An Implementation of Feature-Based Image Metamorphosis

Term Project

Submitted in partial fulfillment
of the requirements of
Multimedia Computing - CS F401

By

ISHAN SHARMA
ID No. 2016B2A70773P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI,
PILANI CAMPUS

March - April 2021

Abstract

In the world of computer graphics and animation softwares one of the most important features that the software should have is to generate smooth transitions between graphics (animation / images / key frames etc).

With reference to images we call this process Image Metamorphosis.

There are many different traditional image metamorphosis algorithms available but they utilise the property of pixel transformation or reference object shape transformation rather than the features in those images.

The paper introduces a new technique called Field Morphing that utilises two-dimensional control primitives by distorting the two images and blending them together.

This algorithm transforms the two images more effectively and fluently than the traditional morphing methods.

Table of Contents

Introduction

Link Reference: Princeton CS Repository Link / ACM Digital Library

The main objective of this paper is to metamorph Image 1 into Image 2 via seamless transition.
Metamorphosis doesn't only mean to cross dissolve one image into another but also to retain the features in the image at a particular location without displacing it too much.

Conventional morphing techniques involve changing / interpolation of pixels but this paper entails on the *computer graphics aspect* of determining the *interpolation of frames using line segments* that emphasis the features in the images, hence the same concept can be extended to 3D using 3D line segments to interpolate voxels.

This paper introduces a new technique called **Field Morphing.** This morphing is influenced by the surrounding two-dimensional control primitives.

Distortion of a single image is achieved using inverse mapping in which every pixel in the destination image gets set to some appropriate source pixel.

How to determine the source pixel?

Line segments are drawn on two images to highlight the feature correspondence between two images. **[Ref Figure 4]**

The paper first introduces the **Transformation With One Pair Of Lines**

(1) for each pixel X in destination image
(2) compute u, v given the line PQ as follows:
(3) $$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2}$$
(4) $$v = \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|}$$
(5) compute X' using u, v on the line $P'Q'$:
(6) $$X' = P' + u\cdot(Q'-P') + \frac{v\cdot Perpendicular(Q'-P')}{\|Q'-P'\|}$$
(7) $$destinationImage(X) = SourceImage(X')$$

Figure 3. [Taken from paper]
Algorithm using a single line pair.

The algorithm transforms each pixel coordinate by rotation, translation, and/or scale, thereby transforming the whole image.

But after analyzing the algorithm we discover that it has the following drawbacks - the scaling is not uniform, but is along the length of the line. Also, shears are not possible to specify.
All the transformations based on a single line pair are affine.

Hence the paper introduces the **Transformation With Multiple Pair Of Lines**

(1) for each pixel X in destination image
(2) $DSUM = (0,0)$
(3) $weightsum = 0$
(4) For each line P_iQ_i in destination image
(5) calculate u, v based on P_iQ_i
(6) calculate X_i' based on u, v and $P_i'Q_i'$
(7) calculate displacement $D_i = X_i' - X$ for this line
(8) $dist$ = shortest distance from X to P_iQ_i
(9) $weight = \left[\frac{length^p}{(a + dist)}\right]^b$
(10) $DSUM+ = D_i * weight$
(11) $weightsum+ = weight$
(12) $X' = X + \frac{DSUM}{weightsum}$
(13) $destinationImage(X) = sourceImage(X')$

Figure 4. [Taken from paper] Algorithm using multiple line pairs and distance error and weighted sum.

A morph operation blends between two images, I0 and I1, to achieve this we define corresponding lines in I0 and I1.

Each intermediate frame I of the metamorphosis is defined by creating a new set of line segments by interpolating the lines from their positions in I0 to the positions in I1.

Both images I0 and I1 are distorted toward the position of the lines in I. These two resulting images are cross- dissolved throughout the metamorphosis, so that at the beginning, the image is completely I0 (undistorted because we have not yet begun to interpolate away from the line positions associated with I0).

Halfway through the metamorphosis it is halfway between I0 and I1, and finally at the end it is completely I1.

