the average face appears female; men and women were represented in an exact 1:1 ratio in the input, so this result indicates that the half-way point when taking the linear average of male and female feature points appears female.

The traits I varied in figure 7-12 are masculinity and nose-size. Nose-size was judged approximately on a scale from 1 to 10. Since nose-size and masculinity are related, there is some interference when moving along each dimension. This raises the need for *orthogonal traits*, which I discuss in Chapter 8. Otherwise, however, the algorithm was a success: nose size and masculinity do visibly vary along their respective axes.

Figure 7-11: The average face.

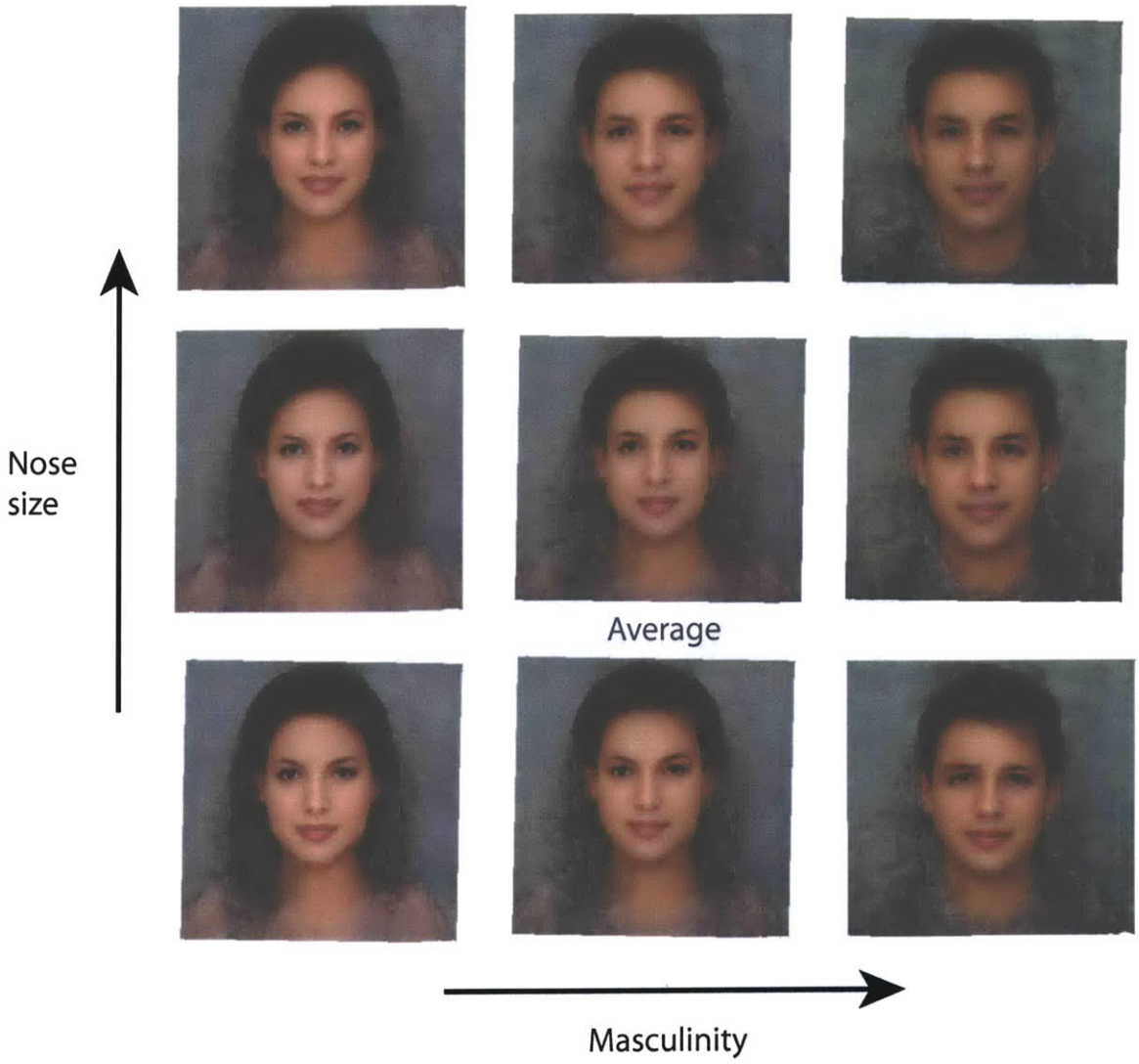

Figure 7-12: Altering masculinity and nose size, starting from the average face. The middle image is the average face. Moving from the bottom images to the top increases nose size; moving from the left images to the right increases masculinity.

Figure 7-13: The average dog.

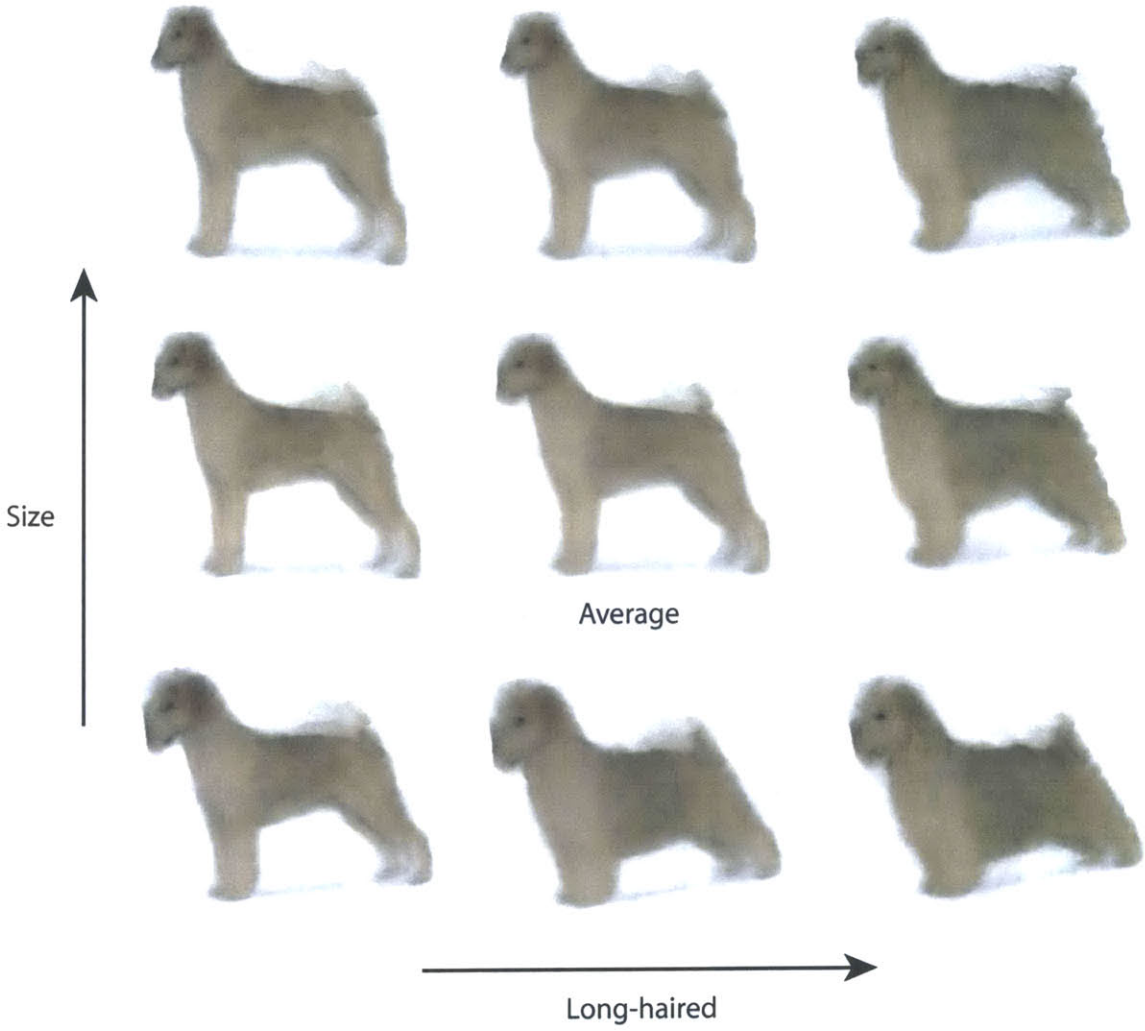

Figure 7-14: Altering the size of a dog, and whether it is a long hair or a short hair. The middle image is the average dog. Moving from the bottom images to the top increases dog size; moving from the left images to the right makes the dog more long-haired.

