Morphing Using Curves and Shape Interpolation Techniques

Henry Johan[†] Yuichi Koiso[‡] Tomoyuki Nishita[‡]

[†]Dept. of Information Science
[‡]Dept. of Complexity Science and Engineering
University of Tokyo
Hongo 7-3-1, Bunkyo, Tokyo 113-0033, Japan
{henry, koiso, nis}@is.s.u-tokyo.ac.jp

Abstract

This paper presents solutions to the feature correspondence and feature interpolation problems in image morphing. The user specifies the correspondence between the source and the target images by drawing input curves on the features of the objects. The correspondence between these curves at the finest level (pixel level) is computed by optimizing a cost function. Based on this correspondence, the input curves are approximated by using Bézier curves. We represent the Bézier curves and the connections among them by using a "dependency graph". Feature interpolation is performed by interpolating the dependency graphs using the edge-angle blending technique. We also propose methods for controlling the transition rates of the shape and the color. We implemented the proposed algorithms in our morphing system which is based on the field morphing technique. From experimental results, our algorithms can generate a smooth morphing animation even when the objects in the source and the target images have different orientations. Also, the user's workload is reduced because the system includes an automatic feature correspondence computation.

Keywords: *Morphing, Shape Interpolation, Transition Control, Warping, Deformation, Animation.*

1 Introduction

Morphing (Metamorphosis) is a technique used to generate a sequence of images that smoothly transform a source image into a target image. This technique is often used to create special effects for motion pictures or television commercials. A conventional method of generating an intermediate image is by cross-dissolving the source and the target images. This approach, however, cannot generate a high-quality image since the features (landmarks) of the objects in the source and the target images are generally not coincident with each other. To assure smooth transformation, first the source and the target images are warped so that the features on both images are aligned to each other and then the colors of the warped images are cross-dissolved.

There are four main issues to overcome in morphing. First, how can we extract the features from the source and the target images, and establish the correspondence between these features. Second, how should we interpolate these features to generate features for the intermediate image. Third, how should we define a warp function, and lastly, how can we control the transition rates of the shape and the color.

The first issue concerns the amount of effort required by the user to specify the correspondence between the source and the target images. It is important to help the user to perform this process easily. The feature interpolation problem becomes complicated, especially when rotation is involved. This paper presents solutions to these problems.

The user establishes the coarse correspondence between the source and the target images by drawing input curves on the both images. The correspondence between these curves at the finest level (pixel level) is computed by optimizing a cost function. Based on this correspondence, the input curves are approximated by using the least squares fitting [14] of piecewise cubic Bézier curves. We call these Bézier curves *"feature curves"*.

We present a method for interpolating the features correctly even when the orientation of the object in the source image is different to the one in the target image. That is, the features do not shrink during the morphing animation. In the preprocessing step, we construct dependency graphs for the both images. A dependency graph represents the feature curves and the connections among them. The features in the intermediate image are generated by applying the edge-angle blending technique [22] on the two graphs.

Our morphing system uses the field morphing technique [2] for the warp computation. The field morphing approach is very simple and easy to implement, yet it can generate impressive effects. Since the features of the images are represented using Bézier curves, we use pairs of Bézier curves to define the mapping from one image to the other. The Bézier curves are extended to what we call *”extended feature curves”* and we use these extended feature curves in the warp computation. We also allow the user to specify transition curves for controlling the transition rates of shape and color. As a result, more interesting animations can be produced.

Section 2 describes previous work on the subproblems in 2D-image morphing. Section 3 discusses our solution to the feature specification problem. Section 4 deals with feature interpolation and shape transition control problems. Section 5 describes the field morphing algorithm and our approach to color transition control. Then, we show some examples to demonstrate the advantages of our methods. The final section concludes this paper and describes the direction for further research.

2 Previous work

First, we list some works related to the problem of finding the correspondence between two polygonal shapes. All of the methods that are going to be described can also be used to calculate the correspondence between two open piecewise linear curves. Sederberg and Greenwood [21] established a correspondence between two polygonal shapes by finding a global minimum solution of a work function. Their method requires the user to specify the values of seven physically based parameters. The drawback is that it is very difficult to specify the values of these parameters in order to achieve the desired result. The method in [25] established the correspondence by maximizing the sum of the dot product between tangent vectors at the sampling points on the source and the target shapes. However, the calculation fails when the orientations of the two shapes differ.

Carmel and Cohen-Or [3] proposed a method that first warps the source object and aligns it with the target object. Then, using the polygon-evolution algorithm, the warped source and target objects are converted into two convex polygons that are projected onto two identical circles. Merging the topologies of the projected objects and projecting them back to the original objects obtains the correspondence.

In feature based morphing, feature interpolation is very important. This problem is similar to the vertex path problem in shape interpolation. Sederberg et al. [22] proposed an algorithm that blends the intrinsic definitions (edge lengths and vertex angles) of two polygonal shapes (edge-angle blending). In Shapira and Rappoport [23], they represent the interior of the two shapes by using *”star-skeletons”* representation. Intermediate shapes are generated by blending the parametric description of these skeletons. Goldstein and Gotsman [6] presented an algorithm which first applies the polygon evolution to the two shapes and then blends the vertex paths resulting from the polygon evolution process.

Another technique for solving the vertex path problem was presented by Carmel and Cohen-Or [3]. They compute the vertex path with a transformation which consists of a rigid part and an elastic part. The rigid part of the transformation performs rotation and translation, while the elastic part performs a linear interpolation between the corresponding vertices. Next is the work of Zhang and Huang [30]. First, they apply the wavelet decomposition to the two shapes, then they blend their low-resolution shapes and after that they perform the wavelet reconstruction.

The shortcoming of these methods is that they only consider the interpolation between two object boundaries. However, in morphing, the user can specify features not only on the boundaries, but also inside or outside the boundaries. Therefore, we cannot use these methods directly in morphing

Recent work on this problem is done by Alexa et al. [1]. Their algorithm computes the compatible triangulation of the two shapes. Next, the optimal least-distorting morphing between each pair of corresponding triangles is determined. The global transformation is defined as a transformation which minimizes the overall local deformation. This method can be used to perform the interpolation like the one shown in Figures 12-14 if we take into account all the unconnected features when computing the compatible triangulation. However, in the paper they did not mention about the method to compute this kind of triangulation.

Although warp computation is not the main theme in this research, we also list some 2D-image warping techniques here. In Wolberg’s book [28], the mesh warping method was introduced. Another mesh-based technique was the work of Nishita et al. [17]. Beier and Neely proposed the field morphing technique [2]. An optimization based on a piecewise linear approximation was proposed by Lee et al. [11] to accelerate the warp computation. In Lee et al. [8], Litwinowicz and Williams [13], the radial basis functions approaches for warp computation were introduced. Wolberg [29], Ruprecht and Muller [19] provided an extensive survey of this method.