Approaching The Solution

First I would need to design a GUI that displays the two images that we want to metamorph.
This GUI would need you to draw the corresponding line segments to highlight features.

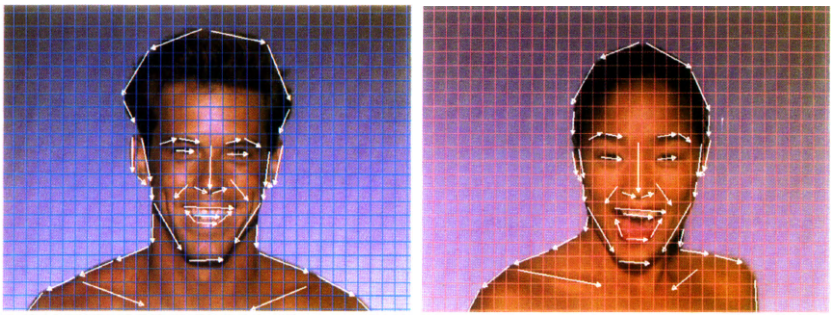

Figure 5. [Taken from paper] Shows the line segment feature highlighting correspondence to the images to be metamorphed.

These line segments would be stored in a separate text file.

Assume we are having two images **Image1** and **Image2**. And the resultant image at any given **time = t** is **ResultImage**.

The image metamorphosis algorithm could be normalised to a scale of time between time = 0 to 1.

At time = 0, **the ResultImage is exactly Image1.**

At time = t, the ResultImage consists of distorted Image1 translucent and distorted Image2 translucent.

At time = 1, **the ResultImage is exactly Image2.**

As time progresses we need to metamorph this Image1 into Image2.

This process according to the algorithm described by Beier-Neely consists of three steps -

1. Distort Image1.
2. Distort Image2.
3. Blend Distorted Image1 and Distorted Image2.

Two functions need to be designed for the above steps - DistortImage and BlendImage.

DistortImage Function: This function takes in the following parameters:

1. Image
2. TimeStamp (Value between 0 to 1)
3. Three Parameters [a, b, p] specified in the algorithm for weight calculation.
4. Line Segments in Image1 and Image2.

This function distorts an image according to the algorithm described in the paper using weighted sum of linear interpolations of the line segments.

BlendImage Function: This function takes in the following parameters:

1. Image1
2. Image2
3. TimeStamp (Value between 0 to 1)

This function linearly blends (cross fades) the two images based on the timestamp.

The resultant pixel (x,y) at time = t would be the weighted average (parameterized with t (time)) of the two image pixels at the location (x,y).

The main driver program shall take the two images Image1 and Image2 and progressively generate the interpolated frames of two images morphed together at different time intervals (time-step of 0.25 units).

All of the resultant images generated can then be combined and converted into a GIF hence achieving the metamorphosis task.

Files Submitted

FileName	Motive
2016B2A70773P_D2.pdf	Final Report
2016B2A70773P_D2.zip	Folder containing the soft copy of the code implemented.

Contents of 2016B2A70773P_D2.zip -

FileName	Motive
README.md	Readme file containing instructions on how to compile and execute the code with dependencies.
Algebra3.hpp, Image.cpp, Image.hpp, LineSegment.hpp, stb_image_write.hpp, stb_image_write.cpp, stb_image.cpp, stb_image.hpp	Publically available library files used for easier implementation to handle images / line segments.
Editor.java	Java file that presents you with two images to draw line segments on it for highlight features in them.
Morph.cpp	C++ file that performs Multiple Line Transformation as described in the paper using line segments made by the above file
Makefile	Compiles the code, also used to remove compiled files if needed.
Morph.sh	Shell Script that executes the code
images	Folder containing some sample images for use
Multiple *.o and Editor.class	Various object files pre-compiled - given so that the user doesn't have to compile.

In case you are not able to run the code, you can also view the working of the code on my system in this video.

Design for the solution

Designing the Editor

As explained above, an editor had to be made which would present the user with two images side by side that they had to morph. The user had to then draw line segments that would highlight the feature correspondence in those images.

Initially I had planned to do everything in C/C++ as it is my primary coding language but due to package limitations I had to resort to using Java Swing GUI.

