Image Morphing with the Beier-Neely Method

by

Feng Zhu
B.Sc., Northwestern Polytechnical University, 2013
A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Image Morphing with the Beier-Neely Method

by

Feng Zhu
B.Sc., Northwestern Polytechnical University, 2013

Supervisory Committee

Dr. Michael Adams, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Pan Agathoklis, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Michael Adams, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Pan Agathoklis, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

The Beier-Neely feature-based image-morphing method is studied. Then, software implementing the Beier-Neely image-morphing method, designed and developed by the author, is presented. The software consists of three programs. The first program is a graphical user interface (GUI) used to manually select feature line segments. The second program is a morphing program that generates a morphing image sequence, where each intermediate frame in the sequence represents a stage in the morphing process. The third program converts the image sequence produced to a video that displays the image-morphing effect.

Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

my Supervisor Dr. Michael Adams, for mentoring, support, encouragement, and patience. Thank you for being punctilious and rigorous through my graduate study. I could not achieve such significant improvement without your guidance.

my Supervisory Committee member Dr. Pan, for being on my Supervisory Committee. Thank you for spending time reviewing my project. I also want to express my gratitude to the staff members in the Department of Electrical and Computer Engineering, Dr. Dong Xiaodai, Janice Closson, Dan Mai, and Amy Rowe.

my friends Weiheng Ni, Darya Ismailova, Xiao Feng, Xiao Ma, for standing together with me and helping me overcome difficulties. I learned a lot from the group projects we did and your valuable suggestions on my studies. Thank you for answering my questions patiently and carefully from trivial programming questions to global design problems. The time we spend together is priceless.

my parents Ming Zhu and Heyan Xu, for supporting me with your unconditional love. Thank you for understanding me and encouraging me all the time. It is your selfless dedication to my growth that makes my dream come true.

DEDICATION

To my family.

Chapter 1

Introduction

1.1 Image Morphing

Image morphing can be defined as an image processing technique to progressively turn one image into another through a smooth transition over a period of time. As the morphing proceeds, the first image is gradually distorted and faded out, at the same time the second image is distorted toward the first image and faded in. Therefore, the early images in the sequence are much more like the first image. The middle image in the sequence is the average of the first image distorted halfway towards the second and the second distorted halfway back toward the first image. For example, when morphing between two faces, the middle image normally looks lifelike, like a real human face, but it is identical to neither the first person nor the second person.

An example of image morphing is shown in Figure 1.1. The images in this example

(a) (b) (c) (d)

Figure 1.1: An example of image morphing. (a) The source image. (d) The target image. (b) and (c) Intermediate frames generated by applying an image-morphing technique.

are produced by using image-morphing software developed by the author. The image in Figure 1.1(a) is called the source image and it is where the morphing starts. The image in Figure 1.1(d) is called the target image and it is where the morphing ends. The two intermediate images in Figure 1.1(b) and (c) are computed by applying an image-morphing method. As we can see from the sequence, image morphing actually performs interpolations of intermediate frames between the source image and the target image.

1.2 Historical Perspective

Image morphing has proven to be a powerful tool for visual effects. Many breath-taking applications in the film and television industries of fluid transformations are easy to find. In the early 1980s, Brigham used a form of morphing in experimental art at the New York Institute of Technology [10]. Traditional techniques for morphing effects include clever cuts such as a character exhibiting changes while running through a forest and passing behind several trees, and an optical cross-dissolve where one image fades out while the other fades in with makeup change, appliances, or object substitution. These classic methods have proven their success in horror films such as *Wolfman*. The first movies with morphing are *Indiana Jones and the Last Crusade* in 1989 and *Willow* in 1998. The first music video with morphing is *Black and White* by Michael Jackson in 1991. Even many Disney animations are made using morphing for speeding production.

One morphing technique was pioneered at Industrial Light and Magic by Smythe in 1990 [18] as a mesh-warping approach. A well-known feature-based morphing approach was proposed by Beier and Neely in 1992 [3] from its application in Michael Jackson's music video *Black or White*. Then Lerios et al. addressed the ghosting artifacts by correcting the warp area and then extending the method to 3D voxels in 1995 [12]. In 1996, Lee et al. proposed an energy minimization method for deriving one-to-one warp functions [5]. After that, Wolberg discussed the popular thin-plate spline interpolation approach based on point correspondences in 1998 [11]. More recently, a computationally-complicated method based on features of line segments was proposed by Schaefer et al. in 2006 [17].

From the related work above, morphing algorithms fall into three categories: mesh warping, feature-based warping, and thin-plate spline interpolation. These different image-morphing methods are explained in the following paragraphs.

In the mesh-warping approach, we choose points representing features in the source and target images, generate meshes of triangles for the two images according to the points we selected, and then warp each triangle through an affine transformation. In this way, the mesh

warping relates features with two sets of non-uniform meshes in the source and target images, dividing the images into small pieces that are mapped onto each other. Each frame in the transformation uses an interpolated mesh as the set of target positions for the input mesh points, and the interpolated mesh is computed by performing linear interpolation between respective points in the two sets.

While meshes, as described above, appear to offer a convenient way to specify feature points, the meshes are sometimes cumbersome to use because of the huge amount of time required to carefully select mesh nodes. The feature-based approach was developed to simplify the user interface using line segments to relate features in the source and target images. This algorithm is based on fields of influence surrounding the feature line segments and offers a high level of control over the morphing process. The feature-based technique has a big advantage over the mesh-warping technique described in Wolberg's book [19]: it is much more expressive. The only positions that are used in the algorithm are the ones that the animator explicitly creates. Since lines and curves can be point sampled, it is sufficient to consider the features on an image to be specified by a set of points.

The components of a warp can be derived by constructing the surfaces that interpolate scattered points [20]. The thin-plate spline interpolation approach is based on this observation. As a conventional tool for surface interpolation over scattered data, the thin-plate spline method aims at finding a "minimally bended" smooth surface that passes through all given points.

The progression of morphing algorithms has been marked by more expressive visual effect and less burden of feature specification. A significant step beyond mesh warping was made possible by the specification of line segment pairs in feature-based morphing. All subsequent algorithms, including the thin-plate splines and energy minimization, sought to improve the smoothness of the computed warp function. They did so at a relatively high computational cost. In terms of the computational speed, mesh warping is the best, according to the work of Alexandra [6]. The computational advantage, however, is greatly offset by the huge amount of time required to carefully select mesh nodes which requires much animator effort. The main disadvantage of the feature-based and thin-plate spline algorithms is the speed. The thin-plate spline approach requires the least animator effort but the speed is the slowest among these three algorithms. The feature-based algorithm keeps a balance between time complexity and animator effort. Due to the acceptable computational complexity, proper degree of implementation difficulty, and proven quality of visual effect, we feel that the feature-based approach is most practical. Therefore, the project described in this report focuses on the well-known Beier-Neely feature-based algorithm. A detailed description of

this method will be presented later in Chapter 2.

1.3 Organization of This Report

The reminder of this report is organized as follows.

Chapter 2 provides the background information to facilitate a better understanding of the project, including some notation used in the next chapters, a description of image-morphing techniques, image warping, forward mapping and reverse mapping, and image blending.

Chapter 3 introduces the Beier-Neely algorithm implemented in our project. First, an overview of the method is presented to give a general idea of this approach. Before presenting more detailed descriptions, we introduce the feature line-segment data format. Then, we describe warping in detail, including linear line-segment interpolation and transformations of line segments.

Chapter 4 focuses on the image-morphing software developed with the Beier-Neely algorithm by the author. To begin, an overview of the software is introduced, which generally explains what the software does as well as its constituent programs. Then, we discuss the prerequisites for using the software involving libraries, compilers, and other software tools, followed by the steps for building the software. Further, a detailed introduction of the programs that constitute the software is provided with regard to their respective functionality and command-line interface. Finally, several software usage examples are provided as concrete references for the user's convenience.

Chapter 5 presents some results produced by the software. Then, we discuss the effect of changing various parameters of the warping in the Beier-Neely method, and analyze the time-complexity of the method through experiments. Lastly, suggestions for obtaining high-quality morphing results are presented with examples.

Finally, Chapter 6 summarizes the work on the project followed by potential future work related to this project.

Chapter 2

Background

2.1 Overview

To help the reader understand the remainder of the report, this chapter provides some necessary background. First, we give some notation used in this report. Then, a general technical definition of image morphing is given. As we will see, image morphing involves two main steps, namely, image warping and image blending. Finally, we further illustrate these two key steps with more details.

2.2 Notation

Before proceeding further, we introduce some basic notation employed throughout the report. The set of real numbers is denoted as $\mathbb{R}$. To denote that a number x is an element of $\mathbb{R}$, we write $x \in \mathbb{R}$. We define the following notation for a subset of the real line $\mathbb{R}$:

$$[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}. \tag{2.1}$$

Vectors are denoted by lowercase letters. For a vector $v = (v_1, v_2, ..., v_n)$, the 2-norm of v is denoted as $||v||$, and defined as

$$||v|| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}. \tag{2.2}$$

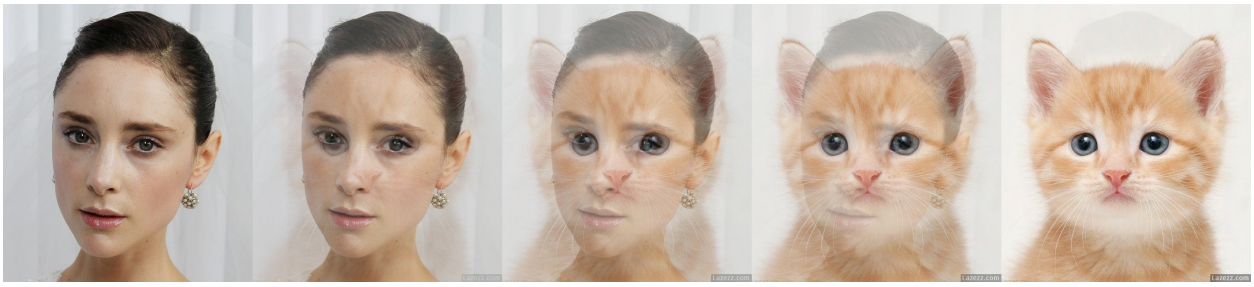


Figure 2.1: Cross-dissolving

For two vectors $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_n)$, the dot product of a and b is denoted as $a \cdot b$, and defined as

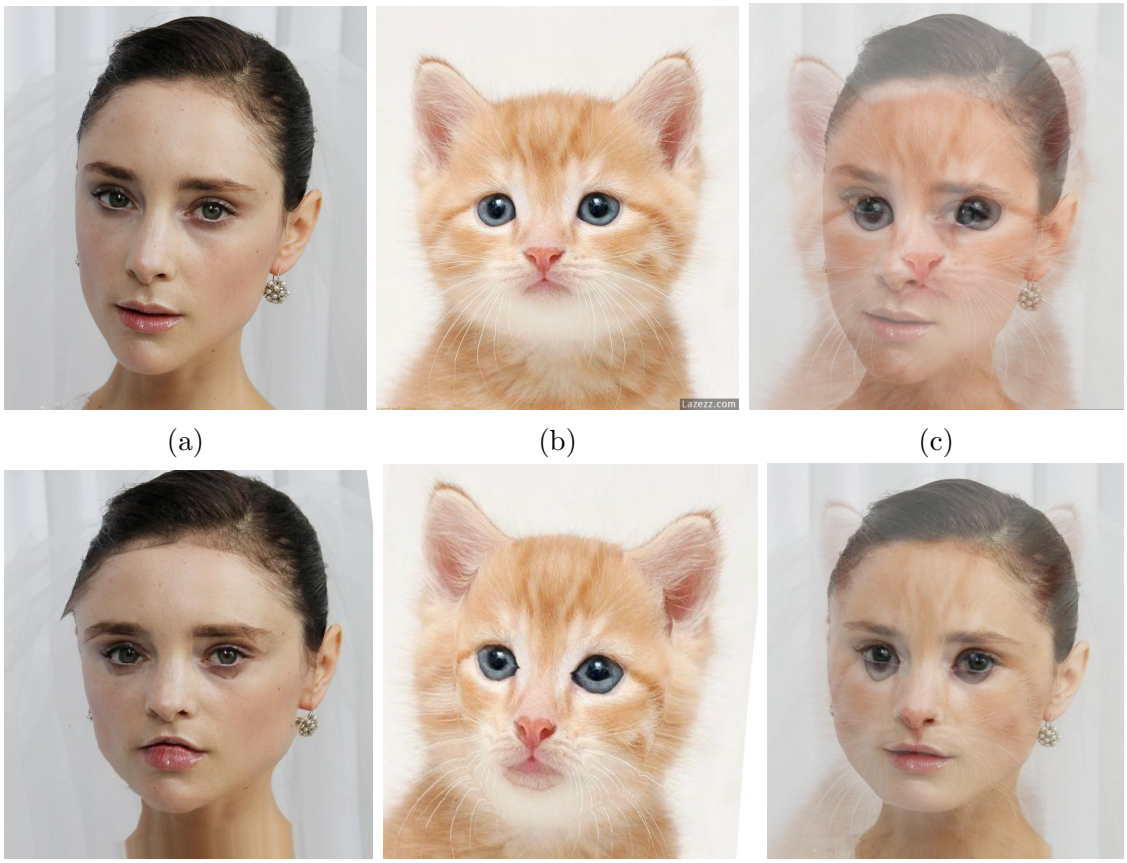
$$a \cdot b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n. \tag{2.3}$$

Points are denoted using uppercase letters. Line segments are denoted by two uppercase letters. For instance, PQ represents a line segment with two endpoints P and Q. In addition, a directed line segment with starting point P and ending point Q is denoted as $\overrightarrow{PQ}$. Since directed line segments frequently occur herein, we will often drop the arrow from the notation for directed line segments in contexts where this should not cause confusion (e.g., simply write PQ instead of $\overrightarrow{PQ}$).

