CS148 Final Report

Feature-Based Image Metamorphosis

Josh Israel, Peng Hui How, Simon Ayzman
{joshuai, phow1, simonayzman}@stanford.edu

August 14, 2014

1 Overview

Our project focuses on morphing between two videos, using an algorithm outlined in the paper Feature-Based Image Metamorphosis[1]. It describes a method of transitioning between two images, using a series of pre-defined, common features (here, lines) in each image to determine how much each pixel should be weighted towards either image.

2 Feature Adder

The *featureAdder* program is a robust interactive image processing program that allows users to specify common features in both the source and destination images. Specifically, the *featureAdder* program takes in the source and destination image (`.png`) files and outputs their respective feature (`.feat`) files. This output is a simple text format containing coordinates for each line segment, which can then be read in by the image morphing program.

In some sense, *featureAdder* is an important bottleneck in the production of morphed images/videos. Once the later programs are working correctly, they can be run automatically without the user's control. However, *featureAdder* needs to be a user-friendly and productive enough tool that makes the process of adding common feature simple and constructive; the users' needs and patterns need to be taken into account. For example, the program is capable of undoing mistakes and/or improperly placed features in the order that they were added and as far back as necessary. Also, *featureAdder* is capable of opening/editing previously made feature files, removing the need to recreate a feature file from scratch every time for the same image. The undo feature is retained even for newly opened files, but the order in which the features were added is not preserved; the program simply alternates between undoing source and destination image features. Lastly, *featureAdder* is capable of taking screenshots of the current work environment. Once the features are written into `.feat` files, they will then be fed into and parsed by the *imageMorph* program introduced in the next section.

The following is a screenshot of the program in use:

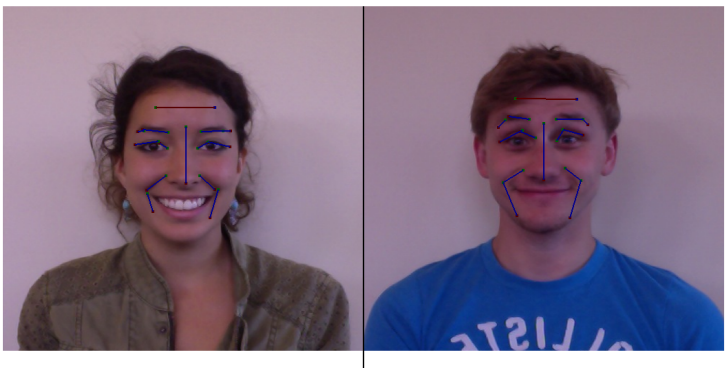

Figure 1: Feature adder in use, the left and right images are source and destination images respectively

3 Image Morph

Now, given an ordered pair of images, one as source and the other as destination, denoted as I_S and I_D respectively, with the corresponding pairs of features specified, the *imageMorph* program generates a video made up of

the the intermediate frames. Our program takes in the following input arguments:

1. The source and destination feature (`.feat`) files
2. The source and destination image (`.png`) files
3. The common prefix of the name of the intermediate images
4. The number[1] of intermediate frames to be generated
5. The choice of easing function[2]

Our program generates the intermediate frames using a classical metamorphosis algorithm that appears in Thaddeus Beier's SIGGRAPH paper[1] titled 'Feature-Based Image Metamorphosis' published in 1992.

Notably, each of the intermediate frames is specified by a time $t \in [0, 1]$, where larger t indicates more complete transformation - namely, one that has higher similarity to the destination image rather than the source image (and vice versa). Suppose there are n intermediate frames, then the time progress t of the m^{th} frame is a function of m and n - in fact, it is known as the easing function. There is nothing special about an easing function; given a fixed n, it just has to satisfy the following properties:

1. t is a nondecreasing function with respect to m
2. $t(0, n) = 0$, $t(n, n) = 1$

In particular, our program offers the options of linear and sinusoidal easing functions to the users, in which $t(m, n) = \frac{m}{n}$ and $t(m, n) = \frac{1}{2} * ((\sin(\frac{m}{n} - \frac{1}{2}) * \pi) + 1)$ respectively.

Regardless of the easing function, the final result of each intermediate image is the linear interpolation between the morphed source image and the morphed destination image, carrying weights of $1 - t$ and t respectively at time t. To clarify, the morphed source image at time t is the result of field morph from I_S to I_D at time t, whereas the morphed destination image is the result of field morph from I_D to I_S at time $1 - t$. The notion of field morph will be made clear shortly in the following paragraph.

So, what exactly is field morphing? As suggested by its name, it is a morphing technique based on a two-dimensional control primitives. More precisely, given a source image and two paired sets of image features (source and destination), one can compute the color of each pixel of the destination image under field morph pixel by pixel. Note that the features here form the field. Our algorithm performs reverse mapping. For each pixel X in the destination image, we find its corresponding pixel X' in the source image.

