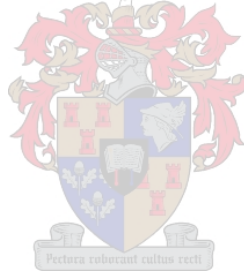


# Morphing in Two Dimensions: Image Morphing

**Magdil Delport**

Thesis presented in partial fulfilment of the requirements for the degree of  
Master of Science at Stellenbosch University.



SUPERVISORS: Prof Ben Herbst, Dr Karin Hunter

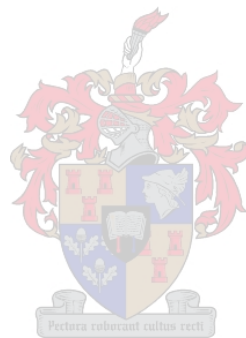
December 2007

# Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature:.....

Date:.....



Copyright © 2007 Stellenbosch University

All rights reserved

# Abstract

Image morphing is a popular technique used to create spectacular visual effects, by gradually transforming one image into another. This thesis explains what exactly is meant by the terms “image morphing” / “warping”, where it is used and how it is done. A few existing morphing techniques are described and finally an implementation using Delaunay triangulation and texture mapping is presented.



# Opsomming

"Image morphing" is 'n gewilde tegniek wat gebruik word om skouspelagtige visuele effekte te skep, deur geleidelik een beeld na 'n ander ander beeld te transformeer. Hierdie tesis verduidelik wat presies met die terme "image morphing" / "warping" bedoel word, waar dit gebruik word en hoe dit gedoen word. 'n Paar bestaande metodes word bespreek en ten slotte word 'n implementasie, wat gebruik maak van Delaunay triangulasie en "texture mapping", beskryf.



# Acknowledgements

I would like to thank all the people who helped me with the completion of this thesis.

My supervisors Prof Ben Herbst and especially Dr Karin Hunter who had to put up with the (sometimes ill-timed) tasks of reading, re-reading and correcting of the numerous draft copies. Their help, guidance and motivation did not go unnoticed and is greatly appreciated.

Soné Swanepoel and my father, Thinus Delport, who helped with the proof readings and the tedious task of finding all the small edit-related errors.

And last but not least all the other people who never stopped believing in me and motivated me to not give up.



# Index

CHAPTER1: INTRODUCTION TO IMAGE MORPHING .....	1
CHAPTER 2: AN OVERVIEW OF WARPING AND MORPHING.....	4
2.1 METAMORPHOSIS AND MORPHING.....	4
2.2 IMAGE MORPHING .....	4
2.3 IMAGE WARPING .....	6
2.4 SOME APPLICATIONS OF IMAGE MORPHING .....	8
2.5 SUMMARY.....	10
CHAPTER 3: WARPING (SPATIAL TRANSFORMATIONS).....	11
3.1 DEFINITION OF AN IMAGE WARP.....	11
3.2 FORWARD VS. REVERSE WARPING .....	12
3.2.1 Forward Warping .....	12
3.2.2 Reverse Warping .....	13
3.3 HOMOGENEOUS NOTATION .....	14
3.4 GENERAL TRANSFORMATION MATRIX.....	14
3.5 AFFINE TRANSFORMATIONS .....	15
3.5.1 Scaling .....	17
3.5.2 Shearing.....	18
3.5.3 Rotation.....	19
3.5.4 Translation .....	20
3.5.5 The Inverse of an Affine Transformation.....	20
3.5.6 Composing Affine Transformations.....	21
3.6 PERSPECTIVE TRANSFORMATIONS.....	22
3.6.1 The Inverse of a Perspective Transformation .....	23
3.7 SUMMARY.....	24
CHAPTER 4: RELATED WORK – EXISTING TECHNIQUES .....	25
4.1 DEFINITION OF IMAGES .....	25
4.2 IMAGE MORPHING .....	25
4.2.1 Cross Dissolve .....	28
4.2.2 Mesh warping.....	28
4.2.3 Field Morphing (Feature Based Image Metamorphosis) .....	31
4.2.4 Snakes and Multilevel Free-Form Deformation.....	35
4.2.5 View morphing .....	37
4.2.5.1 Parallel views.....	38
4.2.5.2 Non-Parallel Views .....	40
4.2.5.3 Singular View Configurations .....	42
4.3 TRANSITION CONTROL.....	42
4.4 SUMMARY.....	45
CHAPTER 5: TRIANGULATION .....	46
5.1 TRIANGULATION VS. DELAUNAY TRIANGULATION .....	46
5.1.1 Triangulation .....	46
5.1.2 Delaunay triangulation .....	47
5.2 PROPERTIES OF DELAUNAY TRIANGULATION .....	47
5.2.1 Uniqueness .....	47
5.2.2 Nearest Neighbor.....	47

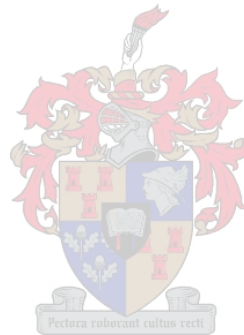
5.2.3 Convex hull .....	48
5.2.4 Empty circumcircle .....	48
5.2.5 Empty circle .....	48
5.2.6 Euler's Formula property .....	48
5.2.7 Maximizes minimum angle .....	48
5.3 ALGORITHMS .....	48
5.3.1 Delaunay Triangulation from Voronoi Diagrams .....	49
5.3.2 Incremental Insertion Algorithm .....	49
5.3.2.1 Lawson's Algorithm .....	50
5.3.3 Divide-And-Conquer Algorithm .....	52
5.3.4 Sweepline Algorithm .....	53
5.3.5 Flipping Algorithm .....	54
5.4 TRIANGLE .....	54
5.5 SUMMARY .....	56
CHAPTER 6: SOFTWARE .....	57
6.1 THE MODEL .....	57
6.1.1 Computational elements .....	57
6.1.1.1 Graphical Object Representation .....	58
6.1.1.2 Transformation specification / representation .....	58
6.1.1.3 Warping Reconstruction .....	58
6.1.1.4 Mapped Object Computation .....	58
6.1.1.5 Shape Blending .....	59
6.1.1.6 Attribute Blending .....	59
6.2 IMPLEMENTATION .....	60
6.2.1 Graphical Object Representation .....	60
6.2.2 Transformation specification / representation .....	60
6.2.3 Warping Reconstruction .....	62
6.2.4 Shape blending .....	64
6.2.5 Attribute Blending .....	64
6.3 THE ACTUAL PROGRAM .....	64
6.3.1 Layout of the Application Window .....	64
6.3.2 Commands .....	66
6.4 MORPHOS .....	67
6.5 FANTAMORPH 3 .....	70
CHAPTER 7 CONCLUSIONS AND FUTURE WORK .....	72
7.1 CONCLUSIONS .....	72
7.2 FUTURE WORK .....	74
BIBLIOGRAPHY .....	76
APPENDIX A .....	82
APPENDIX B .....	87

# List of Figures

Figure 1.1 Morphing consists of two warps followed by a blend .....	2
Figure 2.1 Cross dissolve between woman and cheetah .....	5
Figure 2.2 Image morphing between a woman and a cheetah .....	6
Figure 2.3 Different types of warps .....	7
Figure 2.4 Volume representation of a stack of images (a) without and (b) with interpolated images .....	8
Figure 2.5 Patterns selected by the user and the morphing sequence of the texture ....	10
Figure 3.1 Example of a Warp.....	11
Figure 3.2 Forward Warping .....	13
Figure 3.3 Reverse Warping .....	13
Figure 3.4 Example of scaling .....	18
Figure 3.5 Example of shearing in the u-direction .....	19
Figure 3.6 Example of rotation .....	19
Figure 4.1 Image warping combines warping and cross-dissolving .....	26
Figure 4.2 Cross dissolve between two images .....	28
Figure 4.3 Mesh warping between two images .....	29
Figure 4.4 A single line pair.....	32
Figure 4.5 Single line pair examples .....	33
Figure 4.6 Multiple line pair example.....	33
Figure 4.7 Example of a “ghost” .....	35
Figure 4.8 Example of a snake.....	36
Figure 4.9 A Multilevel free form deformation based morphing.....	37
Figure 4.10 Parallel view morphing .....	39
Figure 4.11 Non-Parallel View morphing.....	41
Figure 4.12 Parallel views .....	42
Figure 4.13 Singular views .....	42
Figure 4.14 Uniform transition rate morph.....	44
Figure 4.15 Non-Uniform transition rate morph .....	44
Figure 5.1 Delaunay triangulation of a random set of points .....	47
Figure 5.2 Voronoi diagram containing Delaunay triangulation.....	49
Figure 5.3 Circumcircle of shaded triangles contains the new vertex.....	50
Figure 5.4 Non-Delaunay triangles are deleted.....	51
Figure 5.5 New triangulation .....	51
Figure 5.6 Ghost Triangles.....	52
Figure 5.7 Merging Step.....	53
Figure 5.8 An Edge Flip .....	54
Figure 5.9 Set of input vertices.....	55
Figure 5.10 Set of output Delaunay triangles .....	55
Figure 6.1 Graphical representation of the control point array .....	60
Figure 6.2 Format of file containing control points.....	61
Figure 6.3 Format of the file containing the triangles .....	61
Figure 6.4 Example of output files created.....	62



Figure 6.5 Initial window.....	65
Figure 6.6 Window displaying the control points, triangulation.....	66
Figure 6.7 Middle image created by the implementation.....	67
Figure 6.8 Typical Morphos User Interface .....	68
Figure 6.9 List of supported feature specification types .....	68
Figure 6.10 Buttons to select between a morph, warp and cross-dissolve action .....	69
Figure 6.11 Window for selecting the transformation technique.....	69
Figure 6.12 A typical Fantamorph3 project workspace .....	70
Figure 6.13 A project workspace with corresponding, user-defined, control points.....	71
Figure 7.1 Foldover problem .....	74
Figure A.1 The start window .....	83
Figure A.2 Triangulation completed .....	84
Figure A.3 Interpolation of the triangles .....	85
Figure A.4 The morphing sequence .....	85
Figure B.1 Example of an Image sequence of a morph created by FantaMorph3 .....	90



# Chapter 1

## Introduction to Image Morphing

The problem of creating a smooth transition from one object to another object is called morphing. More specifically, the problem of creating a smooth transition from one image to another image is called *image morphing*. In other words image morphing can be described as the interpolation from one image to another image. The focus of this thesis is on images and therefore only morphing in two dimensions will be discussed. It is however necessary to state that morphing is not at all restricted to only two dimensions.

To get a clear idea of what is meant by image morphing it is recommended to take a look at something like Michael Jackson's popular music video, "Black and White" that contains continuous transitions (image morphs) between male and female faces. This is but one example. Countless more examples exist and anyone who has a television should at least have seen one or two examples of image morphing in the entertainment industry. It could be anything from gradually ageing a photo of a child to an adult, to transforming a human into something like a werewolf.

The field of morphing has received a lot of attention over the last years and it has reached a state of maturity. Various solutions to address this problem have been submitted, all with their own advantages and disadvantages, but before discussing how it is done it helps to understand what is being done. It is important to note that a morphing sequence consists of two warps (the spatial transformation of the images to align the features specified in both) followed by a blend, as demonstrated in Figure 1.1. Figure 1.2 illustrates an example of how this procedure is used to create an image morph.

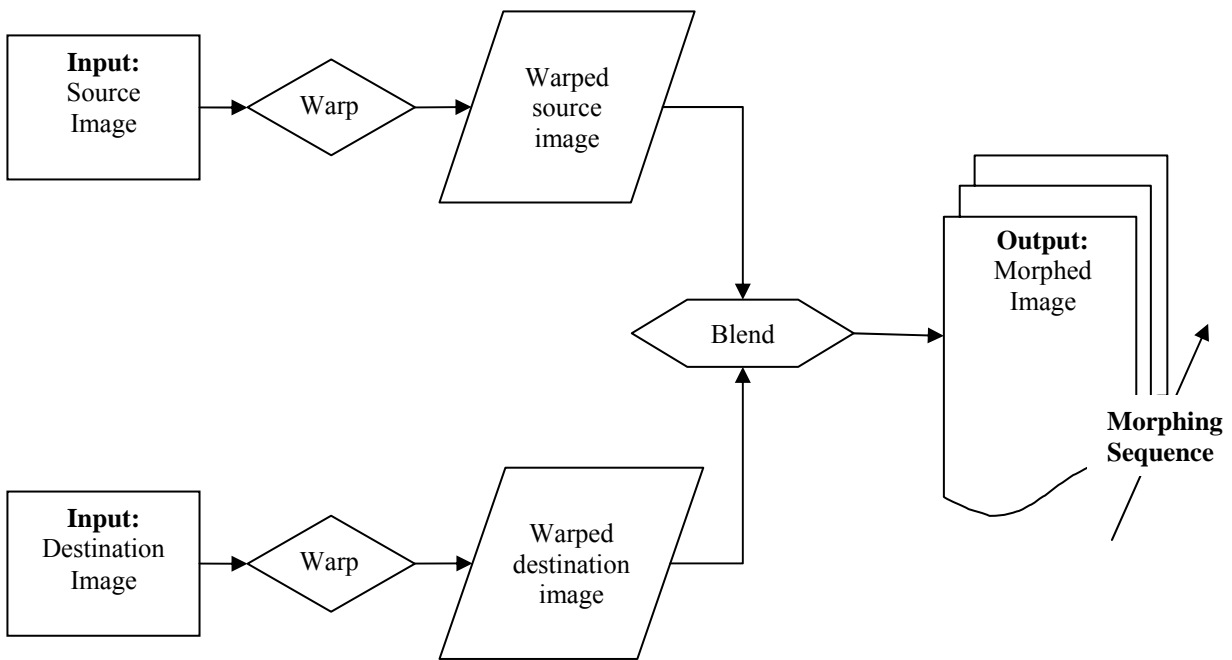


Figure 1.1 Morphing consists of two warps followed by a blend



Figure 1.2 Example of an Image Morph [19]

Chapter 2 provides an explanation of the terms “warping” and “morphing”. It explains how warping fits in with morphing and gives a few applications of morphing and warping.

Warping is an important part of morphing, dealing with aligning the features selected in the images. The more satisfying the warp, the more believable the morph. Chapter 3 gives an overview of warping (spatial transformations) and affine and perspective transformations are discussed as specific examples of warping.

Various techniques for creating a morphing sequence have been developed over the years - the main difference between them being how the warping is performed. Chapter 4 investigates a few popular, existing techniques and describes the basic idea behind each technique.

Delaunay triangulation is a triangulation method often used in computer graphics to sub-divide a surface into a set of triangles. Chapter 5 describes the significant properties that make it such a popular choice for triangulation as well as a few basic algorithms for creating a Delaunay triangulation. The application created for the purpose of this thesis uses a Delaunay triangulation to sub-divide the images into a set of triangles. Instead of warping the entire image, each triangle is warped separately.

Chapter 6 finally demonstrates the contribution of this thesis by describing the implementation of an image morphing sequence. The implementation is given a source and destination image as input and produces the morphing sequence as output. The warping is done by sub-dividing the source and destination images into a set of triangles. Each of the triangles in the source image is interpolated to those in the destination image, while each of the triangles in the destination image is interpolated to those in the source image. Once an intermediate interpolation step is done, OpenGL is used to warp the texture segment of the old triangle to the new triangle (also known as texture mapping). When this is done for all triangles in both images a color blend (color interpolation) is performed between the images to create the morph. A few existing morphing applications are also shown and commented on.

## Chapter 2

# An Overview of Warping and Morphing

### 2.1 Metamorphosis and morphing

According to Wiktionary [53] the word metamorphosis can have the following meanings:

1. “A transformation, such as that of magic or by sorcery.”
2. “A noticeable change in character, appearance, function or condition.”

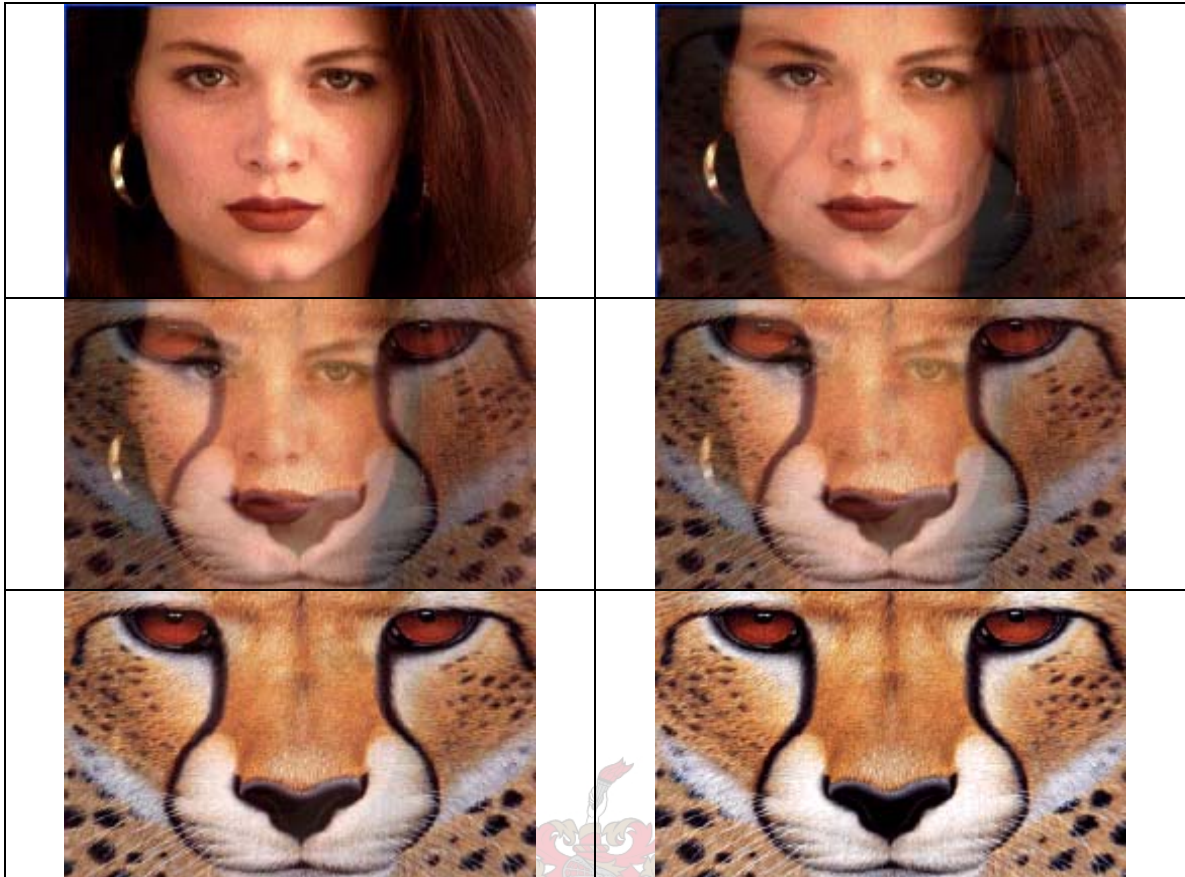
The word “morphing” is derived from the word “metamorphosis, where the morph denotes the changing of appearance of a graphical object.

### 2.2 Image Morphing

Image morphing can be defined as the construction of an image sequence depicting the gradual transition between the two images.



The simplest way to transform one image into another image is to cross-dissolve (better known as “fade”) them. This is achieved by interpolating the color of each pixel over time from the source image to the destination image. However this will not render a very effective visual morph as can be seen in Figure. 2.1. The first image is merely **replaced** by the second image without any warping (spatial deformation).

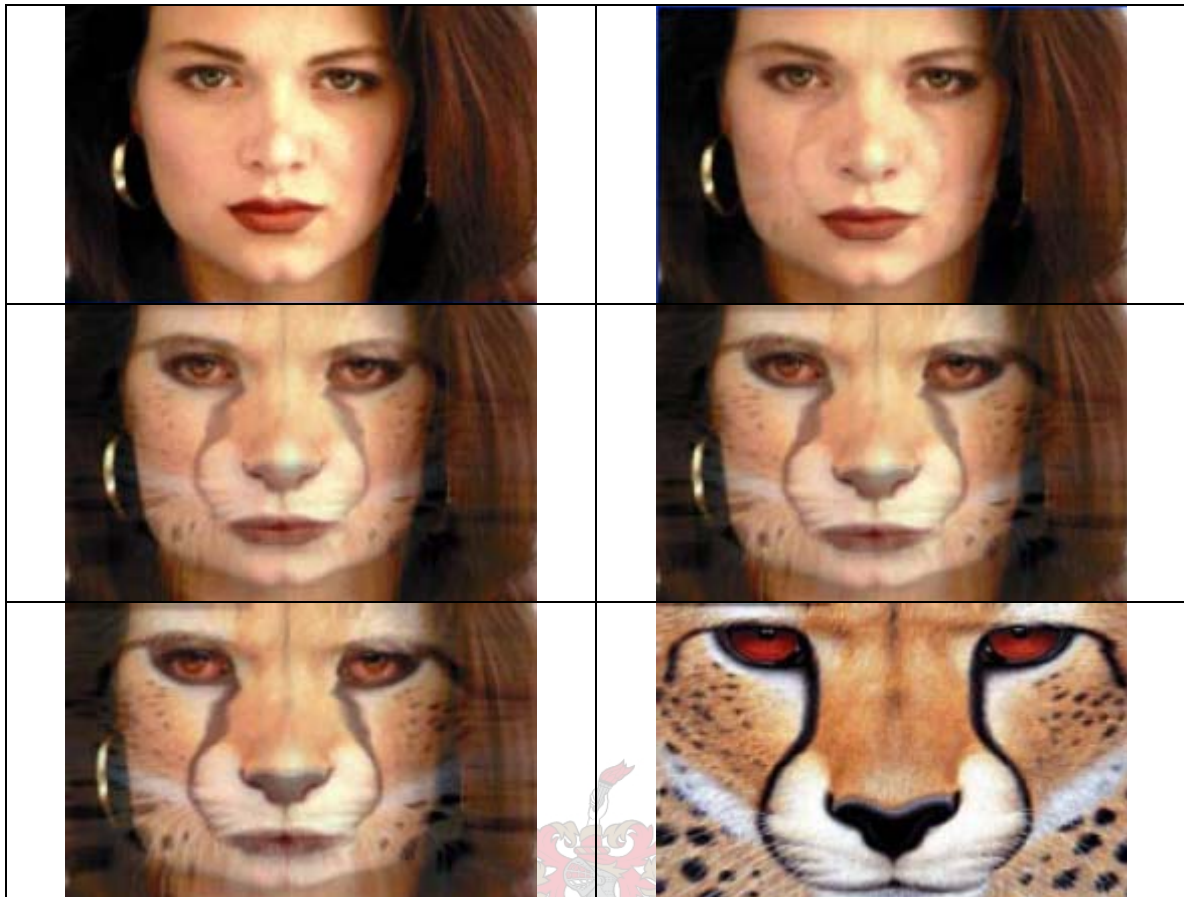


*Figure 2.1 Cross dissolve between woman and cheetah [19]*

A more effective and spectacular method exists and is known as image morphing. Image morphing involves image warping (changing the position of key features in the images) combined with cross-dissolving.