2.3 Image Morphing

Color blending with a changing weight can make one image transform to another. This process, commonly known as image cross-dissolving, is a pixel-by-pixel color interpolation between two images. Figure 2.1 shows a cross-dissolving sequence from a girl to a cat. Each frame in the sequence is a weighted sum of the source and target images without feature alignment, and the target image fades in at the beginning while the source image fades out at the end. We can easily observe that the technique is visually poor because the features of both images are not aligned, resulting in an apparent double-image effect in misaligned areas.

To overcome the poor visual effect of cross-dissolving shown in the preceding example, aligning features in the two images, which is achieved by image warping, is done before image blending. Warping decides the way in which pixels from one image are mapped to the corresponding pixels in the other image. In other words, warping applies 2D geometric



(a) (b) (c)

(d) (e) (f)

Figure 2.2: Image blending with or without warping. (a) The source image. (b) The target image. (c) The result of blending the images in (a) and (b). (d) The warped source image. (e) The warped target image. (f) The result of warping the images in (a) and (b) followed by blending the resulting images in (d) and (e).

transformations to the images to retain geometric alignment between the features in the images, while color interpolation blends the colors in the images. Figure 2.2 illustrates the effect of image warping. Figure 2.2(a) and Figure 2.2(b) are the source and target images, respectively. Figure 2.2(c) shows the result of blending the source and target images without feature alignment. We can clearly see that two separate noses exist in the image. Figure 2.2(d) shows the warped source image and Figure 2.2(e) shows the warped target image. As we mentioned above, applying image warping can greatly improve the morphing visual effect. This improvement can be seen clearly in Figure 2.2(f), where image warping is applied before image blending. Apparently, the girl's face is widened to fit the cat's face. Their noses, mouths, and ears have merged into one. This example illustrates how image morphing can

Algorithm 1 General image morphing algorithm

Input: source image S, target image D
Output: a sequence of morphed images

1: **for** each intermediate frame at stage $t \in [0, 1]$ **do**
2: Warp image S: $W_S = \text{warp}(S, L_t)$
3: Warp image D: $W_D = \text{warp}(D, L_t)$
4: Blend W_S and W_D: $I_t = \text{blend}(W_S, W_D, t)$
5: **endfor**

be achieved by a combination of image warping and image blending.

Image blending can be implemented easily by adding one image to another, while image-warping techniques vary in the way the mapping of pixels is specified. So, the main difference among various morphing algorithms lies in the warping approach. Generally, image morphing starts with an animator establishing the correspondence between source and target images using pairs of image primitives, including mesh nodes, points, line segments, or curves. The correspondence is then used to compute a mapping function that defines the spatial relationship between pixels in both images.

Ignoring the technical details involved in warping and blending, we can give a general image-morphing algorithm as shown in Algorithm 1. This algorithm illustrates how each intermediate frame is produced. The stage t represents the progress of morphing, S and D are source and target images, L_t is a warping parameter at stage t, W_S and W_D are images warped from S and D respectively, and I_t represents the frame generated at stage t. By iterating t from 0 to 1 with an incremental step, we can produce a sequence of frames which is the result of image morphing. Different image-morphing algorithms vary in the strategy used for warping, but the general algorithm including the blending is the same.

2.4 Image Warping

Through image warping, coordinate transformations are performed so that the spatial configuration of the image can be significantly distorted while its coloring can be maintained. Warping transforms each pixel coordinate from one position to another position with a mapping function, and therefore transforms the entire image. Figure 2.3 shows a result of image warping. The pixels with positions P and Q in the original image are mapped to the pixels with positions P' and Q' in the warped image. This example shows the visual effect of image warping: the shape of the original girl's face is clearly widened and shortened. When performing image warping, we control how pixels are mapped so that the features of the source

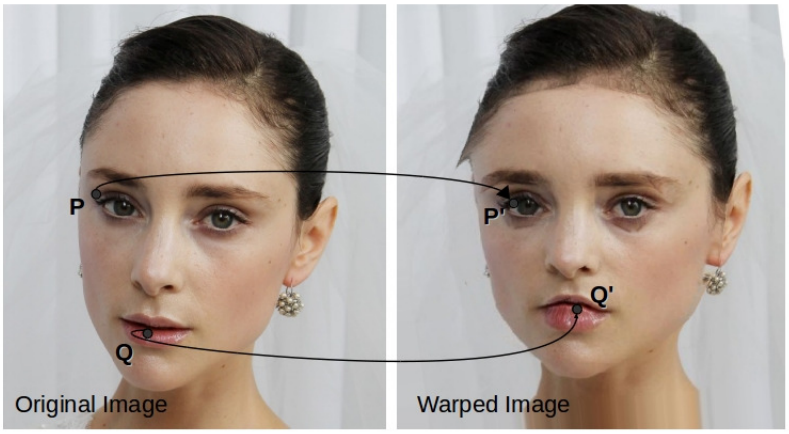


Figure 2.3: Image warping

and target images are matched (i.e., the left eye in the source image will be at the same position as the left eye in the target image). The mapping of pixels can be implemented in two different ways: forward mapping and reverse mapping, which will be further explained in the next section.

2.5 Forward Mapping and Reverse Mapping

Forward mapping and reverse mapping are two ways to warp an image. Forward mapping scans the source image pixel by pixel, copying each one to the appropriate place in the destination image. The destination image is initialized to be blank before warping starts and becomes a warped source image after warping ends. Figure 2.4 shows the forward-mapping process. For each pixel in the source image, the algorithm finds the appropriate pixel to which the source image pixel should be mapped in the destination image. Then copy the source image pixel to the destination image pixel. The method has a problem that some pixels in the destination image may not have any source image pixel mapped to them so that some pixels in the destination image may remain blank. We could perform interpolation to assign values to those empty pixels, but it would take much time to find which ones are undefined. Thus, forward mapping is inefficient to perform.

Reverse mapping scans the destination image pixel by pixel, sampling the correct point from the source image. Figure 2.5 shows the reverse-mapping process. For each pixel in the destination image, the algorithm finds the appropriate pixel from which the destination

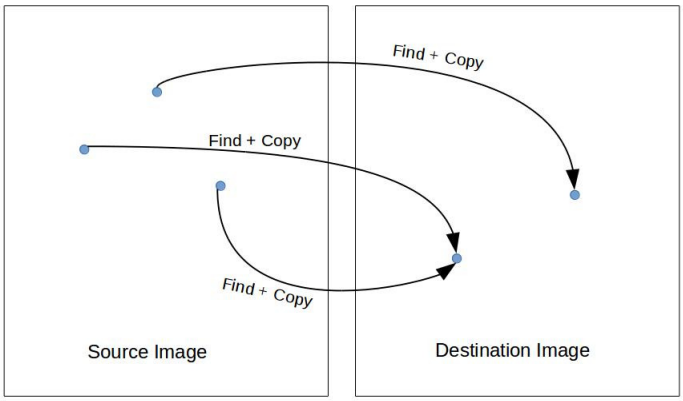


Figure 2.4: Forward mapping

pixel should be sampled in the source image and then copies the source image pixel to the destination image pixel. If the pixel that should be mapped from the source image is outside the source image boundary, we could apply extrapolation using nearest neighbours. The most important feature of reverse mapping is that every pixel in the destination image will be assigned an appropriate value without extra effort, unlike the case of forward mapping. For this reason, the Beier-Neely algorithm uses the reverse-mapping method to construct a warped image.

2.6 Image Blending

Besides image warping, the other important step involved in the morphing process is image blending. As the image warping is taking place, image blending is performed so that the warped images merge into one image. The visual effect of blending is previously shown in Figure 2.2(c). Let A, B, and C denote the source image, target image, and blended image, respectively. Then, $A(x, y)$, $B(x, y)$, and $C(x, y)$ each represent one color component of a pixel with coordinate (x, y) in A, B, and C, respectively. Image blending is described by the equation

$$C(x, y) = \alpha A(x, y) + (1 - \alpha)B(x, y), \quad \text{where } \alpha \in [0, 1]. \tag{2.4}$$

For color images, this calculation in (2.4) is applied for each of the color components.

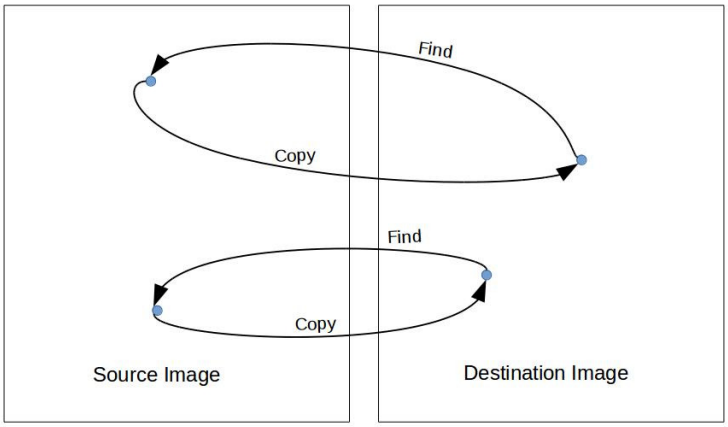


Figure 2.5: Reverse mapping

Chapter 3

The Beier-Neely Algorithm

3.1 Introduction

The Beier-Neely algorithm is a feature-based image-morphing method proposed by Beier and Neely in 1992 [3]. The basic idea of this method is to specify the correspondence between source and target images interactively using a set of line-segment pairs. The mapping of pixels from the source image to the target image is defined using these feature line segments. The output is a sequence of frames representing a morphing of the images. The main work of this project is developing image-morphing software with the Beier-Neely algorithm. So, in this chapter, we introduce this algorithm in detail.

3.2 Feature-Based Image Morphing

Feature-based morphing constructs the warping function using image features. The feature-based method, developed by Beier and Neely in 1992 at Pacific Data Images, was motivated by the desire to simplify the user interface for image morphing. The warping technique in a feature-based method uses two-dimensional control points to specify corresponding features in the source and target images. Each control point exerts a field of influence on its surrounding area with the strength of this field decreasing in proportion to the distance from the control point. The feature-based Beier-Neely algorithm allows users to specify corresponding features using directed line-segment pairs, and therefore provides a greater degree of control over the morphing by offering users the freedom to choose image features. In addition to the straightforward correspondence provided for all points along a directed line segment, the mapping of points in the vicinity of the directed line segment can be determined by the

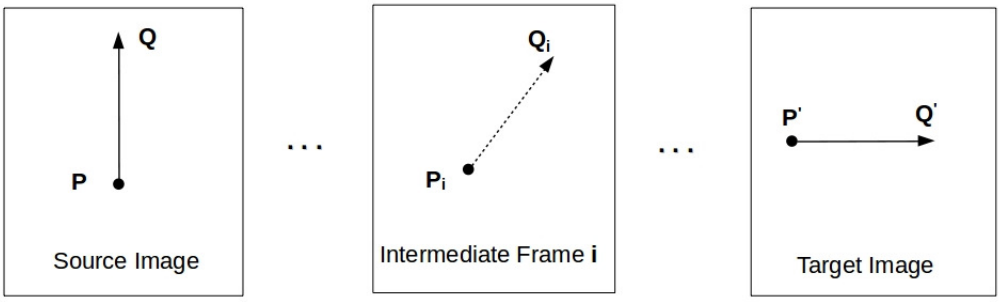


Figure 3.1: Line-segment interpolation

points' perpendicular distance to the directed line segment. Since multiple directed line-segment pairs are usually selected, the displacement of a point is actually a weighted sum of the mappings due to each directed line-segment pair, with the weights attributed to the length of the line segment and the shortest distance between the point and the line segment. Since all of the line segments used in the Beier-Neely method are directed, for convenience, we will often omit the "directed " qualifier in what follows.