Lee et al. [10] developed a method that can generate one-to-one and C^1-continuous warp functions. In [9], Lee et al. presented a new warp generation method that is simpler and faster than the energy minimization method [10]. The warp function generated by this method is one-to-one and C^2-continuous. A computer vision technique called snake [7] is used to help the user to establish the correspondence between the images. An algorithm to transform two similar images with little or no human interaction is proposed by Gao and Sederberg [5], Shinagawa and Kunii [24], and Tam and Fournier [26]. Tal and Elber [27] proposed a hybrid

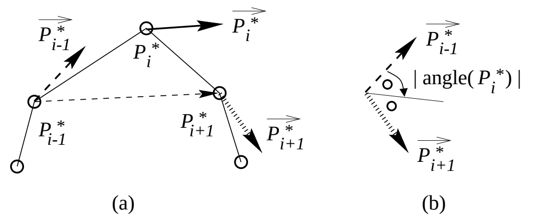

$\overrightarrow{P^*_{i-1}}$
P^*_i
$\overrightarrow{P^*_i}$
P^*_{i-1}
P^*_{i+1}
$\overrightarrow{P^*_{i+1}}$
(a)
$\overrightarrow{P^*_{i-1}}$
$|\,\text{angle}(P^*_i)\,|$
$\overrightarrow{P^*_{i+1}}$
(b)

Figure 1. (a) Tangent vector and (b) angle cost calculations.

approach which combines an image morphing method and a polygonal morphing method.

3 Feature specification

Feature specification is the most tedious operation for the user in the whole morphing process. First, the user determines a portion on the source image and its correspondence on the target image. Then points, lines, or curves are used to specify the locations of these features.

3.1 Feature specification primitive

We use curves as a primitive for specifying features. These curves can be open curves or closed curves. The user specifies the corresponding features of the objects in the source and the target images by drawing two input curves, one in the source image (*"source curve"*) and one in the target image (*"target curve"*). For example, if we want to morph a gorilla to a bear, we draw curves on the features of these animals. These curves represent the correspondence at a coarse level. The correspondence at a finer level are computed automatically.

3.2 Correspondence calculation

After the user specifies two input curves, one on the source image and one on the target image, the next step is to find a correspondence between these curves at the finest level (pixel level). We sample the input curve with a set of equally spaced points and hence obtain piecewise linear curve. Therefore, our problem is reduced to finding a correspondence between two piecewise linear curves. If the input curve is a closed curve, then the piecewise linear curve becomes a polygonal shape.

We extend the work of Sederberg and Greenwood [21] and Cohen et al. [25]. By defining a new cost function, we overcome the problems found in [21] and [25]. Our approach is as follows. Let the curves on the source and the target images being sampled using n and m points, respectively. Without loss of generality, we assume that $n \geq m$. Let P^S_i, $(i = 1, ..., n)$ and P^T_j, $(j = 1, ..., m)$ to be the points on the source and the target curves. For each point P^*_i, (P^S_i or P^T_i) on the curve, we determine the tangent vector $\overrightarrow{P^*_i}$ as follows (see Figure 1(a)).

$$\overrightarrow{P^*_i} = \frac{P^*_{i+1} - P^*_{i-1}}{\|P^*_{i+1} - P^*_{i-1}\|}. \tag{1}$$

We calculate the correspondence by minimizing the sum of a cost function, which is defined as a function of angle cost and parameter cost. Let sign$(x) = 1.0$ when $x \geq 0.0$ and sign$(x) = -1.0$ when $x < 0.0$. Also, let $Z(\overrightarrow{A}, \overrightarrow{B}) = A_x B_y - A_y B_x$. The angle cost at a sampling point P^*_i is defined using the following equation (see Figure 1(b)).

$$\text{angle}(P^*_i) = \frac{1}{2}\text{arccos}(\overrightarrow{P^*_{i-1}} \cdot \overrightarrow{P^*_{i+1}})\ \text{sign}(Z(\overrightarrow{P^*_{i+1}}, \overrightarrow{P^*_{i-1}})). \tag{2}$$

The parameter cost at point P^S_i is simply defined as $\frac{i}{n}$. Similarly, the parameter value at point P^T_j is defined as $\frac{j}{m}$. Using the angle and the parameter costs, we define the cost function as follows.

$$\text{cost}(i, j) = w_1|\text{angle}(P^S_i) - \text{angle}(P^T_j)| + w_2|\frac{i}{n} - \frac{j}{m}|. \tag{3}$$

The angle term extracts the similarity between the two objects and the parameter term assures that two points far apart will not be set to correspond to each other. Our cost function only has two parameters, w_1 and w_2, which are the weights of the angle and parameter costs. In practice, weights of $w_1 = 1.0$ and $w_2 = 3.0$ seem to perform well over a wide range of images.

Adopting this cost function, the optimization problem that needs to be solved is

$$\min \sum_{i=1}^{n} \text{cost}(i, J(i)). \tag{4}$$

This optimization problem is solved subject to $J(1) = \alpha$, $J(n)$ being a direct neighbor of α, and that $J(i)$, $(i = 2, ..., n-1)$ changes in one direction from $J(1)$ towards $J(n)$. We solve this problem using the dynamic programming technique. When the curves are open curves, we use a constraint that the end points of the source and the target curves correspond alternately. If the curves are closed curves, then the global minimum can be calculated using the algorithm in [4] without having the user specifying the initial correspondence.

Figure 2 shows an example of shape interpolation between a turtle and a wolf (same as the example in [25]). The correspondence between the turtle and the wolf are calculated by using our algorithm. In the shape interpolation process, the mouth, the legs, and the tail of the turtle became the mouth, the legs, and the tail of the wolf. This result confirmed that our approach could produce a correct correspondence even when the two shapes have different orientations.

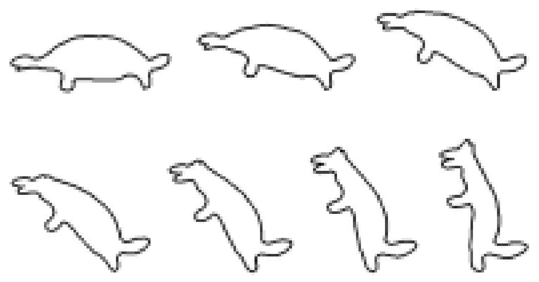


Figure 2. Shape interpolation from a turtle to a wolf.