As can be seen in Figure 2.2 the created visual effect is much more spectacular than that of Figure 2.1. The first image seems to **become** the second image.





*Figure 2.2 Image morphing between a woman and a cheetah [19]*

This technique is the focus of this thesis.

## **2.3 Image Warping**

As was mentioned in Section 2.2 image warping involves changing the position of pixels in the image. The most effective way to explain image warping is to imagine printing an image onto a sheet of rubber and then consider the distortion of the image depending on the forces applied to the rubber sheet (e.g. stretching it) [55].

Figure 2.3 shows examples of different types of image warps.

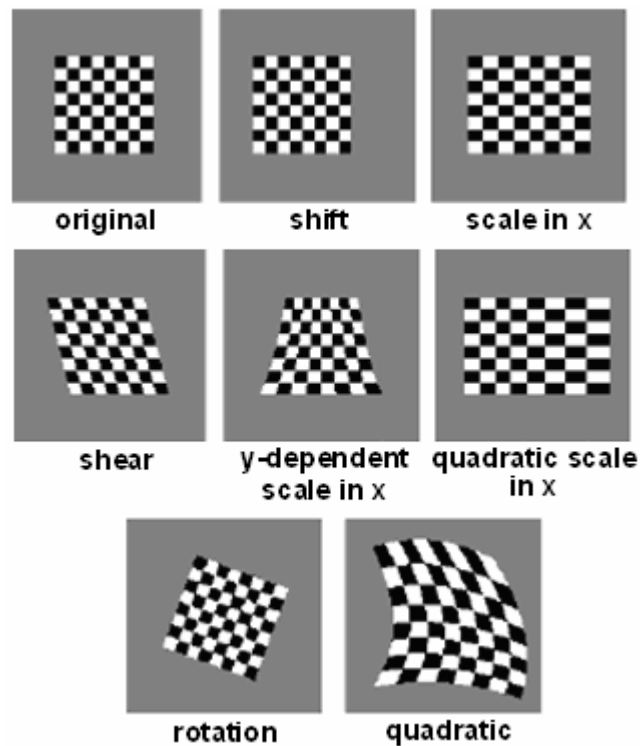


Figure 2.3 Different types of warps

When warping is used to create a morph the idea is to specify a warp that transforms the source image into the destination image. The inverse of the warp should transform the destination image back into the source image.

As the morph progresses, the source image is gradually warped into the destination image and faded out, while the destination image is gradually warped into the source image and faded in. The early images in the morph sequence will be similar to the source image, the middle images of the sequence will be the average of the two images and the last images in the sequence will be similar to the destination image.

It is important to note that the middle image determines the quality of the morph. If it looks believable the animation will look smooth and real.

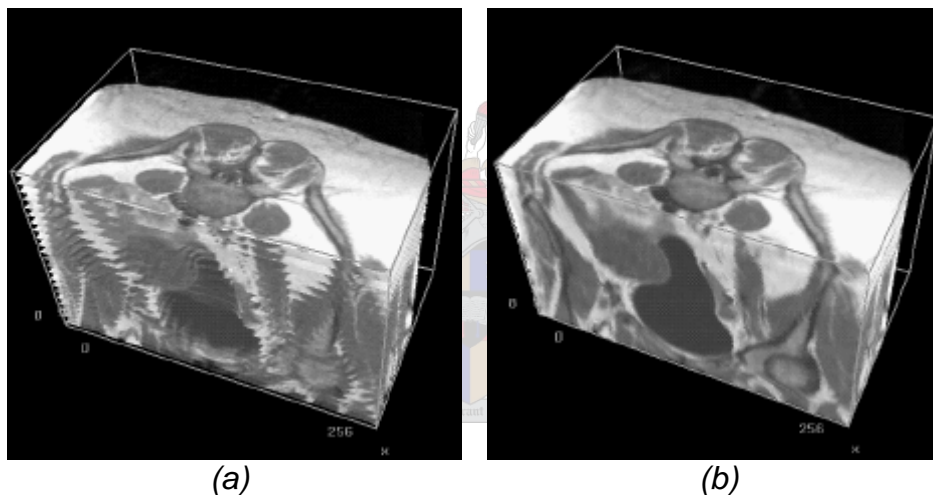
Warping is discussed in more detail in Chapter 3.



## 2.4 Some Applications of Image Morphing

In the medical profession, with modern CRT or NMR scans, slices of the human body can be imaged and combined into 3D models. The distance between such slices is usually much larger than the spatial resolution within each slice. For rendering (especially direct volume rendering) and surface reconstruction, this is undesirable as these require the volume elements to have edges of the same length. To achieve this some interpolation between slices is necessary. A method using image morphing to create the intermediate images between key frames is discussed in [37, 38].

Figure 2.4 displays an example where (b) looks much more realistic than (a), because in contrast to (a), the stack of images in (b) is interpolated.



*Figure 2.4 Volume representation of a stack of images (a) without and (b) with interpolated images [37]*

The entertainment industry is the field in which image morphing is most noticeable. It is used in movies and television to create different kinds of special effects between objects in a scene. The first use of morphing in film was in the French film “Le Magicien” (The Magician) produced in 1899. In this film Georges Méliès, a caricaturist and magician, used stop-start recording techniques to turn himself into his assistant [4].

Méliès built a camera, based on an English projector, and began filming un-staged street scenes. The story goes that one day he was filming at the Place de l’Opéra and his camera jammed just as a bus was passing. After some tinkering, he was able to resume

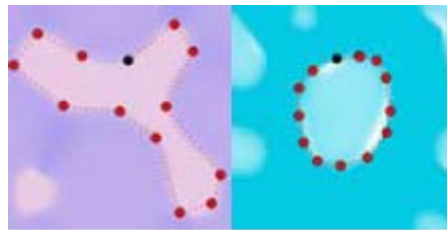
filming. By this time the bus was gone and a hearse was passing in front of the camera. When Méliès screened the film he discovered something spectacular: the moving bus seemed to instantly transform into the hearse. Méliès started planning and staging action for the camera. He built one of the first film studios and made hundreds of short fantasy and trick films based on having control over every element in the frame [4].

The movie “Willow” (produced in 1988) was the first movie to implement morphing [55]. In the movie an ostrich is transformed into a turtle, the turtle is transformed into a tiger and finally the tiger is transformed into a woman. “Willow” can be thought of as the trendsetter for using morphing in movies.

After “Willow” morphing became a well known and sought after technique in the entertainment industry. More examples include “Indiana Jones and the Last Crusade” in a scene where the actor undergoes physical decomposition and the facial animation in the film “The Abyss”. The transformation of the T1000 in “Terminator 2” is another excellent example. Perhaps one of the best examples is the transitions between male and female faces in Michael Jackson’s “Black and White” music video. A must-see for anyone who is still unclear about the meaning of image morphing.

Morphing can also help to animate realistic looking virtual people. Humans are one of the most difficult characters to model and animate, be it in games or movies. The reason for this is because everybody knows exactly what a human should look like and is an expert in recognizing a realistic person. Reconstructing a person in 3D and morphing between two faces are discussed in [29]. Morphing between different 3D human-body shapes is discussed in [30].

Another form of image morphing, dealing with patterned textures (which are really images) is discussed in [32]. Here morphing is used to smoothly transform from one patterned texture to another. The user selects a pattern in the source and destination image thus specifying the feature correspondence between these patterns. Repeated patterns are then automatically detected. Finally the warp function is obtained and the morph is generated. Figure 2.5 shows such an example.



(a)



(b)

Figure 2.5 (a) Patterns selected by the user and (b) the morphing sequence of the texture [32]

## 2.5 Summary

This chapter introduced the terms “image warping” and “image morphing” and listed some warping/morphing applications. In this thesis the terms “morphing” and “image morphing” are used interchangeably, both referring to the morphing of images (unless stated otherwise).

Chapter 3 describes warping, also known in the literature as spatial transformations.

## Chapter 3

### Warping (Spatial Transformations)

Image warping deals with the geometric transformation of digital images. A warp can range from a simple translation, rotation or scale to a combination of these, to a full nonlinear transformation (such as the one shown in Figure 3.1).

In image processing, warping is usually done to remove the distortions from an image (correcting geometric distortions), while in computer graphics it is used to introduce distortions. Another important application would be to neutralize the expression of a face, in facial recognition. In image morphing, warping is used to align the key features (e.g. eyes, mouth, and nose) of the two images being morphed.

#### 3.1 Definition of an Image Warp

A warp (spatial transformation) defines a geometric relationship between each point in the input image and each point in the output image. In other words, a 2D warping is a transformation that distorts a 2D source space into a 2D destination space - it maps a source point  $[u, v]$  to a destination point  $[x, y]$ , as illustrated in Figure 3.1.

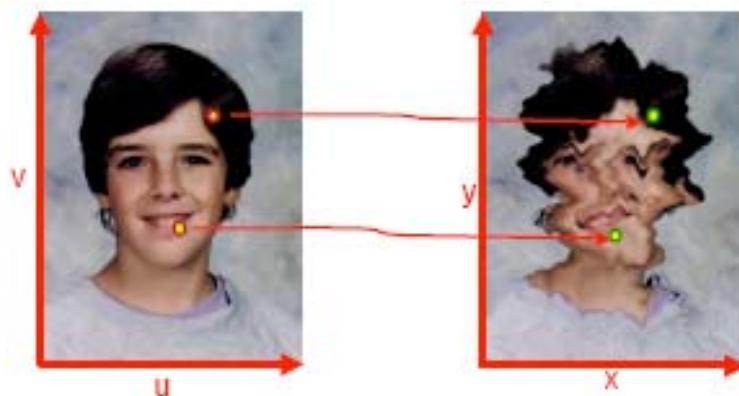


Figure 3.1 Example of a Warp [22]

The general warping function can be written as

$$[x, y] = [X(u, v), Y(u, v)] \quad (3.1.1)$$

relating the output coordinate system to that of the input. Or it can be expressed as

$$[u, v] = [U(x, y), V(x, y)] \quad (3.1.2)$$

relating the input coordinate system to that of the output. Here  $[u, v]$  refers to the input coordinate corresponding to output coordinate  $[x, y]$  and  $X$ ,  $Y$ ,  $U$  and  $V$  are warping functions that specify the spatial transformation. The functions  $X$  and  $Y$  map the input onto the output and therefore (3.1.1) is referred to as forward warping. The functions  $U$  and  $V$  map the output to the input and (3.1.2) is known as reverse warping.

## 3.2 Forward vs. Reverse Warping

### 3.2.1 Forward Warping

Forward warping scans through the source image pixel by pixel and copies each pixel to the appropriate position in the destination image, determined by the  $X$  and  $Y$  warping functions. The warping is straightforward in the continuous domain where pixels can be viewed as points. However in the discrete domain, pixels are seen as finite elements lying on a discrete integer lattice. This can cause two types of problems: holes and overlaps. Holes (patches of undefined pixels) occur when some destination pixels are bypassed in the input-output warping. Overlaps occur when many source pixels map to the same destination pixel.

Figure 3.2 illustrates forward warping. As can be seen, in the graph on the right, holes and overlaps may exist in the destination image.

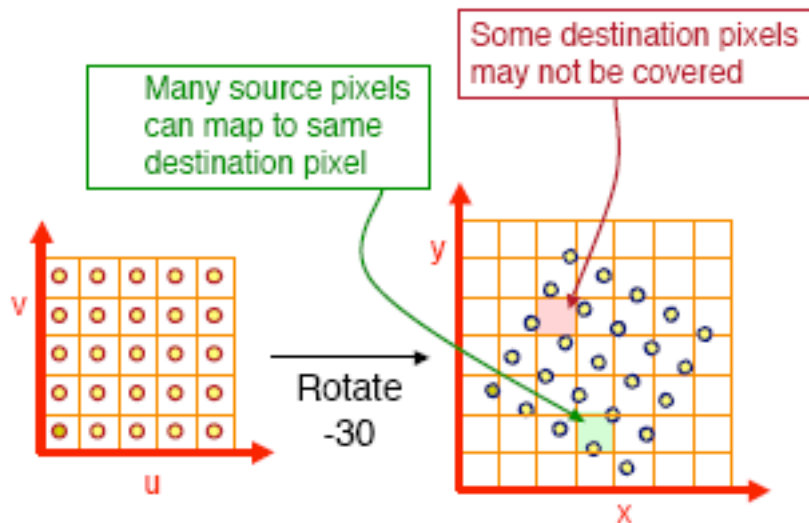


Figure 3.2 Forward Warping [22]

### 3.2.2 Reverse Warping

Reverse warping (also known as inverse or backward warping) scans through the destination image pixel by pixel projecting each output pixel onto the input image via  $U$  and  $V$ . The value of the data sample at that point is copied onto the output pixel. Unlike the point-to-point forward mapping, the reverse warping guarantees that all the output pixels are computed. However the problem of determining whether large holes are left, when sampling the input, still remains. This will happen if large amounts of input data have been discarded while calculating the output. Filtering (interpolation) of the source image is also necessary to retrieve input values at nonintegral (undefined) input positions. Reverse warping is illustrated in Figure 3.3.

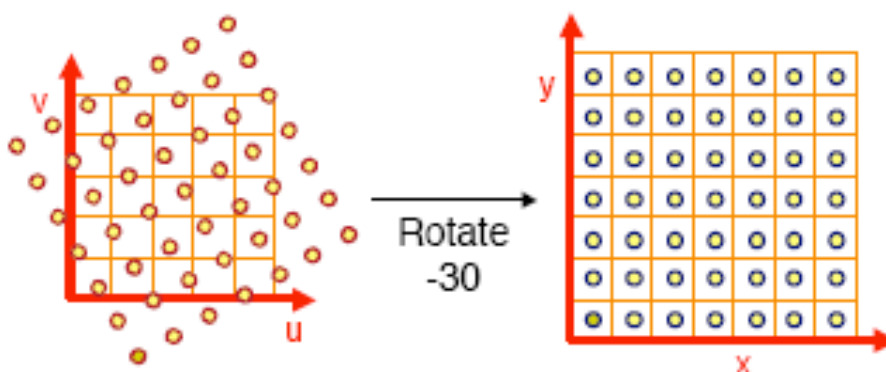


Figure 3.3 Reverse Warping [22]

### 3.3 Homogeneous Notation

The homogeneous representation for points is used to provide a consistent notation for affine and projective mappings. It will be briefly discussed here.

In familiar Euclidean geometry a point in the real plane  $\mathfrak{R}^2$  is represented by vectors of the form  $[x, y]$ . Projective geometry deals with the projective plane (a superset of the real plane) whose homogeneous coordinates are  $[x', y', w]$ . In projective geometry the 2D position vector  $p = [x, y]$  is represented by the homogeneous vector  $p_h = [x', y', 1] = [xw', yw', w']$  where  $w' \neq 0$ . To recover the actual coordinate  $p$  from the homogeneous vector  $p_h$  simply divide by the homogeneous component  $w'$ . For example, the homogeneous vector  $p_h = [x', y', 1] = [xw', yw', w']$  represents the actual point  $[x, y] = [x'/w', y'/w']$ . This division (a projection onto the  $w' = 1$  plane) cancels the effect of multiplication with  $w'$ .

It is thus observed that in homogeneous notation, 2D points are represented by 3-vectors and 3D points are therefore represented by 4-vectors.

### 3.4 General Transformation Matrix

Many simple spatial transformations can be expressed in terms of the general  $3 \times 3$  transformation matrix  $T$  (in the homogeneous coordinate system)

$$[x' \quad y' \quad w'] = [u \quad v \quad w] T \quad (3.4.1)$$

where

$$T = \begin{bmatrix} A & \mathbf{v} \\ \mathbf{t}^T & 1 \end{bmatrix}.$$

It can handle scaling, shearing, rotation, reflection, translation and perspective in 2D. The  $3 \times 3$  matrix  $T$  can be better understood if it is partitioned:

(i) The  $2 \times 2$  sub-matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

specifies the linear transformations for scaling, shearing and rotation.

(ii) The  $1 \times 2$  vector  $\mathbf{t}^T$  produces translation.

(iii) The  $2 \times 1$  vector  $\mathbf{v}$  produces perspective transformation.

A 2D transformation is a transformation (mapping) that distorts a 2D source space into a 2D destination space. Here only affine and perspective transformations will be discussed, described and illustrated in the following sections.

### 3.5 Affine Transformations

Note that the transformations are in terms of the forward mapping functions  $X$  and  $Y$  (transform the source image in the  $uv$ -coordinate system to the destination image in the  $xy$ -coordinate system). Similar derivations will apply for the reverse mapping functions  $U$  and  $V$ .

Formally a transformation  $L(x)$  is linear if and only if

$$L(x + y) = L(x) + L(y) \quad (3.5.1)$$

$$L(\alpha x) = \alpha L(x) \text{ for any scalar } \alpha. \quad (3.5.2)$$

A transformation  $T(x)$  is affine if and only if there exists a constant  $c$  and a linear transformation  $L(x)$  such that

$$T(x) = L(x) + c \quad (3.5.3)$$

is satisfied for all  $x$ . Clearly, all linear transformations are affine transformations.



For affine transformations the forward transformation functions (in Euclidean coordinates) can be given by

$$x = a_{11}u + a_{21}v + a_{31}, \quad (3.5.4)$$

$$y = a_{12}u + a_{22}v + a_{32}. \quad (3.5.5)$$

The general equation for performing an affine transformation (in homogeneous coordinates) can be written as

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}. \quad (3.5.6)$$

Affine transformations are the most common transformations used in computer graphics. They produce combinations of four basic transformations: scaling, shearing, rotation and translation. A succession of these affine transformations can be combined into an overall affine transformation. Some useful properties of affine transformations are:

- **Affine combinations of points are preserved**

An affine combination of two points  $\mathbf{p}_1$  and  $\mathbf{p}_2$  is defined as the point  $\mathbf{w} = a_1\mathbf{p}_1 + a_2\mathbf{p}_2$  where  $a_1 + a_2 = 1$ . Therefore when applying an affine transformation  $T$  to the point  $\mathbf{w}$  we see that  $T(a_1\mathbf{p}_1 + a_2\mathbf{p}_2) = a_1T(\mathbf{p}_1) + a_2T(\mathbf{p}_2)$ . This property is sometimes taken as the definition of an affine transformation

- **Parallelism of lines and planes is preserved**

If two lines (or planes) are parallel in the source space, they will be transformed to parallel lines (or planes).

- **Lines and planes are preserved**

Affine transformations preserve collinearity and flatness. This guarantees that straight lines will transform into straight lines, polygons will transform into polygons, and planar polygons (polygons whose vertices lie in a plane) will transform into planar polygons. In particular a triangle in the source space will be mapped to a

triangle in the destination space or a rectangle in the source space can be mapped to a parallelogram in the destination space, but no more general distortions are possible since the parallelism of lines is preserved. Mapping a rectangle into a general quadrilateral is not possible. For this bilinear, perspective or more complex transformations are needed.

- **Relative ratios are preserved**

Equispaced points on a line will transform into equispaced points on a line, or more generally the relative spacing between points on a line will be preserved.

- **Every affine transformation is composed of elementary operations**

A complex affine transformation can be constructed by composing a number of elementary ones, as discussed in Section 3.5.6.

Below the special cases of affine transformations are discussed. The equations are given for homogeneous coordinates.

### 3.5.1 Scaling

The equation for performing a scaling is



$$[x \quad y \quad 1] = [u \quad v \quad 1] \begin{bmatrix} s_u & 0 & 0 \\ 0 & s_v & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.5.7)$$

By applying the scale factors ( $s_u$  and  $s_v$ ) to the source coordinates all the points are scaled. This type of scaling is more accurately known as scaling about the origin because point P is moved  $s_u$  times farther from the origin in the  $u$ -direction and  $s_v$  times farther from the origin in the  $v$ -direction.