3.3 Warping for Beier-Neely Algorithm

Recalling image warping as explained in Section 2.4, we know warping applies a geometric transformation to images. In the coming sections, we will introduce how image warping is defined in the Beier-Neely algorithm.

3.3.1 Linear Line-Segment Interpolation

From Algorithm 1, we know that each intermediate frame is produced by first warping the source and target images using a warping function at the corresponding stage, and then blending the warped images. Furthermore, the warping function is computed from a set of feature line segments at that stage, which is generated by linearly interpolating the feature line segments between their positions in the source and target images. Figure 3.1 shows the interpolation process for a single pair of line segments. In Figure 3.1, PQ is a feature line segment in the source image and $P'Q'$ is a corresponding feature line segment in the target image. We generate N intermediate line segments $\{P_iQ_i\}_{i=1}^{N}$ by interpolation as follows:

1. Calculate the incremental step ΔP for P: $\Delta P = (P - P')/N$

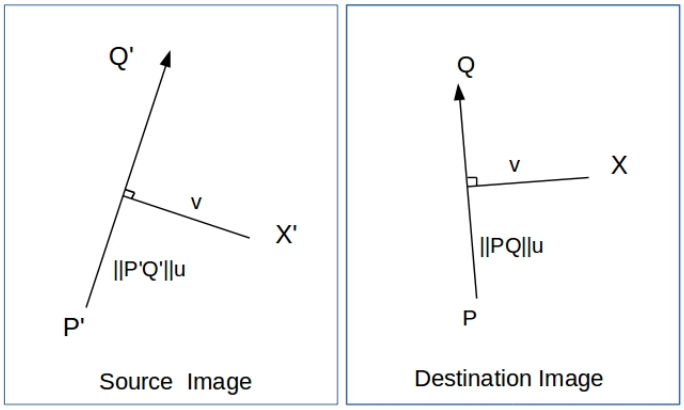


Figure 3.2: Transformation with one pair of feature line segments

2. Calculate the incremental step ΔQ for Q: $\Delta Q = (Q - Q')/N$

3. For each stage i from 0 to $N - 1$, calculate the interpolated line segment P_iQ_i, where $P_i = P + \Delta Pi$ and $Q_i = Q + \Delta Qi$.

3.3.2 Transformation with One Pair of Feature Line Segments

A pair of feature line segments in the source and destination images defines a mapping from a position in the destination image to a corresponding position in the source image. Figure 3.2 shows the case when only one pair of feature line segments is used for the mapping, where X is a position in the destination image, X' is the corresponding position in the source image, PQ is a feature line segment in the destination image, and $P'Q'$ is the corresponding feature line segment in the source image. The mapping function can be described by the equation

$$X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{perp}(Q' - P')}{||Q' - P'||}, \tag{3.1}$$

where

$$u = \frac{\lambda}{||Q - P||}, \tag{3.2}$$

$$v = \frac{(X - P) \cdot \text{perp}(Q - P)}{||Q - P||}, \tag{3.3}$$

$$\lambda = \frac{(X - P) \cdot (Q - P)}{||Q - P||}, \tag{3.4}$$

and perp(α) denotes a vector perpendicular to, and having the same length as, α, where which of the two possible perpendiculars is denoted by perp(α) does not matter as long as consistently chosen. The value λ is the signed distance from P to the orthogonal projection of X onto the line through P and Q. The value u is a normalized version of λ that is scaled such that u goes from 0 to 1 as X's projection moves from P to Q. The value v is the signed perpendicular distance from X to the line through P and Q. The signed perpendicular distance from X' to the line through P' and Q' is v, and the normalized signed distance from P' to X''s projection onto the line through P' and Q' is u. If the X' falls outside image domain, no corresponding true value of X''s coordinate can be determined. The Beier-Neely method as described in the paper [3] does not mention how to solve this problem. So, we have proposed an approach that replaces the X' by the pixel coordinate closest to X' along the image boundary. Later results show this to achieve satisfactory results.

The steps in Algorithm 2 show the transformation process with one pair of feature line segments.

Algorithm 2 Algorithm for transformation with one pair of feature line segments

Input: source image S, feature line segments PQ and $P'Q'$
Output: destination image D

1: **for** each pixel with position X in the destination image **do**
2: calculate u and v for X using (3.2) and (3.3)
3: find X' in the source image with u and v using (3.1)
4: **if** X' falls outside the image domain **then**
5: find the pixel coordinate X'_C closest to X' on the boundary of the source image
6: update X': $X' = X'_C$
7: **endif**
8: **if** X' contains non-integer coordinate **then**
9: find the pixel coordinate X'_I by interpolating the neighbours of X' and rounding the interpolation result
10: update X': $X' = X'_I$
11: **endif**
12: copy the value of the pixel at X' to that of the pixel at X: $D(X) = S(X')$
13: **endfor**

3.3.3 Transformation with Multiple Pairs of Feature Line Segments

Normally, more than one feature needs to be employed in order to obtain a vivid morphing. So, a transformation using multiple pairs of feature line segments is applied. Each feature line segment is associated with a weight that determines the influence that the feature line segment exerts on a pixel. The weight is determined by the shortest distance from the pixel to the feature line segment and should be strongest when the pixel is exactly on the feature line segment, and weaker the further the pixel is from the line. The equation used in the algorithm is

$$w = \left(\frac{l^p}{a + d} \right)^b , \tag{3.5}$$

where l is the length of a feature line segment, d is the shortest distance from a pixel to the feature line segment, and a, b and p are parameters that can be used to change the relative influence of the feature line segments.

The value of a determines the smoothness and precision of the user's control over the warping. A lower value of a implies a tighter control but less smooth warping effect. An increasing value of a results in a less precise control but a more smooth warping effect. The variable b determines how the relative influence of different feature line segments decays with distance. A large value means a pixel will only be affected by the closest feature line segment, and a zero value implies every feature line segment has the same relative influence. To achieve a reasonable morphing result, b should typically be chosen in [0.5, 2]. Another parameter p determines how the length of a feature line segment influences the weight and should be chosen in [0, 1]. A zero value means length has no influence and a higher value means weight is affected more by length.

Figure 3.3 illustrates the case when two pairs of feature line segments are used for the warping. In this example, $P_1'Q_1'$ and $P_2'Q_2'$ are feature line segments for the source image, while P_1Q_1 and P_2Q_2 are the corresponding feature line segments for the destination image. The d_1 is the distance between X and X_1', d_2 is the distance between X and X_2', and X' is the location to sample the source image for X in the destination image. The X' is a weighted average of the two locations X_1' and X_2', computed with respect to the first and second feature line segment pair, respectively.

The algorithm for transformation with multiple pairs of feature line segments is shown in Algorithm 3.

Algorithm 3 Algorithm for transformation with multiple feature line-segment pairs

Input: source image S, feature line-segment set $P_1Q_1, P_2Q_2, ..., P_nQ_n$
Output: destination image D

for each pixel with position X in the destination **do**
  $D_{\mathsf{sum}} = (0, 0)$
  $W_{\mathsf{sum}} = 0$
  for each P_iQ_i in the feature line-segment set **do**
    calculate u and v for X based on P_iQ_i using (3.2) and 3.3
    find X' in the source image with u and v using (3.1)
    calculate displacement $d_i = X' - X$
    calculate the weight w using (3.5)
    $D_{\mathsf{sum}} = d_i w + D_{\mathsf{sum}}$
    $W_{\mathsf{sum}} = w + W_{\mathsf{sum}}$
  endfor
  $X' = X + D_{\mathsf{sum}}/W_{\mathsf{sum}}$
  if X' falls outside the image domain **then**
    find the pixel coordinate X'_C closest to X' on the boundary of the source image
    update X': $X' = X'_C$
  endif
  if X' contains non-integer coordinate **then**
    find the pixel coordinate X'_I by interpolating the neighbours of X' and rounding the interpolation result
    update X': $X' = X'_I$
  endif
  copy the value of the pixel at X' to that of the pixel at X: $D(X) = S(X')$
endfor

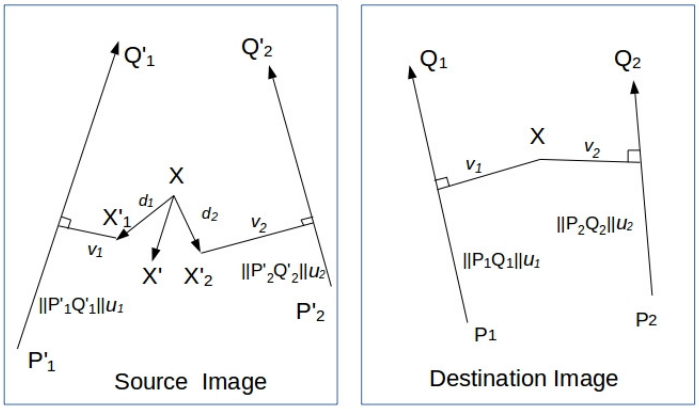


Figure 3.3: Transformation with multiple pairs of feature line segments

3.4 Morphing Between Two Images

With all of the procedures explained previously, we can give a summary of the Beier-Neely algorithm. Morphing between two images consists of the following steps in order:

1. Define feature line-segment sets L_S and L_D for the source image S and the target image D, respectively.

2. Loop t from 0 to 1 with a user-defined number n of intermediate steps. In each step, do the following:

   (a) Perform linear line-segment interpolation to create a new feature line-segment set L_t. This L_t is interpolated from L_S to L_D with regard to the t.

   (b) Warp both the source and the target images with the feature line segments in L_t. Denote W_S as the warped source image and W_D as the warped target image.

   (c) Blend the two warped images W_S and W_D to obtain an intermediate frame I_t in the morphing sequence at stage t.

3.5 Morphing Among Multiple Images

We can also morph among multiple images easily. Morphing among $n + 1$ input images implies a sequence of animations between every two images, resulting in n sequences of

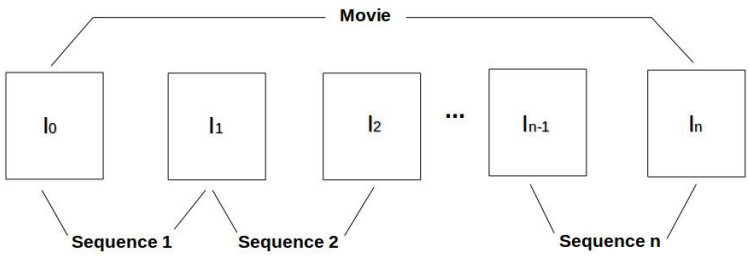


Figure 3.4: Morphing among multiple images

frames. Assuming we have a series of source images called I_0, I_1, I_2, ..., I_n. We can create a long movie consisting of the morphing from I_0 to I_1, then I_1 to I_2, then I_2 to I_3, and so on. Each sequence is obtained by morphing between two images with the algorithm explained before. After finishing morphing between every two images, we concatenate all the sequences together to obtain the entire morphing sequence. This process is illustrated in Figure 3.4.

3.6 Handling Color Images

The Beier-Neely image-morphing algorithm can be applied to both greyscale and color images. When handling color images, we apply blending for each of the color components respectively in the image-blending process. In the warping process, we copy each of the color components from one pixel to another corresponding pixel. Greyscale images contain only one component associated with the intensity information, so we only need to consider this component in the image-blending and warping processes.

Chapter 4

Software

4.1 Introduction

The main contribution of this project is developing software that implements the Beier-Neely feature-based image-morphing algorithm. In the previous chapters, we have introduced the algorithm thoroughly. Now, we will introduce the structure of the software that we developed as well as explain how to build and run the software in order to perform image morphing.