Geometrically, we want the neighborhoods of the features to be preserved - the distances between a pixel and each of the features ought to remain constant, i.e. $d(F_j, X) = d(F'_j, X')$ [3] for each pixel X and each feature F_j. Intuitively, this suggests that farther lines would play a lighter role in the morphism of a point (the heuristic will be explained in detail in the following sections). Essentially, featured-based metamorphosis is just a transformation of a collection of points with multiple pairs of directed line segments (simply known as lines for convenience purpose), each of which represents a specified feature.

With the background information explained, we shall now study the mathematics involved in this algorithm. To understand the transformation with multiple pairs of lines, we ought to first understand the transformation with a single pair of line. As suggested in the previous paragraph, transformation with multiple pairs of lines is none other than combining the transformation with individual pairs of lines, each with different assigned weight (this is entirely dependent on the proximity of the feature line to the pixel).

3.1 Transformation with a Single Pair of Line

Figure 2: Single line pair (left to right: destination, source)

The transformation is straightforward when there is only a single source-destination line pair (F', F). In this case, for each pixel X in the destination image, all we need to do is to separate X into the sum of its parallel component X_u and perpendicular component X_v with respect to the feature line F, i.e., $X = X_u + X_v$ where $X_u \parallel F$ and $X_v \perp F$. In fact, by labeling the feature line with a directed pair (P, Q), we can always write

$$X_u = u \cdot (Q - P), X_v = v \cdot \frac{(Q - P)^{\perp}}{||Q - P||}$$

[1] The larger the number, the smoother the interpolation

[2] only linear and sinusoidal functions for now

[3] Euclidean distance between a feature (essentially as a finite line segment) and a point

for some real numbers u, v so that

$$X = P + X_u + X_v$$

. Now, knowing the value u and v, we can compute X' easily. To find X', we note that

$$X'_u = u \cdot (Q' - P'), X'_v = v \cdot \frac{(Q' - P')^\perp}{||Q' - P'||}$$

so that

$$X' = P' + X'_u + X'_v = P' + u \cdot (Q' - P') + v \cdot \frac{(Q' - P')^\perp}{||Q' - P'||}$$

.

In terms of pseudocode,

```
For each pixel X in the destination image
    find the corresponding u,v
    find the X' in the source image for that u,v
    destinationImage(X) = sourceImage(X')
```

3.2 Transformation with Multiple Pairs of Lines

When multiple pairs of lines (i.e. more than one features) are involved, a weighting of the coordination transformation for each line is performed. The resultant transformation is just a weighted linear combination of the transformations due to each individual feature. In fact, the following is the heuristic of the weight assignment of our algorithm:

$$weight = \left(\frac{length^p}{a + dist} \right)^b$$

where $length$ denotes the length of the feature line segment, $dist$ denotes the shortest distance from current pixel of interest to the feature, and a, b, p are parameters to be adjusted. Note that the nearer the point is from a line, the heavier the weight is.

In terms of pseudocode,

```
For each pixel X in the destination
    DSUM = (0,0)
    weightsum = 0
    For each line PiQi
        calculate u,v based on PiQi
        calculate X'i based on u,v and Pi'Qi'
        calculate displacement Di = Xi' - Xi for this line
        dist = shortest distance from X to PiQi
        weight = pow((pow(length, p) / (a + dist)), b)
        DSUM += Di * weight
        weightsum += weight
    X' = X + DSUM / weightsum
    destinationImage(X) = sourceImage(X')
```

Here, specially note that since each P_iQ_i is a finite line segment, if $u < 0$ then the distance from X to P_iQ_i will just be the distance between X and P_i; similarly, if $u > 1$, then the distance is just that of between X and Q_i.

3.3 Screenshots of Intermediate Frames

The following are some of the screenshots of the intermediate frames obtained. Here, the linear easing function is employed. Here are two rather successful morphing attempts:

Figure 3: Evelyn's Frown to Evelyn's Smile

Figure 4: Evelyn's Smile to Simon's Smile

The following series of images demonstrates our attempt to transform a beaver to a human being with feature-based metamorphosis. Due to the huge disparity between the geometrical features of the source and destination images (note: the ears), we observe aliasing effects in the intermediate frames which is especially obvious halfway through the metamorphosis process. Nonetheless, we still observe decent transformation for their noses.

Figure 5: Beaver Transformed to Human

3.4 Performance

The generation of a single frame takes about 20 seconds. To put it in a larger perspective, generate a sequence of intermediate images with 45 frames (our usual practice) takes around 15 minutes.

3.5 Techniques to Achieve Relatively Seamless Transformations

We noticed a few techniques that smoothen the interpolation between two images with high successful rate:

- Preprocess the images so that both figures are centered
- Align the prominent features, especially nose, mouth, eyes, and jawlines
- The subjects in the source and destination images should have nearly identical sizes, except under the special circumstances where deliberate contraction or expansion is intended

4 Auto Features

As mentioned earlier, one of the major pain points of using the image morph algorithm is that is that it requires a very tedious, manual process of creating all the feature lines. So, something we wanted to experiment with was whether it's possible to automatically generate the features. Instead of drawing all the features by hand, imagine if there was a programmatic preprocessing step to the morphing algorithm that produced all the features needed.