3.3 Bézier curve-fitting

This process starts by subdividing the input curve into several curves based on the correspondence, then approximating these curves with cubic Bézier curves using the least squares fitting method [14]. In a preprocessing step, we use the absolute values of the angle costs of all sampling points on the source, target curves and consider them as functions $f(i), g(i)$, where i is the index of the sampling points. Then, we search for the local maxima of the functions $f(i)$, $g(i)$. For the subdivision process we developed an iterative algorithm. Without loss of generality, we assume that the sampling points in the source curve are more than the one in the target curve. Before we perform the subdivision process, we first determine an initial point P_{init} and a last point P_{last}. When the curve is an open curve, the first point and the last point of the source curve are used as P_{init} and P_{last}, respectively. On the other hand, if the curve is a closed curve, one of the local maxima of function f is used as both P_{init} and P_{last}. Next, we initialize $P^S_{now} = P_{init}$. The detail of the curve-fitting algorithm is as follows.

1. $P^S_{start} = P^S_{now}$, $src = 0$, $tgt = 0$. Let P^T_{start} to be the corresponding point to P^S_{start}.
2. While ($P^S_{now} \neq P_{last}$)
   - 2.1. Let P^T_{now} to be the corresponding point to P^S_{now}.
   - 2.2. If P^S_{now} is a local maxima of $f(i)$ then $src = src + 1$.
   - 2.3. If P^T_{now} is a local maxima of $g(i)$ then $tgt = tgt + 1$.
   - 2.4. If (src = 2) or (tgt = 2) or (V$(P^S_{start}, P^S_{now}) > \theta$) or (V$(P^T_{start}, P^T_{now}) > \theta$) then break.
   - 2.5. P^S_{now} = successor(P^S_{now}).
3. Approximate the segments from P^S_{start} to P^S_{now} on the source curve, P^T_{start} to P^T_{now} on the target curve by using cubic Bézier curves.
4. If ($P^S_{now} \neq P_{last}$) then go to step 1.

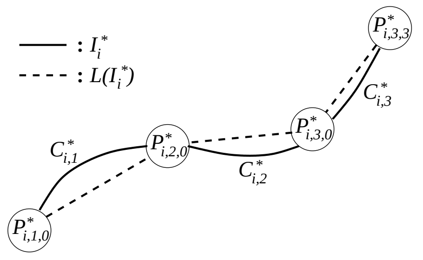


Figure 3. Example of an input curve.

Function V(A, B) (see step 2.4.) returns the angle between the tangent vectors at A and B. By checking the angle between the tangent vector at the starting points P^S_{start}, P^T_{start} and at the current points P^S_{now}, P^T_{now}, we ensure that the result of the subdivision can be approximated by using a cubic Bézier curve, especially when large segments of the curve are of the same angle costs. In our implementation, we set the value of θ to $\frac{3}{4}\pi$.

4 Feature interpolation

In morphing, linear interpolation is usually used for interpolating the features. However, it fails when the two shapes have different orientations. The feature interpolation problem becomes complicated since the user can specify features not only on the boundaries, but also inside or outside the boundaries. As a result, other shape interpolation methods such as [22, 23, 6, 3, 30] cannot be used directly in morphing because they only consider the interpolation between two object boundaries. Our proposed algorithm overcome this problem.

Before we describe our algorithm, we need to define some keywords and notations. Assume that there are n input curves. The input curves I_i^* (see Figure 3), (I_i^S or I_i^T), $(i = 1, ..., n)$ are then approximated using Bézier curves. We call these Bézier curves *"feature curves"*. Let m_i^* to be the number of Bézier curves used to approximate I_i^*. Then $I_i^* = \{C^*_{i,1}, ..., C^*_{i,m_i}\}$, where $C^*_{i,j}$ is a Bézier curve. Here, $C^S_{i,j}$ corresponds to $C^T_{i,j}$. Let $P^*_{i,j,0}, ..., P^*_{i,j,3}$ to be the control points of $C^*_{i,j}$. We define the polyline $L(I_i^*)$ as follows.

$$\begin{aligned} L(I_i^*) &= \{P^*_{i,1,0}, P^*_{i,2,0}, ..., P^*_{i,m_i-1,0}, P^*_{i,m_i,0}, P^*_{i,m_i,3}\} \\ &\quad \text{if } I_i^* \text{ is an open curve,} \\ L(I_i^*) &= \{P^*_{i,1,0}, P^*_{i,2,0}, ..., P^*_{i,m_i-1,0}, P^*_{i,m_i,0}\} \\ &\quad \text{if } I_i^* \text{ is a closed curve.} \end{aligned} \tag{5}$$

The idea of our approach is to represent the feature curves and the connections among them using a directed graph called *"dependency graph"*. The node of the dependency graph represents an end point of a feature curve. The

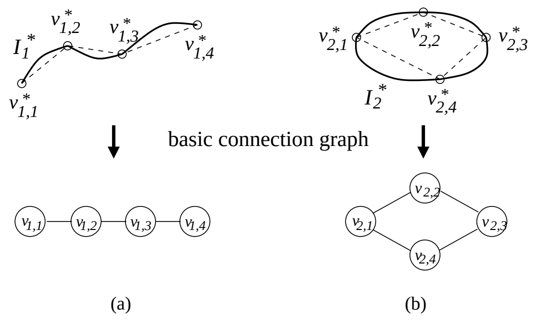


Figure 4. Example of basic connection graphs in the case of (a) open curve and (b) closed curve.

edge of the graph represents a directed line segment connecting its two nodes, and hence establishes a dependence relationship between nodes of the graph. We build the dependency graph for each of the two input images. The intermediate features are generated by applying the edge-angle blending technique [22] between these two graphs.

4.1 Preprocessing step

In the preprocessing step, we construct a dependency graph for the two input images. The dependency graph is constructed by generating a *"connection graph"* and then traversing the connection graph starting from the base vertex in a breadth-first manner and assigning a direction to each edge. At the same time, the intrinsic definitions of the edges are computed and stored in the nodes of the dependency graph.

4.1.1 Connection graph construction

Let γ_i to be the number of vertices in $L(I_i^*)$. We redefine $L(I_i^*) = \{v_{i,1}^*, ..., v_{i,\gamma_i}^*\}$. We can consider the polyline $L(I_i^*)$ as an undirected graph. We call this graph *"basic connection graph G_i"*. The nodes and the edges of graph G_i represent the vertices and the line segments in the polyline $L(I_i^*)$. We define $V(G_i) = \{v_{i,1}, ..., v_{i,\gamma_i}\}$ to be the set of G_i's nodes. Figure 4(a) shows an example of a basic connection graph when I_i^* is an open curve, whereas Figure 4(b) shows an example in the case of a closed curve. We call the edges in these examples *"feature edges"*, since these edges represent feature curves.