4.2 Overview

The image-morphing software that we developed allows the generation of a morphing animation using two or more user-specified image files. Notice that the software supports PPM [16] format color images. The three programs in the software are as follows:

- `select_features`
- `morph_images`
- `frames_to_video`

We first run the `select_features` program to select feature line segments for each input image file. The `select_featues` program provides a graphical user interface (GUI) that can be used to load source and target images, and specify feature line segments for each image. The specified feature line segments for each image could be saved. Then we run the `morph_images` program with the input images and their corresponding feature line segments to produce a sequence of intermediate frames. While the `morph_images` program

is running, information such as the number of images read, the current progress, and the time consumption can be printed. Lastly, the user uses the `frames_to_video` program to convert the sequence of intermediate frames to a video. The result of our image-morphing software would be a video that displays the morphing visual effect for the user-specified images and feature line segments.

4.3 Prerequisites

The image-morphing software should work on most Linux systems with a C++ compiler that supports C++11. The compiler version on the author's machine is GCC 4.8.2 [8]. So, this version is capable of compiling the programs in the morphing software. The software also uses libraries such as SPL [1], CGAL [4], OpenGL [15], and GLUT [9]. In addition, another free software FFmpeg [7] is used to convert image sequences to video. The libraries mentioned above must be installed before one can build and use our software. The versions of the tools that have been verified to work with our software are:

- GCC 4.8.2
- SPL 1.1.15
- CGAL 4.5.2
- OpenGL/GLUT 3.0
- FFmpeg 2.5.3

4.4 Building the Software

Before using the software, it must first be built. The Make [13] utility is used for automatically building executable programs from source code. This program will look for a file named Makefile [14], which specifies how to compile and link source files to generate target programs. The user should first set the current working directory to the directory of the Makefile. The user might also need to modify the Makefile included in the software package (e.g., set some variables at the start of the Makefile based on the library installation on the user's system). Then, the user should delete all the object files and executable files generated from any previous building processes by running the command:

```
make clean
```

Then, the user can generate the executable programs using the command:

```
make
```

4.5 Feature Data File Format

Before introducing the programs in our software, we explain the feature data file format used in the implementation. Recalling the description of the Beier-Neely method in Chapter 3, we know that the features in images are specified as directed line segments. In order to store the features in a file for the morphing program to use, we defined a feature data file format. A feature data file stores a list of feature line segments which are selected from the corresponding image. The first line of the file is the number of features contained in the list. From the second line to the last, each line refers to a feature line segment. Four whitespace separated fields are used to store the coordinates of the two endpoints of a directed line segment. Figure 4.1 is an example of feature data file. As we can see, the first entry is 9, which indicates that nine feature line segments are stored in the file. In this example, the second entry represents a feature line segment PQ with its endpoints coordinates $P(125, 325)$ and $Q(147, 338)$.

4.6 The `select_features` Program

The `select_features` program is a GUI, where a user can manually select feature line segments and save them for use in the morphing process. Generally, a window will be created to display the source and target images for morphing, and then the user can draw feature line segments on the images by mouse clicking, dragging, and releasing. Figure 4.2 shows an example of the appearance of the graphics window when the program runs with two particular test images (a circle and a square).

During the selection process, the user can edit the data using the 'Undo' and 'Clear' buttons, and save the data in the left and right windows by clicking the 'Save' button. At anytime during the selection process, the user could quit the program by typing 'q' on the keyboard. By clicking 'Save' button followed by typing 'q', the user could quit the program with saving the data. The data would not be saved if the user types 'q' only. After all the feature line segments have been selected, the user could use Enter key to save the feature line segments and exit the program. After the user has finished this process, each image would have a corresponding data file storing the image's feature line segments.

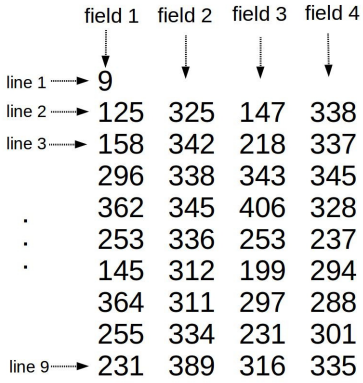


Figure 4.1: Sample feature data file

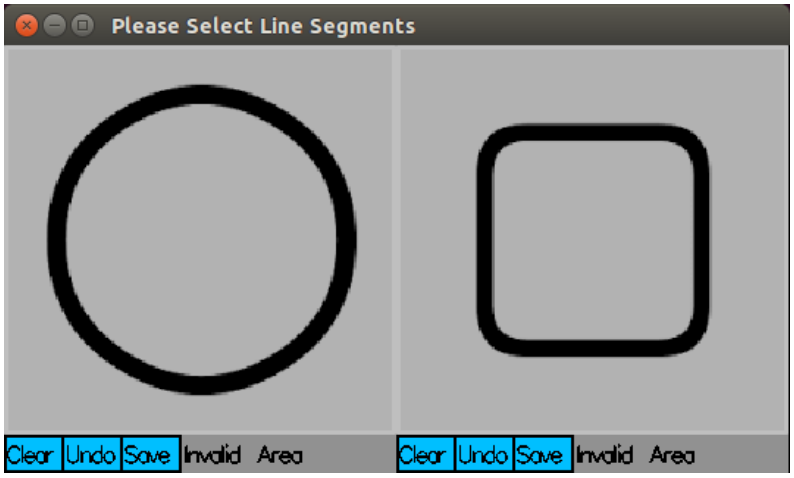


Figure 4.2: Screenshot of the graphics window produced by `select_features` program

4.6.1 Command-Line Interface

SYNOPSIS

`select_features [options]`

OPTIONS

`-i $Image`	Specifies an image for processing to be in the file named `$Image`. This option is required and should be specified twice. The first image file specified would be accepted as the source image, and the second image specified would be accepted as the target image.
`-f $featureSet`	Specifies a feature data file to be `$featureSet`. Feature line segments selected by the user would be saved to this specified file. If the file already exists with feature data, the feature line segments stored in the file will be loaded when the program starts. This option is required and should be specified twice. The first feature data file specified would be accepted as the source image feature data file, and the second feature data file specified would be accepted as the target image feature data file.

4.6.2 How to Use the GUI

The GUI displays one image on the left and the other image on the right, with their original aspect ratio. To make sure all the details in the images can be seen clearly, the user can adjust the window size. The user can draw feature line segments on the images by mouse clicking, dragging, and releasing. Mouse clicking triggers a starting point of a feature line segment, dragging the mouse changes the feature line segment's direction and length dynamically, and releasing the mouse terminates the current feature line segment. Notice that the feature line segments should be carefully selected to represent the features of an object, rather than randomly selected. In addition, there is a one-to-one correspondence between source image features and target image features, which means a feature in the source image must have a corresponding feature in the target image. Figure 4.3 shows the case when 36 features are selected from the source and target images. Each arrow is a directed line segment that

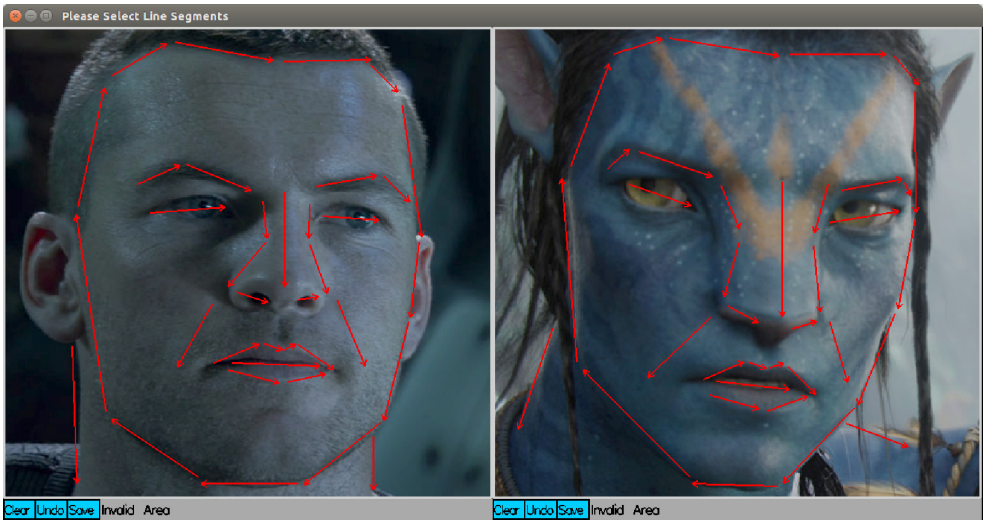


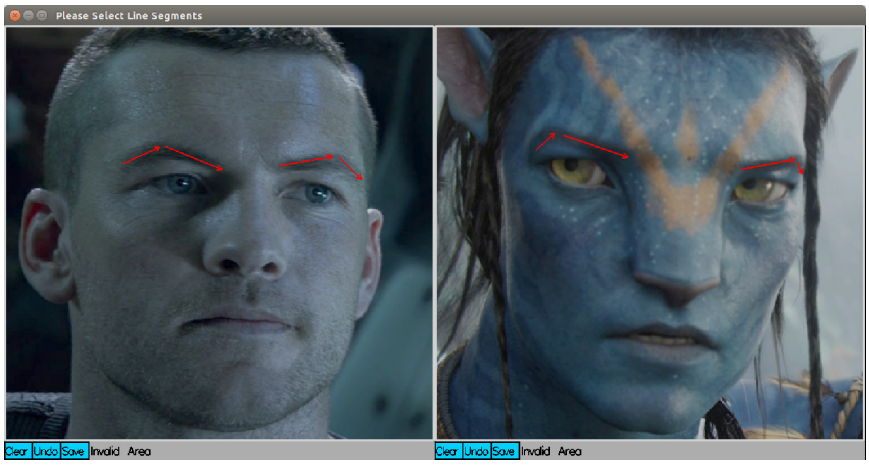
Figure 4.3: Screenshot of the graphics window during feature line segment selection process

represents a feature of the face.

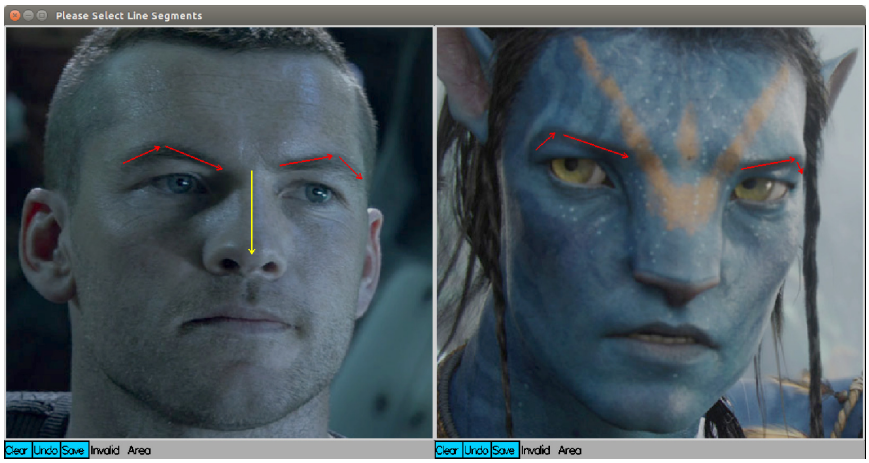
To supplement the above description, a video demonstration on how to use the program can be found online at:

`https://www.youtube.com/watch?v=ItXOV5-8JBY`

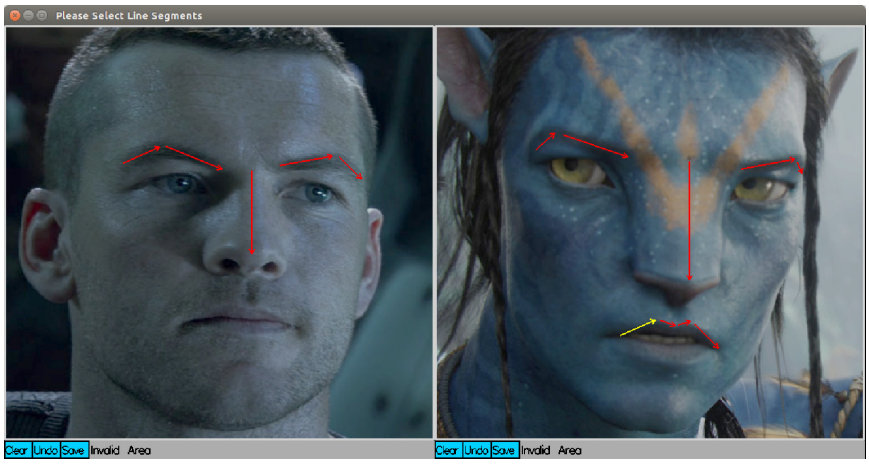
As mentioned previously, the feature line segments in the source and target images have a one-to-one correspondence. Sometimes the user has selected many feature line segments for one image before selecting ones for the other image. Therefore, the order of the feature line segments might be forgotten easily. To help the user select features in the same order for both images, a highlighted arrow (yellow) is provided for the user's convenience. By checking where the highlighted arrow appears, the user knows where the next feature should be drawn on the other image. To further explain this feature-correspondence issue, we provide three examples shown in Figure 4.4. Figure 4.4 shows the screenshot of the graphics window under three conditions. From Figure 4.4(a), we can see both images have four features selected, so no highlighted arrow is shown on the images. From Figure 4.4(b), we can see a highlighted arrow along the nose on the left image, indicating that the left image has more features currently selected. If the next selection is to be on the right image, it should also refer to the feature associated with the nose. Figure 4.4(c) shows another example where a highlighted arrow is drawn on the right image, indicating that more features have been selected on the right image and the next feature to be selected for the left image should be for the same feature on the lip.