Figure 7-15: The average fish.

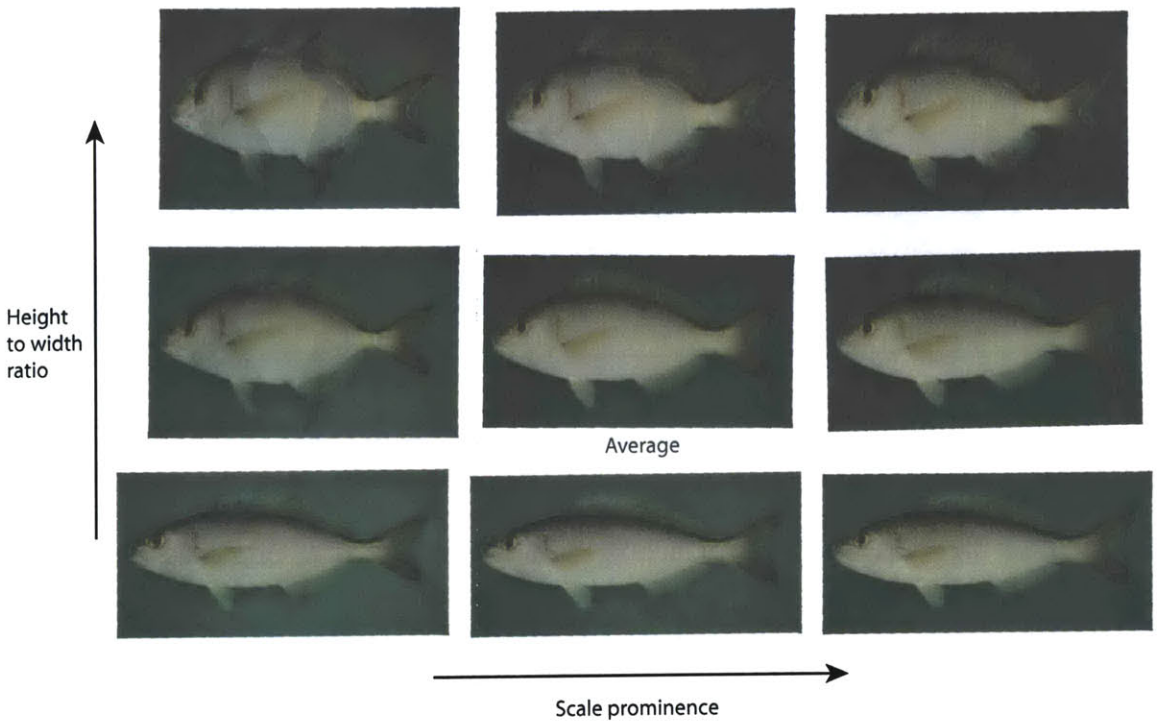

Figure 7-16: Altering the ratio of a fish's height to its width, and the prominence of the scales on the fish. The middle image is the average fish. Moving from the bottom images to the top increases the height to width ratio of the fish; moving from the left images to the right increases the prominence of the scales on the fish.

Results in the Dogs Dataset

The Dogs dataset has the worst of the results, in both alignment and in texture compatibility. This dataset had the largest variability in three-dimensional geometry of the models; this is a severe problem for two-dimensional image morphing. When two separate three-dimensional areas are projected onto the same x, y positions in the image plane of another model, misalignments will inevitably occur. There are currently no techniques to correct for this. (View morphing [14] corrects for a smooth variation of three-dimensional transformation when morphing between two images; it does not handle conflicting two-dimensional projections of three-dimensional geometry.)

Besides large-scale geometric alignment, the greatest problem in the Dogs dataset is texture incompatibility. Image morphs assume that on the small scale, color is constant and cross-fading textures returns the average color. This is not true in the Dogs dataset, because the edges of the individual hairs of a dog create large fluctuations in color. When cross-fading multiple dogs, these colors are not aligned, and so create gray, as in the background of the Faces dataset. (The background of the Dogs dataset were all the same - uniform white - so this problem does not occur there.)

Excepting these two problems, the results of the Dogs dataset are as expected. Varying the traits of the dog set worked well: there is clearly a progression of long hair that is independent of dog size.

Results in the Fish Dataset

The results in the Fish dataset are on the whole quite good. Different fin structures of the fish are well aligned, so that the average fish appears to have a physically realizable fish geometry. There are a few ghosting effects around the tail fin and outside the convex hull of the fish.

When altering traits, the scale prominence trait did not perform as expected. This is a measure of how visible the scales on each fish are: whether the fish appears smooth or scaly. The input morphs did not align the scales, however, since they were on too small a scale. The height to width ratio trait performed well. Note how the Zacor fish dominates the low scale prominence, high ratio of height to width fish. The Zacor fish was the only fish taller than it was wide.

7.2.3 Parameters of Input Morph Evaluation

Input morphs are evaluated during morph registration to determine both the reference image and the paths to the reference image that create reference morphs. The relative importance of chain length, derivative discontinuity, foldover, and holes in a chained morph can be set by the user. In figure 7-17, I show the effect of making each of these values the most important value. I show examples for only the Faces dataset, since it has the only cyclic input graph. (Morph evaluation is not necessary in an acyclic graph.)