We build the connection graph by linking together the basic connection graphs that are located close to each other by putting edges among them. We call these edges *"non-feature edges"*. Before we describe the linking algorithm, we need to define the distance between two basic connection graphs. Let distance(A, B) returns the Euclidean distance between vertex A and vertex B. The distance between $L(I_i^*)$, $(i = 1, ..., n)$ and $L(I_k^*)$, $(k = 1, ..., n)$ is defined as

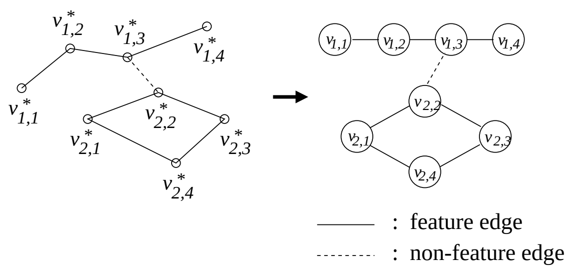


Figure 5. Connection graph by linking two basic connection graphs.

follows.

$$\text{distance\_line}(L(I_i^*), L(I_k^*)) = \min \text{distance}(v_{i,j}^*, v_{k,l}^*), \quad (j = 1, ..., \gamma_i), (l = 1, ..., \gamma_k), \tag{6}$$

where $v_{i,j}^* \in L(I_i^*)$, $v_{k,l}^* \in L(I_k^*)$. Since a basic connection graph G_i represents $L(I_i^S)$ and $L(I_i^T)$, the distance between two basic connection graphs G_i and G_k is

$$\text{distance\_BCG}(G_i, G_k) = \min(\text{distance\_line}(L(I_i^S), L(I_k^S)), \text{distance\_line}(L(I_i^T), L(I_k^T))). \tag{7}$$

The linking operation is as follows.

1. For $(i = 1, ..., n)$, $(j = i + 1, ..., n)$, set dist$[i, j]$ = dist$[j, i]$ = distance_BCG(G_i, G_j). Let dist$[i, j]$ = distance(v_{G_i}, v_{G_j}), where $v_{G_i} \in G_i$ and $v_{G_j} \in G_j$. If dist$[i, j] < \epsilon$ (ϵ: user specified constant), then put a non-feature edge between v_{G_i} and v_{G_j} and set connected$[i, j]$ = connected$[j, i]$ = True. Otherwise, set connected$[i, j]$ = connected$[j, i]$ = False.
2. $Q = \{G_1\}$, $R = \{G_2, ..., G_n\}$.
3. While (R is not empty)
   - 3.1. Search for (i, j), ($G_i \in Q$ and $G_j \in R$) such that dist$[i, j]$ is minimum.
   - 3.2. If (connected$[i, j]$ = False) then put a non-feature edge between v_{G_i} and v_{G_j}.
   - 3.3. $Q = Q \cup \{G_j\}$, $R = R - \{G_j\}$.

After we applied the above algorithm to the basic connection graphs, we obtained the connection graph, which guarantees that a path always exists from any arbitrary node to the other nodes. Figure 5 shows an example of a constructed connection graph.

4.1.2 Dependency graph construction

Figure 6 shows the dependency graph with respect to the connection graph in Figure 5. To build the dependency graph, we first create a root node for the base vertex. A vertex that minimizes $\|v_{i,j}^S - v_{i,j}^T\|, (i = 1, ..., n), (j =$

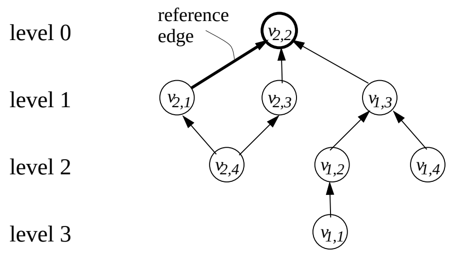


Figure 6. Dependency graph for the connection graph in Figure 5.

$1, ..., \gamma_i)$ is chosen as the base vertex. In Figure 5, we assume that $v_{2,2}$ is the base vertex. Next, one of the feature edges that attached to the base vertex is chosen as a reference edge. For the connection graph in Figure 5, $\overline{v_{2,2}v_{2,1}}$ is chosen as the reference edge. The intrinsic definition [22] of the reference edge are calculated with respect to the x-axis. The intrinsic definitions of the remaining edges which have the base vertex as their end point, $\overline{v_{2,2}v_{2,3}}$ and $\overline{v_{2,2}v_{1,3}}$, are calculated with respect to the reference edge.

We then traverse the connection graph starting from the nodes $(v_{2,1}, v_{2,3}, v_{1,3})$ that share edges with the base vertex in the breadth-first manner and assign a direction to each edge. If we visit node $v_{k,l}$ from node $v_{i,j}$, we store the intrinsic definition of $\overline{v_{i,j}v_{k,l}}$ in node $v_{k,l}$. For instance, in Figure 6, node $v_{1,2}$ holds the intrinsic definition of edge $\overline{v_{1,3}v_{1,2}}$ with respect to edge $\overline{v_{2,2}v_{1,3}}$. Note that a node can store more than one intrinsic definition and thus can have more than one dependency.

4.2 Intermediate features generation step

The intermediate features at time t are computed as follows. Let $v_{a,b}$ to be the base vertex. First, we compute the shape transition rate at each node. Section 4.2.1 discusses this in detail. After that, we compute the coordinates of the base vertex $v^I_{a,b}$ using the linear interpolation of $v^S_{a,b}$ and $v^T_{a,b}$ at a transition rate of SR$(v^S_{a,b}, t)$.

$$v^I_{a,b} = (1.0 - \text{SR}(v^S_{a,b}, t))v^S_{a,b} + \text{SR}(v^S_{a,b}, t)v^T_{a,b}. \qquad (8)$$

Here, SR$(v^S_{i,j}, t)$ returns the transition rate of node $v^S_{i,j}$ at time t. Then, we traverse the dependency graph, starting from the root (base vertex) in a breadth-first manner, and compute the coordinates of the nodes in the order visited. Suppose that the visited node $v_{i,j}$ has d dependencies (see Figure 7). We compute the new position of $v^I_{i,j}$ as follows.

$$v^I_{i,j} = \frac{\sum_{k=1}^{d} w_k v^I_{i,j,k}}{\sum_{l=1}^{d} w_l}. \qquad (9)$$

Here, $v^I_{i,j,k}$ $(k = 1, ..., d)$ are the new positions of $v_{i,j}$ with respect to each dependency. We calculate $v^I_{i,j,k}$ using the

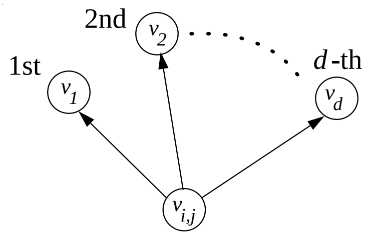


Figure 7. Node $v_{i,,j}$ with d dependencies.

