Implementation of Morphing algorithm with Feature-based Image metamorphosis

Srinath Setty Swati Rallapalli
{srinath, swati}@cs.utexas.edu
The University of Texas at Austin

1 Introduction

In this project, we built a tool based on the algorithm described by Beier and Neely in Feature-Based Image Metamorphosis [2]. *Morphing* is an image processing technique to progressively transition from one graphical object into another. This visual technique can be used to create animations where one object gets transformed into another over a period of time or transform the image of one person into another person's image.

The morphing algorithm in feature-based metamorphosis applies a combination of *Image warping* and *Cross-dissolving*. While cross-dissolving is simple, image warping is accomplished by giving control to the user. An user specifies the corresponding features on the source and destination images. This will be an input to the morphing algorithm. With such a morphing algorithm, the first image is progressively distorted and disappears, while the second image starts totally distorted but the distortion reduces gradually and fades in. In simple words, the result of this morphing algorithm looks like the first image at the start, at every point in the time line closer to the source, it has more influence from the source image, at every point in the time line closer to the destination image, it has more influence from the destination image and at the end, it is the destination image.

2 Algorithms

For completeness, we are writing the equations and algorithms given in [2] here.

2.1 Algorithm with one pair of lines

In this setup, one line is given for source image, and another for the destination image that specify the corresponding features in the source and the destination image. For e.g., for facial morphing (i.e., when the source and destination images are faces), one could specify a line along the forehead for the source and destination images.

"

$$u = \frac{(X-P).(Q-P)}{||Q-P||^2}$$

$$v = \frac{(X-P).Perpendicular(Q-P)}{||Q-P||^2}$$

$$X' = P' + u.(Q'-P') + \frac{v.Perpendicular(Q'-P')}{||Q-P||^2}$$

where *Perpendicular*() returns the vector perpendicular to, and the same length as, the input vector. The value *u* is the position along the line, and *v* is the distance from the line. The value *u* goes from 0 to 1 as the pixel moves from *P* to *Q*, and is less than 0 or greater than 1 outside that range. The value for v is the perpendicular distance in pixels from the line." [2]

Algorithm 1 Transformation with a single line (taken from [2])

```
1. for each pixel $X$ in the destination image
2.     find the corresponding $u$ and $v$
3.     find the $X'$ in the source image for that $u$ and $v$
4.     $destinationImage(X) = sourceImage(X')$
```

2.2 Algorithm with multiple lines

If the user specifies multiple lines, then the morphing algorithm can produce more visually appealing artifacts. We set the parameters given in [2] to the following values: $a = 300$, $b = 1$, and $p = 0$.

3 Implementation

Our implementation is about 3200 lines of C code, which used many of the pieces of the Impressionist project [1] like bitmap handling, fltk application etc.

We implemented a simple user interface to load source and destination images, allow user to specify control lines on them. The tool then allows the user to run the morphing algorithm with that input and it generates intermediate images as `bmp` files. One could use a command like `convert` to create a video out of these image files. The result would be a video that shows the morphing for the user specified images and control lines.

We also enhanced the user interface to allow users to use the grid lines to make better decisions about the control lines. Figure 1 shows a screen shot of the UI with grid lines and control lines.

3.1 Source code

The source code we developed for this project can be accessed from `http://www.cs.utexas.edu/users/srinath/morphing.tar.gz`

3.2 Artifact

We used the tool to create an artifact and can be viewed at `http://www.youtube.com/watch?v=PSa5bh_rbKI`

Algorithm 2 Transformation with multiple lines (taken from [2])

```
1.  for each pixel $X$ in the destination image
2.      $DSUM = (0,0)$
3.      $weightsum = 0$
4.      for each line $P_iQ_i$
5.          calculate $u$, $v$ based on $P_iQ_i$
6.          calculate $X'_i$ based on $u$, $v$, and $P'_iQ'_i$
7.          calculate displacement $D_i = X'_i - X_i$ for this line
8.          compute distance, $d =$ shortest distance from $X$ to $P_iQ_i$
9.          compute weight, $w = (\frac{length^p}{a+d})^b$
10.         $DSUM = DSUM + D_i \times w$
11.         $weightsum = weightSum + w$
12.     $X' = X + \frac{DSUM}{weightsum}$
13.     $destinationImage(X) = sourceImage(X')$
```

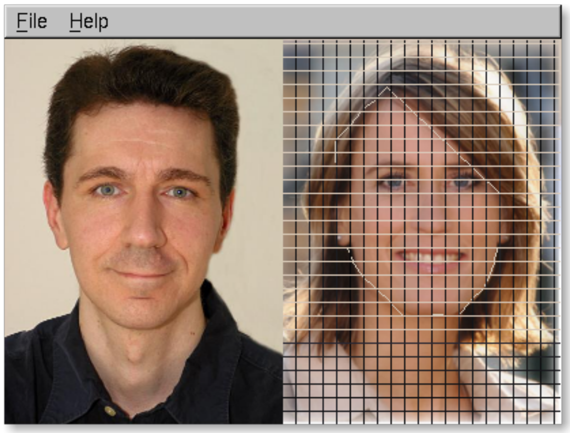

Figure 1—User Interface showing the grid lines and user defined lines to extract features

3.3 Known bug

The lines disappear from the left side window because of some unknown bug in the implementation of the user interface.

4 Acknowledgments

We are grateful to the developers of the Impressionist code base [1]. It helped us to start from the Impressionist code base to quickly build an fltk application. Thanks to Prof. Don Fussell and Peter Djeu for helping us scope this project.

References

[1] Impressionist. `http://userweb.cs.utexas.edu/~fussell/courses/cs384g/projects/impressionist/impressionist.tgz`.

[2] T. Beier and S. Neely. Feature-based image metamorphosis. *SIGGRAPH Comput. Graph.*, 26(2):35–42, 1992.