The images are nearly identical. The largest difference occurs when the morph

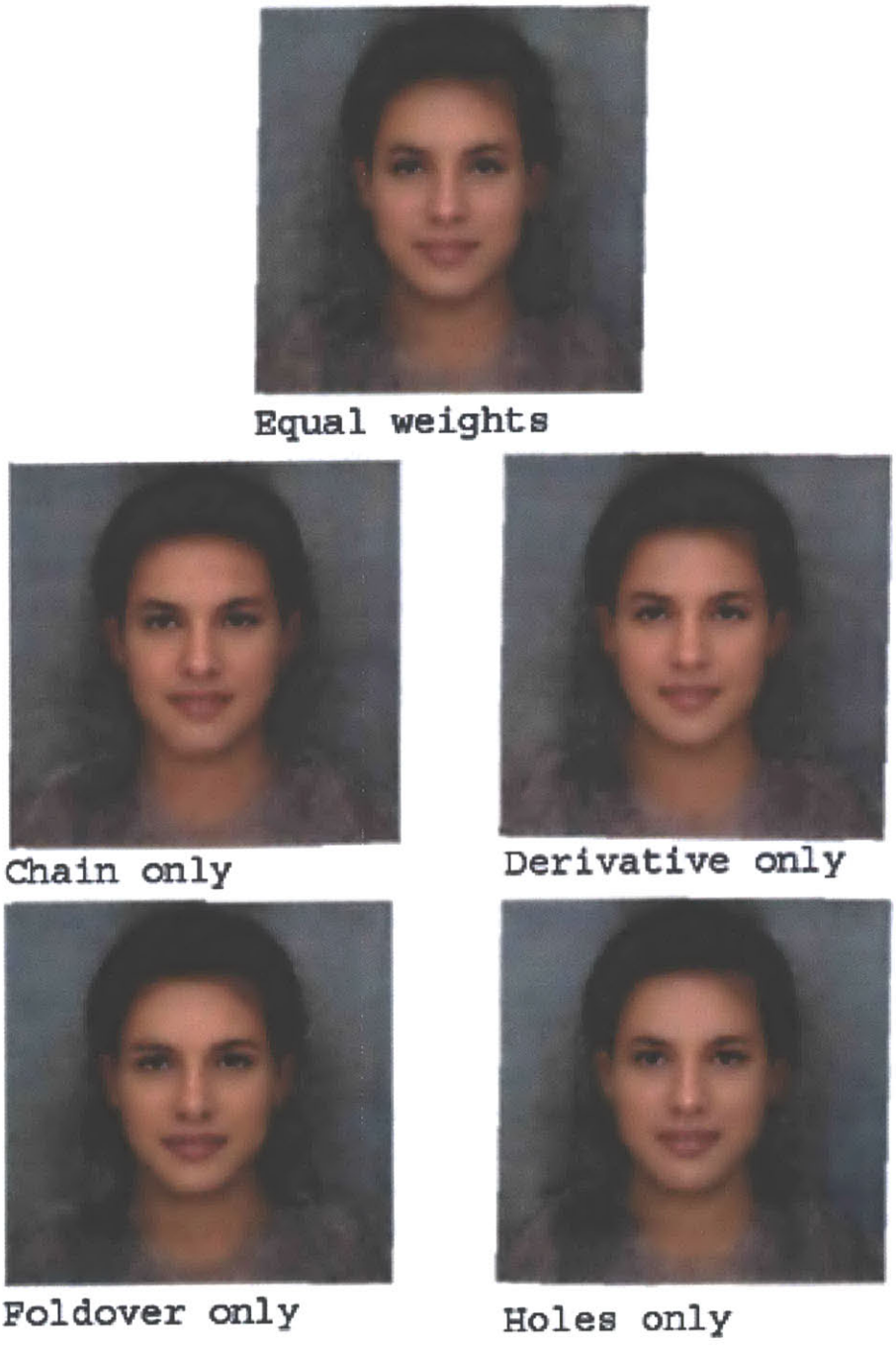

Figure 7-17: Varying morph evaluation parameters when creating the average image of the Faces dataset. Morph evaluation parameters were set as listed for each image: equal weights for all evaluation parameters in the top image, and 0 weight for all but one parameter in the other four images. k was set equal to 5 in the best k paths algorithm. Geometry and texture dimensionality reduction used linear combinations of basis vectors, and texture vectors were used directly to specify the multimorph image.

evaluation is entirely based on chain length. This case appears to have better alignment in the right forehead of the face. In practice, therefore, I set chain length weight to 3, and the weight for derivative discontinuities, foldover, and holes to 1.

7.2.4 Parameters of Best k Paths Algorithm

In morph registration, reference morphs are created by merging the best k paths. I show the effects of varying k in figure 7-18. As for morph evaluation, I show examples only for the Faces dataset (The best k paths algorithm is not used in an acyclic graph.)

Again, differences in the images are slight. $k = 5$ has a smoother edge, and a slightly smaller area of misaligned forehead in the upper right. In this dataset, however, merging more than one path does not significantly effect the results. I set $k = 5$ for images in the Faces dataset.

7.2.5 Parameters of Geometry Dimensionality Reduction

Geometry dimensionality reduction may be done using linear combinations of input vectors, linear combinations of singular vectors, LLE, or isomap. Linear combinations of singular vectors, LLE, and isomap are characterized by the dimensionality chosen for their embedding space. When using LLE and isomap, the results are further affected by the number of neighbors used when computing the reduction and reprojection of the high-dimensional vectors. I show the effects of these parameters on the average image for each dataset in figures 7-19, 7-20, and 7-21.

In each of these tests, the texture dimensionality reduction was set to linear combinations of input vectors, and texture vectors were used directly to specify the multimorph image. For the Faces dataset, morph evaluation weights were set to (3,1,1,1) as discussed above, and the number of merged paths was set to $k = 5$.

For each dataset, 6 images are shown: the average image using linear combination of input vectors, linear combinations of singular vectors, and then two images each of LLE and isomap. The first of the LLE and isomap images were computed using 12 neighbors in both reduction and reprojection. The second images were computed using 20 neighbors.

Although the results clearly vary depending on the reduction chosen, it is difficult to judge which result is more "correct".

When using linear combinations of singular vectors, LLE, or isomap; reducing the embedding dimensionality degraded results considerably (not shown). Since user input in my system uses traits instead of specifying embedding coordinates directly, it was not critical to use a reduced embedding space, and so I always use the maximum embedding dimensionality (the number of vertices, minus one).

7.2.6 Parameters of Texture Dimensionality Reduction

As with geometry dimensionality reduction, texture dimensionality reduction may be done using linear combinations of input vectors, linear combinations of singular vectors, LLE, or isomap. An additional choice is whether embedding coordinates

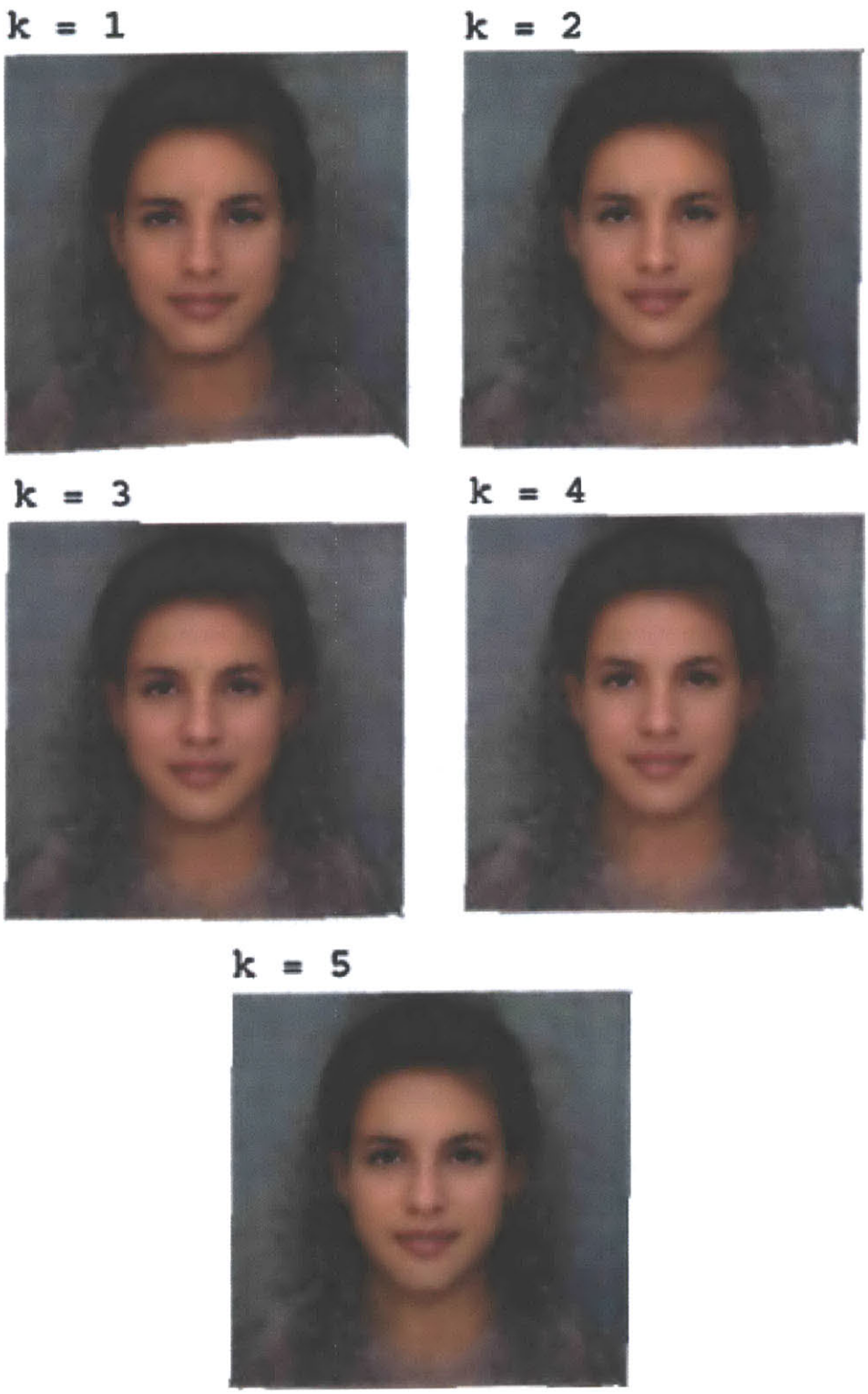

Figure 7-18: Varying best k paths parameters when creating the average image of the Faces dataset. Morph evaluation weights were set to (length, derivative, foldover, holes) = (3,1,1,1). k in the best k paths algorithm was set as listed for each image. Geometry and texture dimensionality reduction used linear combinations of basis vectors, and texture vectors were used directly to specify the multimorph image.