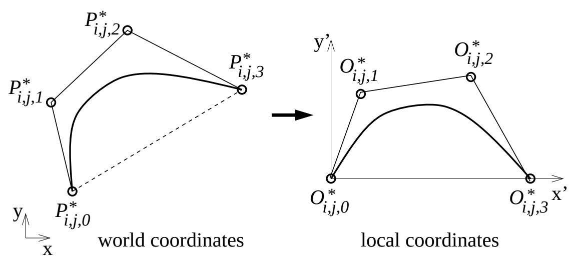


Figure 8. Conversion from the world coordinates to the local coordinates system.

edge-angle blending technique. w_k is the weight of the k-th dependency. To generate a smooth interpolation, edge with small changes on its length (the difference between the length of the edge in the source graph and in the target graph is small) should not suffer a great distortion on its length during the interpolation. Therefore, the weight of such edge should be large. Let, length$(\overline{v_{i,j}v_{k,l}})$ returns the length of edge $\overline{v_{i,j}v_{k,l}}$. We define w_k as follows.

$$w_k = \frac{1}{\beta + |\text{length}(\overline{v^S_k v^S_{i,j}}) - \text{length}(\overline{v^T_k v^T_{i,j}})|} \qquad (10)$$

β (a small value) is used to avoid division by zero. In our implementation, we set the value of β to 10^{-5}.

After we have calculated the coordinates of the nodes, we have the coordinates and the transition rates at the end points of the feature curves. Since the feature curves are cubic Bézier curves, the next task is to calculate the coordinates of the rest of their control points. At the preprocessing step, for each feature curve $C^*_{i,j}$, we define a local coordinate system with line $\overrightarrow{P^*_{i,j,0}P^*_{i,j,3}}$ as the x-axis and compute the local coordinates $O^*_{i,j,k}$ of $P^*_{i,j,k}$, $(k = 1, 2)$ with respect to this coordinate system (see Figure 8).

We calculate the coordinates of $P^I_{i,j,k}$, $(k = 1, 2)$ at the intermediate image as follows. First, we compute the transition rate α_k at $P^S_{i,j,k}$ using the linear interpolation of $SR(P^S_{i,j,0}, t)$ and $SR(P^S_{i,j,3}, t)$ at a rate of $k/3$. Then, we calculate the local coordinates $O^I_{i,j,k}$ using the linear interpolation of $O^S_{i,j,k}$ and $O^T_{i,j,k}$ at a rate of α_k. Finally, we transform $O^I_{i,j,k}$ into coordinates with respect to the world coordinates system.

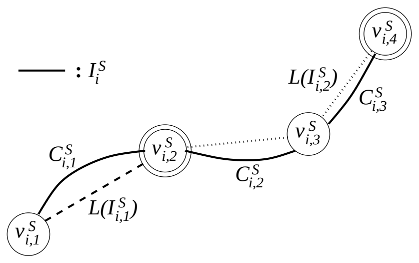


Figure 9. Subdivision of $L(I_i^S)$ into $L(I_{i,1}^S)$ and $L(I_{i,2}^S)$.

4.2.1 Features transition control

We use a Bézier curve to define a transition curve. To control the transition rates of the features, we allow the user to specify the transition curves at the points in $L(I_i^S)$. Assume that the transition rates of some points in $L(I_i^S)$ are specified. Then, by using these points, we can subdivide $L(I_i^S)$ into several polylines $L(I_{i,j}^S)$. If only one of the end points of $L(I_{i,j}^S)$ has its transition rate specified (for instance, $L(I_{i,1}^S)$ in Figure 9), then we set the transition rate at the other end point as the transition rate at this point.

The transition rates of the remaining points in $L(I_{i,j}^S)$ are computed using the linear interpolation of the transition rates at $L(I_{i,j}^S)$'s end points. Let the transition rates at the end points of $L(I_{i,j}^S)$ to be α_1 and α_2. Assume that $L(I_{i,j}^S)$ has $\gamma_{i,j}$ vertices. The transition rate at the k-th point ($k = 1, ..., \gamma_{i,j}$) is a linear interpolation of α_1 and α_2 at a rate of $k/\gamma_{i,j}$. If the user does not specify the transition rates for any of the node points, then we set the transition rates for all of the node points with a linear function.

5 Field morphing

In our morphing system, the field morphing technique [2] is used for the warp computation. Since the features of the images are represented using Bézier curves, we use pairs of Bézier curves to define the mapping between the input images. However, for computing the warp, we use extended feature curves, which are the extended version of Bézier curves.

5.1 Extended feature curve

Figure 10 shows an example of an extended feature curve. The extended feature curve C is constructed of three parts. The center segment is a Bézier curve C_c and attached to C_c's end points are two line segments C_h and C_t. The gradients of these line segments are the same as the gradients at the end points of the Bézier curve.

Each pixel has (u, v) coordinates with respect to each extended feature curve. We define u as a parameter on the extended feature curve, and v as a signed distance from the pixel to the extended feature curve. Let P_0,...,P_3 be the control points of Bézier curve C_c. Assume that length(C_c) represents the length of Bézier curve C_c. The u coordinate of any pixel X is computed by first projecting X onto the extended feature curve, thus yielding a point X_p. If X_p is on C_c, the value of u is such that $C_c(u) = \sum_{i=0}^{3} P_i B_3^i(u) = X_p$. When X_p is on C_h, $u = -(\frac{\|P_0 X_p\|}{\text{length}(C_c)})$. In the case when X_p is on C_t, $u = 1.0 + (\frac{\|P_3 X_p\|}{\text{length}(C_c)})$.

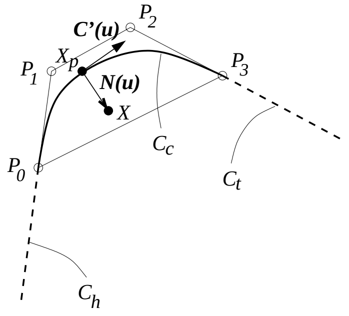


Figure 10. Example of an extended feature curve.