(a)



(b)



(c)

Figure 4.4: Screenshot of the GUI in three cases. (a) The same number of features selected in the two images. (b) More features selected on the left image. (c) More features selected on the right image.

4.7 The `morph_images` Program

The `morph_images` program reads in multiple input images in PPM format and their corresponding feature data files. Each feature data file contains feature line segments selected for the corresponding image.

4.7.1 Command-Line Interface

SYNOPSIS

```
morph_images [options]
```

OPTIONS

`-i $image` Specifies an image file for morphing to be the file named `$image`. This option is required and should be specified at least twice. The morphing would start from the first image to the last image in the order we specify the image files.

`-f $featureFile`
Specifies a feature data file for an image to be `$featureFile`. This option is required and should be specified at least twice with the one-to-one correspondence between features in the files specified.

`-n $frameNum` Specifies the number of frames generated between two successive images to be `$frameNum`. The option can be specified multiple times and the order of this option matters since the number of frames can be different between different images. This option is required.

`-b $baseName` Specifies the base name of the output frames to be `$baseName`. This option is required.

`-d $digits` Specifies the digits for the file name of each intermediate frame to be `$digits`. For example, in a file named basename_xxxx.ppm, the option `-d` is specified as 4. This is optional. The default value is associated with the number of frames (e.g., the `-n` option). For example, when the frame number goes from 0 to 99, the `-d` will be specified as 2.

`-a` `$param_a` Specifies the a parameter for warping (refer to (3.5)) to be `$param_a`. This is optional. The default value is 1.0. Note that the default value does not guarantee good morphing quality.

`-m` `$param_m` Specifies the b parameter for warping (refer to (3.5)) to be `$param_m`. This is optional. The default value is 1.0. Note that the default value does not guarantee good morphing quality.

`-p` `$param_p` Specifies the p parameter for warping (refer to (3.5)) to be `$param_p`. This is optional. The default value is 0.5. Note that the default value does not guarantee good morphing quality.

`-s` `$save_flag`
Sets the flag indicating if frames should be saved to `$save_flag`. The option can be specified as 0 or 1. This option is by default 1, which enables the saving of frames.

`-g` `$dsp_flag` Sets the flag indicating if the morphing result should be displayed to `$dsp_flag`. The option can be specified as 0 or 1. This option is by default 1, which enables the displaying of the morphing result.

`-t` `$time_flag`
Sets the timing flag to `$time_flag`. The timing flag indicates if the time consumption should be printed. The option can be specified as 0 or 1. This option is by default 1, which enables printing the timing information.

4.8 The `frames_to_video` Program

The `frames_to_video` program is used to convert a sequence of morphed images to a video. This program creates video files using the Ffmpeg [7].

4.8.1 Command-Line Interface

SYNOPSIS

```
frames_to_video [options]
```

OPTIONS

`-i`	`$basename`	Sets the base name for input files to be `$basename`. This option is required.
`-o`	`$out_file`	Sets the output file name to be `$out_file`. This option is required.

4.9 Software Usage Examples

So far, we have introduced each of the programs in our software. The programs are executed in this order:

- `select_features` (line-segment selection program)
- `morph_images` (image-morphing program)
- `frames_to_video` (convert-to-video program)

In what follows, we present several examples as a reference for the usage of the software.

4.9.1 First Software Usage Example

Suppose that we have a source image file named `source.ppm` and a target image file named `target.ppm`. We want to select some feature line segments for each image, and then save the feature line-segment files as `sourceFeature.dat` and `targetFeature.dat` separately. This task can be accomplished by running the `select_features` program as follows:

```
select_features -i source.ppm -f sourceFeature.dat -i \
target.ppm -f targetFeature.dat
```

Then we generate 50 frames between these two images based on `sourceFeature.dat` and `targetFeature.dat`, and each frame has a basename `frame`. The frames are to be stored in a folder called `output` under the current directory. The parameters used for the warping function are default values. We want to display the morphing result when the morphing is completed. This task can be accomplished by executing the following command:

```
morph_images -i source.ppm -f sourceFeature.dat -i target.ppm \
 -f targetFeature.dat -n 50 -b output/frame -g -s
```

Once the preceding command completes, all frames have been generated and saved. Next, we want to create a MP4 video file named `out.mp4`. This task can be accomplished by executing the following command:

```
frames_to_video -i output/frame -o out.mp4
```

4.9.2 Second Software Usage Example

Suppose that we have a source image file named `image1.ppm`, a target image file named `image2.ppm`, and a feature data file named `feature1.dat` for the source image. We only need to select features for the target image and then save them as a file named `feature2.dat`. This task can be accomplished by running the program as follows:

```
select_features -i image1.ppm -f feature1.dat -i image2.ppm \
-f feature2.dat
```

Then we want to generate 100 frames between these two images based on `feature1.dat` and `feature2.dat`. The a, b, and p parameters used for warping are 0.8, 1.2 and 1.0, respectively, instead of the default values. Each frame will have a basename `figure`. We also want the time consumption information printed to screen. This task can be accomplished by executing the following command:

```
morph_images -i image1.ppm -f feature1.dat -i image2.ppm -f \
feature2.dat -n 100 -b figure -s -t -a 0.8f -m 1.2f -p 1.0f
```

After all frames are generated, we want to create a MP4 video named `animation.mp4`. This task can be accomplished by executing the following command:

```
frames_to_video -i figure -o animation.mp4
```

4.9.3 Third Software Usage Example

Suppose that we have a sequence of image files named `image1.ppm`, `image2.ppm`, ..., `image5.ppm` in the folder `group_photos` under the command directory. Their feature data files are also provided: `feature1.dat`, `feature2.dat`, ..., `feature5.dat`. Each morphing between two images contains 50 frames with names like `frame_0006.ppm` in a folder named `out` under the current directory. The parameters for warping are to be the default values. This task can be accomplished by running the following command:

```
morph_images -i image1.ppm -f feature1.dat -i image2.ppm -f \
feature2.dat -i image3.ppm -f feature3.dat -i image4.ppm -f \
feature4.dat -i image5.ppm -f feature5.dat -n 50 -d 4 \
-b out/frame -s -t
```

After all frames are generated, we want to create a video named `animation.mp4`. This task can be accomplished by executing the following the command:

```
frames_to_video -i frame -o animation.mp4
```

Chapter 5

Results and Analysis

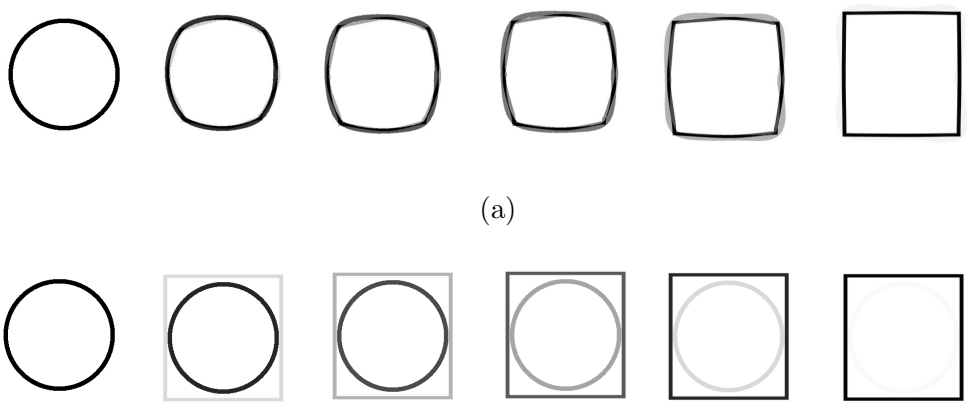
5.1 Overview

In this chapter, we first provide several image-morphing results produced by our software, verifying its effectiveness and satisfactory performance. Then, we perform some experiments to analyze the Beier-Neely method. The first part of this experimental evaluation concerns the three parameters a, b, and p appearing in (3.5). In particular, we consider how these parameters affect the quality of morphing. Next, we examine how the time complexity of the Beier-Neely algorithm is affected by the number of frames, image size, and number of features. Lastly, some useful suggestions on how to obtain high-quality morphing results are given.

5.2 Examples of Morphing Results

This section gives two examples of image-morphing results produced by our software. The first example is shown in Figure 5.1. Figure 5.1(a) shows the morphing from a circle to a square. Figure 5.1(b) is a sequence of cross-dissolved only images. From Section 2.3, we know that cross-dissolving is a blending of images with a changing weight and has a very poor visual effect, so we can confirm the effectiveness of our morphing result by giving Figure 5.1(b) as a comparison. As we can see in Figure 5.1(b), while the circle gradually fades out and square fades in, the shapes of the circle and square remain unchanged. The image sequence in Figure 5.1(a) is considerably more effective, since the contour of the circle is gradually narrowed and sharpened to a square.

The preceding example shows the morphing result with very simple source and target



(a)

(b)

Figure 5.1: Morphing from circle to square. (a) Morphing result. (b) Cross-dissolving result.

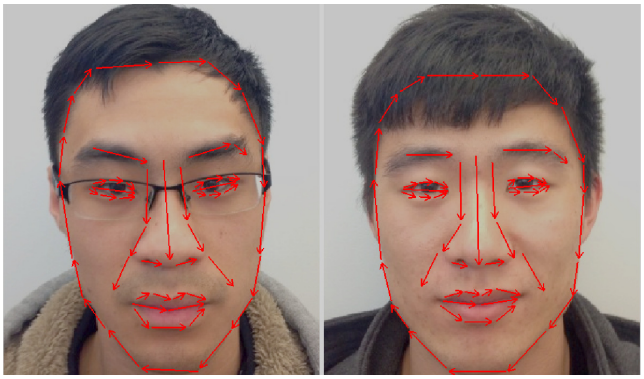
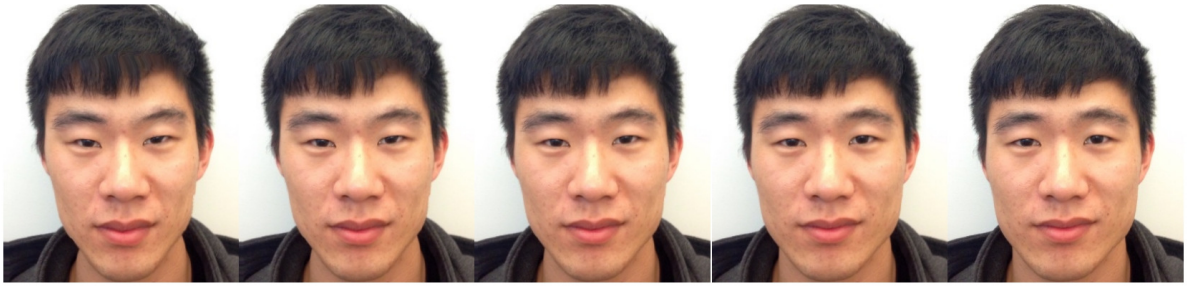


Figure 5.2: Two face images with corresponding feature line segments selected

images. To further demonstrate the effectiveness of our image-morphing software, we need to perform experiments on more complex images with more feature information. Therefore, in the next example, we consider morphing human face images shown in Figure 5.2.

