



# **Synthetic Images of Faces using a Generic Head Model**

A thesis presented for  
the degree of Doctor of Philosophy  
at the University of Aberdeen

Andrew C. Aitchison  
BSc, University of Bristol

Submitted December 1991

Accepted June 1992

# Abstract

Realistic images of faces of individual people have been generated by computer by texture mapping a photograph of the individual onto a wireframe model of their head. In previous work each individual had to be measured in order to build this model.

We use a *generic* head model to describe the shape of a human head. This model gives us “world knowledge” of the shape of a head, to create the new images we distorted it to fit the face in the photograph, creating a new, *specific*, head model. This specific model can be distorted further to change the expression before it is projected onto a viewing plane. We then map the texture from the photograph of the person, one triangle of the model at a time, onto the viewing plane to create a novel view of the individual. Texture from more than one image may be used, by selecting the most appropriate image for each triangle of the model.

The generic  $\rightarrow$  specific transformation is implemented by making each edge of the model into a spring, displacing some of the vertices to fit the photograph and then allowing the spring-frame model to relax to a position of least energy.

**key phrases** computer graphics, image warping, texture mapping, multiple texture sources, synthetic images, wireframe model, generic model, specific model, spring-frame model, human heads, human faces.

# Declaration

I declare that this thesis

- has been composed by me
- has not been accepted in any previous application for a degree

and that

- the work is my own
- all quotations have been distinguished and the sources of information acknowledged.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Why create synthetic images of faces ? . . . . .	1
1.2	Previous work: Animation . . . . .	3
1.3	Previous work: Simulation . . . . .	7
1.4	The Work of this Thesis . . . . .	9
<b>2</b>	<b>Distorting Pictures in 2 Dimensions</b>	<b>10</b>
2.1	Distorting Images . . . . .	12
2.2	The Delaunay Triangulation of a Set of Points . . . . .	13
2.3	Modelling the Distortion . . . . .	21
2.4	Generating the Distorted Picture . . . . .	23
2.5	Implementation of the Distortion Algorithm . . . . .	25
2.6	Applications . . . . .	30
<b>3</b>	<b>Generating Novel 3-Dimensional Views</b>	<b>33</b>
3.1	Representing the Generic Model . . . . .	36
3.2	Data for the Generic Head . . . . .	37
<b>4</b>	<b>Building Generic Head Models</b>	<b>40</b>
4.1	Approximating a Triangulation . . . . .	43

4.2	Faugeras' Point Insertion Algorithm . . . . .	45
4.2.1	The Algorithm Described by Faugeras . . . . .	46
4.2.2	Faugeras' Pseudo-code for the Complete Algorithm . . . . .	49
4.2.3	Criticisms of Faugeras's Method . . . . .	49
4.3	A Point Removal Algorithm . . . . .	51
4.3.1	Removing a Vertex is a Local Operation . . . . .	52
4.3.2	Remembering Removed Vertices . . . . .	53
4.3.3	Outline of the algorithm . . . . .	54
4.3.4	Removing a single vertex from the neighbourhood triangulation . . . . .	55
4.3.5	Calculating the thickness of a triangle . . . . .	57
4.3.6	In what order do we remove the vertices ? . . . . .	60
<b>5</b>	<b>The Specific Head Model</b>	<b>64</b>
5.1	Globally Matching the Generic Model to the Photograph . . . . .	66
5.2	Finding the Depth Components of the Model Vertices . . . . .	69
5.3	Locally Improving the Match . . . . .	72
5.3.1	Finding the Minor Vertices: Delaunay Triangulation Method	73
5.3.2	Finding the Minor Vertices Using a Spring-Frame Triangulation . . . . .	76
5.4	MBASIC . . . . .	80
5.5	Conclusion . . . . .	80
<b>6</b>	<b>Displaying the Head</b>	<b>81</b>
6.1	Expression . . . . .	81
6.2	Display . . . . .	82
6.3	Results . . . . .	87