We approached this with facial detection. The idea is to pick out features like eyes, ears, nose, mouth, etc., mimicking the major things typically outlined during the manual process. We did this using Viola-Jones object detection[2], which is built into OpenCV out-of-the-box as "Haar Feature-based Cascade Classifiers"[3]. The strategy was to take the detection outputs (rectangles), and turn them into features that could be fed into *imageMorph*. To achieve this, we simply created feature lines from the top and left side of the rectangle.

Unfortunately, the approach did not succeed. It was very difficult to find images where detection worked on the same parts of the face. For example, one image might have the eyes, nose, and ears, while the other would detect the mouth instead of the ears. Further, even the same aspects were detected, the enclosing rectangles were wildly different. The best we could do are the two images below.

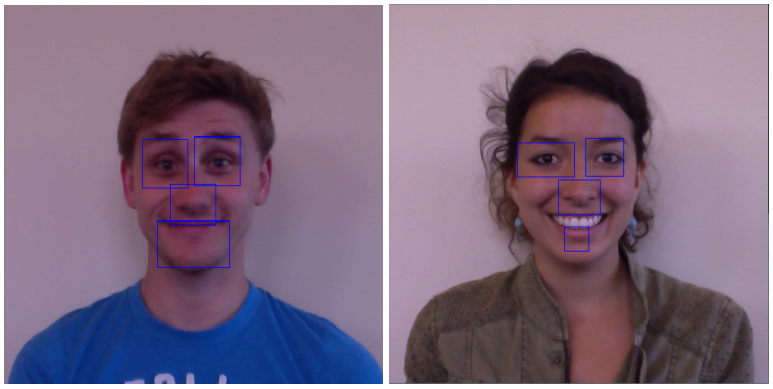

Figure 6: Features selected with the facial detection approach (from left to right: source, destination)

This created an image morph with frames like those below, which is clearly undesirable

Figure 7: Intermediate frames of the metamorphosis above

Future work could perhaps investigate whther training the classifiers specifically for the kind of pictures being taken improves the results. However, this would mitigate most (or all) of the value of *auto*-feature. Another possible tact could be to rely on the assumption that the two faces have to be relatively aligned for the image morphing to work, and do some kind of pre-averaging or weighting between the analogous detected rectangles.

5 Feature-Based Video Metamorphosis

Video metamorphosis extends directly from image metamorphosis in that the common feature identification is done between each of respective frames of the videos. Another program was written, called *frameExtractor*, that splits up a video into a series of frames that will then be used for this metamorphosis. In order for the metamorphosis to be successful, both videos need to have the same action take place so that as an action begins in the source video, it morphs into the completion of that action in the destination video. Key frames need to be identified between the two videos to ensure that any variations in the timing/speed of the actions still play out smoothly in the final video. Then, features are added to each keyframe–source and destination–and *imageMorph* is run on the resulting feature files. An appropriate intermediate frame is selected from the generated frames such that it fits accurately as the current moment in the overall video. This feature addition and subsequent frame selection is repeated for all the keyframes. Finally, all the selected morphed frames are compiled together and combined into an overall video using another program called *combiner*.

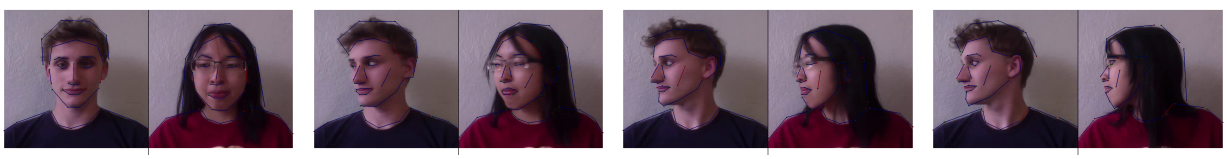

Figure 8: Frames extracted from source and destination videos (from left to right: source, destination)

Below are the results of the this animated morphing

Figure 9: Video metamorphosis

6 Relevance Outside The Field of Computer Science

The most obvious example is the aforementioned music video. The algorithm produces an effect that simply looks very interesting, especially in video form.

7 Remark About the Teamwork

Since we are doing a team project with three group members, we hereby report the task separation between the group members:

- Josh: Video-processing tools and auto-feature
- Peng Hui: Image morphing algorithm
- Simon: Feature adder and video morphing

8 Acknowledgement

We would like to express our utmost gratitude towards Katherine, and all the teaching assistants of CS 148 who provided valuable feedback to the project. We would also like to thank all our models that appeared in our report, without them, we would not be able to experiment on the quality of our metamorphosis program.

References

[1] Thaddeus Beier, *Feature-Based Image Metamorphosis*. Silicon Graphics Computer, Mountain View, California, 1992.

[2] Viola, P. and Jones, M. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition.*

[3] Bradski, G., *OpenCV*, opencv.org.

[4] Michael Jackson, *Black or White*, http://youtu.be/F2AitTPI5U0?t=5m27s