I used Java Swing GUI to design a window that would display two images, and using mouse drag user and plot the line segments that would highlight the feature correspondence in those images.

The mouse characteristics were easy to code using the MouseListener and MouseMotionListener classes in Java.

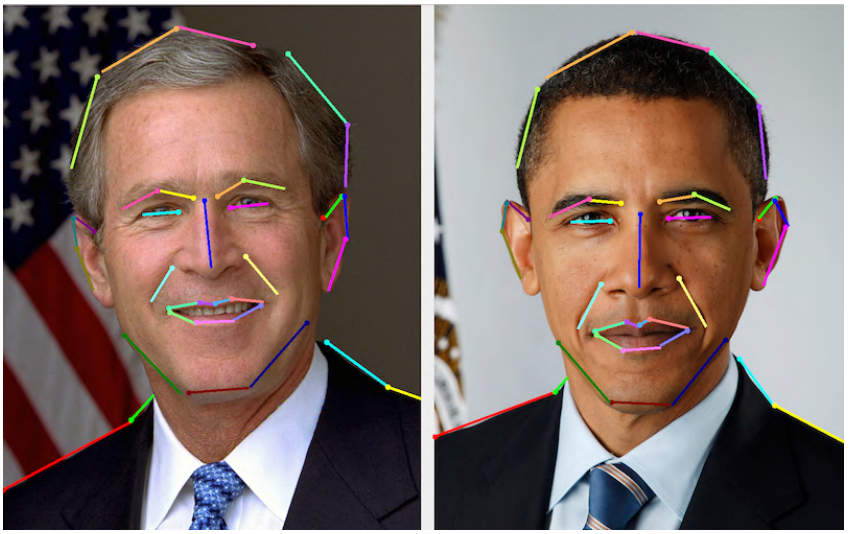

Figure 6. Shows the line segments feature highlighting correspondence to the images to be metamorphed.

As the user draws the line segments on the image, the function saveCorrespondences writes the line segment information in a normal text file. This text file contains the number of segments and these 8 values of line segments -

1. Left Image Start Point X
2. Left Image Start Point Y
3. Left Image End Point X
4. Left Image End Point Y
5. Right Image Start Point X
6. Right Image Start Point Y
7. Right Image End Point X
8. Right Image End Point Y

Coding the Algorithm

The C++ part of the code takes care of running the actual multiple line transformation algorithm described in the paper.

The script takes input of two images Image1 and Image2 (considering you want to metamorph Image1 into Image2) and the Line Segments File generated above by the editor.

The program uses the following parameter values a = 0.5 =, b = 1, p = 0.2 as recommended in the paper.

The program generates a new image morphed for every time increment of t = 0.05.

The program first reads all the line segments from the file and adds it in the vector array of type Class LineSegment.

The morphing function has been explained above as two distortions of two images and 1 blending of both images.

The distortion function simply calculates the weighted sum of bilinear interpolations as described in the paper.

The blending function calculates the weighted average of two pixels at the same location of the image to make it appear as blending (translucent-transitioning).