To begin, we specify the source and target images with their respective feature line segments. As is shown in Figure 5.2, 53 line segments are selected corresponding to numerous facial features. Having selected the features, we run our program to perform morphing. Figure 5.3 shows the morphing result, including some intermediate results. Figure 5.3(a) shows a sequence of warped source images, Figure 5.3(b) shows the corresponding sequence of warped target images, and Figure 5.3(c) shows the morphing result. In Figure 5.3(c), each image is also a blending of the corresponding images in Figure 5.3(a) and (b). The result

(a)



(b)

(c)

Figure 5.3: Face morphing. (a) A sequence of warped source images. (b) Corresponding sequence of warped target images. (c) Result of the facial morphing.

is a very convincing morph that can be seen by the realism of the images in the morphing sequence. In the warping of the source image, the most significant reconfiguration is the eyebrow line which has been straightened through the warping. Another apparent change is that the hairline of the face in the source image has been lowered while the forehead has been narrowed. In the warping of the target image, the most significant change would be in the contour of the face on the target image, which has now been elongated and narrowed to create a slender shape. On close inspection of the centre image in Figure 5.3(c), we can see the collar shape of the source image is not changing to the other gradually and naturally. This could be improved by adding more feature line segments around the collar area. The video of this morphing example is available online at:

`https://youtu.be/C_DX8CD7JHc`

5.3 Hardware and Software Setup

Before analyzing the time complexity of the Beier-Neely algorithm, we briefly introduce the hardware and software setup used for timing. The system used has the following characteristics:

- Processor: Intel Core i7-4510U CPU @ 2.00GHz×2
- Memory: 979.8 MiB
- Graphics: Gallium 0.4 on SVGA3D
- OS type: 64-bit
- Version of GCC: 4.8.2
- Compiler optimization level: `-O3`

5.4 Effect of Parameters

Recalling (3.5), we know that three parameters a, b, and p, determine the visual effect of image warping. To analyze the effect of the parameters, we run three tests with different values assigned to the three parameters. In each of the experiments, one of the three parameters takes its highest value while the other two parameters are each assigned their respective lowest values. Figure 5.4 shows the result of our first experiment with parameters:

$a = 2.0$, $b = 0.5$, and $p = 0.0$. In this experiment, parameter a takes its highest value, while b and c each take their respective lowest values. Figure 5.5 shows the result of our second experiment with parameters: $a = 0.1$, $b = 2.0$, and $p = 0.0$. In this experiment, parameter b takes its highest value, while a and c each take their respective lowest values. Figure 5.6 shows the result of our third experiment with parameters: $a = 0.1$, $b = 0.5$, and $p = 1.0$. In this experiment, parameter c takes its highest value, while a and b each take their respective lowest values.

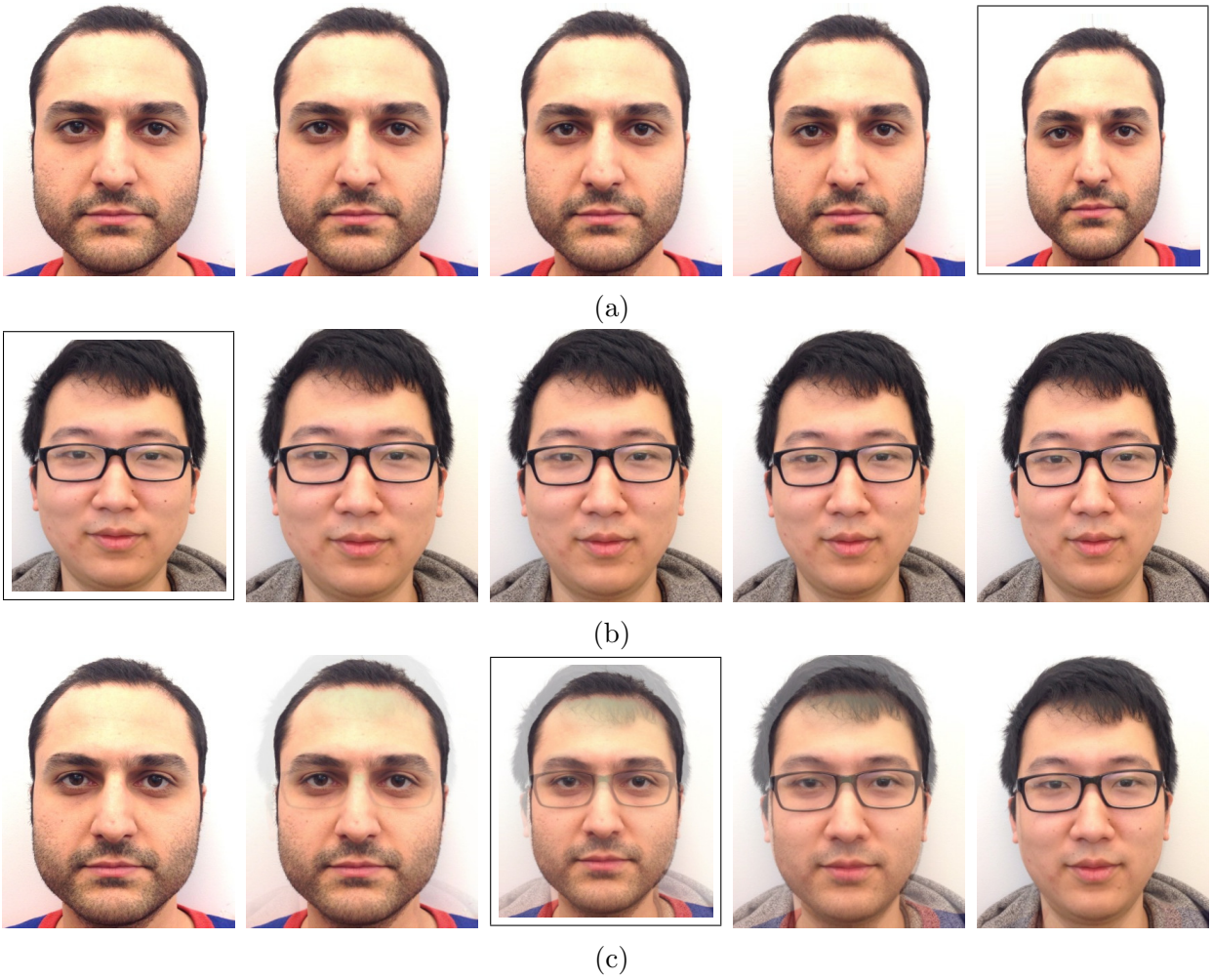
In each of Figures 5.4, 5.5, and 5.6, the three images appearing in boxes are the ones showing the most significant visual effects. From the warped images in each of the three experiments, we can see that having a high value of a or p in the first and third experiments causes very little facial warping, while a high value of b has the most effect. This is directly related to the quality of the warping, which is much more significant in the second experiment where b is the highest. From the previous explanation about the mathematics in (3.5), we know that a large b value has a dramatic influence on the shortest distance between a pixel and a feature line segment, so the morphing results shown in our experiments match the theory. Through further experiments, we are able to conclude that a good choice of parameters for face morphing is typically given by: $a = 1.0$, $b = 2.0$, and $p = 0.5$.

5.5 Time-Complexity Analysis

The time complexity of Beier-Neely algorithm is $O(n), O(p), O(w)$, where n is the number of feature line segments, p is the number of pixels in the source and target images, and w is the amount of computation required for one pair of feature line segments [2]. In what follows, we analyze the time complexity experimentally.

5.5.1 Effect of Number of Frames

First, we consider the relationship between the number f of frames to be generated and the corresponding time consumption t. In this experiment, the source and target images employed were of size 512 by 512 and 31 feature line segments were selected. We ran the morphing software ten times with the number of frames set to 0, 10, 20, ..., 90, and recorded the elapsed time in each case. The other parameters used in morphing were fixed. Figure 5.7(a) shows a graph of the results. As we can see from the graph, the relationship between time consumption t and the number f of frames is a linear function that can be described approximately as $t = 1.4f$.

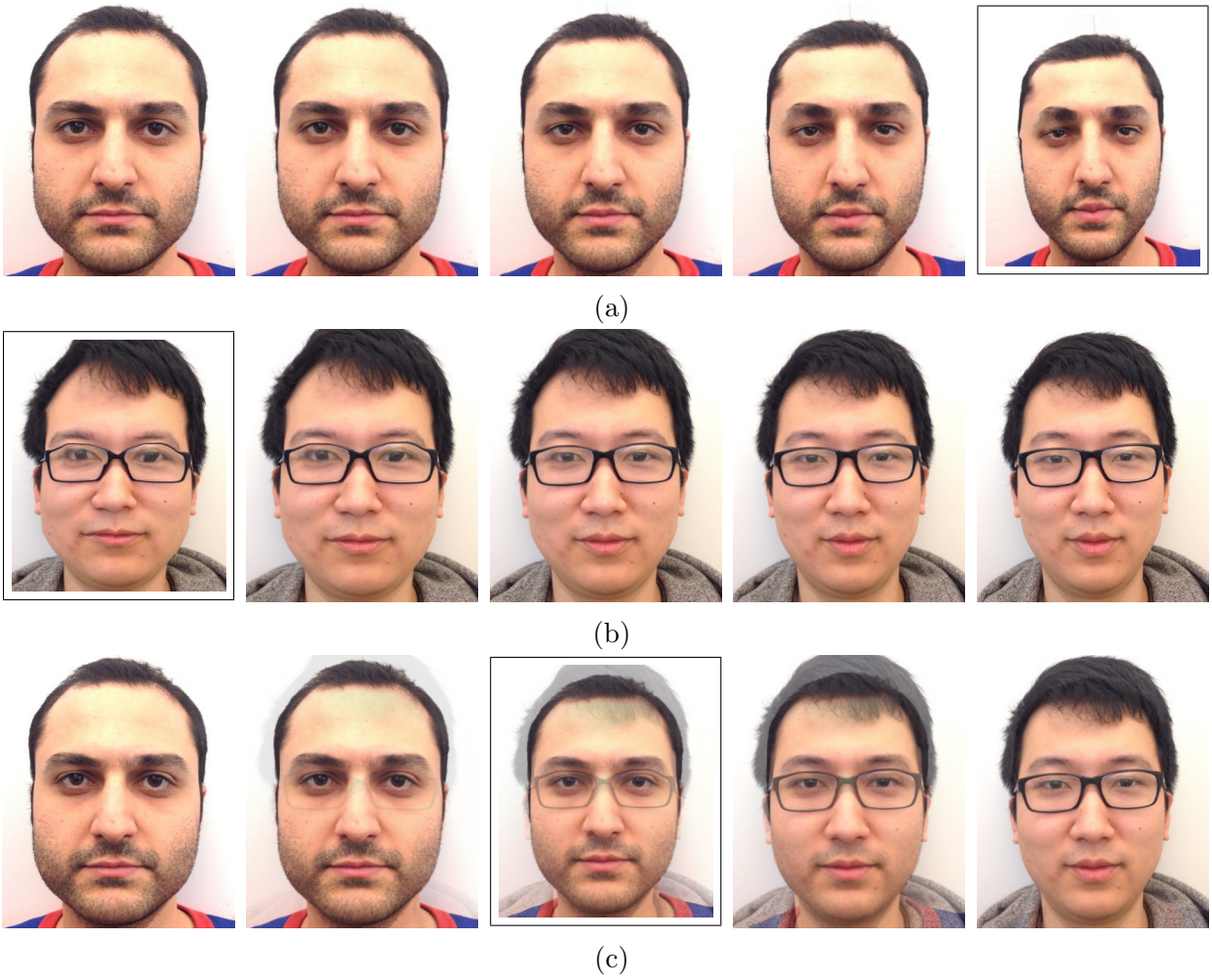


(a)

(b)

(c)

Figure 5.4: Morphing with parameters: $a = 2.0$, $b = 0.5$, and $p = 0.0$. (a) The sequence of warped source images, (b) the corresponding sequence of warped target images, and (c) the resulting sequence of blended images.