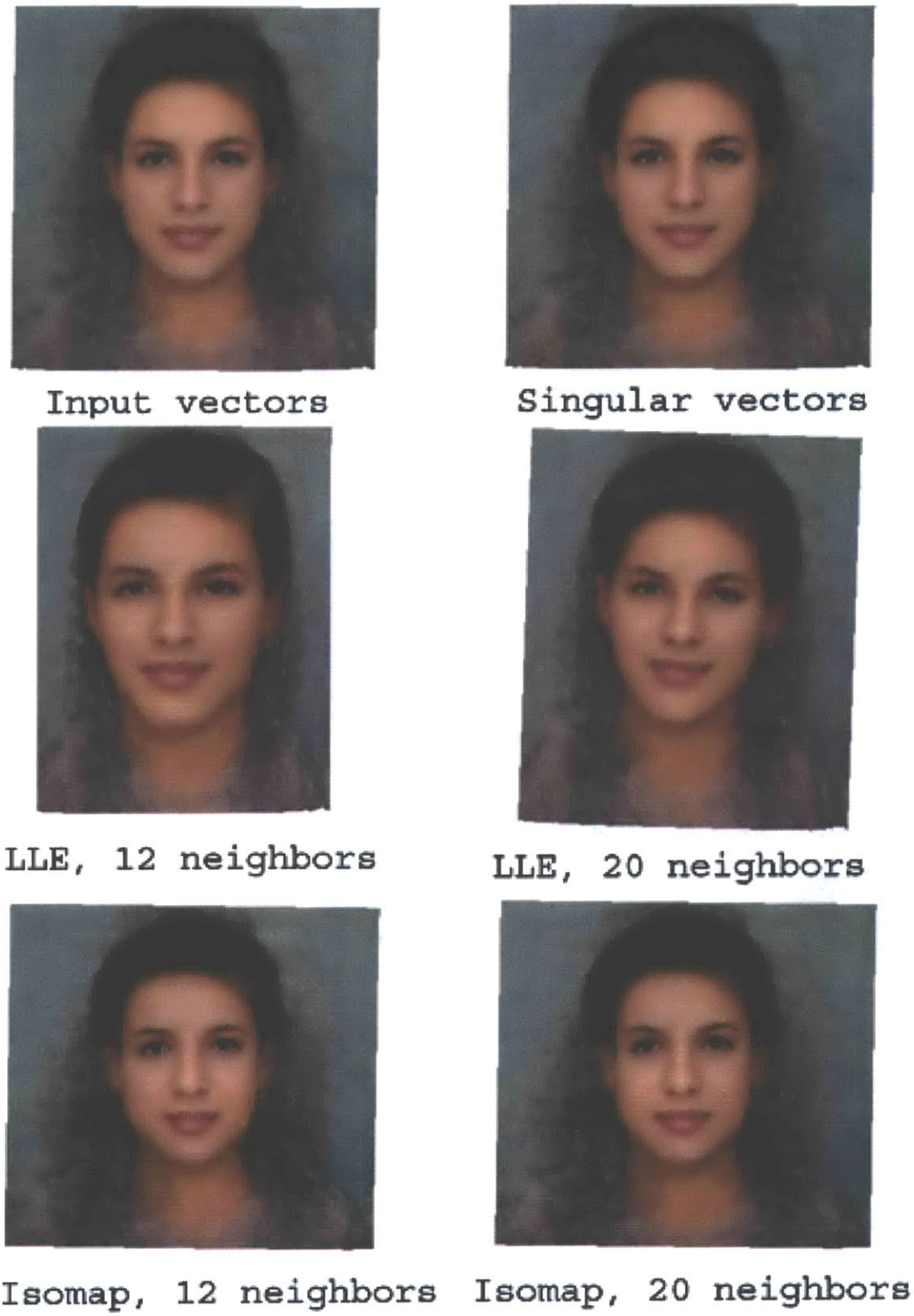

Figure 7-19: Varying the type of geometry dimensionality reduction used to create the average image in the Faces dataset.

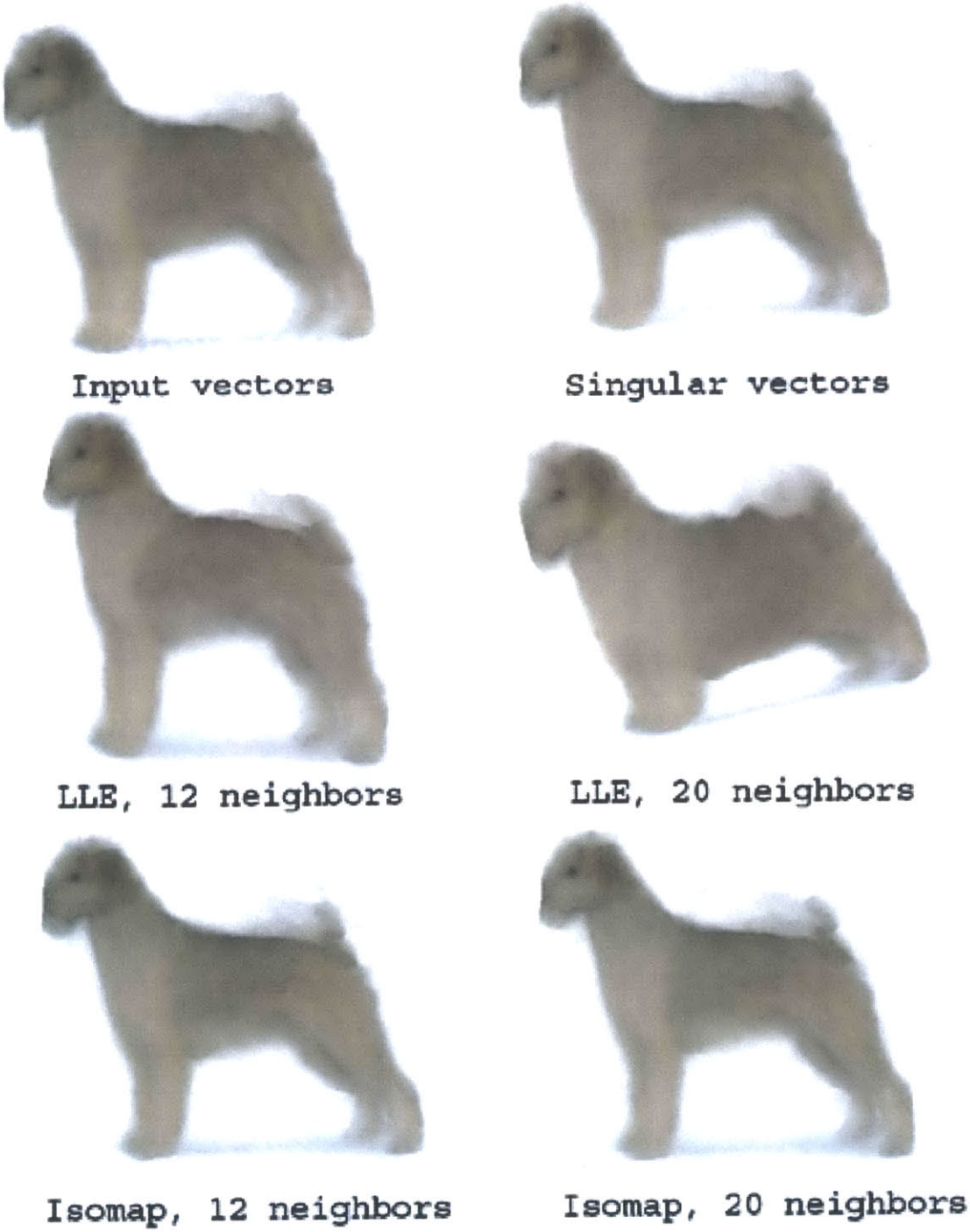

Figure 7-20: Varying the type of geometry dimensionality reduction used to create the average image in the Dogs dataset.