Enlargements are specified with positive scale factors greater than one, reductions with positive scale factors smaller than one. If a scale factor is negative there is a reflection about a coordinate axis. In other words it will cause the image to be mirrored. If the scale

factors are identical the transformation is a *uniform scaling*. Non-identical scale factors will alter the proportions of the image – this is called a *differential scaling*.

In Figure 3.4 an example of scaling is shown.

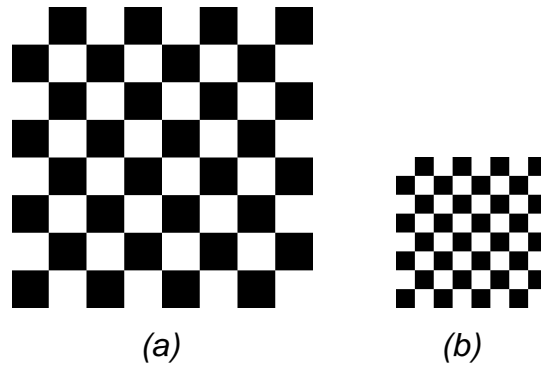


Figure 3.4 Example of uniform scaling where (a) is the original image and (b) the scaled image

### 3.5.2 Shearing

An example of a shear can be seen in Figure 3.5. A shear in the  $v$ -direction is given by:

$$[x \ y \ 1] = [u \ v \ 1] \begin{bmatrix} 1 & h_u & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (3.5.8a)$$

where  $h_u$  specifies what fraction of the  $u$ -coordinate of the point is added to the  $v$ -coordinate.

Similarly a shear in the  $u$ -direction is given by

$$[x \ y \ 1] = [u \ v \ 1] \begin{bmatrix} 1 & 0 & 0 \\ h_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.5.8b)$$

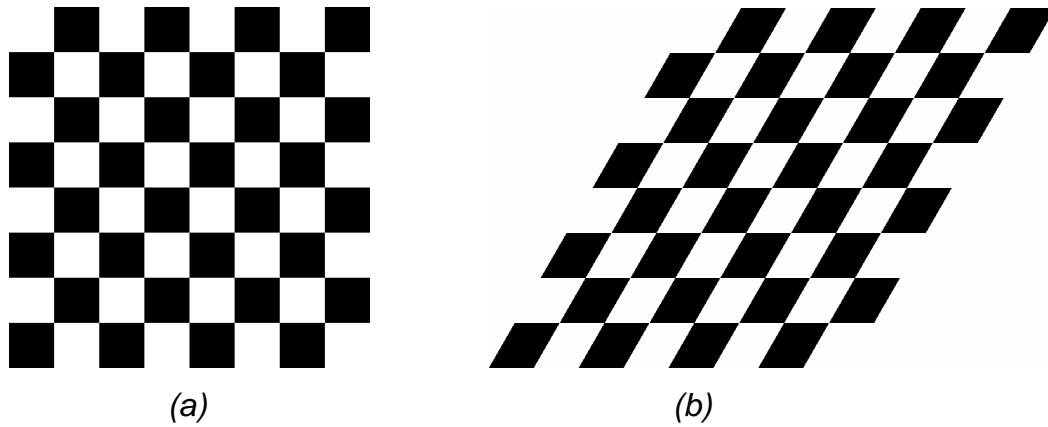


Figure 3.5 Example of shearing in the  $u$ -direction, where (a) is the original image and (b) the sheared image

### 3.5.3 Rotation

A fundamental graphics operation is the rotation of a figure about a given point through some angle. See Figure 3.6 for an example. The equation

$$[x \ y \ 1] = [u \ v \ 1] \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.5.9)$$

(in homogeneous coordinates) will cause a clockwise rotation (about the origin) for positive values of  $\theta$  on all the points in the  $uv$ -plane.

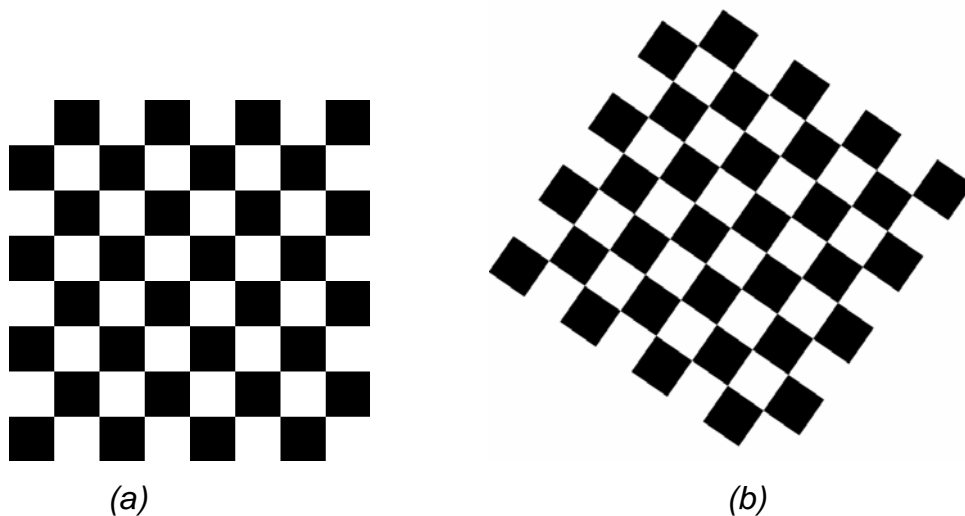


Figure 3.6 Example of rotation, where (a) is the original image and (b) the rotated image

### 3.5.4 Translation

Often an image needs to be translated to a different position. By adding offsets  $t_u$  and  $t_v$  to  $u$  and  $v$ , respectively, all the points in the  $uv$ -plane are translated to a new position. The transformation is then

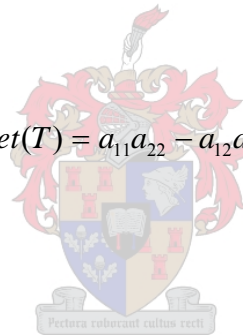
$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_u & t_v & 1 \end{bmatrix}. \quad (3.5.10)$$

### 3.5.5 The Inverse of an Affine Transformation

The inverse of an affine transformation is affine itself. All affine transformations of interest are nonsingular, which means that the determinant of the transformation matrix in (3.5.6)

$$\det(T) = a_{11}a_{22} - a_{12}a_{21} \quad (3.5.11)$$

is nonzero.



The inverse of transformation matrix  $T$  can be determined from the adjoint  $adj(T)$  and determinant  $\det(T)$  of matrix  $T$ . It is known, from linear algebra, that

$$T^{-1} = adj(T) / \det(T) \quad (3.5.12)$$

where the adjoint of a matrix is simply the transpose of the matrix of cofactors [47].

Therefore to calculate  $T^{-1}$  the following is done:

Firstly  $C$ , the matrix of cofactors, is built

$$C = \begin{bmatrix} a_{22} & -a_{21} & a_{21}a_{32} - a_{31}a_{22} \\ -a_{21} & a_{11} & a_{31}a_{12} - a_{11}a_{32} \\ 0 & 0 & a_{11}a_{22} - a_{21}a_{12} \end{bmatrix}, \quad (3.5.13)$$

then  $C$  is transposed to get  $C^T$

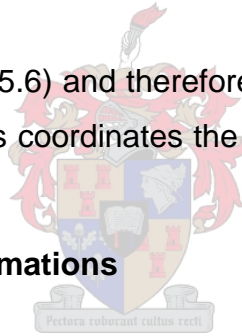
$$\text{adj}(T) = C^T = \begin{bmatrix} a_{22} & -a_{21} & 0 \\ -a_{21} & a_{11} & 0 \\ a_{21}a_{32} - a_{31}a_{22} & a_{31}a_{21} - a_{11}a_{32} & a_{11}a_{22} - a_{21}a_{12} \end{bmatrix}, \quad (3.5.14)$$

and finally each element of  $C^T$  is scaled by  $1/\det(T)$  to form  $T^{-1}$

$$T^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{21} & 0 \\ -a_{21} & a_{11} & 0 \\ a_{21}a_{32} - a_{31}a_{22} & a_{31}a_{21} - a_{11}a_{32} & a_{11}a_{22} - a_{21}a_{12} \end{bmatrix}. \quad (3.5.15)$$

Note that (3.5.15) is of the form (3.5.6) and therefore an affine transformation. Also note that since working in homogeneous coordinates the constant  $1/\det(T)$  can be discarded and  $\text{adj}(T)$  can be used as  $T^{-1}$ .

### 3.5.6 Composing Affine Transformations



It is rare that just one transformation is performed. Usually an application requires that a compound transformation is built out of several elementary ones. Multiple transformations can be combined into a single composite transformation. This is called composing or concatenating the transformations. When two affine transformations

$$T_1 = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}, \quad T_2 = \begin{bmatrix} b_{11} & b_{12} & 0 \\ b_{21} & b_{22} & 0 \\ b_{31} & b_{32} & 1 \end{bmatrix}$$

are composed, the resulting transformation

$$T_1 T_2 = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & 0 \\ b_{21} & b_{22} & 0 \\ b_{31} & b_{32} & 1 \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & 0 \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & 0 \\ a_{31}b_{11} + a_{32}b_{21} + b_{31} & a_{31}b_{12} + a_{32}b_{22} + b_{32} & 1 \end{bmatrix}$$

is also affine, since  $T_1 T_2$  is also in the form of an affine transformation.

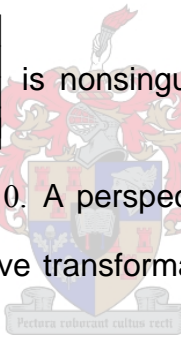
### 3.6 Perspective Transformations

Perspective transformations are easily manipulated in homogeneous matrix notation.

The forward transformation function can be written as

$$\begin{bmatrix} x' & y' & w \end{bmatrix} = \begin{bmatrix} u' & v' & q \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (3.6.3)$$

where the matrix  $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$  is nonsingular and where  $[x, y] = [x'/w, y'/w]$  for  $w \neq 0$ , and  $[u, v] = [u'/q, v'/q]$  for  $q \neq 0$ . A perspective transformation is produced when  $[a_{13} \ a_{23}]^T \neq 0$ . Note that a perspective transformation is an affine transformation when  $a_{13} = a_{23} = 0$ .



The Euclidean representation of this kind of transformation is

$$x = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}}, \quad (3.6.1)$$

$$y = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}}. \quad (3.6.2)$$

A perspective transformation (also known as a projective mapping) is a transformation that maps straight lines to straight lines.

Perspective transformations have the following useful properties:

- **They preserve lines in all orientations.**

Straight lines map onto straight lines.

- **They permit quadrilateral-to-quadrilateral mappings.**

Warping a rectangle into a general quadrilateral is possible.

- **Perspective transformations can be composed by concatenating their matrices.**

A complex perspective transformation can be constructed by composing a number of elementary ones.

### 3.6.1 The Inverse of a Perspective Transformation

The inverse of a perspective transformation is a perspective transformation. It can be determined in terms of the adjoint of the transformation matrix  $T$

$$T^{-1} = \text{adj}(T) / \det(T) \quad (3.6.4)$$

where  $\text{adj}(T)$  is the adjoint of  $T$  and  $\det(T)$  is the determinant of  $T$ .

In homogeneous algebra however, the adjoint matrix can be used instead of the inverse matrix whenever an inverse matrix is needed. This is possible because two matrices that are (nonzero) scalar multiples of each other are equivalent in the homogeneous coordinate system, so there is no need to divide by the determinant (a scalar). The inverse transformation is thus

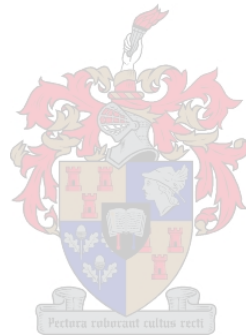
$$[u \quad v \quad q] = [x' \quad y' \quad w] \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}. \quad (3.6.5)$$



### 3.7 Summary

This chapter introduced some basic warp functions (spatial transformations) providing some background on the different types of transformations. It showed how these transformations can happen, where they are used and some limitations to consider.

The next chapter investigates a few existing image morphing techniques.



# Chapter 4

## Related work – Existing techniques

Various methods for computing image morphs exist. In this chapter a number of these techniques will be discussed. These are however not the only existing techniques - there are numerous others, but mentioning all of them would simply be impossible.

### 4.1 Definition of Images

An image can be described as a 2D graphical object defined by the function [17]

$$f : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n \quad (4.1.1)$$

where  $U$  is the shape of the image - usually a rectangle of the plane,  $\mathbb{R}^n$  is the attribute space (commonly the 3D RGB color space where the color attributes are specified), and the function  $f$  is the attribute function of the image. For each point  $p \in U$ ,  $f(p)$  defines the attributes of  $p$ . In the simplest case this is the color attribute, but it can also be other attributes such as opacity, scene depth, etc.

### 4.2 Image Morphing

Image morphing methods produce a smooth transition from the source image to the destination image by interpolating the position and then the colour of pixels in the two images. Figure 4.1 gives a simple representation of morphing.

This means all image morphing techniques follow the same basic steps. It begins by selecting corresponding control points in each image. These control points are then used to compute a warping (a transformation/mapping) function that defines the spatial relationship between all the points in the two images. The warping functions are used to interpolate the corresponding positions of control points across the morphing sequence.

Once both images are warped into alignment a cross-dissolve is generated between the two images. A cross-dissolve is merely an interpolation of pixel values.

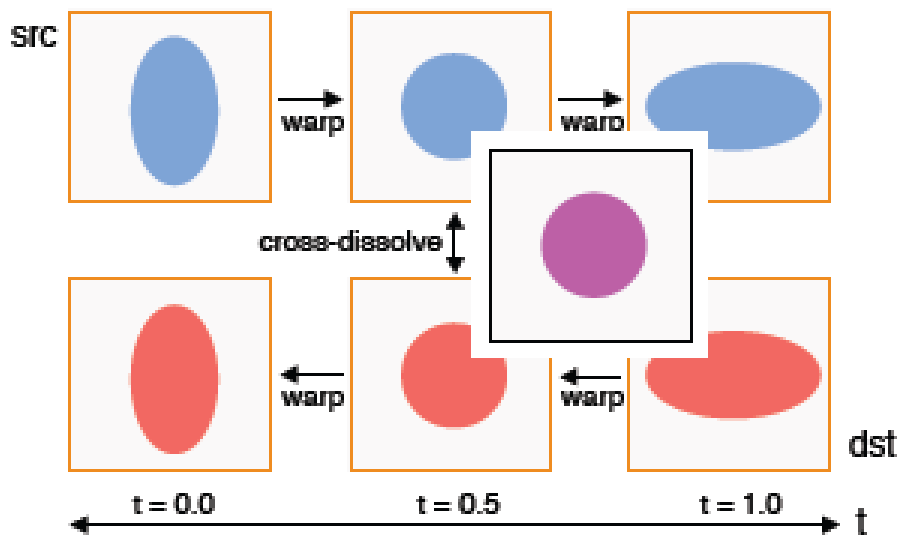


Figure 4.1 Image warping combines warping and cross-dissolving [22]

Since color interpolation between images is rather simple, research in image morphing has been concentrated on creating warp functions from the specified corresponding features. Note that it is important for each control point to be represented in both the source and destination images, because no correspondence between the images can be established if a control point is missing in one them. This type of feature selection restricts the application of the traditional morphing techniques. This restriction can be dealt with by techniques involving nearest neighbor pixel color interpolation or the use of a background color [6, 41, 51].

The following image morphing techniques will be discussed in more detail in the sections that follow:

**(i) *Cross dissolve***

Cross-dissolving can be described as a color interpolation. In this technique the source image is simply faded into the destination image.

**(ii) *Mesh warping***

Using non-uniform quadrilateral meshes to specify corresponding control points a warp is computed from the corresponding mesh points via something like spline interpolation.

**(iii) *Field morphing***

Field morphing uses a set of line segments to specify features in an image. A pair of lines on the two images will determine a warp from their local coordinate system. When more than one pair of lines is specified a weighted average is used to determine the influence of each line pair on the images.

**(iv) *Snakes and multilevel free-form deformations***

In this technique snakes are used to simplify feature specification. A multilevel free-form deformation (MFFD) technique is used to derive a  $C^2$ -continuous and one-to-one warp that exactly satisfies the feature correspondence. This technique is based on 2D B-spline approximation [26].

**(v) *View morphing***

Unless special care is taken, most morphing techniques do not preserve 3D shape. Usually morphing between images with the same 3D shape will result in shapes that are mathematically different. Seitz and Dreyer [41] devised a method that preserves the 3D shape of the morphed objects.

### 4.2.1 Cross Dissolve

Before the development of morphing, transitions between two images were generally done by cross dissolving (e.g. a linear color interpolation to fade from one image to another). A cross dissolve is usually applied to the whole image and in effect the texture of the source image is transformed to the texture of the destination image by blending the color of the pixels. The result is poor because of the double exposure effect that is apparent in regions where the features of the source image do not align with those in the destination image, as can clearly be seen in Figure 4.2. However, cross dissolving is implemented as a critical part of the implementation in the techniques discussed below.



*Figure 4.2 Cross dissolve between two images [54]*

### 4.2.2 Mesh warping

The two-pass mesh warping algorithm was pioneered at Industrial Light & Magic by D. Smythe for use in the 1988 movie “Willow” [46, 55]. It has since been used successfully in movies such as “The Abyss”, “Indiana Jones and the Last Crusade” and “Terminator 2”.

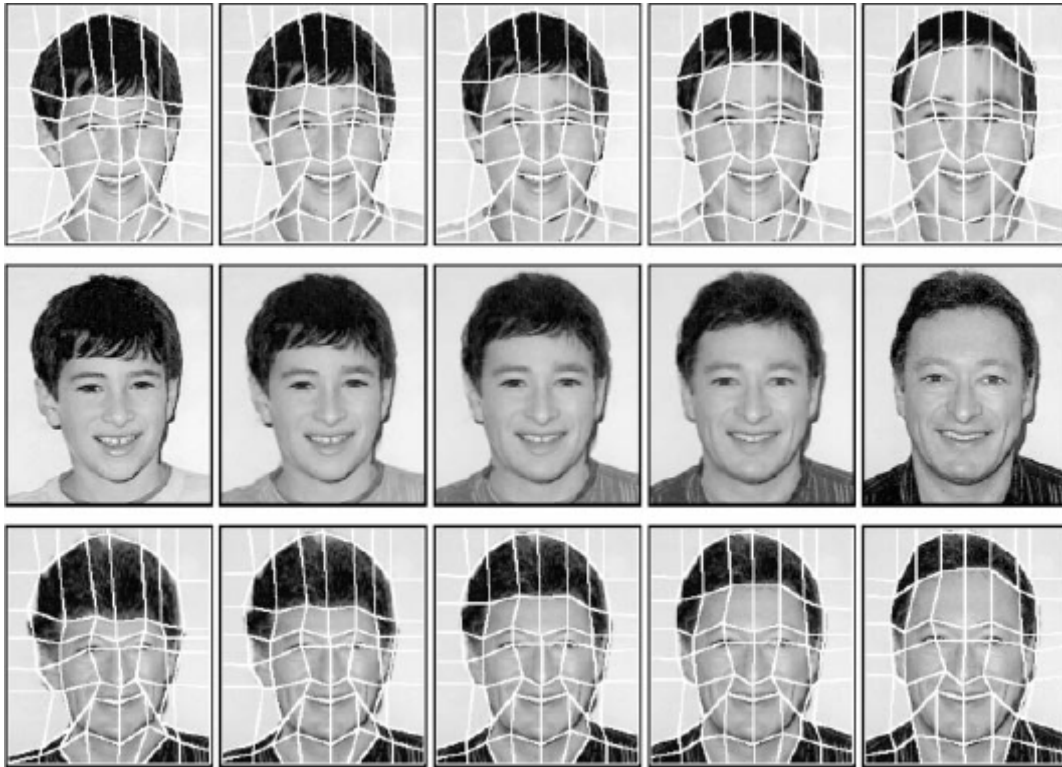


Figure 4.3 Mesh warping between two images [54]

The technique [55] can be described as follows: Consider two images - the source image denoted as  $I_s$  (top-left image in Figure 4.3) and the destination image as  $I_d$  (bottom-right image in Figure 4.3). Each image has a mesh overlay. The source image has mesh  $M_s$  overlaid and the destination image has mesh  $M_d$  overlaid.  $M_s$  specifies the coordinates of the control points in  $I_s$  and  $M_d$  specifies their corresponding positions in  $I_d$ .  $M_s$  and  $M_d$  are used to determine the spatial transformation that maps all the points in  $I_s$  onto the points in  $I_d$ . No folding or discontinuities are allowed in the meshes and for simplicity they are constrained to have frozen outer borders.