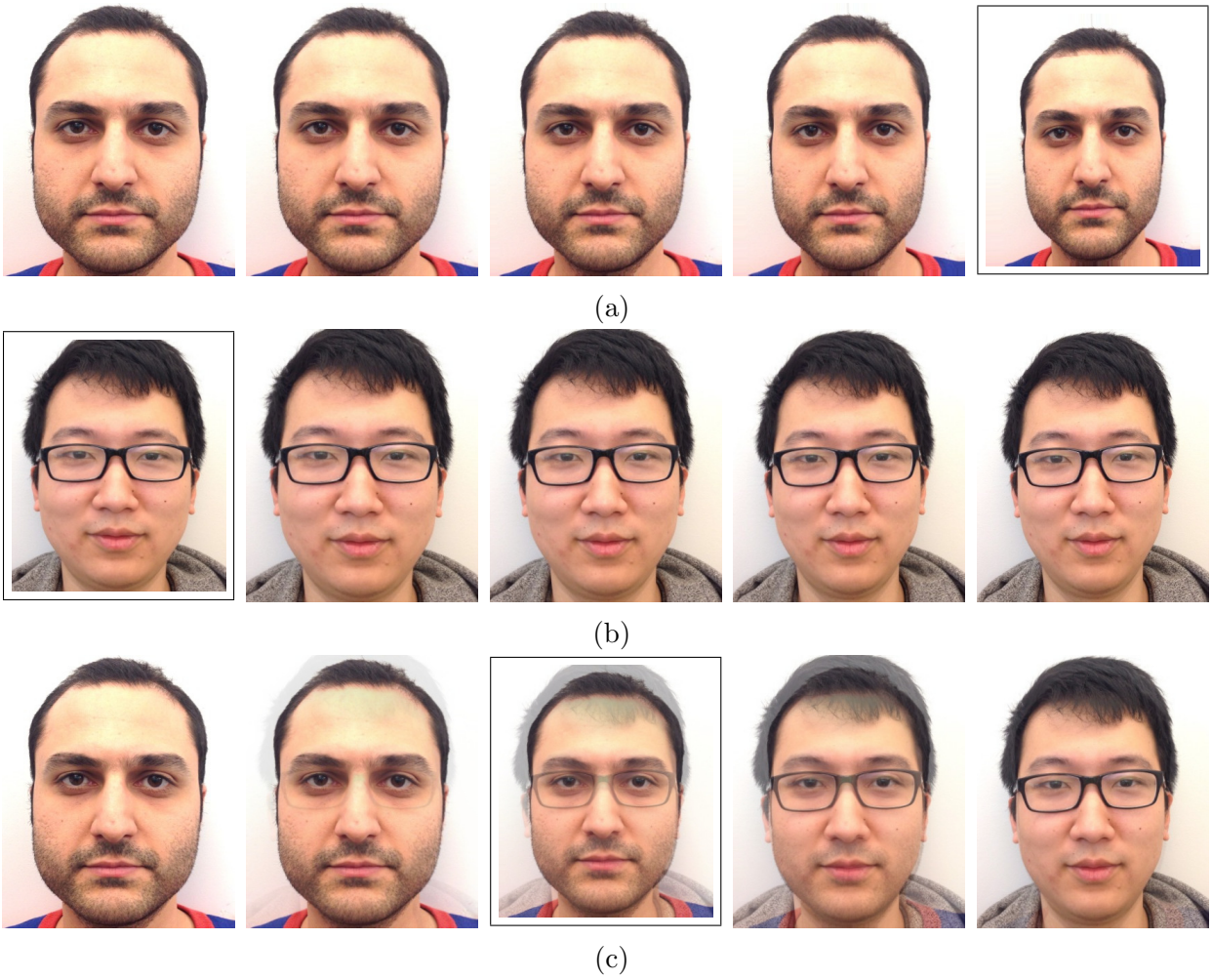


(a)

(b)

(c)

Figure 5.5: Morphing with parameters: $a = 0.1$, $b = 2.0$, and $p = 0.0$. (a) The sequence of warped source images, (b) the corresponding sequence of warped target images, and (c) the resulting sequence of blended images.

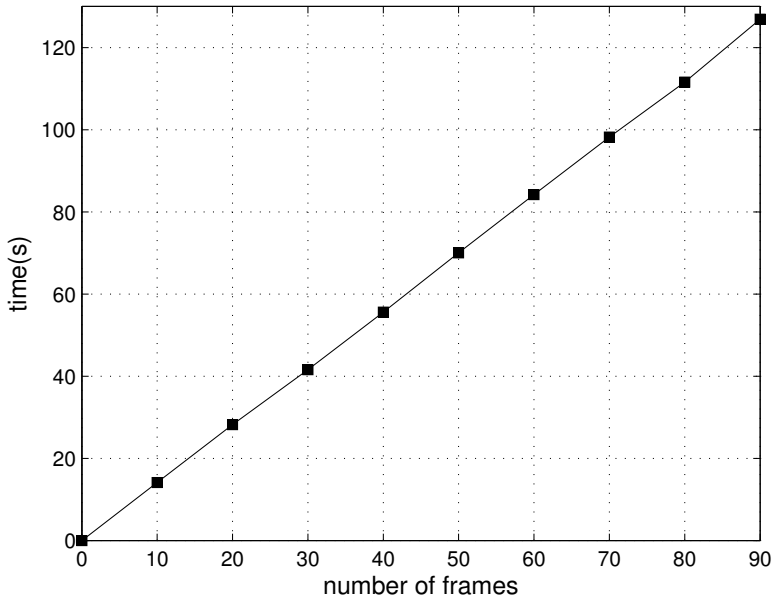


(a)

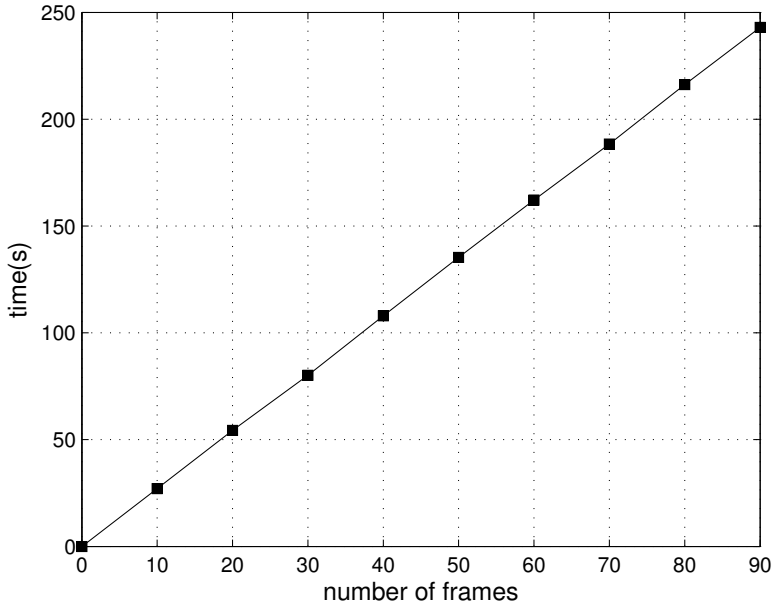
(b)

(c)

Figure 5.6: Morphing with parameters: $a = 0.1$, $b = 0.5$, and $p = 1.0$. (a) The sequence of warped source images, (b) the corresponding sequence of warped target images, and (c) the resulting sequence of blended images.



(a)



(b)

Figure 5.7: Time consumption versus the number of frames generated. (a) An experiment with images of size 512 by 512 and 31 features. (b) An experiment with images of size 540 by 600 and 56 features.

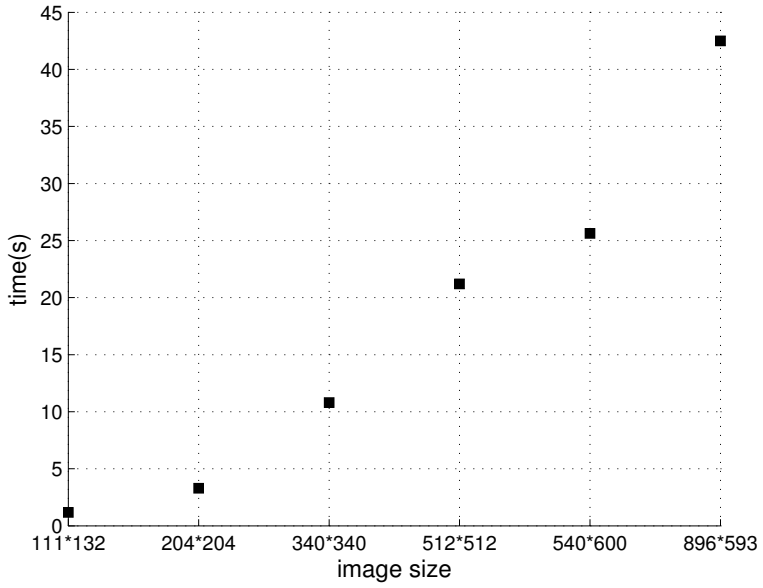


Figure 5.8: Time consumption versus image size.

To further support our conclusion, we conducted another experiment where the source and target images used were of size 540 by 600 and 56 feature line segments were employed. We also ran the program ten times with number of frames set to 0, 10, 20, ..., 90, and recorded the elapsed time in each case. Figure 5.7(b) shows a graph of the results. From the trend, we can describe the mathematical relationship approximately as $t = 2.7f$.

From the two experiments above, we can conclude that the time consumption t and the number f of frames are linearly related. That is, we have $t = kf$, where k is a constant related to the size of image and the number of features.

5.5.2 Effect of Image Size

Next, we explore the relationship between time cost t and the image size s in pixels. In this experiment, we ran the image-morphing program six times with differing image sizes but the same number of frames and the same number of features. The other parameters used in morphing were fixed. As is shown in Figure 5.8, the relationship between time complexity and image size is proportional, which indicates that a larger image costs more time to finish the morphing. To quantify the result, we express the image size in terms of the total number of pixels. If the size of generated frame is W by H, the number of pixels is $N = WH$. In each experiment, we divided the time consumption t by the number of pixels N to calculate

the average time consumption per pixel $T = t/N$. The results are shown in Figure 5.9.

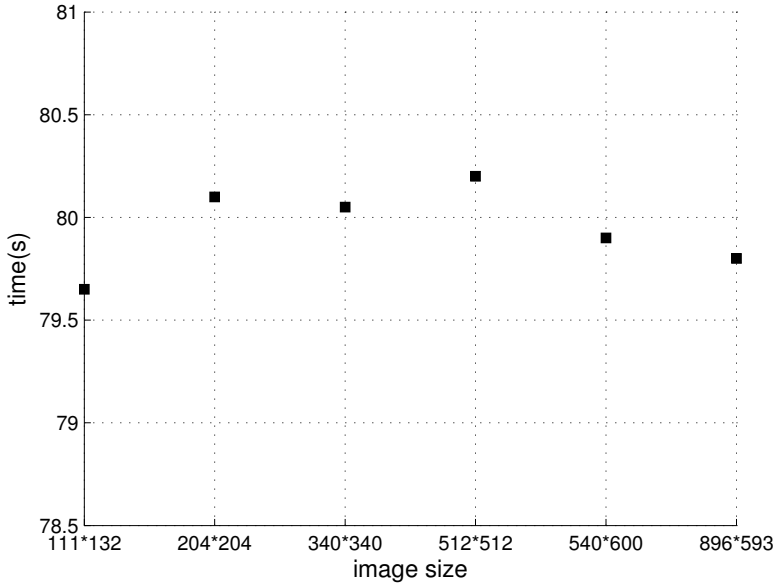


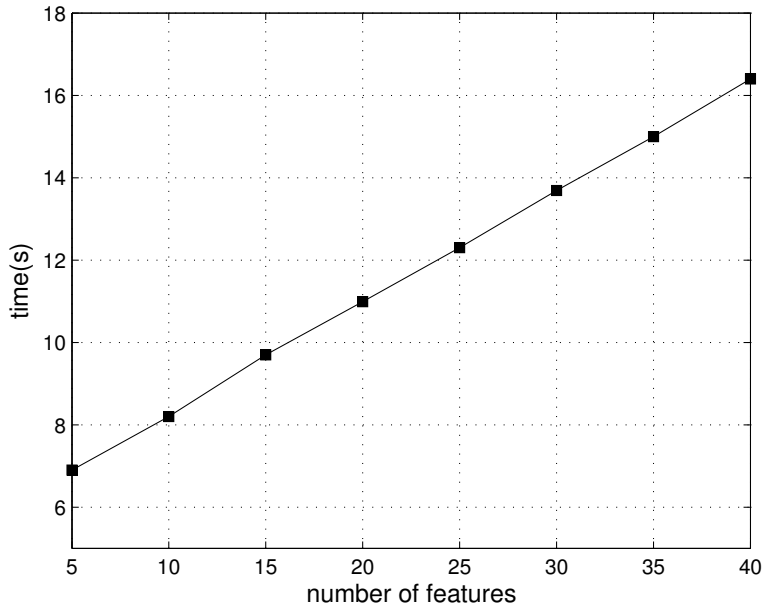
Figure 5.9: Time consumption per pixel for generated frames

We can see that the time cost on each pixel is a constant of about 8 μs/pixel. To evaluate the constant, we repeated the six experiments on the same images with different number of features used. The result is that the constant changes over the different number of features. Thus, it can be concluded that the time consumption for each pixel is a constant c related to the number of features and frames.