Figure 7-21: Varying the type of geometry dimensionality reduction used to create the average image in the Fish dataset.

should be reprojected back onto a high dimensional texture vector, which is used directly in a single warp to the shape of the reprojected geometry vector, or whether the embedding coordinates should be reprojected onto coefficients of the input images, to specify the weights in a cross-fade of the n_v warped input images. I show the effects of varying these parameters in figures 7-22, 7-23, 7-24, 7-25, 7-26, and 7-27.

In each of these tests, the geometry dimensionality reduction was set to linear combinations of input vectors. For the Faces dataset, morph evaluation weights were set to (3,1,1,1) as discussed above, and the number of merged paths was set to $k = 5$.

For each dataset, 12 images are shown: the 4 types of dimensionality reduction (LLE and isomap two images each), evaluated on the left with cross-fade coefficients and on the right with direct warping of texture vectors.

It is clear in these results that more cross-faded images produce a better output image. When using LLE or Isomap, which only combine 12 or 20 neighboring images, ghosting strongly degrades the quality of the output image. This is most visible in the Faces dataset. The results are more severe when combining 12 images than when combining 20. I use linear combinations of input vectors in practice, since using isomap or LLE does not appear to improve results.

Surprisingly, the results when using texture vectors directly are nearly identical to the results when using cross-fade coefficients. Since using texture vectors directly is far more efficient (one image is warped, instead of n_v), I use direct warping of texture vectors in practice.

7.3 Conclusion

This concludes my discussion of results for the implementation of my multimorph algorithm. In the next chapter, I discuss my views on the success of this technique, and areas for future work.

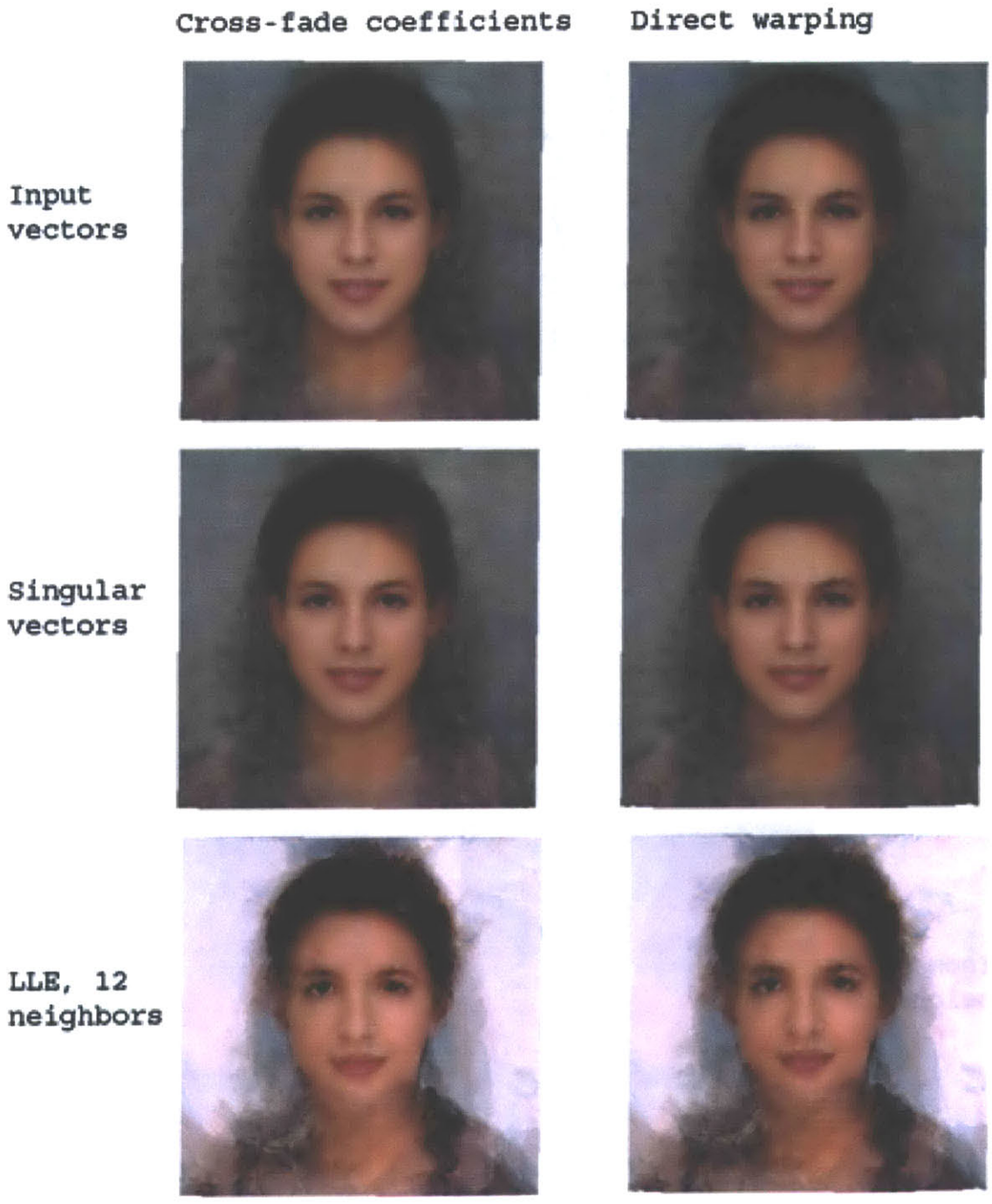

Figure 7-22: Varying the type of texture dimensionality reduction used to create the average image in the Faces dataset.