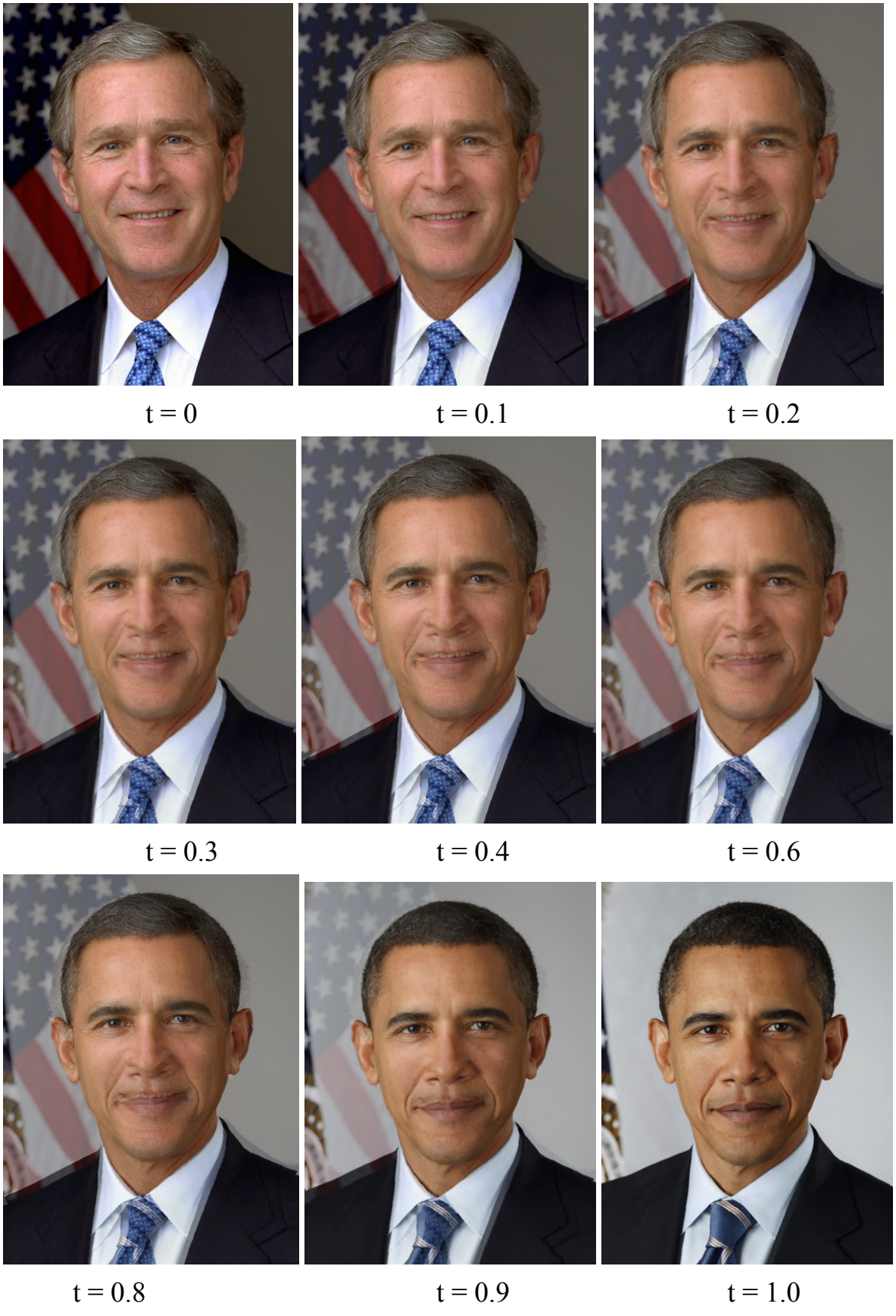

Figure 7. Shows the resultant metamorphed images.

Conclusion and Future Work

Feature-based Image Metamorphosis is a useful visual animation tool that transforms one image to another in a smooth fashion. The animation can generate an image of a subject that looks like a real subject, but is actually a combination of the subjects in the source and destination images.

According to Beier-Neely, this technique has one big advantage over the mesh warping technique: it is much more expressive. The only positions that are used in the algorithm are ones the animator explicitly created.

We can observe that Feature-based image metamorphosis is indeed very slow. So, better techniques can be invented to increase the speed. Also, we can have better algorithms with local control and not global control, as is the case with this algorithm. Ghosting and problems in linear interpolation can also be removed for improved future works.

References

[1] Neely S & Beier T, *"Feature-based image metamorphosis".* SIGGRAPH '92, July 1992, Pages 35-42

[2] https://web.cs.wpi.edu/~matt/courses/cs563/talks/morph.html

[3] https://en.wikipedia.org/wiki/Morphing

[4] https://en.wikipedia.org/wiki/Bilinear_interpolation

[5] https://stackoverflow.com/questions/4906736/working-with-images-in-c-or-c

[6] https://stackoverflow.com/questions/18027833/adding-image-to-jframe/18027889

[7] https://github.com/nothings/stb