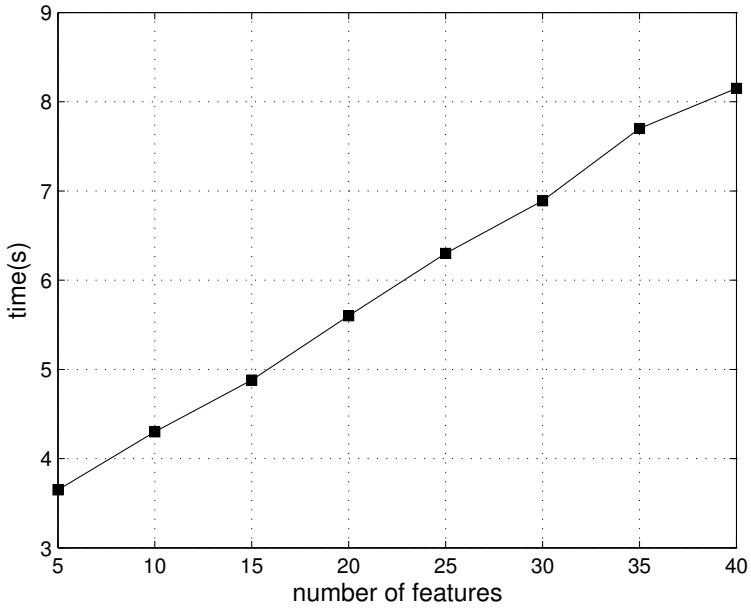
Based on this observation, we can further conclude that the relationship between time consumption t and image size is also a linear relation that can be described as $t = cN$, where N is the total number of pixels, and c is a constant decided by the number of features and frames.

5.5.3 Effect of Number of Features

Next, we consider the relationship between the time consumption t and the number n of features used for morphing. In this experiment, the images we used were of size 512 by 512 and the number of frames was set to 10. The other parameters used in morphing were fixed. We ran the software several times with the number of features set to 5, 10,..., 40, and recorded the elapsed time in each case. Figure 5.10(a) shows the time consumption of the



(a)



(b)

Figure 5.10: Time consumption versus the number of features. (a) An experiment with images of size 512 by 512 and 10 frames generated. (b) An experiment with images of size 340 by 600 and 10 frames generated.

eight experiments. As we can see, the relationship between time cost t and number n of features is a linear relationship that can be approximately described as: $t = 0.27n + 5.57$, where the two coefficients were calculated based on the values of the points on the chart.

To again show the linear relationship, we repeated the experiment on a different pair of source and target images. Figure 5.10(b) shows the time consumption of eight experiments with different number of features. The images used were of size 340 by 600, and the number of frames was set to 10. We ran the software several times with number n of features set to 5, 10,..., 40, and recorded the elapsed time in each case. From the graph, it is clear that the relation between the time consumption and the number of features is also linear. From the numerical results, we can describe the relationship as: $t = 0.13n + 3$.

By analyzing the two examples above, we can conclude that the relation between time consumption t and the number n of features used is a linear relationship $t = kn + b$, where k and b are two coefficients decided by the image size and the number of frames to be generated.

5.6 Obtaining High-Quality Morphing

To achieve a realistic morphing result, the user should carefully choose the source and target images as well as the feature line segments. In what follows, we will make several suggestions on how to obtain high-quality morphing results.

5.6.1 Choosing Appropriate Images

To obtain good morphing results, we should choose the source and target images with similar composition, including background color, object proportion, and the type of object appearing in the image. Too much difference between the source and target images takes much effort to ensure a seamless transformation, and might result in very low-quality morphing. In addition, similar types of objects have similar features so we are able to easily select corresponding features for the objects. For example, morphing a cat's face to a coffee mug is irrational since they have very different features, while morphing a cat's face to a dog's face is reasonable since the two objects have similar features.

Figure 5.11 shows an example in which the source and target images are of vastly different types, resulting in a low-quality morph. As we can see, the two ears of the cat approach to each other, the shape of the face narrows, and a handle grows on the right side of the face. The mug, however, has no obvious change, and therefore the visual effect of this morphing is poor. This is caused by the significant difference between the cat's face and the

Figure 5.11: Morphing from a cat to a mug

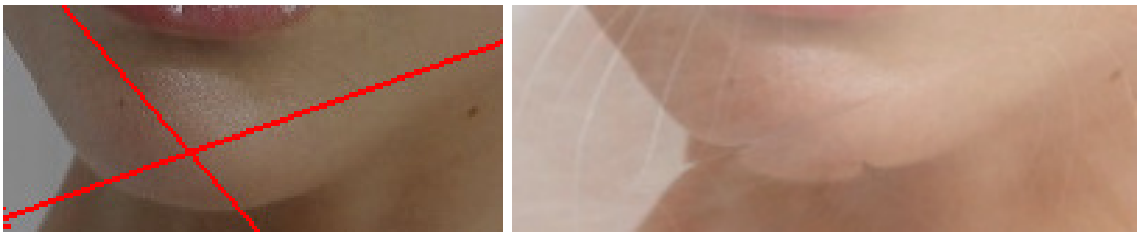
mug. During the feature selection process, it is extremely difficult to choose corresponding features between these two objects. For example, we can select a feature near the nose of the cat, but we are not able to select a corresponding feature for the mug since the mug does not have a nose. The video of this morphing example can be found online at:

`https://youtu.be/-LxRXvyceko`

5.6.2 Choosing Appropriate Feature Line Segments

To obtain good morphing results, we should also keep a correct correspondence between feature line segments for all images. This consistency can be guaranteed by always following the highlighted arrow appearing in the GUI. A detailed instruction on feature selection has been presented in Section 4.6.2. Besides keeping a correct correspondence, we should make sure that feature line segments selected do not overlap or intersect with each other. Any intersection or overlap might cause problems in the line-segment interpolation process, which might lead to distortion in some areas in the resulting images. Figure 5.12 shows a distortion problem caused by line intersection. From Figure 5.12(a), we can see two feature line segments intersect with each other in the lower jaw area. Figure 5.12(b) shows a frame from the morphing sequence with the intersected feature line segments, and we can see an obvious distortion in the jaw area.

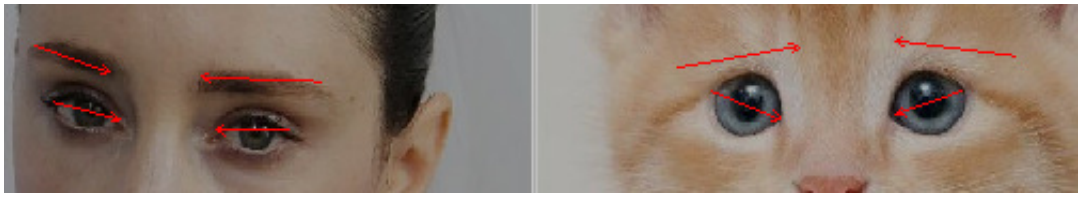
For good morphing results, we should also select enough features to ensure the smoothness of morphing. The more features are used, the more image details are considered, leading to a smoother morphing. Figure 5.13 shows the morphing results with different number of features selected for the same source and target images. From Figures 5.13(a) and (b), we can see four and eighteen features are selected in the eye area, respectively. Figures 5.13(c) and (d) show the morphing results produced from the cases in Figures 5.13(a) and (b),



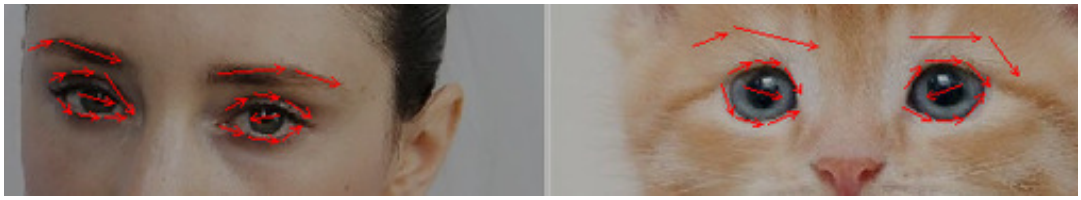
(a) (b)

Figure 5.12: Distortion caused by intersected feature line segments. (a) Two feature line segments selected in the jaw area. (b) A distortion in the jaw area in the morphed image.

respectively. Both images in Figures 5.13(a) and (b) come from the middle frame in the morphing sequence. According to our discussion earlier in this section, using more feature line segments will result in a better morphing result. This can be proved by observing Figures 5.13(c) and (d). As we can see, Figure 5.13(d) is more precise than Figure 5.13(c). If we carefully observe the left eye area, Figure 5.13(c) has a "double image" effect while Figure 5.13(d) is a more realistic eye without a "double image" effect. From this experiment, we can conclude that using more features leads to a higher morphing quality. Thus, the user should select as many features as possible to achieve a good morphing result.



(a)



(b)

(c)

(d)

Figure 5.13: Visual effect with different number of features. (a) Four features selected. (b) Eighteen features selected. (c) Middle frame from the morphing sequence in case (a). (d) Middle frame from the morphing sequence in case (b).

Chapter 6

Conclusions

In this project, we have studied the Beier-Neely feature-based image-morphing algorithm. In our work, we have implemented the method in software. Reasonable morphing results have been achieved. From the performance of the software, we can see the algorithm works well to generate a convincing and pleasing morphing visual effect. We also evaluated the performance of the Beier-Neely algorithm by analyzing results produced by our morphing software. The effect of the three parameters for the warping process was explored and discussed. The time complexity of Beier-Neely algorithm was also analyzed by experimenting with different numbers of frames, different image sizes, and different numbers of features. Based on our experimental results, we can conclude the relation between these factors mentioned above and the time complexity is a linear relationship. At last, several useful suggestions on obtaining high-quality morphing results were provided for the convenience of users.

The Beier-Neely feature-based morphing algorithm has been shown to be effective by our software implementation and results obtained with it. The morphing algorithm, however, still has some disadvantages that could be addressed in future work. When morphing among a large number of images, the user has to draw feature line segments in a high number of source and target image files to produce a smooth morphing sequence, which is a time-consuming and tedious task. In future work, some research on automated feature selection could be done to reduce the amount of user input.

Bibliography

[1] M. D. Adams. SPL, Signal Processing Library, 2015. http://www.ece.uvic.ca/~mdadams/SPL.

[2] A. M. Bagade and S. N. Talbar. Image morphing concept for secure transmission of image data contents over internet 1. *Journal of Computer Science*, 6(9):987–992, 2010.

[3] T. Beier and S. Neely. Feature-based image metamorphosis. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 35–42. ACM, 1992.

[4] CGAL, Computational Geometry Algorithms Library, 2015. http://www.cgal.org.

[5] W. H. Farrand and J. C. Harsanyi. Mapping the distribution of mine tailings in the coeur d'alene river valley, idaho, through the use of a constrained energy minimization technique. *Remote Sensing of Environment*, 59(1):64–76, 1997.

[6] A. V. Feciorescu. Image morphing techniques. *Journal of Industrial Design and Engineering Graphics*, 6(1):25–28, 2010.

[7] ffmpeg, ffmpeg tool, 2014. https://www.ffmpeg.org.

[8] gcc, GNU Compiler Collection, 2014. https://gcc.gnu.org/gcc-4.8.

[9] GLUT, OpenGL Utility Toolkit, 2015. https://www.opengl.org/resources/libraries/glut.

[10] J. Gomes. *Warping and morphing of graphical objects*, volume 1, pages 14–15. Morgan Kaufmann, 1999.

[11] S. Lee, G. Wolberg, and S. Y. Shin. Polymorph: Morphing among multiple images. *Computer Graphics and Applications, IEEE*, 18(1):58–71, 1998.

[12] A. Lerios, C. D. Garfinkle, and M. Levoy. Feature-based volume metamorphosis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 449–456. ACM, 1995.

[13] Make, Make Manual, 2014. `http://www.gnu.org/software/make/manual`.

[14] Makefile, Makefile Conventions, 2014. `https://www.gnu.org/prep/standards/html_node/Makefile-Conventions.html`.

[15] OpenGL, Open Graphics Library, 2015. `https://www.opengl.org`.

[16] PPM, portable pixmap format, 2013. `http://netpbm.sourceforge.net/doc/ppm.html`.

[17] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 533–540. ACM, 2006.

[18] D. B. Smythe. A two-pass mesh warping algorithm for object transformation and image interpolation. *Rapport technique*, 1030:31, 1990.

[19] G. Wolberg. *Digital image warping*, volume 10662. IEEE computer society press Los Alamitos, 1990.

[20] G. Wolberg. Image morphing: a survey. *The visual computer*, 14(8):360–372, 1998.