We compute v as follows. Let d to be the distance between pixel X and the extended feature curve. The tangent vector on X_p is $C'(u)$ and the normal vector is $N(u)(N(u) = X - X_p)$. We define $z = Z(C'(u), N(u))$. If $z > 0.0$ then $v = d$. When $z < 0.0$ then $v = -d$. If $z = 0.0$ then $v = 0.0$ since X is located on the extended feature curve.

5.2 Field morphing algorithm

Field morphing uses an inverse mapping approach to warp the source and the target images. In inverse mapping, for every pixel in the intermediate image, we compute its corresponding pixels in the two input images. Thus, we guarantee that every pixel in the intermediate image gets set to an appropriate value.

Before we warp the source and the target images, we use the method described in Section 4 to compute the feature curves in the intermediate image. These feature curves are then converted to extended feature curves. We use these extended feature curves to define the mapping from one image to the other. Figure 11 shows the mapping computation. The complete warping algorithm can be found in [2].

In the warp computation, we have to calculate the (u, v) coordinates for every pixel with respect to each extended feature curve. Therefore, we have to compute the projection from a pixel to a Bézier curve. We employ the Bézier Clipping method [16] to solve this problem (see [18]). Because we use curves, there is a possibility that we can get more than one closest point for a single curve. Let $X_{p_1}, ..., X_{p_n}$

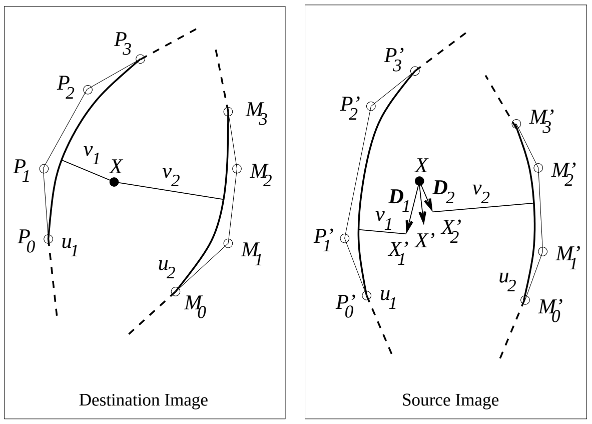


Figure 11. Warp computation in field morphing.

to be the closest points. Assume (u_i, v_i) to be the (u, v) coordinates of X_{p_i}. We solve this problem by recording the value of the parameter u, u_{last}, which belongs to the last pixel processed, and choose the point which minimizes $|u_j - u_{last}|, (j = 1, ..., n)$ as the closest point. To accelerate the warp computation, we adopted an adaptive computation method similar to [11].

After we warped the source and the target images, the next thing to do is to blend the colors of the warped images to produce the final intermediate image. We allow the user to control the transition rates of the colors (see Section 5). Here, we assume that we know the color transition rate for each pixel in the source image. For every pixel X in the intermediate image at time t, let X_s and X_t to be the corresponding pixels in the source and the target images, respectively. The color of X is determined by using the following equation.

$$\begin{aligned}\text{Inter}(X, t) = (1.0 - \text{CR}(X_s, t))\text{Src}(X_s) + \\ \text{CR}(X_s, t)\text{Tgt}(X_t). \qquad (11)\end{aligned}$$

Here, $CR(X_s, t)$ returns the color transition rate at time t of pixel X_s in the source image.

5.3 Color transition control

The user controls the color transition rates by specifying color transition primitives on the source image and designating a transition curve for each primitive. We use points and curves as the color transition primitives.

For each pixel X in the intermediate image at time t, let X_s to be the corresponding pixel in the source image. To determine the color at pixel X, we need to know the color transition rate of pixel X_s (see Equation 11). The color transition rate at time t of pixel X_s is computed by interpolating the transition rates at time t of all primitives. Assume that there are n color transition primitives CP_1,...,CP_n. Let $CP_i(t)$ to be a function that returns the transition rate at time t of the i-th primitive. First, we compute the weight w_i of each primitive CP_i by

$$w_i = \left(\frac{1.0}{p + \text{distance}(X_s, CP_i)^q} \right)^r, \qquad (12)$$

where distance(X_s, CP_i) returns the distance between pixel X_s and primitive CP_i. p, q, and r are user specified parameters. In our tests, we use $p = 1.0$, $q = 2.0$, and $r = 2.0$. Then, we use the following equation to calculate the color transition rate of pixel X_s at time t.

$$\text{CR}(X_s, t) = \frac{\sum_{i=1}^{n} w_i.CP_i(t)}{\sum_{j=1}^{n} w_j}. \qquad (13)$$

Since the weight w_i of each primitive at a pixel does not depend on time t, we can calculate look-up tables of the weights in advance and use them when generating the intermediate images. As a result, we can compute the color transition rates very fast. When there are n color transition primitives, we create n look-up tables, i.e. one for each primitive. The weight of the i-th primitive at pixel (x, y) is stored in the (x, y) element of the i-th look-up table. To accelerate the computation of the look-up tables, we adaptively subdivide the image plane until the error is below a given threshold, and then we use bilinear interpolation to calculate the weights.

6 Examples

In our experiment, we use SGI O2 (MIPS R12000 270Mhz) to perform the computation. Figures 12 - 14 show the examples of shape interpolation. In each example, Figure (a) shows the result of our algorithm, whereas Figure (b) shows the result when we did not consider the non-feature edges in the connection graph (i.e. simply applying the edge-angle blending technique between each pair of corresponding features in the two images). We also marked the portions of the intermediate shapes which exhibit artifacts by using circles.

Figure 12 shows a shape interpolation of a woman tilting her face. The orientations of the woman's face in the source and in the target shapes are only slightly different. As a result, when we performed the interpolation independently for each pair of corresponding features, the results did not exhibit many artifacts. However, we still can see that some features overlapped with each other at the woman's hair. On the other hand, we did not encounter this problem when we used our algorithm.

The next two examples clearly show the advantages of our algorithm. In Figure 13, the gorilla and the bear have

Table 1. Image sizes, numbers of input curves and feature curves, and computation times of the morphing examples.

Figure	Size	#Input curves	#Feature curves	CPU time (secs)
15	270×320	7	46	1.4
16	450×505	28	103	6.7
17	400×284	5	33	1.3

different orientations. The example in Figure 14 is an interpolation between two cartoon characters. The shapes contain many features that are not connected to each other. Moreover, several parts of the source and the target shapes have different orientations. By combining all these features into one dependency graph, we could produce a smooth interpolation sequence.

The examples demonstrate that our method works well even when there are many unconnected features and when the source and the target shapes have different orientations. The feature interpolation process can be performed in short computation time. The preprocessing step for the examples in Figures 12 and 13 can be computed in about 0.01 seconds. For the example in Figure 14, the preprocessing step took about 0.02 seconds of computation time. The animation sequence of the features can be generated in real time.