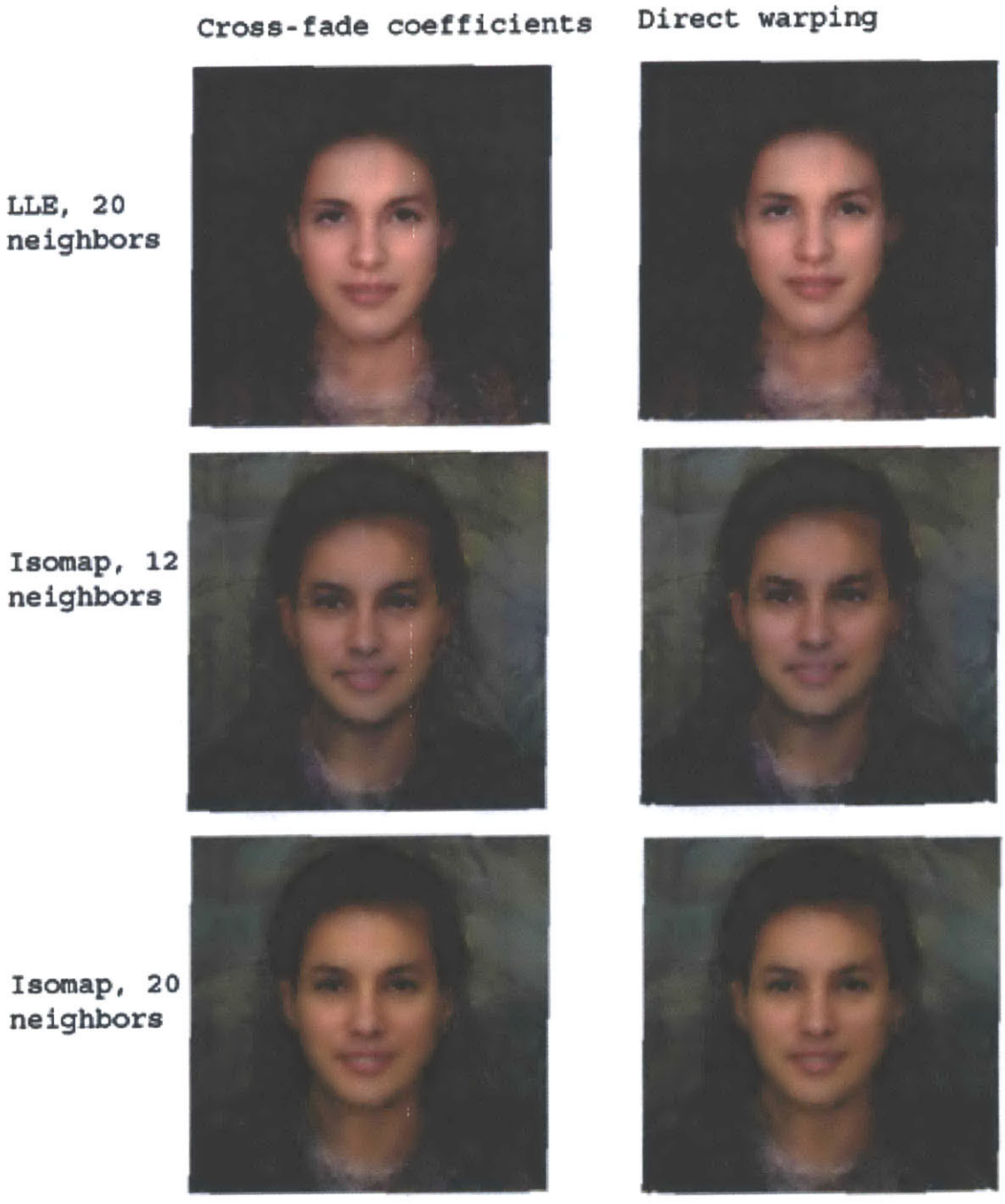

Figure 7-23: Varying the type of texture dimensionality reduction used to create the average image in the Faces dataset.

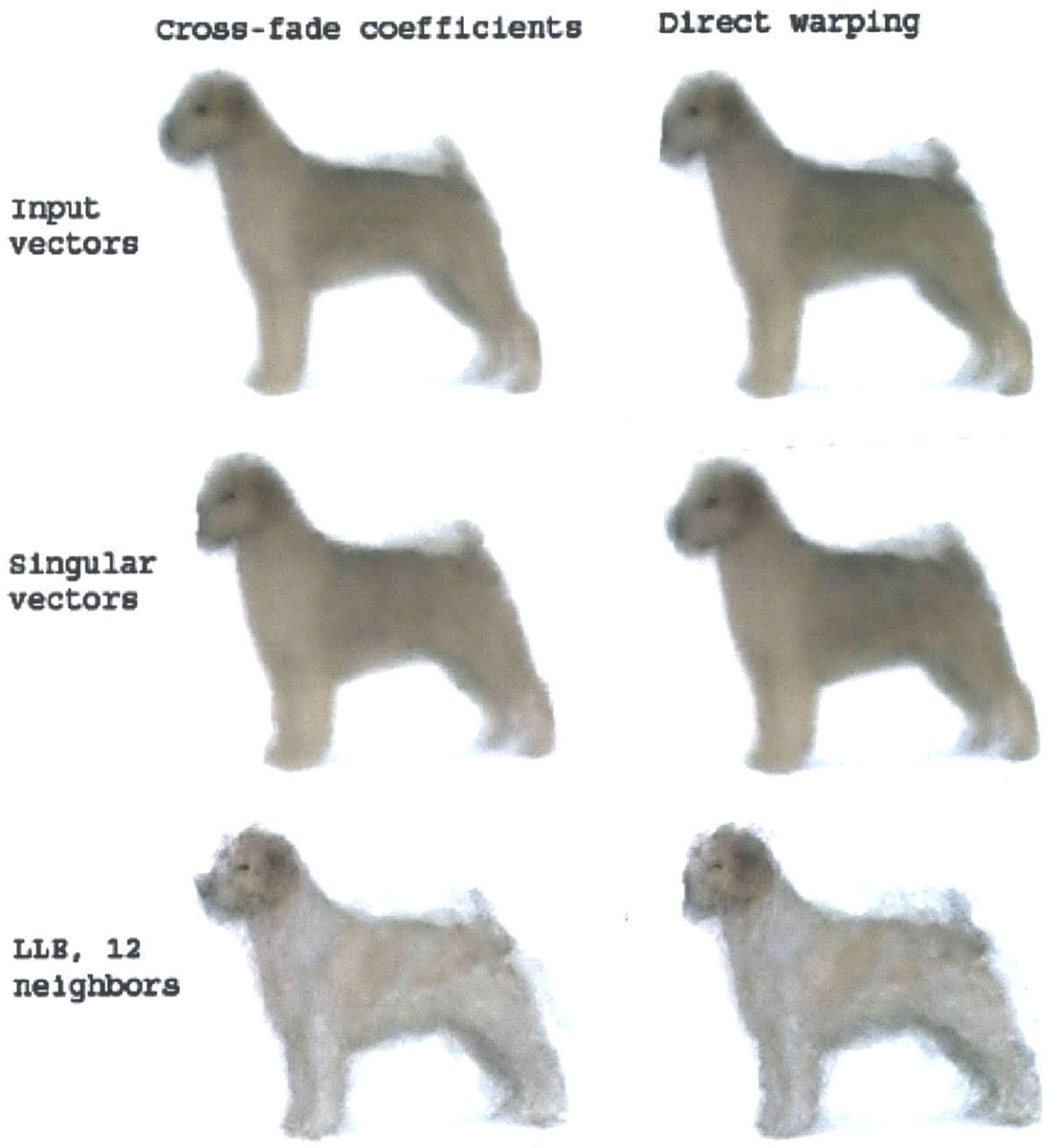

Figure 7-24: Varying the type of texture dimensionality reduction used to create the average image in the Dogs dataset.