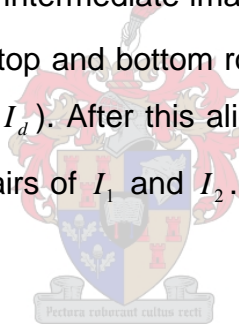
Each intermediate image (every image between  $I_s$  and  $I_d$ ) can therefore be computed by the four step process described below. The number of intermediate images is specified by the user.

```

for each intermediate image  $f$  do
    Linearly interpolate mesh  $M$  between  $M_s$  and  $M_d$ 
    Warp  $I_s$  to  $I_1$  using meshes  $M_s$  and  $M$ 
    Warp  $I_d$  to  $I_2$  using meshes  $M_d$  and  $M$ 
    Linearly interpolate image  $I_f$  between  $I_1$  and  $I_2$ 
end

```

Figure 4.3 shows this process. In the top row of the figure, mesh  $M_s$  is transformed to mesh  $M_d$ , producing an intermediate mesh  $M$  for each frame. These meshes are used to transform  $I_s$  to the intermediate image defined by mesh  $M$ . The bottom row shows the exact same process in reverse order, where  $I_d$  is transformed to the intermediate image. This process is done to maintain the alignment of control points between  $I_s$  and  $I_d$  as they both transform to some intermediate image, producing the pairs of  $I_1$  and  $I_2$  images, respectively shown in the top and bottom rows (excluding of course the top-left image,  $I_s$  and bottom-right image  $I_d$ ). After this alignment is obtained a cross-dissolve is done between the successive pairs of  $I_1$  and  $I_2$ . This can be seen in the middle row of Figure 4.3.



The example in Figure 4.3 used Catmull-Rom spline interpolation to determine a correspondence of all pixels and Fant's algorithm was used to resample the image in a separable implementation [9, 55].

The disadvantage of using this method is that in the simplest version of this technique the user must specify in advance how many control points will be used. These points must then be moved to the correct locations. However, points left unmodified by mistake, or points that could not be matched are still used in this algorithm. The user will often feel that he has too much control in some areas and not enough in others.

Another problem is that the algorithm breaks down for large rotational distortions (bottleneck problem [55]). The intermediate image in the algorithm might be distorted to such an extent that the information is lost.

### 4.2.3 Field Morphing (Feature Based Image Metamorphosis)

This technique, developed by Beier and Neely [2] simplifies the task of feature specification. Instead of using meshes and splines to specify features, this technique makes use of line segments. A pair of corresponding line segments (one defined relative to the source image, the other relative to the destination image) defines a mapping from one image to the other (this is explained below).

Using reverse mapping (to ensure that each pixel in the destination image is set to an appropriate value) a pair of corresponding lines in the source and destination images defines a coordinate mapping from the destination image pixel coordinate  $\mathbf{X}$  to the source image pixel coordinate  $\mathbf{X}'$ . For line  $PQ$  the position of  $\mathbf{X}$  along the line is given by

$$u = \frac{(\mathbf{X} - \mathbf{P}) \cdot (\mathbf{Q} - \mathbf{P})}{\|\mathbf{Q} - \mathbf{P}\|^2}. \quad (4.1.1)$$

The value of  $u$  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. For a pixel  $\mathbf{X}$  not on the line  $PQ$  the perpendicular distance (in pixels) to the line is given by

$$v = (\mathbf{X} - \mathbf{P}) \cdot \mathbf{n}_{PQ} \quad (4.1.2)$$

where  $\mathbf{n}_{PQ}$  is the unit vector perpendicular to  $PQ$ . There are two perpendicular vectors and either one can be used as long as it is used consistently.



Finally the mapping of  $\mathbf{X}$  to  $\mathbf{X}'$  is given by

$$\mathbf{X}' = \mathbf{P}' + u(\mathbf{Q}' - \mathbf{P}') + v\mathbf{n}_{\mathbf{P}'\mathbf{Q}'}. \quad (4.1.3)$$

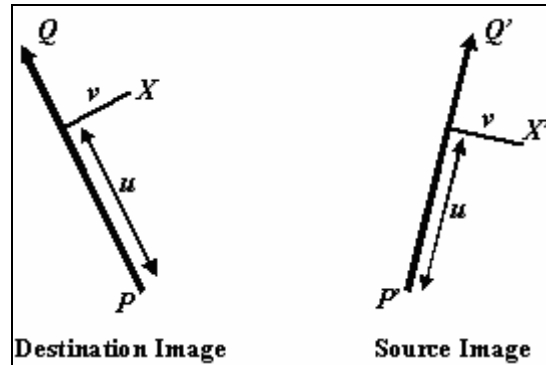


Figure 4.4 A single line pair

For a single line pair (see Figure 4.4) the algorithm, as described in [2], is given as:

**For** each pixel  $\mathbf{X}$  in the destination image **do**  
    Compute  $u$  and  $v$  in the destination image  
    Compute the  $\mathbf{X}'$  in the source image for that  $u$  and  $v$   
    Set the destination image pixel  $\mathbf{X}$  to the value of the source image pixel  $\mathbf{X}'$   
**End**

Each pixel coordinate is transformed by a rotation, translation and or scaling, thereby transforming the whole image. However, some affine transformations, such as shears and uniform scales, are not possible to perform with this method. Figure 4.5 shows a few examples.

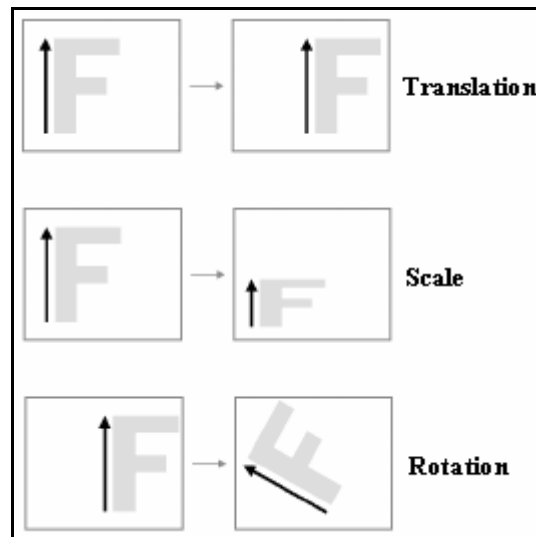


Figure 4.5 Single line pair examples [2]

Because more than one feature is usually needed for an acceptable transformation, multiple feature line pairs will be necessary. Figure 4.6 shows an example.

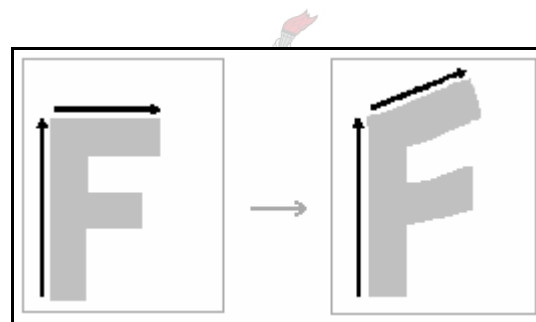


Figure 4.6 Multiple line pair example [2]

Multiple pairs of lines can specify more complex transformations. The displacement of a point in the source image is a weighted sum of the transformations due to each line pair, with the weights depending on distance and line length. For each line pair a position  $\mathbf{X}'_i$  is calculated. Then a displacement  $\mathbf{D}_i$  (the difference between the pixel location in the source and destination image) is calculated as

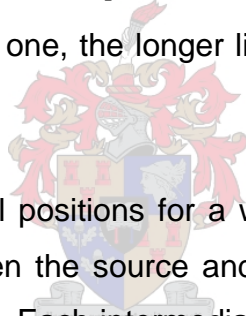
$$\mathbf{D}_i = \mathbf{X}'_i - \mathbf{X}. \quad (4.1.4)$$

Finally a weighted average of these displacements is calculated. The weight assigned to each line should be strongest when the pixels fall exactly on the line and weaker when the pixels are further away from the line. The equation is

$$weight = \left( \frac{length^p}{(a + dist)} \right)^b \quad (4.1.5)$$

where *length* is the length of the line, *dist* is the distance from pixel **X** to the line, and *a*, *b* and *p* are constants that are varied / chosen to control the warp.

If *a* is barely greater than zero and *dist* is zero, the weight approaches infinity. With this value for *a* the user knows that the pixels on the line will go exactly where he/she wants them to go. Larger values for *a* will supply a smoother warping, but with less precise control. The variable *b* determines how the relative strength of different lines falls off with distance. For large values of *b*, pixels are only affected by the lines nearest to them. If *b* is zero, pixels will be affected equally by all lines. Values of *b* in the range [0.5, 2] are the most useful. The value of *p* is usually in the range [0, 1]. If it is zero all lines have the same weight. If it is one, the longer lines will have a greater weight than the shorter lines.



The procedure for calculating pixel positions for a warped image is then as follows: A morphing operation blends between the source and destination image. Corresponding lines are defined in the two images. Each intermediate image of the morph is defined by creating a new set of line segments by interpolating the lines from their positions in the source image to their positions in the destination image. The source and destination images are distorted towards the lines in the intermediate image. These two resulting images are cross dissolved throughout the morph.

The authors [2] used two different methods for interpolating the lines. One method simply interpolates the endpoints of each line. The other method interpolates the center position and orientation of each line as well as the length of each line. In the first case, if a line is rotated it would shrink in the middle of the morph.

This technique is much more specific than the 2-pass mesh warping technique, discussed above. In this algorithm the only positions that are used are the ones explicitly defined by the user. Everything that is specified is moved exactly where the user wants

them and everything else is blended smoothly based on those positions. Adding new lines will increase the control in that area, without affecting the rest of the areas too much.

A disadvantage of this algorithm is that sometimes unexpected interpolations are generated between the lines. The algorithm guesses what should happen when far away from the line segments, and sometimes the result is wrong. This problem usually manifests itself as a “ghost” of a part of the image showing up in some unrelated part of the interpolated image and is caused by an unforeseen combination of the specified line segments [2]. Figure 4.7 displays an example of a ghost. Additional line pairs must sometimes be supplied to counter the bad effects of a previous set of lines.

Other disadvantages are speed and control. All line pairs must be considered for every source pixel before the mapping is known. An optimization based on piecewise linear approximation is offered in [28].

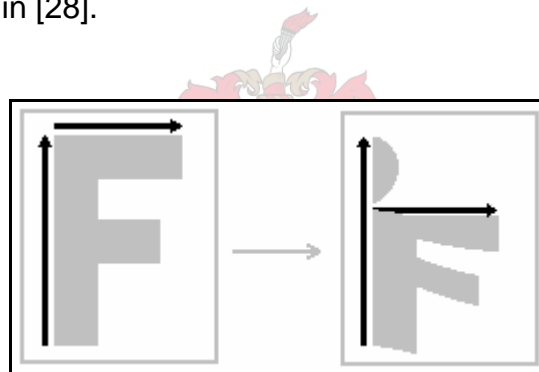


Figure 4.7 Example of a “ghost” [2]

#### 4.2.4 Snakes and Multilevel Free-Form Deformation

In this technique *snakes*, a popular technique in computer vision, are used to specify features in the source and destination images. Snakes [23] are energy-minimizing splines that move under the influence of image and constraint forces. They simplify feature specification because primitives must only be positioned close to the features. Image forces push the snake towards salient image features such as lines and edges while constraint forces pull the snake to a desired image feature among nearby ones, thereby refining their final positions and making it possible to capture the exact position

of a feature. The use of snakes relies a great deal on the features in an image being well defined by their edges.

To specify a feature a snake is initialized by positioning a polyline (connected control points) close to a feature. A sequence of points is then uniformly sampled on the polyline. As the snake minimizes its energy it wriggles itself and finally locks onto the feature. Figure 4.8 illustrates an example where (a) is the image with the rough manually specified connected control points and (b) is the same image after the snake has been applied to the linked control points.

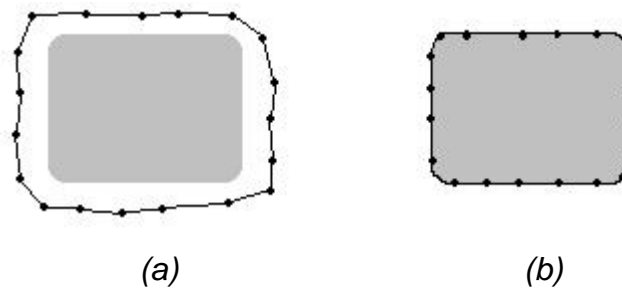


Figure 4.8 Example of a snake

When placing a feature specification primitive  $f_s$  on the source image  $I_s$ , a primitive  $f_d$  is also deposited on the destination image  $I_d$ .  $f_s$  is moved repeatedly or a snake is generated from  $f_s$  to specify a feature on  $I_s$ .  $f_d$  is then moved to designate the corresponding feature on  $I_d$  and a snake is initiated if necessary.

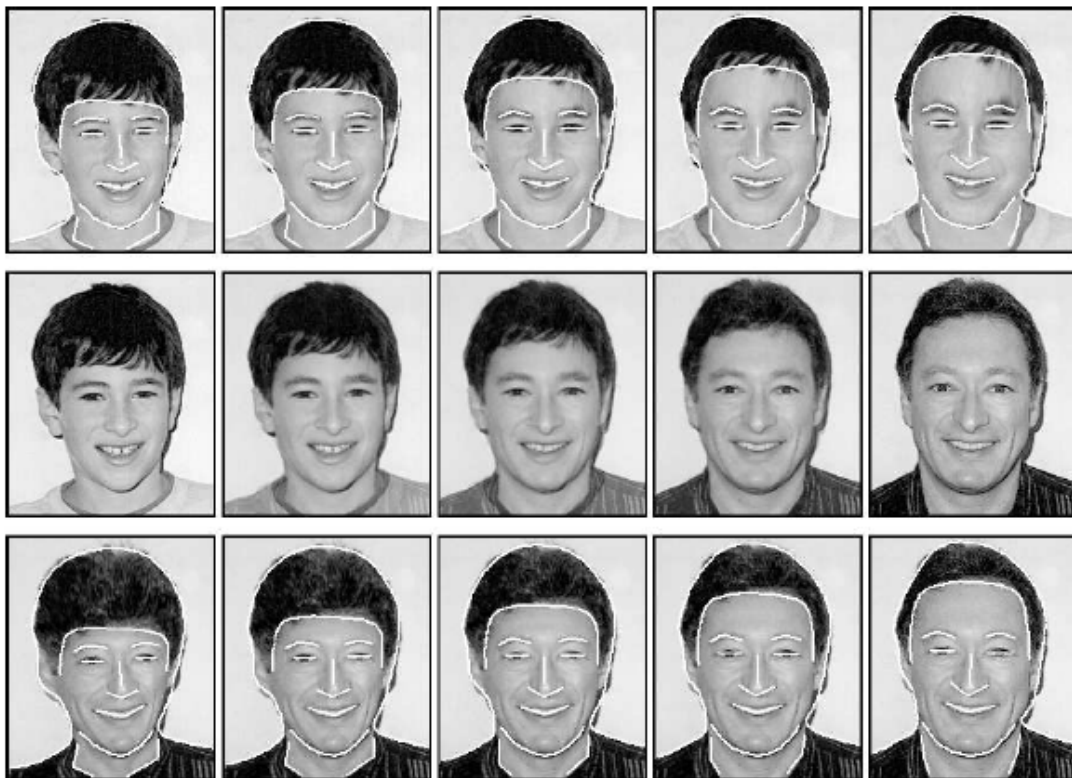
If  $f_s$  and  $f_d$  are polylines, the correspondence between them is established by their vertices. The correspondence between two snakes can be derived from the polylines that provide their initial positions. The feature correspondence between the source and destination image is translated to a set of point pairs sampled on the feature primitives.

Once the features are specified multilevel free-form deformations (MFFD) are used to achieve  $C^2$ -continuous, one-to-one warps among control point pairs [26]. The image is overlaid with a rectangular grid in the  $xy$ -plane. The grid is a regular lattice, slightly larger than the image, with intersection of the lattice corresponding to a pixel on the image. The image is deformed by manipulating the parallelepiped lattice overlaid on it. The basis function for the free-form deformation (FFD) is a bivariate cubic B-spline

tensor product, so that there is local control. This makes it possible to locally manipulate the lattice when a point on the grid is moved to the specified position. Therefore the new lattice producing this movement can even be computed for a large number of control points.

The one-to-one property is achieved by applying a sequence of FFD functions in lattices of finer densities, making sure that in each application the maximum displacement for the control points of a certain level does not exceed the threshold for that level. This process is repeated until the control points reach the deformation first specified by the user. Although this method guarantees smooth results the computational cost for it is very high.

Figure 4.9 displays an example of a multilevel free-form deformation based morphing.



*Figure 4.9 A Multilevel free-form deformation based morphing [54]*

#### **4.2.5 View morphing**

The question arises whether the methods discussed above preserve 3D shape. That is, does a morph of two different views of an object produce new views of the same object?

The answer is usually no, unless special care is taken. Usually morphing between images with the same 3D shape will result in shapes that are mathematically different. This means that the techniques discussed above can not handle changes in viewpoint or object pose. Seitz and Dreyer [41] devised a method that preserves 3D shape under interpolation. An image transformation is shape-preserving if from two images of a particular object, it produces a new image representing a view of the same object [12].

Computing the morph, using this technique, requires: **(i)** as with all the previous methods two images  $I_0$  and  $I_1$ , but in this case the images represent views of the *same* 3D object, **(ii)** a correspondence between the pixels in the images and **(iii)** unlike the previous mentioned methods, the projection matrices,  $\Pi_0$  and  $\Pi_1$ , for each image are also necessary.

Pixel correspondence can be achieved by user input and by automatic interpolation provided by existing morphing techniques, while the projection matrices can be computed using methods that require the internal camera parameters or the 3D positions of a number of image points [10].

#### 4.2.5.1 Parallel views

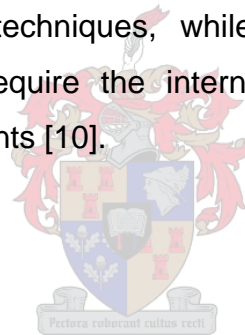


Figure 4.10 displays the morphing of parallel views. Linear interpolation of corresponding pixels in parallel views with image planes  $I_0$  and  $I_1$  creates image  $I_{0.5}$  - another parallel view of the same scene. The figure represents the case where two images,  $I_0$  and  $I_1$ , of a point  $\mathbf{P}$  on some object are taken from two different camera view points,  $\mathbf{C}_0$  and  $\mathbf{C}_1$ .

A camera is presented by a  $3 \times 4$  homogeneous projection matrix of the form  $\Pi = [H | -HC]$ , where the vector  $\mathbf{C}$  is the Euclidean position of the camera's optical center and the  $3 \times 3$  matrix  $H$  specifies the position and orientation of its image plane with respect to the world coordinate system.

Suppose the camera has moved from the world origin to position  $(C_x, C_y, 0)$  and it has also been zoomed out, as can be seen by the focal length changing from  $f_0$  to  $f_1$ .

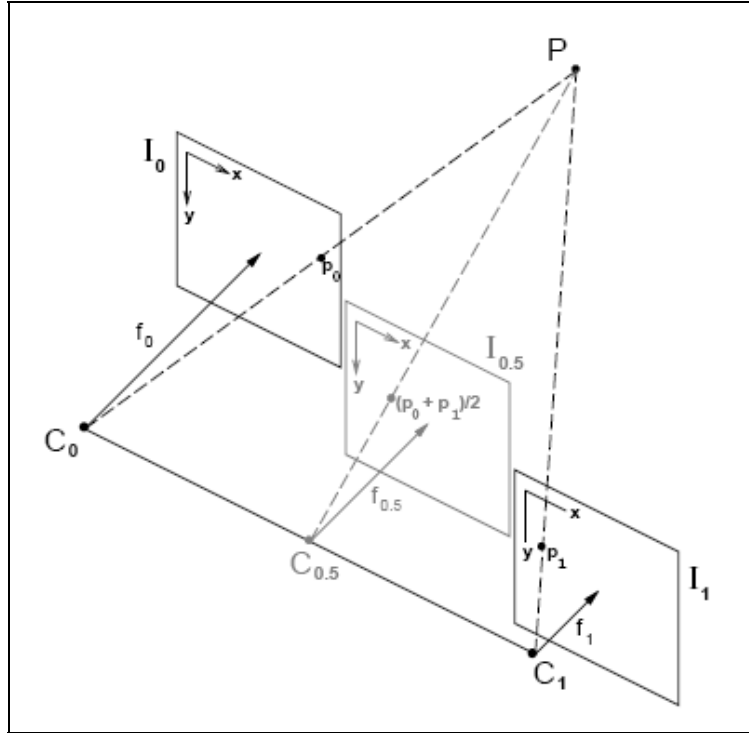


Figure 4.10 Parallel view morphing [41]

According to Chen and Williams [6] linear image interpolation produces new perspective views when the camera is moved parallel to the image plane. Therefore the projection matrices for the images can be written as follows [41]

$$\Pi_0 = \begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad (4.1.6)$$

$$\Pi_1 = \begin{bmatrix} f_1 & 0 & 0 & -f_1 C_x \\ 0 & f_1 & 0 & -f_1 C_y \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (4.1.7)$$

Let  $\mathbf{p}_0 \in I_0$  and  $\mathbf{p}_1 \in I_1$  be the projections of scene point  $\mathbf{P} = [X \ Y \ Z \ 1]$ . The point  $\mathbf{p}_t$  (on an intermediate frame) can then be calculated via linear interpolation as follows [41]



$$\mathbf{p}_t = (1-t)\mathbf{p}_0 + (t)\mathbf{p}_1 = (1-t)\frac{1}{Z}\Pi_0\mathbf{P} + (t)\frac{1}{Z}\Pi_1\mathbf{P} = \frac{1}{Z}\Pi_t\mathbf{P} \quad (4.1.8)$$

where

$$\Pi_t = (1-t)\Pi_0 + (t)\Pi_1. \quad (4.1.9)$$

Image interpolation thus produced a new view, whose projection matrix  $\Pi_t$ , is a linear combination of  $\Pi_0$  and  $\Pi_1$ , representing a camera at position  $\mathbf{C}_t = (tC_x, tC_y, 0)$  with focal length  $f_t = (1-t)f_0 + (t)f_1$ .