Examples of morphing are shown in Figures 15 - 16 (see Color Plate). In each example, Figure (a) shows the source and the target images and their associated feature curves and Figure (b) shows the morphing sequence (from left-to-right and top-to-bottom). Figure 15 is an example of a morphing from a gorilla to a bear. Here, we have to put additional feature curves in order to reduce the ghost. Still, some of the intermediate images exhibit the ghost problem. For instance, between the rear legs in the 2nd through the 6th images of the morphing sequence. The ghost between the rear legs are caused by the fact that there are spaces between the gorilla's rear legs, whereas there is no space between the bear's rear legs. Figure 16 shows a morphing between two cartoon characters. In the morphing sequence, the colors were set to change from the legs towards the head. Similar to Figure 15, we put some additional feature curves to avoid the ghost problem.

Figure 17 (see Color Plate) shows a series of images that transform a butterfly to another type of butterfly. In Figure 17(a), we controlled the color transition rates such that the colors change gradually from the bottom towards the top of the butterfly. We also varied the shape transition rates so that the lower wings transform faster than the upper wings. For comparison, we generated a morphing sequence between the source and the target butterflies using the same transition rates everywhere (Figure 17(b)). It is obvious that by allowing the transition rates to vary across the image, we can generate a more interesting animation.

The image size, the number of input curves and feature curves, and the computation time for each example are shown in Table 1. By computing the correspondence between the features automatically, we can reduce the user's workload on specifying corresponding input curves. This is cause by the fact that the user can specify two long curves and the correspondence between the similar areas in the two curves are computed automatically. For the examples in Figures 15 and 17, the user specified only few input curves. In the case of Figure 16, the user specified 23 input curves on the features of the objects (five additional curves are used to reduce the ghost problem). This number is reasonable considering the shapes and the textures of the objects in the source and the target images. However, sometimes the automatic correspondence computation yields a result which differs from what the user expects. To solve this problem, the user specifies the correspondence at few points and then repeats the correspondence computation.

7 Conclusions and discussions

In this paper, we have proposed a method for computing the correspondence (at pixel level) between two curves and a method for generating the intermediate feature curves for the morphing computation. To establish the correspondence between two input curves, we developed an automatic algorithm based on minimizing a cost function. The method tends to set correspondence between similar areas in the two curves. As a result, feature specification process becomes easier. For the feature interpolation, we developed an algorithm that uses a directed graph to represent the relationships among the feature curves. For the interpolation computation, we employed the edge-angle blending technique [22]. From experimental results, the algorithm produces a smooth transformation even when the user specified many unconnected features. However, we have not been able to prove that the algorithm can prevent the intersections between features during the interpolation.

We have also presented methods for controlling the transition rates of shape and color. The shape transition control is a part of the feature interpolation. For controlling the color transition, the user specifies the color transition primitives (points, curves) and assigns a transition curve for each primitive. Each primitive has a field of influence, and by interpolating the transition rates at the primitives, we compute the color transition rates of the remaining pixels.

We have implemented the proposed algorithms in our morphing system. This system is based on the field morphing technique [2]. Since the features in the images are represented by using Bézier curves, we use pairs of Bézier curves to define a mapping between the source and the target images. In our system, the user uses a tablet for drawing the input curves. To make this operation more comfortable, we are planning to adopt an image segmentation tool, for instance intelligent scissors [15].

The proposed feature correspondence and shape interpolation algorithms can also be used in conjunction with other morphing algorithms such as the radial basis function technique [8, 13] and the multilevel free form deformation technique [9]. However, the shape interpolation algorithm still has a room for improvement. Since, we employ the edge-angle blending technique for interpolating the dependency graphs, it mights distort the area of the intermediate shape. In our future research, we want to solve this problem by developing a multi-resolution shape interpolation algorithm for interpolating the dependency graphs.

References

[1] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proc. of SIGGRAPH 2000*, 2000.

[2] T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (Proc. of SIGGRAPH'92)*, Vol. 26, No. 2, pages 35-42, 1992.

[3] E. Carmel and D. Cohen-Or. Warp-guided object-space morphing. In *The Visual Computer*, Vol. 13, pages 465-478, 1997.

[4] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours. In *ACM of Communications*, Vol. 20, No. 10, pages 693-702, 1977.

[5] P. Gao and T. Sederberg. A work minimization approach to image morphing. In *The Visual Computer*, Vol. 14, pages 390-400, 1998.

[6] E. Goldstein and C. Gotsman. Polygon morphing using a multiresolution representation. In *Proc. of Graphics Interface'95*, pages 247-254, 1995.

[7] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. In *International Journal of Computer Vision*, pages 321-331, 1988.

[8] S. Lee, K.Y. Chwa, J. Hahn, and S.Y.Shin. Image morphing using deformable surfaces. In *Proc. of Computer Animation*, pages 31-39, 1994.

[9] S. Lee, K.Y. Chwa, S.Y. Shin, and G. Wolberg. Image metamorphosis using snakes and free-form deformations. In *Proc. of SIGGRAPH'95*, pages 439-448, 1995.

[10] S. Lee, K.Y. Chwa, J. Hahn, and S.Y. Shin. Image morphing using deformation techniques. In *The Journal of Visualization and Computer Animation*, Vol. 7, No. 1, pages 3-23, 1996.

[11] T. Lee, Y.C. Lin, L. Lin, and Y. Sun. Fast feature-based metamorphosis and operator design. In *Computer Graphics Forum (EUROGRAPHICS'98)*, Vol. 17, No. 3, pages 15-22, 1998.

[12] A. Lerios, C. Garfinkle, and M. Levoy. Feature-based volume metamorphosis. In *Proc. of SIGGRAPH'95*, pages 449-456, 1995.

[13] P. Litwinowicz and L. Williams. Animating images with drawings. In *Proc. of SIGGRAPH'94*, pages 409-412, 1994.

[14] D. Moore and J. Warren. Least-squares approximations to Bezier curves and surfaces. In *Graphics Gems II*, pages 406-411, 1991.

[15] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *Proc. of SIGGRAPH'95*, pages 191-198, 1995.

[16] T. Nishita, T. Sederberg, M. Kakimoto. Ray tracing trimmed rational surface patches. In *Computer Graphics (Proc. of SIGGRAPH'90)*, Vol. 24, No. 4, pages 337-345, 1990.

[17] T. Nishita, K. Fujii, and E. Nakamae. Metamorphosis using Bezier clipping. In *Proc. of Pacific Graphics'93*, pages 162-173, 1993.