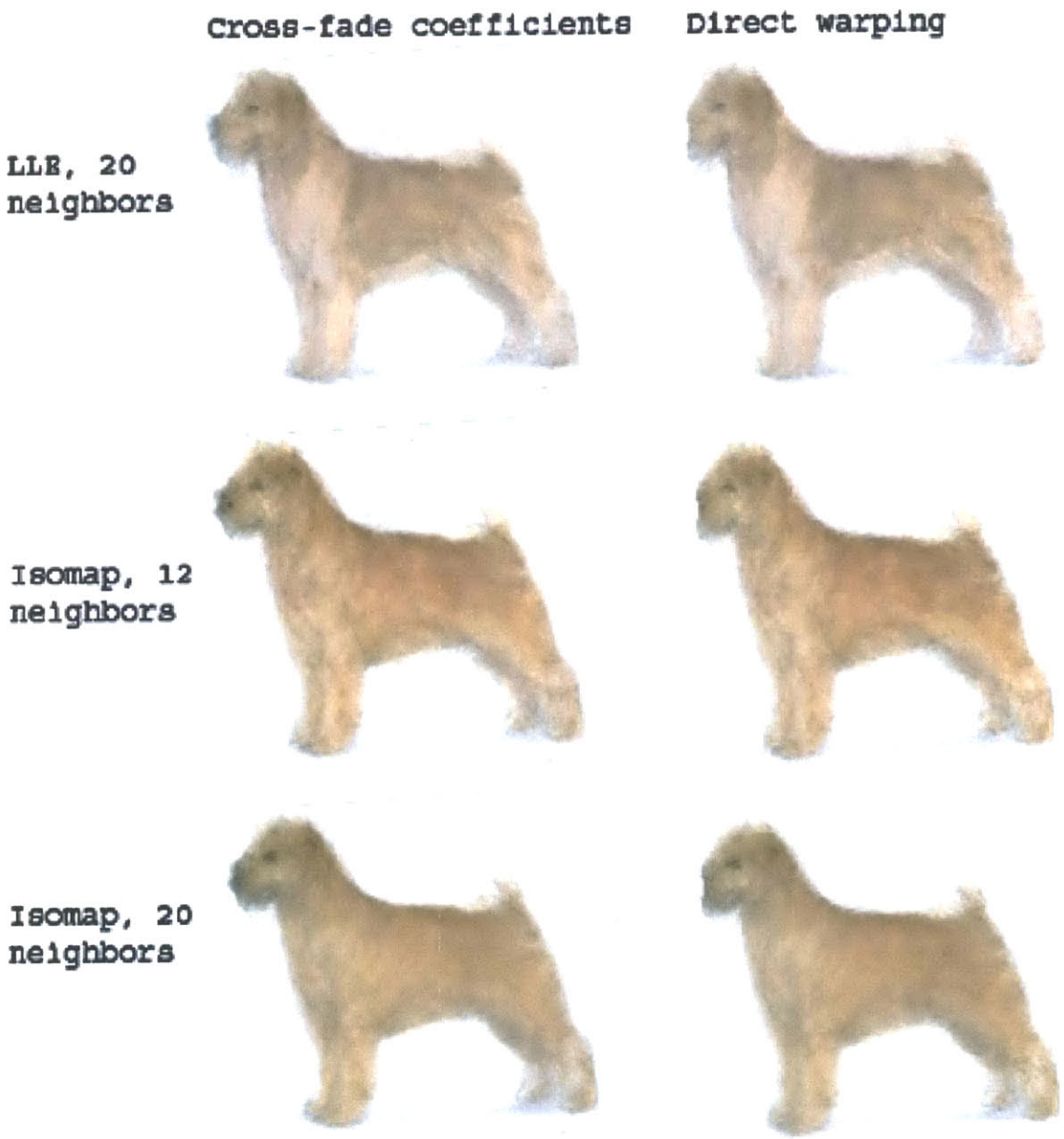

Figure 7-25: Varying the type of texture dimensionality reduction used to create the average image in the Dogs dataset.

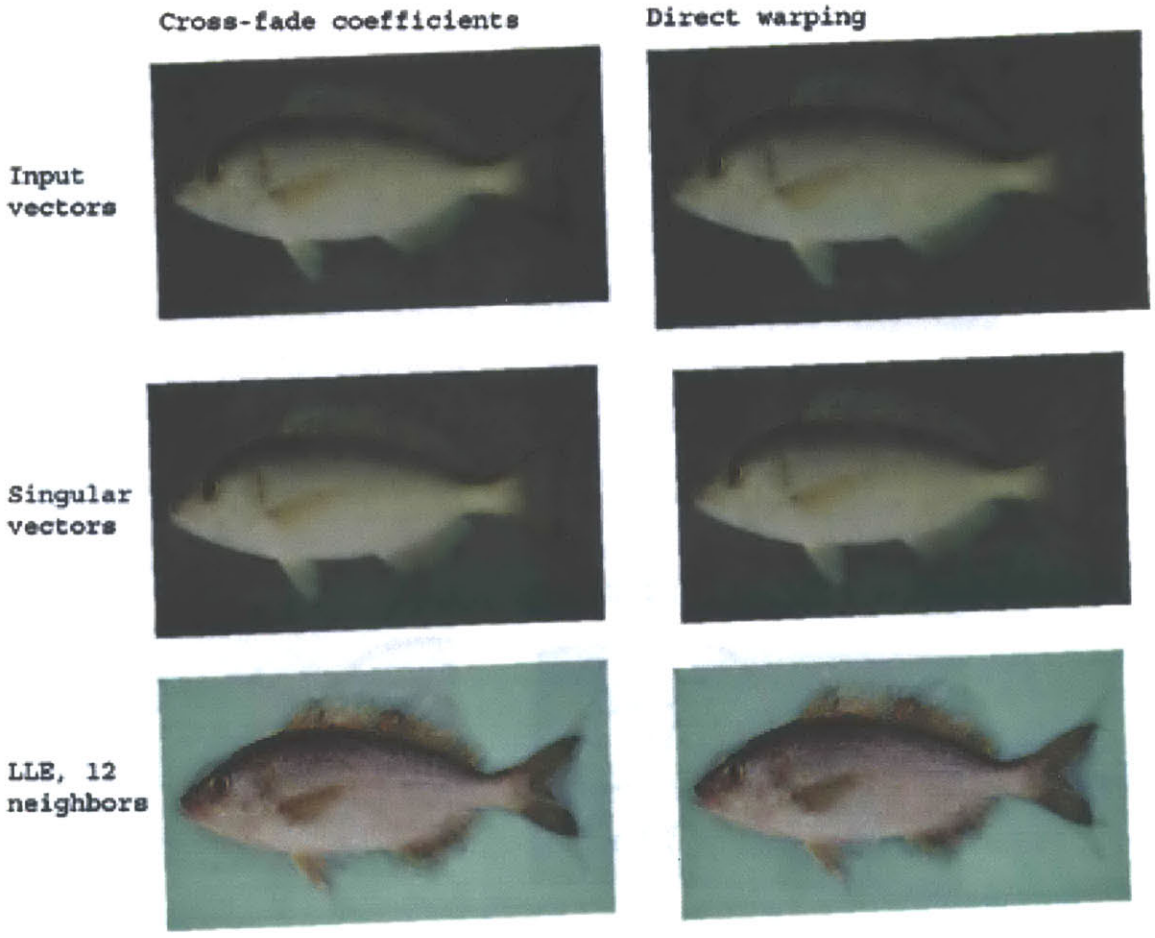

Figure 7-26: Varying the type of texture dimensionality reduction used to create the average image in the Fish dataset.