This means that interpolating images created by parallel cameras will produce the illusion of simultaneously moving the camera from  $\mathbf{C}_0$  to  $\mathbf{C}_1$  and continuously zooming. The image interpolation is shape-preserving because it creates new views of the same object [41].

#### 4.2.5.2 Non-Parallel Views

When dealing with non-parallel initial views it is only necessary to pre-warp them to corresponding parallel views and use the method described above to produce an interpolated image. This type of view morphing can be done in three steps [41]: (i) pre-warp, (ii) morph and (iii) post-warp. Figure 4.11 illustrates this process.

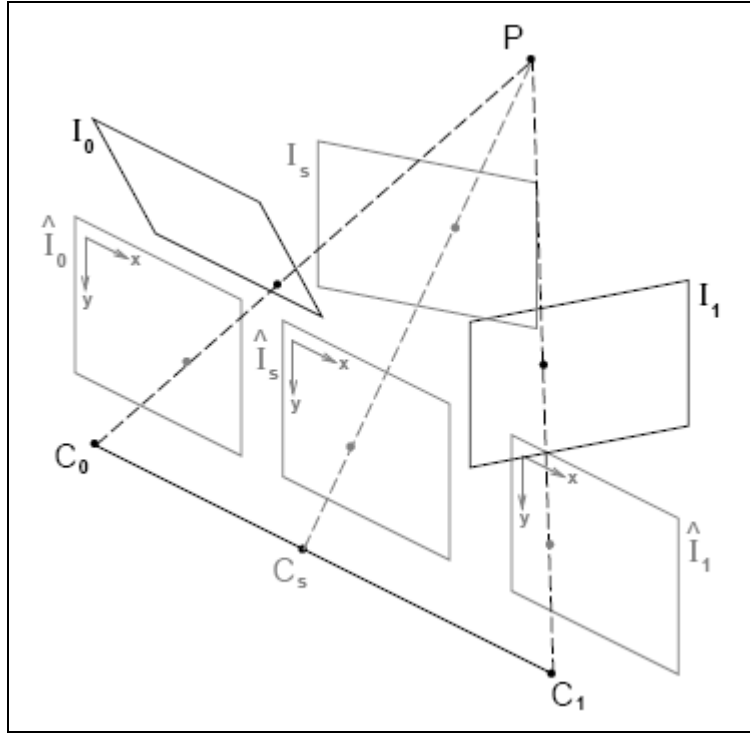


Figure 4.11 Non-Parallel View morphing [41]

### (i) Pre-warp

Pre-warping will bring the two image planes into alignment without changing the optical centers of the two cameras. It is done by applying projective transformations  $H_0^{-1}$  to image  $I_0$ , and  $H_1^{-1}$  to image  $I_1$  producing pre-warped images  $\hat{I}_0$  and  $\hat{I}_1$ . Assuming  $H$  is a  $3 \times 3$  matrix containing the location and orientation of the image plane, the projection matrices for  $I_0$ ,  $I_1$  and  $\hat{I}_s$  respectively are

$$\Pi_0 = [H_0 \mid -H_0 C_0], \quad (4.1.10)$$

$$\Pi_1 = [H_1 \mid -H_1 C_1] \quad (4.1.11)$$

and

$$\Pi_s = [H_s \mid -H_s C_s]. \quad (4.1.12)$$

Then

$$\hat{I}_0 = H_0^{-1} I_0 \text{ and } \hat{I}_1 = H_1^{-1} I_1 \quad (4.1.13)$$

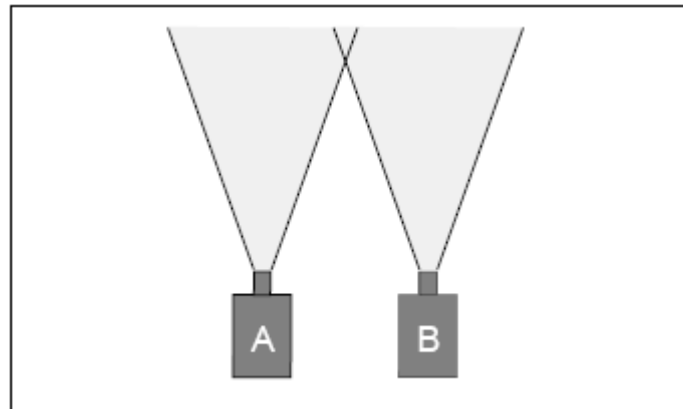
### (ii) Morph

$\hat{I}_s$  is produced by linearly interpolating pixels of  $\hat{I}_0$  and  $\hat{I}_1$  using equation (4.1.8)

### (iii) Post-warp

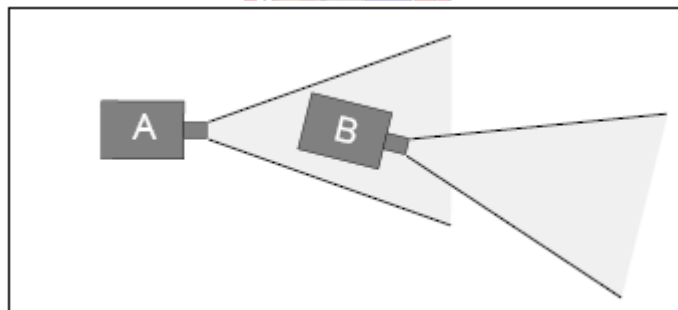
Finally  $I_s$  is produced by applying  $H_s$  to  $\hat{I}_s$ .

#### 4.2.5.3 Singular View Configurations



*Figure 4.12 Parallel views [41]*

For parallel views the optical center of one camera is never included in the field of view of the other camera as can be seen in Figure 4.12.



*Figure 4.13 Singular views [41]*

Reprojection changes only the viewing direction of a camera – not its field of view. That means that for a pair of views, where the optical center of one camera is within the field of view of the other camera, the view cannot be made parallel by pre-warping. Figure 4.13 displays a pair of singular views for which pre-warping cannot be computed.

## 4.3 Transition Control

Setting up the rate at which warping and color blending takes place during a morph sequence is called the transition control. Most of the morphing techniques discussed above make use of a uniform transition rate - this means that the positions of the features in the source image changes to their corresponding destination positions at a fixed, constant rate. When the transition rate is different from part to part for in-between images in a morph sequence, interesting results can be expected. Such non-uniform transition functions can improve the visual effect of the morph.

In mesh-based techniques transition control is achieved by assigning a transition curve to each mesh node. This can be difficult when complicated meshes are used to specify features. According to [35] the transition speed can be defined by a Bézier function defined on the mesh. Other techniques use a deformable surface model to manage transition control by selecting a set of points on an image and specifying a transition curve for each point [27, 54].

In Figure 4.14 a uniform transition function is applied to the warping of the source and destination images. This can be seen in the top and bottom rows of the figure. Notice that all the points in the source and destination images are moving at a uniform rate. The two rows (top and bottom) of images are then added together to result in the middle row of in-between images. Geometric alignment is maintained between the two rows (sets) of in-between warped images, before color blending morphs them into the final morph sequence (middle row).

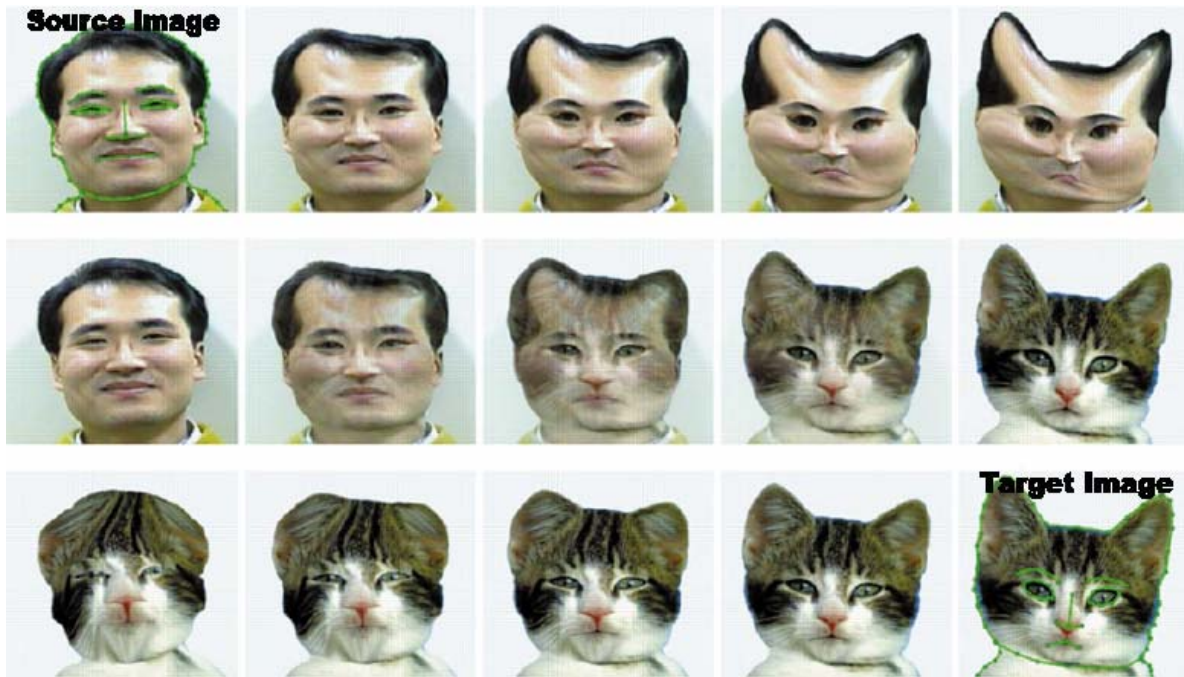


Figure 4.14 Uniform transition rate morph [54]

Figure 4.15 shows what will happen when a non-uniform transition function is applied to the same source and destination images. In this example the transition function is defined to accelerate the warping of the nose in the first few images, while leaving the shape of the head as it is for the first half of the sequence. The top and bottom row are color blended to form the morph sequence (as seen in the middle row).



Figure 4.15 Non-Uniform transition rate morph [54]

## 4.4 Summary

This chapter discussed some of the most well known warping and morphing techniques for two dimensions. The biggest difference in the techniques is the way that the warping is done. The blending part of the morph is rather straight forward and is implemented in all the techniques as a color interpolation.

A more general discussion of morphing is given in [1] where morphing is used to describe objects as a composite of other objects. A set of objects produced by morphing among multiple objects forms a mathematical space, called the morphing space. A number of properties of morphing functions and morphing spaces are discussed and general algorithms for the synthesis and analysis of objects in morphing spaces are also provided.

The next chapter will focus on Delaunay triangulation, an important element in the warping section of the morphing implementation of this thesis.



# Chapter 5

## Triangulation

Triangulation is a word used for the general problem of subdividing a complex domain into a disjoint collection of triangles. The simplest region into which a planar object can be decomposed is a triangle.

Triangulation plays a vital part in the implementation created for the purpose of this thesis. It subdivides each of the input images (the source and destination image) into a triangular mesh of sub-images. These triangular sub-images are then warped and blended to produce the morph from the source image into the destination image.

The triangulation method used is Delaunay triangulation. A set of control points are specified by the user, for each input image. This set of points is then used as input for the Delaunay triangulation. Delaunay triangulation, a well known and very popular triangulation method, is chosen since it maximizes the minimum angle of the generated triangles. This chapter gives a brief overview of Delaunay triangulation.

### 5.1 Triangulation vs. Delaunay Triangulation

For the purpose of this thesis triangulation is only considered in two-dimensions (2D).

#### 5.1.1 Triangulation

A triangulation of a set of points  $P$  in 2D is a set of triangles  $T$  whose:

- (i) vertices are collectively  $P$ ,
- (ii) union is the convex hull of  $P$ ,
- (iii) interiors do not intersect each other.



### 5.1.2 Delaunay triangulation

The Delaunay triangulation for a set of points,  $P$ , in 2D is the triangulation  $DT(P)$  of  $P$  such that no point in  $P$  falls in the interior of the circumcircle of any triangle (the circle that passes through all three of its points) in  $DT(P)$ . Figure 5.1 displays the Delaunay triangulation of a random set of points.

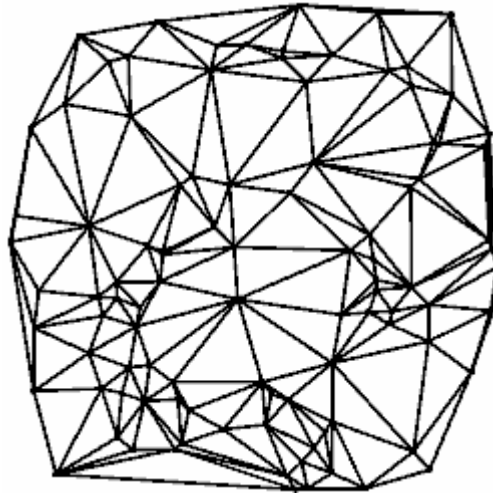


Figure 5.1 Delaunay triangulation of a random set of points.

## 5.2 Properties of Delaunay Triangulation

### 5.2.1 Uniqueness

The Delaunay triangulation for a given vertex set is unique. That means that a given set of data points will produce the same triangulation regardless of the order in which the points are given.

### 5.2.2 Nearest Neighbor

In the Delaunay triangulation every vertex is connected by lines to its nearest neighbors in such a way that all lines are edges of triangles and do not intersect.



### 5.2.3 Convex hull

The exterior face of the Delaunay triangulation is the convex hull of the vertex set [34].

### 5.2.4 Empty circumcircle

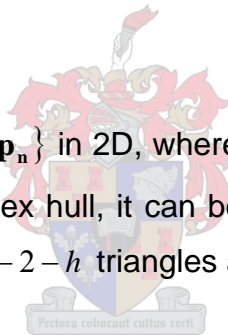
The circumcircle of a triangle is the unique circle that passes through all three of its vertices. Every triangle of a Delaunay triangulation has an empty circumcircle [34].

### 5.2.5 Empty circle

Two points  $\mathbf{p}_i$  and  $\mathbf{p}_j$  are connected by an edge in the Delaunay triangulation if and only if there is an empty circle passing through  $\mathbf{p}_i$  and  $\mathbf{p}_j$  [34].

### 5.2.6 Euler's Formula property

Given a set of points,  $P = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$  in 2D, where  $n$  is the number of points and  $h$  is the number of vertices on the convex hull, it can be proved by Euler's formula that the Delaunay triangulation will have  $2n - 2 - h$  triangles and  $3n - 3 - h$  edges [34].



### 5.2.7 Maximizes minimum angle

Delaunay triangulation maximizes the minimum angle of all the angles of the triangulation and tends to avoid skinny triangles [34].

## 5.3 Algorithms

Various algorithms exist for constructing Delaunay triangulations. Some of them will be briefly discussed below. A comparison of existing algorithms can be found in [48].

### 5.3.1 Delaunay Triangulation from Voronoi Diagrams

The Voronoi diagram of a set of points  $P$  is a subdivision of the plane into a set of polygons in such a way that each Voronoi polygon of point  $p_i$  contains all locations that are closer to  $p_i$  than to any other point of  $P$ . For an explanation on calculating Voronoi diagrams see [12, 13, 20].

In 2D, the geometric dual of the Voronoi diagram is the Delaunay triangulation. This means that the Delaunay triangulation can be obtained by drawing line segments between two vertices, if their Voronoi polygons have a common edge. See Figure 5.2.

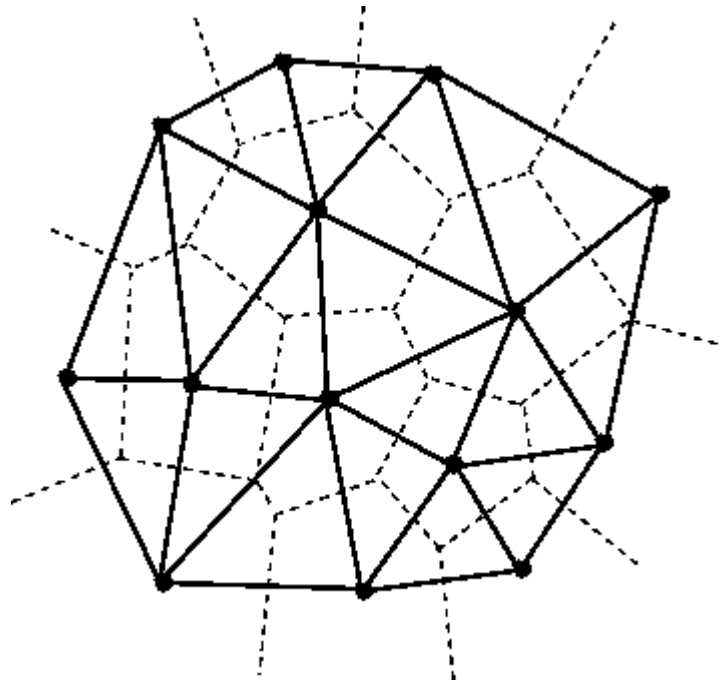


Figure 5.2 Voronoi diagram (dashed lines) containing Delaunay triangulation (solid lines)

### 5.3.2 Incremental Insertion Algorithm

The most straightforward method for computing a Delaunay triangulation starts by forming a triangle, surrounding all the given vertices. The algorithm proceeds by repeatedly inserting one vertex at a time and then re-triangulating the parts of the graph that are affected. When a vertex is inserted, a search is done to locate all the triangles whose circumcircles contains the vertex. Those triangles are then removed and that part of the graph is triangulated again.

The earliest such algorithm was introduced by Lawson [24] and is based upon edge flips.

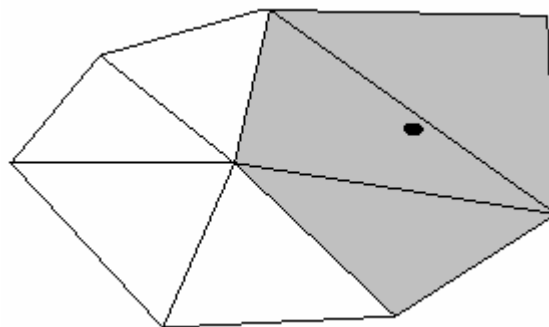
### 5.3.2.1 Lawson's Algorithm

This algorithm is also known as the diagonal swapping algorithm. Circumcircles are calculated for all new triangles, created when a vertex is added to an existing triangular mesh. If any of the neighboring vertices lie inside the circumcircle of any triangle, then a quadrilateral is formed by the triangle and its neighbor. The diagonals of the quadrilateral are flipped (see Section 5.3.5) to create a new triangulation. This process repeats until there are no more faulty triangles (triangles with a non-empty circumcircle) and no more flips required.

### 5.3.2.2 Watson's Algorithm

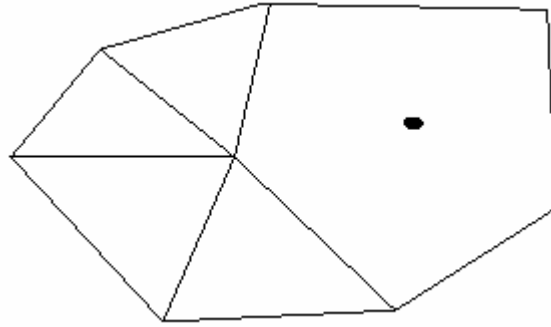
Bowyer [5] and Watson [50] simultaneously introduced an algorithm that does not use edge flips.

When a new vertex is inserted a search is made for all the triangles whose circumcircles contain the new vertex (see Figure 5.3 – the circumcircle of the shaded triangles contains the new vertex).



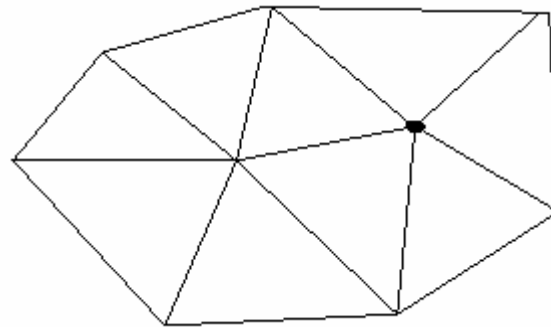
*Figure 5.3 Circumcircle of shaded triangles contains the new vertex*

These triangles are no longer Delaunay, and are thus deleted (see Figure 5.4). All the other triangles are left undisturbed as they will stay Delaunay. The set of deleted triangles form an insertion polygon, which is left vacant by the deletion of the triangles.



*Figure 5.4 Non-Delaunay triangles are deleted*

Each vertex of the insertion polyhedron is then connected to the new vertex with a new edge as seen in Figure 5.5. These new edges are Delaunay.



*Figure 5.5 New triangulation*

A way to speed up this method is by keeping the history of the triangulation in the form of a tree-structure. Elements replacing a conflicting element in an insertion are called its children. When a parent conflicts with a point to be inserted, so does its children. This provides a fast way of getting the list of triangles to remove (which is the most tedious part of any incremental insertion algorithm) [52].

### 5.3.3 Divide-And-Conquer Algorithm

In this algorithm a line is drawn recursively to split the vertices into subsets, until each subset contains two or three vertices each. Each of these subsets is easily triangulated to form edges or triangles. Finally (and this is the difficult part) the subsets are merged together.

In the implementation discussed in [20] each triangulated subset is surrounded with a layer of “ghost” triangles. The ghost triangles are connected to each other in a ring about a vertex at infinity (a single edge being represented by two ghost triangles). This is illustrated in Figure 5.6. On the left is the representation of an isolated edge and on the right is a representation of an isolated triangle. The dashed lines are the ghost triangles. The white vertices all represent the same vertex at infinity, while the black vertices represent the vertices in the subset.