[18] T. Nishita and H. Johan. A Scan Line Algorithm for Rendering Curved Tubular Objects. In *Proc. of Pacific Graphics'99*, pages 92-101, 1999.

[19] D. Ruprecht and H. Muller. Image warping with scattered data interpolation. In *IEEE Computer Graphics and Application*, Vol. 15, pages 37-43, 1995.

[20] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *Computer Graphics (Proc. of SIGGRAPH'86)*, Vol. 20, No. 4, pages 151-160, 1988.

[21] T. Sederberg and E. Greenwood. A physically based approach to 2D shape blending. In *Computer Graphics (Proc. of SIGGRAPH'92)*, Vol. 26, No. 2, pages 25-34, 1992.

[22] T. Sederberg, P. Gao, G. Wang, and H. Mu. 2D shape blending: An intrinsic solution to the vertex path problem. In *Proc. of SIGGRAPH'93*, pages 15-18, 1993.

[23] M. Shapira and A. Rappoport. Shape blending using the star-skeleton representation. In *IEEE Computer Graphics and Application*, Vol. 15, pages 44-51, 1995.

[24] Y. Shinagawa and T.L. Kunii. Unconstrained automatic image matching using multiresolutional critical-point filters. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 9, pages 994-1010, 1998.

[25] S. Cohen, G. Elber, and R. Yehuda. Matching of freeform curves. In *Computer Aided Design*, Vol. 29, No. 5, pages 369-378, 1997.

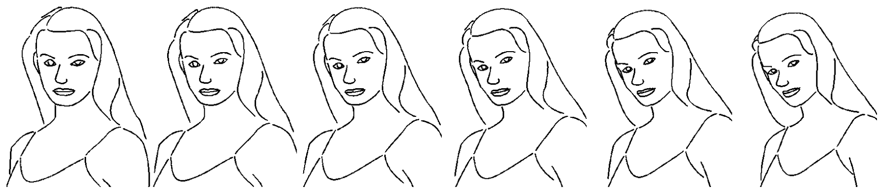
[26] R.C. Tam and A. Fournier. Image interpolation using unions of sphere. In *The Visual Computer*, Vol. 14, pages 401-414, 1998.

[27] A. Tal and G. Elber. Image morphing with feature preserving texture. In *Computer Graphics Forum (EUROGRAPHICS'99)*, Vol. 18, No. 3, pages 339-348, 1999.

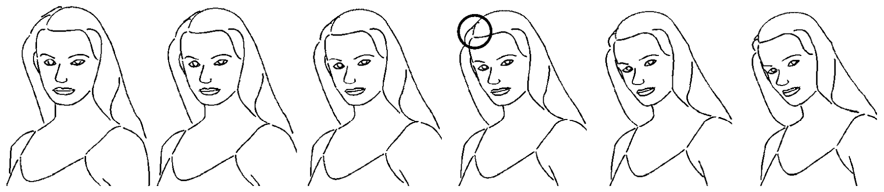
[28] G. Wolberg. *Digital image warping*. IEEE Computer Society Press, 1990.

[29] G. Wolberg. Image morphing: a survey. In *The Visual Computer*, Vol. 14, pages 360-372, 1998.

[30] Y. Zhang and Y. Huang. Wavelet shape blending. In *The Visual Computer*, Vol. 16, pages 106-115, 2000.

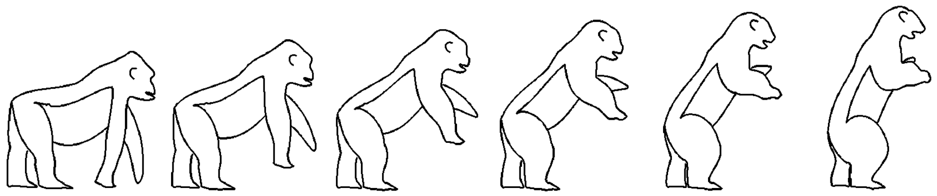


(a)

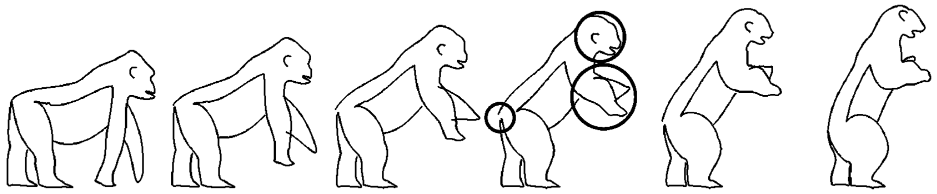


(b)

Figure 12. Woman tilting her face.

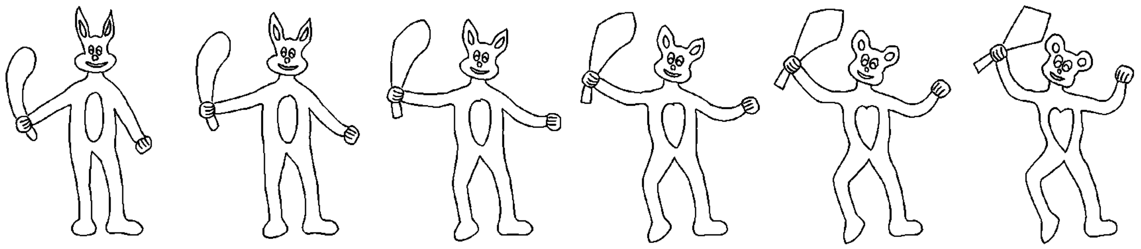


(a)

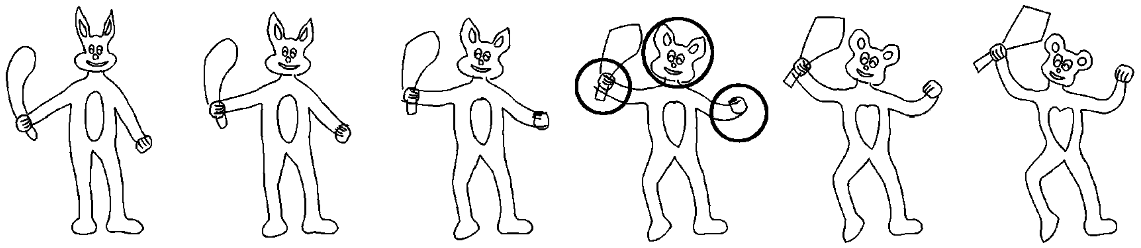


(b)

Figure 13. Shape interpolation from a gorilla to a bear.



(a)



(b)

Figure 14. Shape interpolation between two cartoon characters.