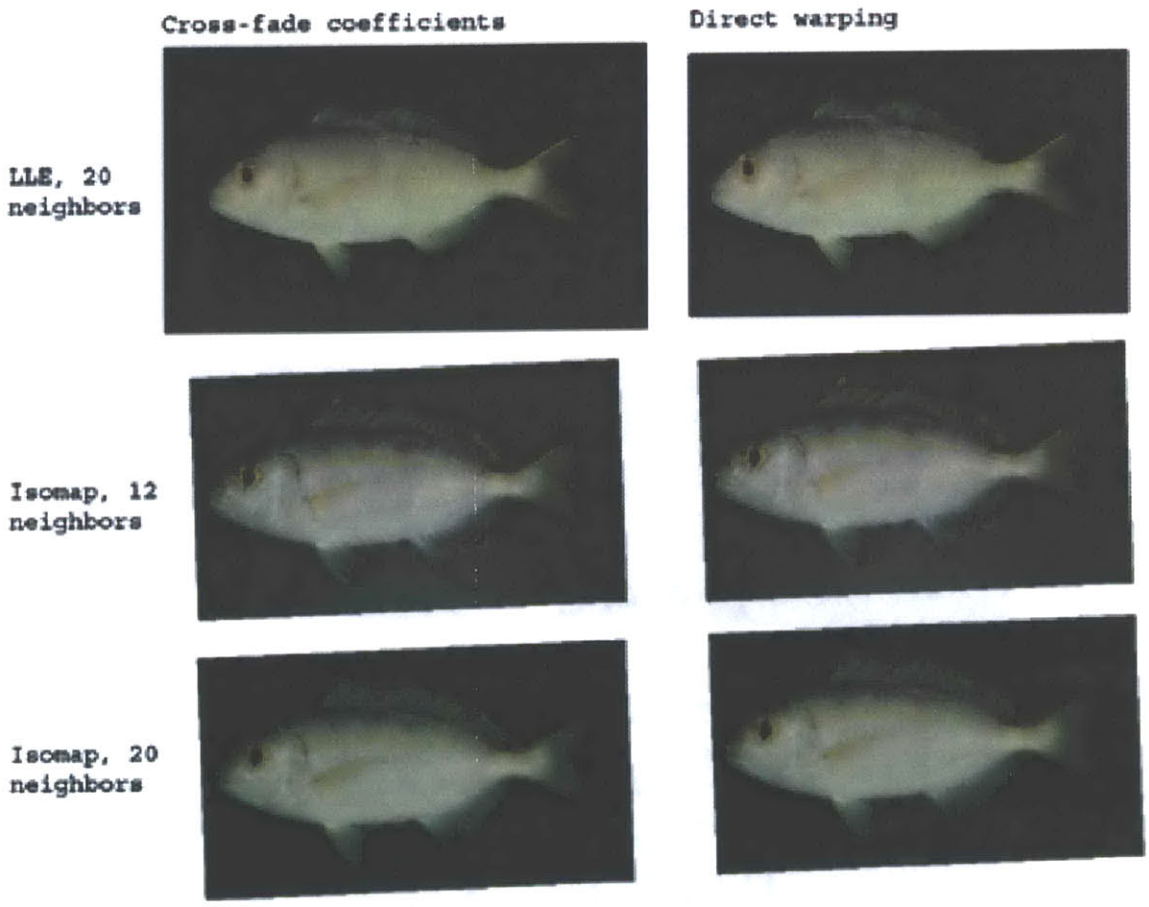

Figure 7-27: Varying the type of texture dimensionality reduction used to create the average image in the Fish dataset.

Chapter 8

Conclusions and Future Work

8.1 Performance of Algorithm

8.1.1 Success

My multimorph algorithm has partially successful results. If images can be successfully aligned, then the output images are reasonable combinations of the input images.

Setting the position of features in the output image to linear or non-linear interpolations of the features in the input images works well, if the combined features are reasonable similar. Reasonable in this case means that there is no pathological data in which neighboring points in one image are far removed from each other in another image. If the warps between images are fundamentally simplitudal transformations of local areas that are correctly stitched together, then my interpolations of the geometry are quite successful.

Interpolating texture does not work so well as interpolating geometry. The texture combinations in this algorithm work best when the textures being combined are regions of solid color. The texture part of the algorithm is then just choosing the color of each region from the colors of the input images for that region. When the input images have not been separated finely enough, then non-corresponding features in the texture merge unaligned, creating either gray pixels or ghosting effects.

Traits work well; in my datasets increasing trait values successfully increased the attributes on which I based the traits.

8.1.2 Failure

The failure of the algorithm is evident in imperfect alignment and the inability of my system to capture the set of input images precisely. Imperfect alignment is visible in the Dogs dataset: large areas of the output image are only present in some of the warped images, creating ghosting when they are combined.

Part of this problem is due to a fundamental inability of digital image morphs to capture three-dimensional geometry. Multiple three-dimensional areas in one image projected onto the same area in the image plane of another image can not be aligned.

The other part of the problem is in morph registration. Input edge morphs from the same image may produce excellent but completely different warps of the image. Chaining these edge morphs tries to warp space in one of the destination images to the other destination image when no corresponding area exists. This is a similar problem to the inability to capture three-dimensional geometry, and is also impossible to fix.

Less fundamental problems in my implementation stem from dimensionality reduction. Combining x and y positions of features in several images through direct linear averages does not produce compelling results. My techniques for non-linear combinations, however, suffer from a sparsity of input data. Each of my datasets has on the order of 100 images, which correspond to points in a 10000-dimensional vector space. This is a vastly under-constrained system, and the techniques I use - isomap and LLE - require the input to be much denser to work correctly. The effects of this failure are that combinations are essentially linear, and stray from the manifold of input images considerably.

8.1.3 Final Judgement

As is, my implementation creates interesting combinations of images of similar geometry. These images, however, are not high-quality, and are therefore useful primarily as artistic inspiration and not as production images. If the problem of dimensionality reduction can be fixed to better capture the space of input images, then my algorithm will be more successful.

8.2 Future Work

8.2.1 Three-dimensional Morphing

The largest obstacle to good output images in my system is the use of digital image morphs - warps of projections of three-dimensional geometry. The alternative to digital image morphs is morphing three-dimensional geometry. This was done for facial meshes by Blanz and Vetter [2]. Their algorithm could benefit from the steps used in my algorithm for morph registration, dimensionality reduction, and traits. Other possible expansions of morphing into three dimensions are *Metamorphosis of Arbitrary Triangular Meshes* by Kanai, Suzuki, and Kimura [8], *Feature-Based Volume Metamorphosis* by Lerios, Garfinkle, and Levoy [11], and *Shape Transformation Using Variational Implicit Functions* by Turk and O'Brien [16]. Combining these methods for two-object three-dimensional morphing with my methods for combining multiple source objects is an area for future work.

8.2.2 Background Recognizer

When correspondence in the background of images is not specified, the warps of these regions cross-fade to gray, or worse, ghosting. It would be a useful feature to detect these areas and remove them by setting their pixels in the output image to a

background image or color. These areas can by automatically marked by taking the convex hull of the correspondence features in the input morphs. Alternatively, they can be detected from the warped images used in the final cross-fade by checking for variance of each pixel over a threshold.

8.2.3 Orthogonal Traits

In the Faces dataset, increasing the nose size trait was not independent of increasing masculinity. Since traits are represented in my system as gradient vectors at each embedding point, it is possible to convert them into an orthogonal basis. Handling this conversion so that trait vectors increase one trait while keeping others constant is an area for further work in traits.

8.3 Contribution

My work in this thesis is a contribution to the field of digital image morphing. I describe a process to create morphs of multiple source images. My discussion of this process is more in-depth than has been given before. Merging redundant user input, using non-linear combinations of input images to produce output images, and modeling traits as non-linear functions are all novel contributions.

Bibliography

[1] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Proceedings of SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques*, 1992.

[2] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques*, pages 187–194, 1999.

[3] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[4] American Kennel Club. World Wide Web electronic publication. http://www.akc.org/, June 2002.

[5] R. Froese and D. Pauly. World Wide Web electronic publication. http://www.fishbase.org/, June 2002.

[6] A. Goshtasby. Piecewise linear mapping functions for image registration. *Pattern Recognition*, 6:459–466, 1986.

[7] Michael J. Jones and Tomaso Poggio. Multidimensional morphable models. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 683–688, 1998.

[8] Takashi Kanai, Hiromasa Suzuki, and Fumihiko Kimura. Metamorphosis of arbitrary triangular meshes. *IEEE Computer Graphics amd Applications, Perceiving 3D Shape*, 20(2), Mar/Apr 2000.

[9] J. Lasenby, W. J. Fitzgerald, C. J. L. Doran, and A. N. Lasenby. New geometric methods for computer vision. *International Journal of Computer Vision*, 36(3):191–213, 1998.

[10] Seungyong Lee, George Wolberg, and Sung Yong Shin. Polymorph: Morphing among multiple images. *IEEE Computer Graphics Applications*, 18:58–71, 1998.

[11] Apostolos Lerios, Chase D. Garfinkle, and Marc Levoy. Feature-based volume metamorphosis. In *Proceedings of SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques*, 1995.

[12] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, December 2000.

[13] D. Ruprecht and H. Muller. Image warping with scattered data interpolation. *IEEE Computer Graphics and Applications*, 15:37–43, March 1995.

[14] Steven M. Seitz and Charles R. Dyer. View morphing. In *Proceedings of SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques*, pages 21–30, 1996.

[15] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, December 2000.

[16] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques*, 1999.

[17] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[18] Thomas Vetter, Michael J. Jones, and Tomaso Poggio. A bootstrapping algorithm for learning linear models of object classes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 40–46, 1997.

[19] George Wolberg. *Digital Image Warping.* IEEE Computer Society Press, third edition, 1994.

[20] George Wolberg. Image morphing: A survey. *The Visual Computer*, 14:360–372, 1998.