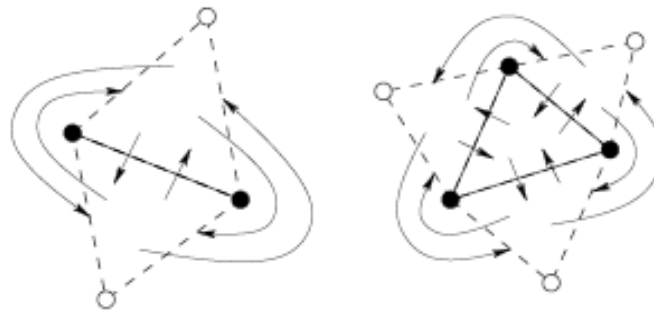


Figure 5.6 Ghost Triangles [43]

Ghost triangles are used for the merging step. Some are transformed into real triangles during this step; two triangulations are sewn together by fitting their ghost triangles together. Some edge flips are also needed. After the merging step, the ghost triangles are removed. See Figure 5.7 where dashed lines represent ghost triangles and triangles displaced by edge flips and where the shaded triangles are non-Delaunay triangles and needs to be displaced by edge flipping.

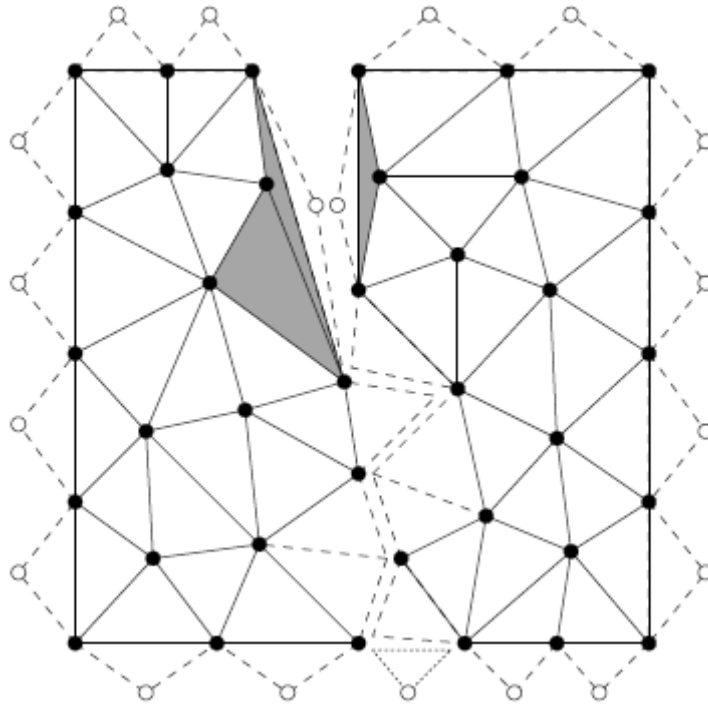


Figure 5.7 Merging Step [43]

Another algorithm which makes use of the divide-and-conquer method is discussed in [8].

### 5.3.4 Sweepline Algorithm

In 2D a horizontal line is swept from the bottom to the top (or from left to right). This line, called a sweepline, is halted at so-called event locations where the status of the sweepline is updated. Between events the sweepline does not have to be halted because its status does not change. The status of the sweepline and the type of events depend on the application. For the construction of the Delaunay Triangulation such an algorithm has been implemented by [12]. An event occurs when the sweepline reaches a point in the point set or when it passes a circle formed by three adjacent vertices of the current mesh boundary. New elements are created and the status of the boundary edges is updated [11].

### 5.3.5 Flipping Algorithm

The flipping algorithm starts with an arbitrary triangulation and searches for an edge that is not locally Delaunay. Each edge on the boundary (convex hull) of the triangulation is considered to be locally Delaunay. Edges that are not on the boundary are checked. Whenever the flip algorithm identifies an edge that is not locally Delaunay, that edge is flipped. Flipping an edge means deleting it, thereby combining the two neighboring triangles into a single containing quadrilateral and then inserting the crossing edge of the quadrilateral. In Figure 5.8 the triangulation on the left is flipped to form the triangulation on the right.

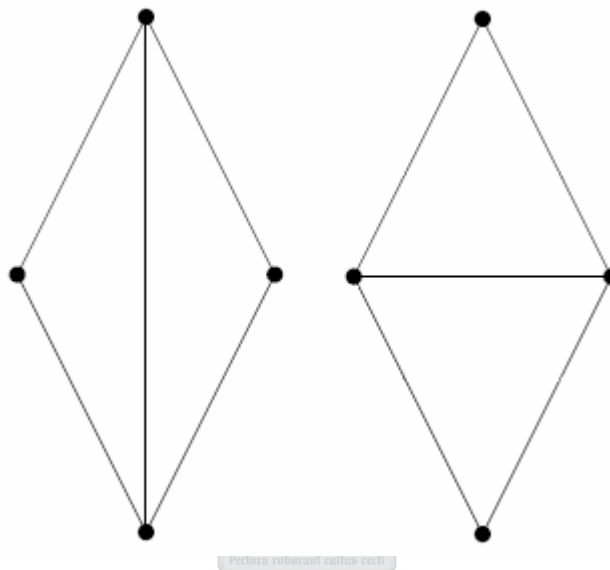


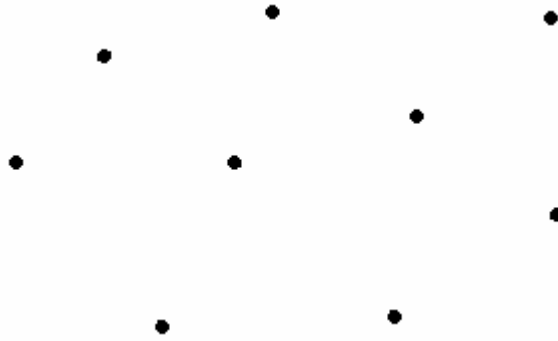
Figure 5.8 An Edge Flip

The success of the algorithm relies on the fact that if any edge of the triangulation is not Delaunay, it means that there is an edge that is not locally Delaunay and that can be flipped [44].

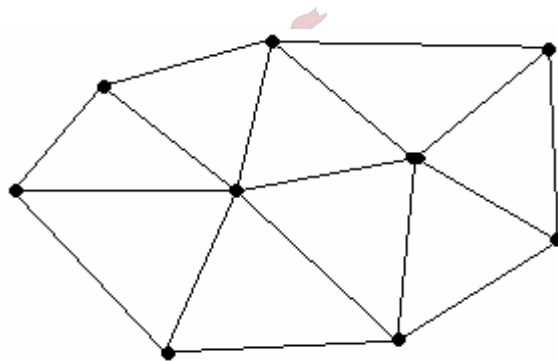
## 5.4 Triangle

“Triangle” is a C-program created for 2D mesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations and Voronoi diagrams. The program was created by Jonathan Richard Shewchuck and is available on the web [43, 45]. It guarantees quality mesh generation by making use of Rupert’s Delaunay refinement algorithm [36].

Triangle's default behavior is to find the Delaunay triangulation for a set of vertices. This is the only function of Triangle that is used/explored for the purpose of this thesis. Triangle is given a set of vertices coordinates as input and produces the vertices connected as Delaunay triangles as output. An example can be seen in Figure 5.9 and Figure 5.10.



*Figure 5.9 Set of input vertices*



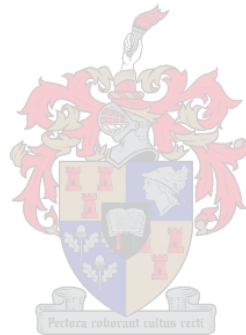
*Figure 5.10 Set of output Delaunay triangles*

In the implementation created for the purpose of this thesis Triangle is used to subdivide the images into a set of triangles. It is given as input the coordinates of the control points as well as the coordinates of the four corner points of the source image. This process will be discussed in more detail in the following chapter.



## 5.5 Summary

This chapter discussed Delaunay triangulation and the relevance of mentioning it in this thesis. The next chapter will describe the implementation created for this thesis. It will discuss how Delaunay triangulation, warping and blending were combined to create the morphing system. A few existing implementations will also be looked at for interest sake.



# Chapter 6

## Software

This chapter describes a basic model that can be used to create a warping and morphing system, describes the software implemented for the purpose of this thesis and looks at a few examples of existing morphing software.

### 6.1 The Model

The model will clearly separate the representation from the computation and it is decomposed into several computational elements [18, 19].

A morphing system will usually be designed for a specific graphical object (such as images), user interface and warping/morphing technique, depending on the application.

#### 6.1.1 Computational elements

As previously mentioned, a morph consists of two warps (one for the source object and one for the destination object) and a blending operation. Based on this concept of the morphing operation a morphing system should consist of the following six basic elements:

1. Graphical object representation
2. Transformation specification/representation
3. Warping Reconstruction
4. Mapped Object Computation
5. Shape Blending
6. Attribute blending

### **6.1.1.1 Graphical Object Representation**

A data structure that will describe the graphical object, encapsulating the description of its shape and attributes, should be defined. A typical application would only handle a specific type of graphical object - such as an image, represented by a matrix of pixel values. A more generic application should be able to handle a variety of graphical objects, for example images, surfaces, plane curves, volumes, etc. Once again the representation will depend on the specification of the user and on what the application will be used for [18].

### **6.1.1.2 Transformation specification / representation**

Specifying the transformation will consist of a discrete representation of the transformation (which is usually obtained from user input) [19]. Typically a user will specify values for the warp as a discrete set of points at an initial and final state, for example the control points in the source image and their corresponding points in the destination image.



### **6.1.1.3 Warping Reconstruction**

The warping reconstruction uses the discrete representation of the transformation to compute the transformation values at any points of interest. Interpolating the key states in time and extending the transformation values to all points of the graphical object is also included here [18, 19].

### **6.1.1.4 Mapped Object Computation**

Here the elements of the representation of the graphical object are enumerated, by traversing its structure, to apply the warping to the graphical object. The computation of the warped/transformed object is closely related to the reconstruction of the warp/transformation (6.1.1.3) and in some cases both these elements are combined in the actual implementation [18].

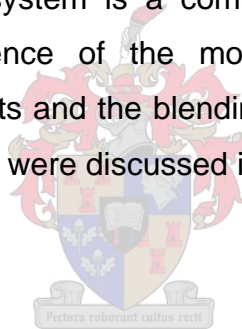
### 6.1.1.5 Shape Blending

Shape blending is necessary when there is not a perfect alignment between the geometry of the two graphical objects. When working with images this step is usually not present because the image boundaries should be perfectly aligned, however when working with other graphical objects this step is of fundamental importance [18].

### 6.1.1.6 Attribute Blending

This operation is used to compute the resulting attribute function from the attributes in the source and destination graphical objects. This method is dependent on the nature of the attributes. For example when working with images, this operation will blend the color of the source pixel with that of the destination pixel using some kind of blending function.

Although a morphing technique system is a combination of all six of the elements discussed above, the real essence of the morph is contained in the type of transformation, the graphical objects and the blending operation being used [18]. Some of the various morphing techniques were discussed in Chapter 4.



## 6.2 Implementation

The morphing system, created for the purpose of this thesis, was implemented in Visual C++ and OpenGL and runs under Windows. It also makes use of “Triangle”, a C-program implemented by Jonathan Richard Shewchuck [45]. This section will describe the implementation in terms of the model described in 6.1.

### 6.2.1 Graphical Object Representation

The implementation was created specifically for handling images. Images are made up of pixels (picture elements), each being a specific color. The eye, not being able to see the individual pixels blends them together to form the overall picture. The image is stored in the computer memory in the form of a pixmap (pixel map) - a matrix of numbers between 0 and 255. Each number representing the value (color) of the pixel stored at that position.

24-bit bitmap files (\*.bmp) are used for storing the images. OpenGL reads the \*.bmp files and stores it as a texture.

### 6.2.2 Transformation specification / representation

The user selects a number of control points from each image, using the mouse pointer. These control points are stored as  $xy$ -coordinates in an array (each image has its own array) containing all the control points, as displayed in Figure 6.1.

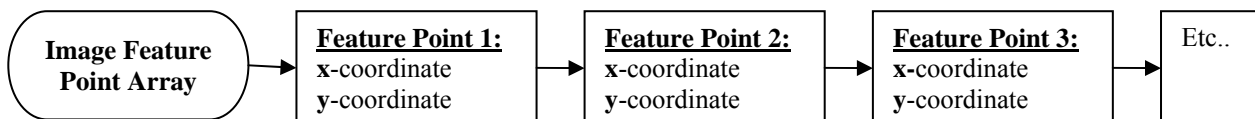


Figure 6.1 Graphical representation of the control point array

These points are then written to a file to be used by the triangulation program. The format of the file is illustrated in Figure 6.2.

```
<number of feature points> <dimension of each point> <0???\>  
  
<array index of point> <x-coordinate> <y-coordinate>  
<array index of point> <x-coordinate> <y-coordinate>  
<array index of point> <x-coordinate> <y-coordinate>  
...etc...
```

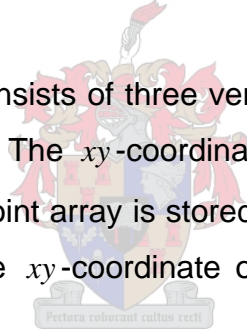
*Figure 6.2 Format of file containing control points*

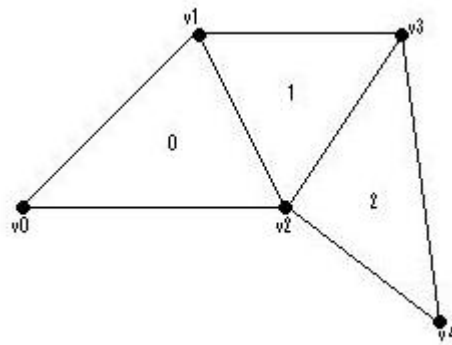
A Delaunay triangulation is performed on the control points to create a number of triangles. These triangles are stored in a file with the format shown in Figure 6.3:

```
<number of triangles> <no of points in each triangle> <0???\>  
  
<triangle index> <index of vertex1> <index of vertex2> <index of vertex3>  
<triangle index> <index of vertex1> <index of vertex2> <index of vertex3>  
<triangle index> <index of vertex1> <index of vertex2> <index of vertex3>  
...etc...
```

*Figure 6.3 Format of the file containing the triangles*

Each triangle has an index and consists of three vertices. Each vertex being one of the previously specified control points. The  $xy$ -coordinate is not stored in the file. Only the index of the vertex in the control point array is stored. By using this index and searching for it in the control point array the  $xy$ -coordinate of the vertex can be obtained. See Figure 6.4.





**Feature point array:**

v0 = pt [0] = (0,10)  
 v1 = pt [1] = (7,20)  
 v2 = pt [2] = (10,10)  
 v3 = pt [3] = (15,20)  
 v4 = pt [4] = (17,0)

**Triangles:**

T0 = (0,1,2)  
 T1 = (1,3,2)  
 T2 = (3,4,2)

**Output file  
containing  
feature points:**

```
5 2 0
0 0 10
1 7 20
2 10 10
3 15 20
4 17 0
```

**Output file  
containing  
triangles:**

```
3 3 0
0 0 1 2
1 1 3 2
2 3 4 2
```

Figure 6.4 Example of output files created

### 6.2.3 Warping Reconstruction

After selecting the corresponding control points in both the source and destination images, these points are used as input for “Triangle” (the triangulation program used). “Triangle” sub-divides the images into a set of Delaunay triangles.

Once the images are sub-divided into a set of triangles the warping from the source image to the destination image is done by interpolating, over time, each triangle in the source image to the corresponding triangle in the destination image. Once the new, intermediate, triangle is known the warping of the texture segment (image segment) inside that triangle is performed by OpenGL's texture mapping function.





### 6.2.4 Shape blending

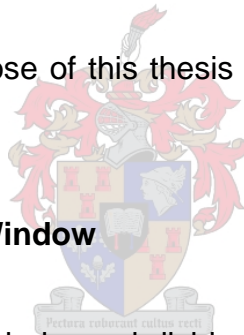
Although both images are expected to be rectangular they don't have to be the same size. The program will resize both images to be the same size.

### 6.2.5 Attribute Blending

The colors of the two images are gradually blended over time, using OpenGL's blending function. At the beginning of the transformation the source image is fully visible and the destination image is not visible at all. As the transformation moves from the source image to the destination image, the source image will gradually become less visible as the destination image becomes more visible.

## 6.3 The Actual Program

The program created for the purpose of this thesis is discussed in this section. Source code is available on request.



### 6.3.1 Layout of the Application Window

The program consists of a parent window subdivided into 4 smaller windows (see Figure 6.5):

1. The top-left sub-window contains the source image.
2. The bottom-left sub-window contains the destination image.
3. The top-right sub-window, when prompted, displays the morphing sequence. If required, this window can also be prompted to display the warping of either the source image or the destination image.
4. The bottom-right sub-window, when prompted, displays the interpolation of the triangles.



*Figure 6.5 Initial window*

The user has to select a number of corresponding control points in each image. Note that these points have to be chosen in the same order. The selection of the control points determines the quality of the morph. The more points selected, the more satisfying the morph will be. There is a limit of 25 points that can be selected.

Once the control points are specified the user must select the triangulation command. This command will perform a Delaunay triangulation on the selected control points. The files needed for the triangulation program are created and the triangulation program is executed on the source image. The same triangulation is used for the destination image; however the destination control point coordinates are used for the vertices of the triangles. The results are then displayed over each image.

Once the triangulation is completed, the user can select the command for calculating and displaying the morphing sequence. The morph is calculated with 10 intermediate images.

Figure 6.6 displays the triangulation and morphing sequence.

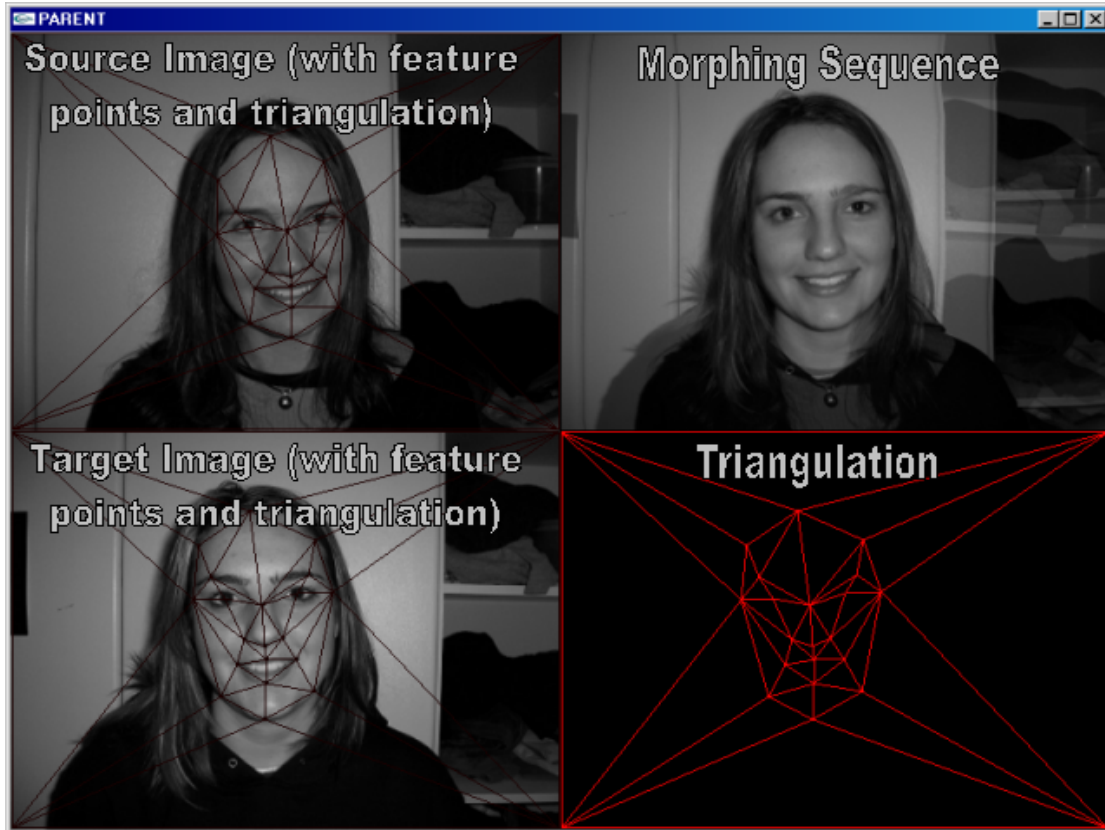
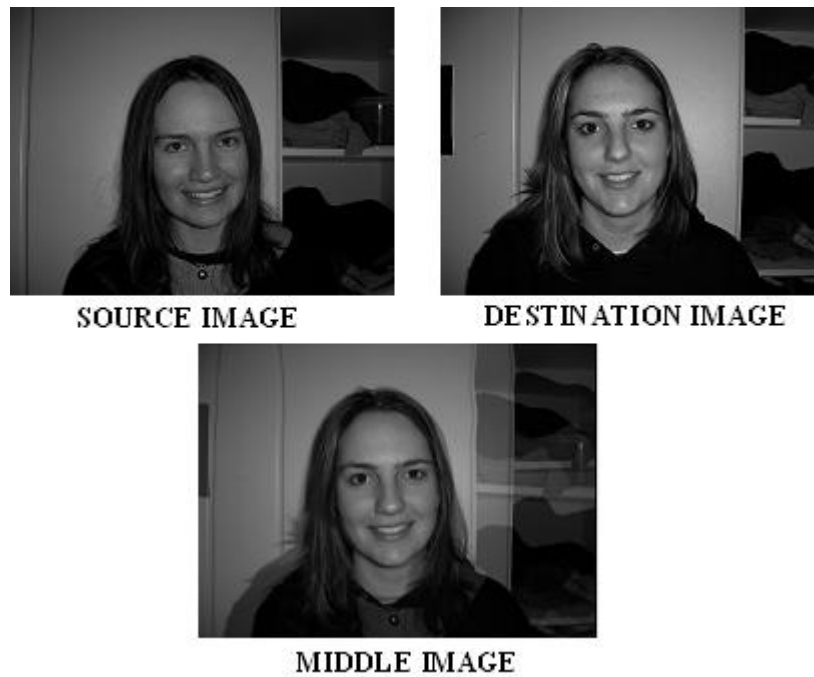


Figure 6.6 Window displaying the control points, triangulation and the morphing sequence

### 6.3.2 Commands

- “t” generates the Delaunay triangulation for the source and destination image and draws them over the images.
- “q” / “w” interpolates the triangles from the source triangles to the destination triangles and visa versa.
- “z” / “x” generates the morph from the source image to the destination image and visa versa.

Figure 6.7 displays the source image (top-left), the destination image (top-right) and the middle image of the morphing sequence created by the implementation discussed above. Because the middle image is believable this will result in a satisfying morphing sequence.



*Figure 6.7 Middle image created by the implementation*

## 6.4 Morphos

Morphos [19] was implemented in C++ and currently runs on Windows. It uses OpenGL for the 3D interface and for real-time previews of image warps. Its main purpose is to be used for research and experimentation. The source codes are available to view and manipulate and therefore Morphos is a “living” system that can be evolved and improved over time.

The version of Morphos that was investigated supports:

- (a) images,
- (b) polygonal curves and
- (c) surfaces.

Several techniques are implemented in the system. They include:

- (i) meshes,
- (ii) features,
- (iii) point specifications,
- (iv) linear and exponential dissolve attribute combinations,
- (v) field-based, radial-basis functions,

- (vi) two-pass spline mesh warping,
- (vii) projective warping, etc.

This gives the user a wide variety of techniques to choose and compare from.

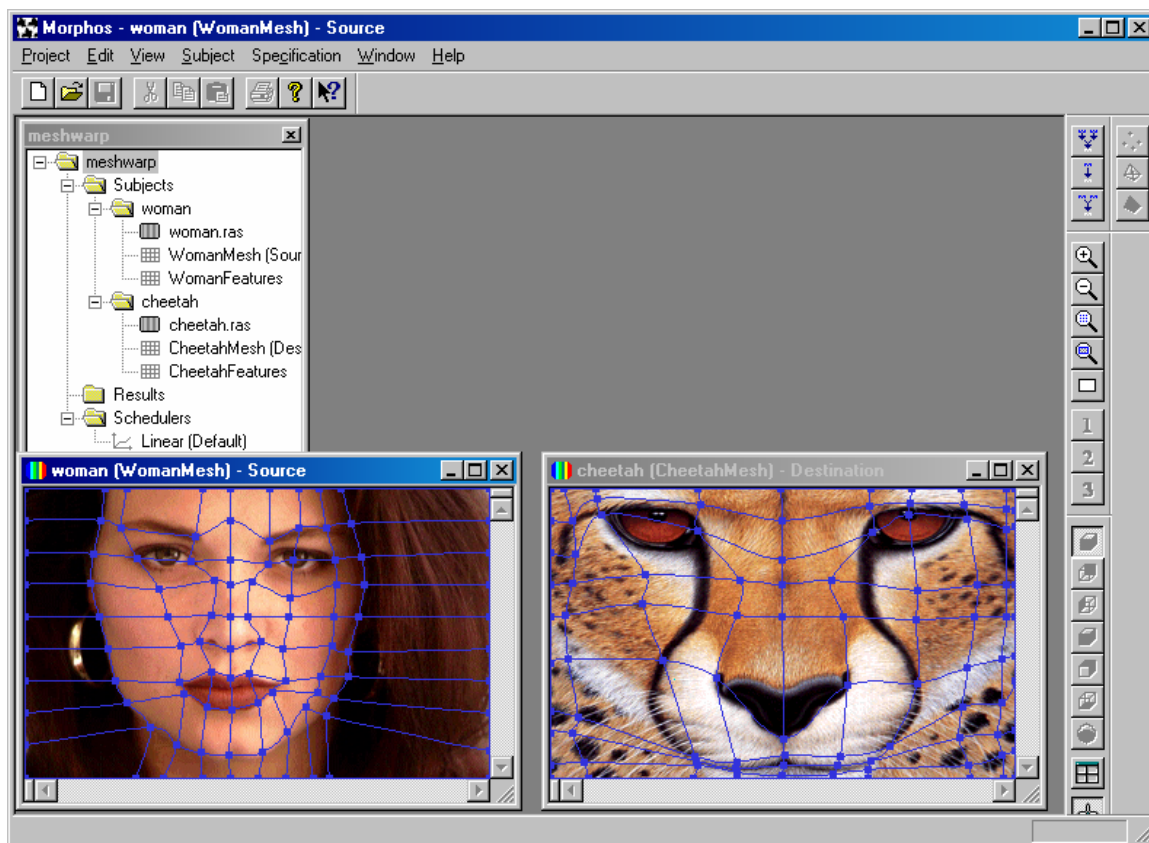


Figure 6.8 Typical Morphos User Interface

A basic Morphos project workspace will look something like Figure 6.8. The source and destination images, together with their corresponding features are specified. The features can be represented by one of the types in Figure 6.9.

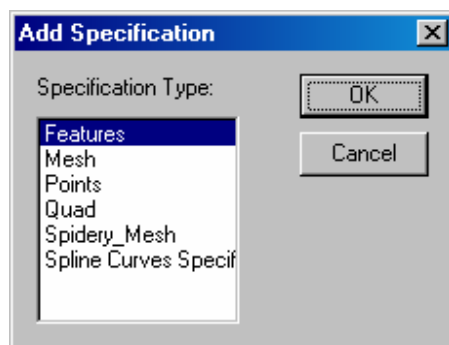


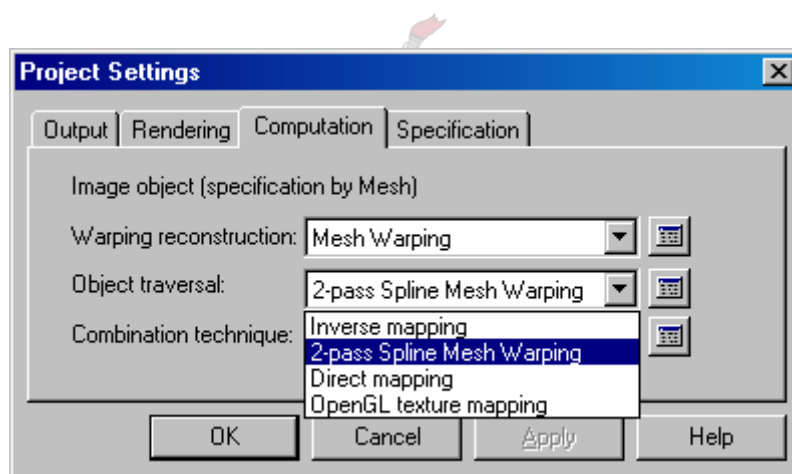
Figure 6.9 List of supported feature specification types

When the images have been loaded and the features specified, the warping, the cross dissolve and the morph can be viewed separately by clicking on one of the buttons in Figure 6.10. Clicking on the top button will display the entire morphing sequence, clicking on the middle button will display the warp and clicking on the bottom button will display the cross-dissolve.



*Figure 6.10 Buttons to select between a morph, warp and cross-dissolve action*

When clicking on one of these buttons the user is also prompted to select one of the available techniques (shown in Figure 6.11) to be used for calculating the transformations.



*Figure 6.11 Window for selecting the transformation technique*

Once this is done the user will be able to view the transformation as a number of images. Therefore it can once again be stated that Morphos is perfect to use for comparing different techniques and also for getting a feel for morphing, warping and cross-dissolving in general. It is also useful to introduce some of the different types of graphical objects that exist.

## 6.5 FantaMorph 3

Abrosoft's FantaMorph 3 is a commercial product that is specifically designed and optimized for images. It can easily create fantastic image morphs and warp movies. FantaMorph supports most image formats including BMP, JPEG, TIFF, PNG, GIF, TGA, PCX, and even professional 32-bit with alpha formats. The morph/warp sequence can then be exported to an Image Sequence, an AVI-file, an animated GIF file, a Flash file, a screen saver, a standalone EXE and various other file-formats. FantaMorph3 takes advantage of hardware acceleration and the rendering speed easily goes up to several hundred frames per second. The high speed makes it possible to play final effects in real time without exporting to a file.

For the purpose of this thesis only the trial version was explored.

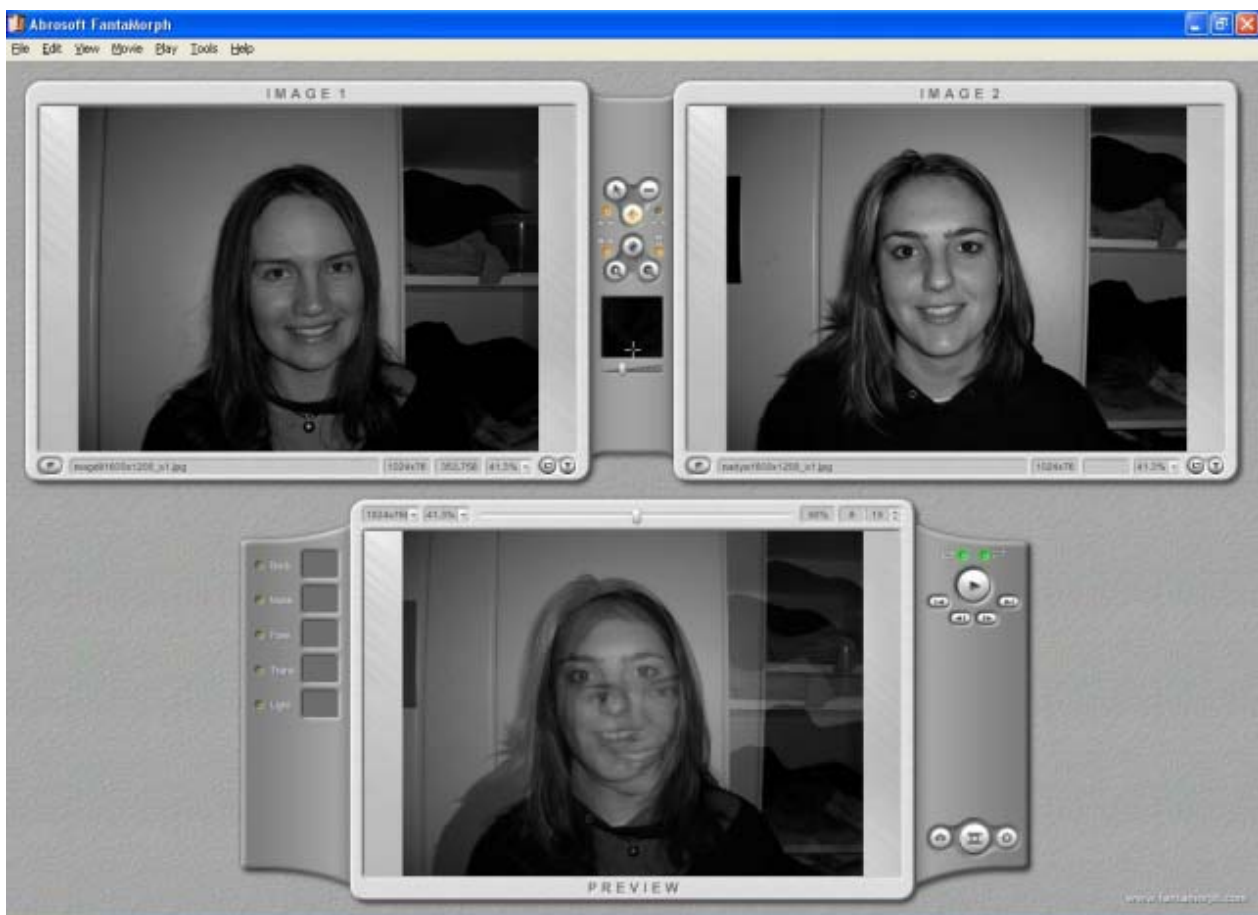
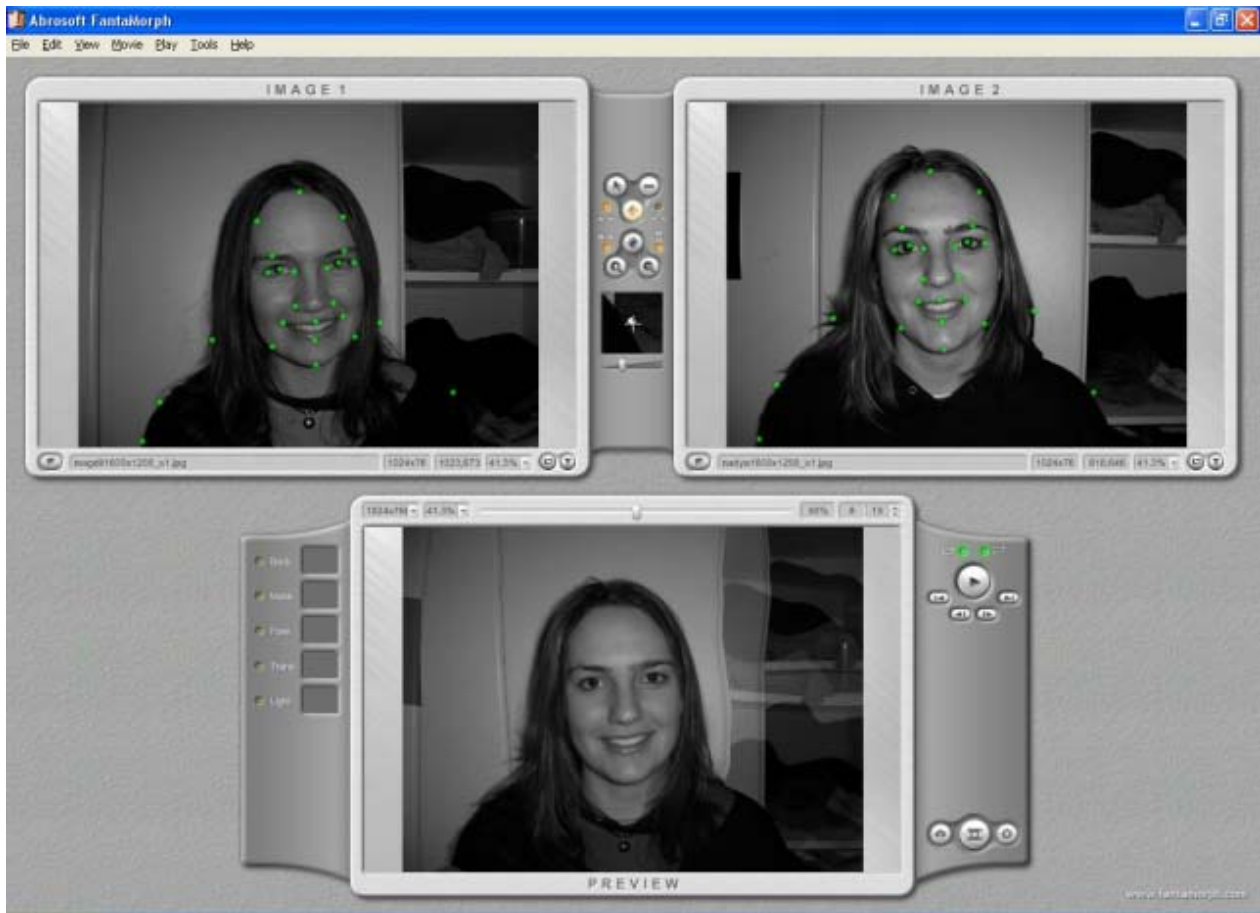


Figure 6.12 A typical Fantamorph3 project workspace



A typical FantaMorph3 project will look something like Figure 6.12. It contains a source image (left), a destination image (right) and a preview of the result (bottom).

Features can be specified by selecting corresponding points in the source and destination images as seen in Figure 6.13.



*Figure 6.13 A project workspace with corresponding, user-defined, control points*

The bottom-center frame in Figure 6.13 (marked as “Preview”) will display the morph result as a movie clip.

Appendix C displays an image sequence created by FantaMorph3, using the images and control points in Figure 6.13. The images are displayed left to right, top to bottom.



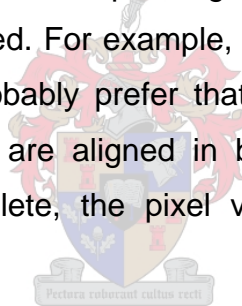
# Chapter 7

## Conclusions and Future Work

### 7.1 Conclusions

The problem of finding a way to create a smooth transition from one image to another image (known as image morphing) was discussed in this thesis. Because the focus of this thesis was on images, only morphing in two-dimensions (and more specifically morphing of images) was discussed.

As explained in the text a morph consists of two warps, followed by a blend. The term “warp” refers to the geometric transformation of an image. In other words transforming the images in such a way that the corresponding features of each image are aligned before the images are color blended. For example, if a morph is done from one person to another person, one would probably prefer that features such as the eyes, nose, mouth, ears, etc in both images are aligned in both images before they are color blended. With the warping complete, the pixel values of the images are blended (interpolated) to finish the morph.



A few existing techniques to create image morphs were discussed, where the only significant difference between them is the manner in which the warping is performed. The two-pass mesh algorithm being one of the first techniques created had the problem of requiring a finite number of control points in the mesh. This lead to sometimes having too many control points in some region and/or too few in another region. Thus the user might not have sufficient control over all the areas.

Field morphing gives the user much more control over specific areas that require it and areas that do not are basically left unchanged. The problem with this technique was that sometimes unexpected interpolations (referred to as “ghosts”) could be generated.

A tedious step in morphing is specifying the control points. If the control points are not correctly specified the morph is bad. Snakes simplify this task, because control points must only be placed close to the features. The snake minimizes its energy and locks onto the feature. Snakes used together with multilevel free-form deformations guarantees smooth morphing results, but for a very high computational cost.

Finally a technique, called “view morphing”, was discussed. The advantage of this technique being that it is shape-preserving.

Delaunay triangulation was introduced, because it is a very popular triangulation method. Various techniques to create a Delaunay triangulation exist and some of them were mentioned. A program, called “Triangle”, created by Jonathan Richard Shewchuck [43, 45] was used to generate the Delaunay triangulation of the images used in the morphing technique implemented in this thesis.

Finally an implementation to generate an image morphing sequence was presented. The technique starts by subdividing the source and destination images each into a set of triangles. After completing the triangulation, the triangles in the source image are interpolated over time to the corresponding triangle in the destination image. Once a new, intermediate triangle is known a texture mapping is performed (handled by OpenGL) to map the texture of the old triangle to the new triangle. With this warping part of the morph completed, the images are color blended to complete the last step of the morph.

This technique was found to render a smooth, real-time transition between the source and destination image. It is a simple idea (especially because OpenGL can automatically handle most of the warping by means of texture mapping), yet it compares exceptionally well with the other techniques that were discussed.

A problem that can however be experienced with this technique is foldovers. This happens when lines connecting points cross over each other, as displayed in Figure 7.1.

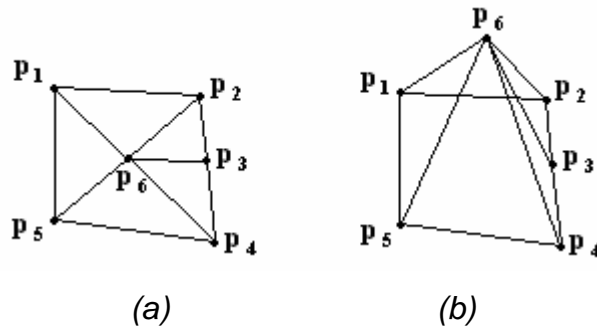
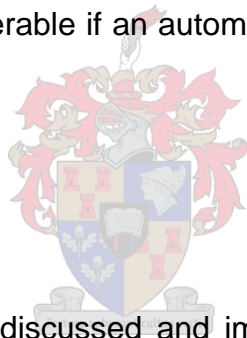


Figure 7.1 Foldover problem

In Figure 7.1, (a) is the source triangulation and (b) is the destination triangulation, where the foldover problem occurred.

When this happens the triangle in which the overlap occurred will not morph correctly. All the other triangles will however morph correctly. Currently it is up to the user to select the control points in such a way that no triangle edges will overlap to overcome this problem; however it would be preferable if an automated solution could be implemented to overcome this.



## 7.2 Future Work

Most of the morphing techniques discussed and implemented require a great deal of input from the user. The user is required to define the corresponding control points. The result of this is that a large deal of the quality is dependent on the user. If the user does a good job, then the resulting morph will be pleasing, however if the user does a bad job then no matter how optimal the algorithm/implementation, the resulting morph will not be pleasing. A method to optimally automate this process still needs to be discovered. Perhaps a technique combining some sort of automated edge-detection with a minimum or no user input.

Another part of morphing that should receive more attention is the transition control of the blending function. In most of the techniques discussed a simple uniform cross-dissolve is used, however as mentioned earlier the use of non-uniform transitions may result in more spectacular visual effects.

Image morphing is usually done over two images. Morphing over multiple images could also be explored in more detail, as discussed in [25]. If morphing can be done over multiple images, then why limit it to only two?



# Bibliography

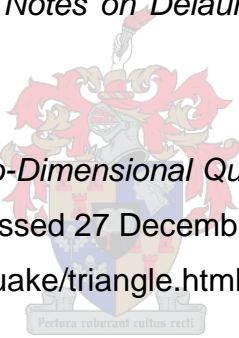
- [1] Alexa M, Müller W (1999) The morphing space. *Seventh International Conference in Central Europe on Computer Graphics and Visualization (Winter School on Computer Graphics)*, pages 329–336, ISBN 80-7082-490-5. Held in University of West Bohemia, Plzen, Czech Republic
- [2] Beier T, Neely S (1992) Feature-Based Image Metamorphosis. *Computer Graphics (Proceedings of SIGGRAPH '92)*, **26**(2), pp. 35-42
- [3] Bern M, Eppstein D (1992) Mesh Generation and Optimal Triangulation. *Computing in Euclidean Geometry* (Ding-Zu Du and Frank Hwang, editors), Lecture Notes Series on Computing, **1**, pp. 23-90, World Scientific, Singapore
- [4] Bordwell D, Thompson K (1997) “The Power of Mise-en-Scene.” *In Film Art, An Introduction*. McGraw-Hill, New York
- [5] Bowyer A (1981) Computing Dirichlet Tessellations. *Computer Journal*, **24**(2), pp. 162-166
- [6] Chen S.E, Williams L (1993) View interpolation for image synthesis. *Computer Graphics (Proceedings of SIGGRAPH '93)*, pp. 279–288
- [7] Chen B, Dachille F, Kaufman A (1999) *Forward Image Warping*. State University of New York, Stony Brook
- [8] Cignoni P, Montani C, Scopigno R (1998) De Wall: A fast Divide and Conquer Delaunay Triangulation Algorithm in  $E^d$ . *Computer-Aided Design* **30**(5), pp. 333-341
- [9] Fant K.M (1986) A Nonaliasing, Real-time Spatial Transformation Technique. *IEEE Computer Graphics and Applications*, **6**(1), pp. 71-80

- [10] Faugeras O.D (1993) Three Dimensional Computer Vision, A Geometric Viewpoint. MIT Press, Cambridge, MA
- [11] Fleischmann P (1999) Dissertation: Mesh Generation for Technology CAD in Three Dimensions, Technischen Universität. Wien, Austria
- [12] Fortune S (1987) A Sweepline Algorithm for Voronoi Diagrams, *Algorithmica*, **2**(2), pp. 153-174
- [13] Fortune S (1992) Voronoi Diagrams and Delaunay Triangulations. *Computing in Euclidean Geometry* (Ding-Zu Du and Frank Hwang, editors), Lecture Notes Series on Computing, **1**, pp. 193-233, World Scientific, Singapore
- [14] Froumentin M, Labrosse F, Willis P (2000) A Vector-based Representation of Image Warping. *Eurographics 2000*, **19**(3)
- [15] Gao P, Sederberg T (1998) A work minimization approach to image morphing. *Visual Computer*, **14**, pp. 390-400
- [16] Giacon P, Siggan S, Tommasi G, Busti M (2005) Implementing DSP Algorithms Using Spartan-3 FPGAs. *Xcell Journal*, Second Quarter 2005
- [17] Gomes J, Costa B, Darsa L, Velho L (1996) Graphical Objects. *The Visual Computer*, **12**, pp. 269-282
- [18] Gomes J, Costa B, Darsa L, Velho L (1998) A System's Architecture for warping and morphing of graphical objects. *Proceedings of SIBGRAP'98*, SBC
- [19] Gomes J, Darsa L, Costa B, Velho L (1999) *Warping and Morphing of Graphical Objects*, Morgan Kaufmann Publishers, Inc, San Fransisco
- [20] Guibas L, Stolfi J (1985) Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics*, **4**(2), pp. 74-123

- [21] Heckbert, P.S. (1989) *Fundamentals of Texture Mapping and Image Warping*. Master's Thesis. (Technical Report No. UCB/CSD 89/516). University of California, Berkeley
- [22] Humphreys G (2004) *Image Warping, Compositing and Morphing*. Course Slides for CS445:Intro Graphics. University of Virginia, Fall 2004  
URL: [www.cs.virginia.edu/~gfx/courses/2004/Intro.Fall.04/handouts/03-morph.pdf](http://www.cs.virginia.edu/~gfx/courses/2004/Intro.Fall.04/handouts/03-morph.pdf)
- [23] Kass M, Witkin A, Terzopoulos D (1988) Snakes: Active Contour Models. *International Journal of Computer Vision*, pp. 321–331
- [24] Lawson C.L (1977) Software for  $C^1$  Surface Interpolation. *Mathematical Software III* (John Rice editor), pp. 161-194, Academic Press, New York
- [25] Lee S-Y, Wolberg G, Shin S.Y (1998) Polymorph: Morphing Among Multiple Images. *IEEE Computer Graphics and Applications*, **18**(1), pp. 58-71
- [26] Lee S-Y, Chwa K-Y, Shin S.Y, Wolberg G (1995) Image Metamorphosis Using Snakes and Free-Form Deformations. *Computer Graphics (Proceedings of SIGGRAPH '95)*, pp. 439-448
- [27] Lee S-Y, Chwa K-Y, Shin S.Y, Hahn J (1996) Image Metamorphosis Using Snakes Deformation techniques. *The Journal of Visualization and Computer Animation*, **7**(1), pp. 3-23
- [28] Lee T, Lin Y-C, Lin L, Sun Y (1998) Fast Feature-Based Metamorphosis and Operator Design. *Proceedings of EuroGraphics '98*, Computer Graphics Forum, **17**(3), pp. 15 - 22
- [29] Lee W.S, Magnenat-Thalmann N (1998) Head Modeling from Pictures and Morphing in 3D with Image Metamorphosis based on triangulation. *Modelling and Motion Capture Techniques for Virtual Environments, Proc. Captech98, Springer LNAI LNCS Press, Geneva, pp.254-267.*

- [30] Lee W.S, Magnenat-Thalmann N (2001) Virtual Body Morphing. *Proc. Computer Animation, Seoul, Korea*.
- [31] Leros A, Garfinkle C.D, Levoy M (1995) Feature-Based Volume Metamorphosis. *Computer Graphics (Proceedings of SIGGRAPH '95)*, **29**, pp. 449-456
- [32] Liu Z, Liu C, Shum H-Y, Yu Y Pattern Based Texture Metamorphosis. *In Pacific Graphics*, pp. 184-191
- [33] Maxwell E.A (1946) *The Methods of Plane Projective Geometry, based on the Use of General Homogeneous Coordinates*, Cambridge University Press, London
- [34] Mount D.M (2005) *Lecture notes for CMSC754: Computational Geometry*. Department of Computer Science, University of Maryland, Fall 2005
- [35] Nishita T, Fujii T, Nakamae E (1993) Metamorphosis using B´ezier clipping. *In Proceedings of the First Pacific Conference on Computer Graphics and Applications*, pp. 162–173, Seoul, Korea, World Scientific Publishing Co.
- [36] Ruppert J (1995) A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, **18**(3), pp 548-585
- [37] Ruprecht D, Müller H (1994) Deformed Cross-Dissolves for Image interpolation in Scientific Visualization. *The Journal of Visualization and Computer Animation*, **5**(3), pp. 167-181
- [38] Ruprecht D, Müller H (1995) Image Warping with Scattered Data Interpolation. *IEEE Computer Graphics & Applications*, March, pp. 37-43
- [39] Sederberg T.W, Greenwood E (1992) A physically Based Approach to 2D Shape Blending. *Computer Graphics (Proceedings of SIGGRAPH '92)*, pp. 35-42



- [40] Sederberg T.W, Gao O, Wang G, Mu H. (1993) 2D Shape Blending: An Intrinsic solution to the vertex path problem. *Computer Graphics (Proceedings of SIGGRAPH '93)*, pp 15-18
- [41] Seitz S, Dyer C (1996) View Morphing. *Computer Graphics (Proceedings of SIGGRAPH '96)*, pp. 21-30
- [42] Shapira M, Rappoport A (1995) Shape Blending using the star skeleton representation. *IEEE Computer Graphics and Applications*, **15**(2)
- [43] Shewchuk J.R (1996) Triangle: Engineering a 2D quality mesh generator and Delaunay Triangulator. *First Workshop on Applied Computational Geometry*, pp. 124-133, Carnegie Mellon University, Pittsburgh, Pennsylvania
- [44] Shewchuck J.R (1999) *Lecture Notes on Delaunay Mesh Generation*. University of California, Berkeley
- [45] Shewchuck J.R, *Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator*. [Online]. Last Accessed 27 December 2006  
URL: <http://www.cs.cmu.edu/~quake/triangle.html>
- 
- [46] Smythe D.B (1990) A Two-pass Warping Algorithm for Object Transformation and Image Interpolation. ILM Technical Memo #1030, Computer Graphics Department, Lucasfilm Ltd.
- [47] Strang G (1980) *Linear Algebra and its Applications*. 2<sup>nd</sup> Edition, Academic Press, New York
- [48] Su P, Drysdale R.L (1996) A Comparison of Sequential Delaunay Triangulation Algorithms
- [49] Tal A, Elber G (1999) Image Morphing with Feature Preserving Texture. *Computer Graphics Forum*, 18(3), pp. 339-348, Blackwell Publishers, Oxford

- [50] Watson D.F (1981) Computing the n-dimensional Delaunay Tesselation with Application to Voronoi Polytopes. *Computer Journal* **24**(2), pp. 167-172
- [51] Welsh, S. C. (2003). *Pseudo-3D animations using 2D image morphing techniques*. Unpublished PhD thesis, Monash University, Clayton, Victoria, Australia
- [52] *Wikipedia, The Free Encyclopedia – Morphing*. [Online] Last Accessed 27 December 2006  
URL: <http://en.wikipedia.org/wiki/Morphing>
- [53] *Wiktionary, A Wiki-Based Open Content Dictionary*. [Online]. Last Accessed 27 Desember 2006  
URL: <http://en.wiktionary.org/wiki/metamorphosis>
- [54] Wolberg G (1998) Image Morphing: A Survey. *The Visual Computer*, **14**(8/9), pp. 360-372
- [55] Wolberg G (1990) *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California
- [56] Wolberg G (1996) Recent Advances in Image Morphing. *Computer Graphics (Proceedings of SIGGRAPH '96)*, pp 64-71
- [57] Zanella V, Fuentes O (2004) An Approach to Automatic Morphing of Face Images in Frontal View. *Proceedings of 2004 Mexican International Conference on Artificial Intelligence (MICAI)*, Mexico City, Mexico, Lecture Notes in Artificial Intelligence 2972
- [58] Coxeter H.S.M (1978) *Non-Euclidean Geometry*. University of Toronto Press

# Appendix A

## User Guide for the Implementation Discussed in Section 6.3

The name of the program file is “Morph.exe”. This file together with “triangle.exe” and “glut32.dll” is needed to run the application. These files will be provided on request.

### **STEP 1: Before Running the Application**

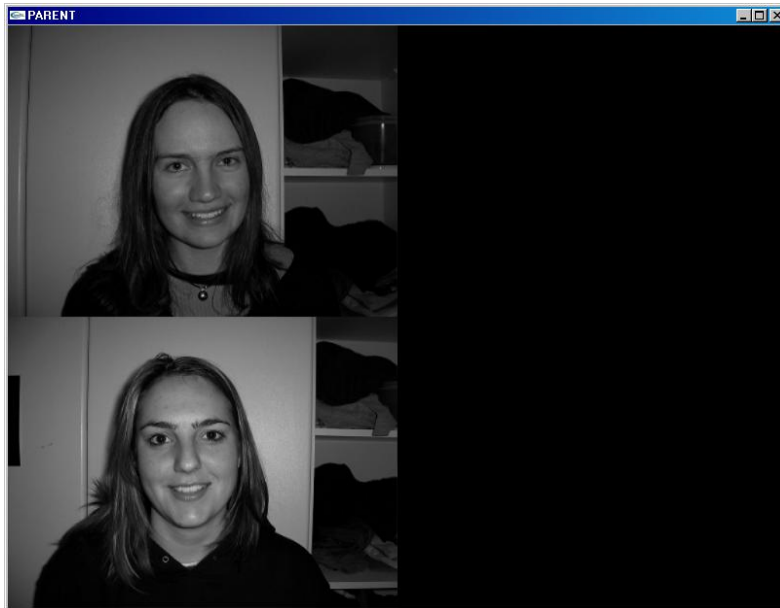
The application only runs on Windows. Once the application is running do not open any other applications on top of it as this will result in the application not working properly.

Make sure that a folder “c:\Morph” exists on the hard drive. If it does not exist then create it. This is the folder where the application will search for the triangulation program and input images. This folder must therefore contain the following three items:

1. The triangulation program, provided with the source code, named “triangle.exe”.
2. The source image. This can be any bitmap (\*.bmp) image (color or grey) named “source.bmp”.
3. The destination image. This can be any bitmap (\*.bmp) image (color or grey) named “dest.bmp”.

Once the above folder is created run the “Morph.exe” file. This file can be located anywhere, as long as the file “glut32.dll” is in the same location.

A window similar to the one in Figure A.1 should be displayed.



*Figure A.1 The start window*

The top-left sub-window will contain the image named “`source.bmp`”.

The bottom-left sub-window will contain the image called “`dest.bmp`”.

## **STEP 2: Selecting the Corresponding Control points**

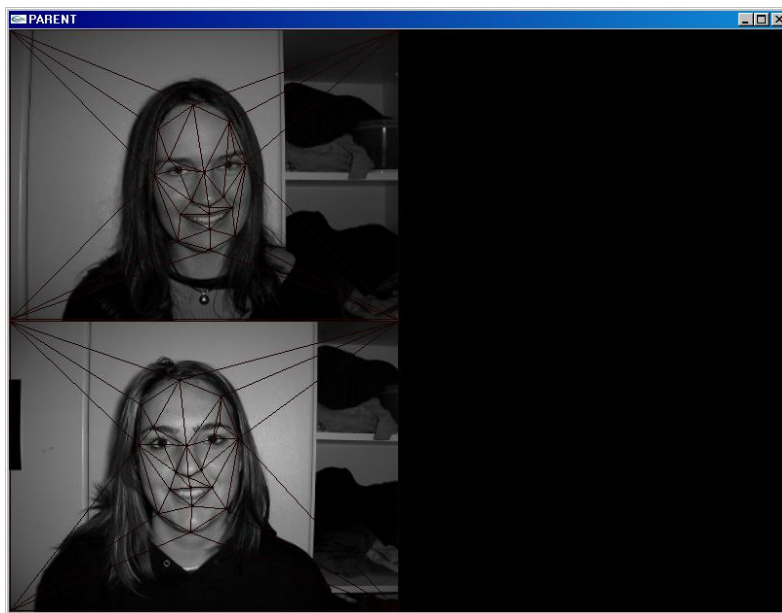
Once the previous step is completed, the control points must be selected in each image. This can be done by moving the mouse to the desired location and left-clicking. A small dot will appear to display the point. A total of up to 25 points are allowed. The four corner points are automatically included (which makes the maximum number of points 29)

The order in which the points are selected is very important. It must be the same for both images. That means that for example if the order for the source image is: left eye center, right eye center, middle of nose, left mouth corner, right mouth corner, etc. then the order for the destination image **MUST** also be: left eye center, right eye center, middle of nose, left mouth corner, right mouth corner, etc. If this requirement is not met, the morph will turn out to be catastrophic. It is therefore suggested to select a point in the source image and then immediately select the corresponding point in the destination image and then following this method until all the points are selected to avoid making a mistake.

### **STEP 3: Triangulation**

The next step is the Delaunay triangulation. When pressing any of the keys on the keyboard, make sure the mouse pointer is anywhere on the open application. Press “t” on the keyboard. This will prompt the application to call the “triangle.exe” file, which is responsible for the triangulation. The file “triangle.exe” was created by Jonathan Richard Shewchuck and is available on the web [45].

Figure A.2 is a demonstration of what the window should look like at this point.

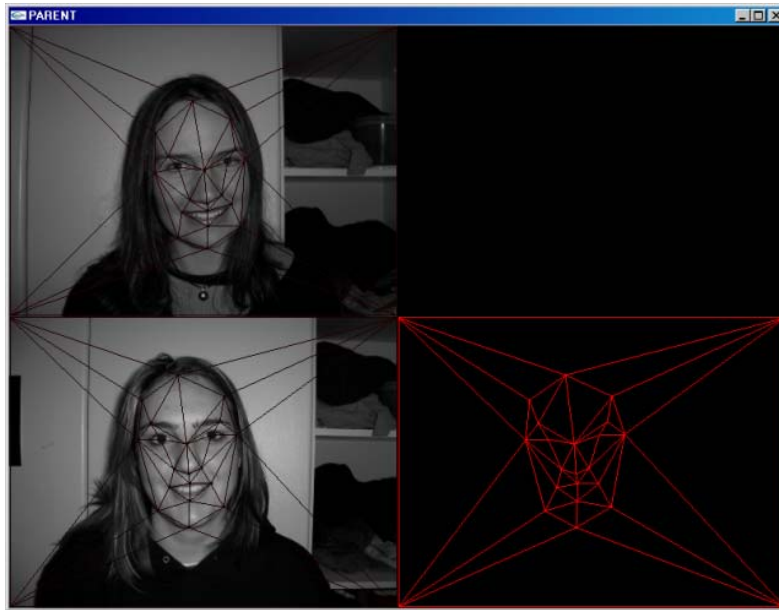


*Figure A.2 Triangulation completed*

### **STEP 4: Interpolation of the Triangles**

It is possible to see how the triangles are interpolated from the source triangulation to the destination triangulation. Pressing “q” on the keyboard will display the interpolation, from the source triangles to the destination triangles, in the bottom-right sub-window. Pressing “w” on the keyboard will display the interpolation, from the destination triangles to the source triangles.

Figure A.3 displays the application window, with the visible interpolating triangles.

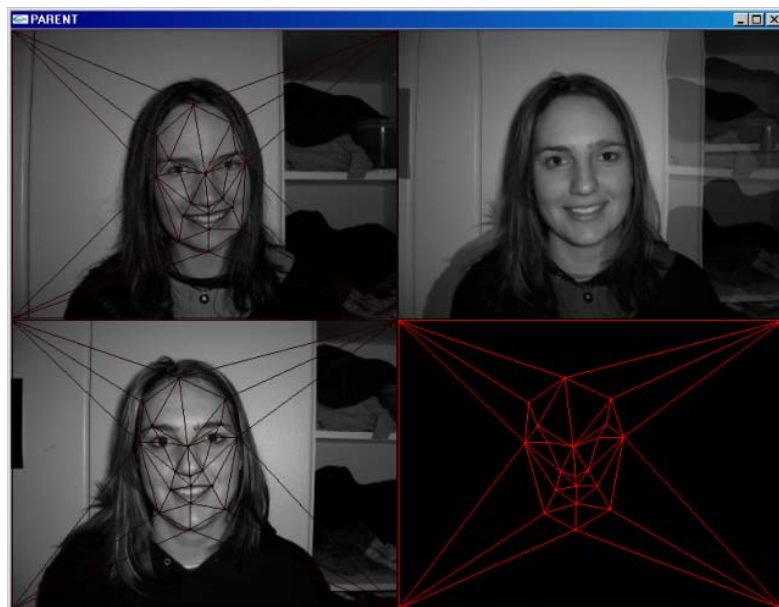


*Figure A.3 Interpolation of the triangles*

### **STEP 5: Morphing**

To view the entire morphing sequence (displayed in the top-right sub-window) press either “z” or “x” on the keyboard. “z” morphs from the source image to the destination image and “x” morphs from the destination image to the source image.

Figure A.4 shows what the final window would look like.



*Figure A.4 The morphing sequence*

To end the application, simply click the cross in the top-right corner of the application window.



# APPENDIX B

## An Image Morphing Sequence Created by FantaMorph3

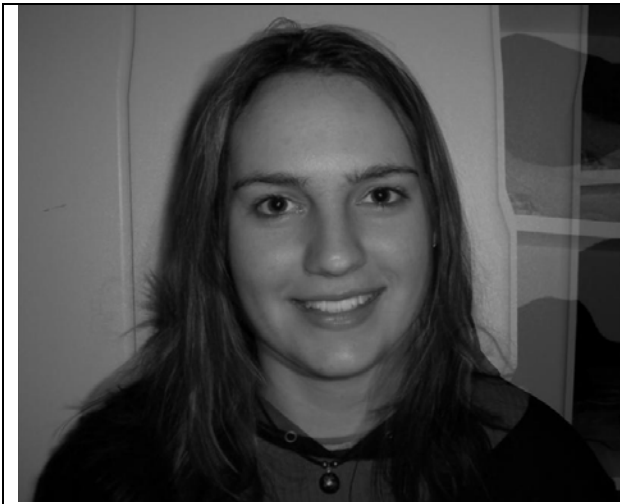
The images are displayed left to right, top to bottom.

**SOURCE IMAGE**











**DESTINATION IMAGE**

*Figure B.1 Example of an Image sequence of a morph created by FantaMorph3*