<b>7</b>	<b>Using Texture From Several Images</b>	<b>90</b>
7.1	Experiments Using More Than One Picture of an Object . . . . .	91
7.2	The Coffee Mug Model . . . . .	91
7.3	Taking the Pictures . . . . .	92
7.4	Combining Several Sources of Texture . . . . .	93
7.4.1	Matching Points . . . . .	94
7.5	Illumination . . . . .	94
7.5.1	Specular Reflection and Highlights . . . . .	95
7.6	Conclusions . . . . .	96
<b>8</b>	<b>Some Applications</b>	<b>99</b>
8.1	Average Faces and Face Merges . . . . .	100
8.2	Standardized Faces for Principal Component Analysis . . . . .	105
8.2.1	Principal component analysis of face images . . . . .	105
8.2.2	Improving PCA . . . . .	107
<b>9</b>	<b>Summary</b>	<b>109</b>
<b>A</b>	<b>Fourier-domain Depth from Shading</b>	<b>115</b>
A.1	Lighting Models . . . . .	115
A.1.1	The Lambertian reflectance function . . . . .	116
A.1.2	The Lunar reflectance function . . . . .	116
A.2	A Peculiar Interaction . . . . .	117
A.3	Pentland's algorithm . . . . .	117
A.4	Application of the algorithm . . . . .	119
A.5	Conclusions . . . . .	121
<b>B</b>	<b>Candide: Rydfalk's Wireframe Model</b>	<b>122</b>

<b>C Source Code</b>	<b>124</b>
C.1 Generic $\rightarrow$ Specific Model Transformation . . . . .	124
C.1.1 vectors.h . . . . .	124
C.1.2 3d_stuff.h . . . . .	125
C.1.3 delaunay.h . . . . .	126
C.1.4 specific.h . . . . .	127
C.1.5 specific_delaunay.c . . . . .	128
C.1.6 specific.c . . . . .	132
C.2 Drawing the Model . . . . .	137
C.2.1 render.h . . . . .	137
C.2.2 render3d.c . . . . .	138
C.2.3 draw.h . . . . .	141
C.2.4 draw.c . . . . .	142
C.3 Triangulation Reduction: Faugeras' Method . . . . .	147
C.3.1 vector_arithmetic.p . . . . .	147
C.3.2 faugeras_dist.p . . . . .	148
C.3.3 shortestPath.p . . . . .	149
C.3.4 faugeras.p . . . . .	150
C.4 Triangulation Reduction: Point Removing Method . . . . .	154
C.4.1 shuffle.p . . . . .	154
C.4.2 ian_dist.p . . . . .	155
C.4.3 calculateThickness.p . . . . .	157
C.4.4 triangulate_polygon.p . . . . .	158
C.4.5 approx_ian.p . . . . .	159
C.4.6 main program . . . . .	161

<b>D Glossary</b>	<b>162</b>
-------------------	------------

<b>Bibliography</b>	<b>165</b>
---------------------	------------

# List of Figures

2.1	3 Mac-a-Mug faces . . . . .	11
2.2	Three faces created by our method. . . . .	11
2.3	A Voronoi tessellation and its dual . . . . .	13
2.4	A walk to the triangle containing a point . . . . .	16
2.5	Equivalence of the max-min angle criterion and the circumcircle test. . . . .	17
2.6	The circumcircle test . . . . .	19
2.7	The new vertex placed in the insertion polygon. . . . .	20
2.8	Scanning a triangle: top half. . . . .	26
2.9	Scanning a triangle: bottom half. . . . .	26
2.10	Comparison of two texture-mapping algorithms . . . . .	29
2.11	A set of control points which are mis-triangulated by the new algorithm . . . . .	32
3.1	Candide texture-mapped . . . . .	35
4.1	The original head data . . . . .	41
4.2	A naïve approximation of the original head . . . . .	42
4.3	Approximating a triangulation or just the points ? . . . . .	50
4.4	Faugeras' icosahedron . . . . .	51
4.5	Neighbourhood Triangulations . . . . .	55

4.6	Examples of the Reduced Neighbourhood Triangulation when the removed vertex lies on the boundary . . . . .	56
4.7	Swapping the Diagonals . . . . .	56
4.8	Vicki approximated with $\epsilon = 2\text{cm}$ . . . . .	58
4.9	Vicki approximated by the sophisticated thickness calculation . . . . .	58
4.10	Sophisticated triangle thickness calculation . . . . .	59
4.11	Sophisticated, random approximation . . . . .	61
4.12	2 passes of the sophisticated, random approximation . . . . .	62
4.13	3 passes of the sophisticated, random approximation . . . . .	62
5.1	A face, showing the Delaunay triangulation of the major points . . . . .	65
5.2	The generic head globally transformed to match the photograph . . . . .	70
5.3	The Delaunay triangulation of the major points . . . . .	73
5.4	The points of the generic model globally transformed to fit the major points . . . . .	74
5.5	Equilibrium of the spring model . . . . .	79
6.1	The author rotated 4 times . . . . .	88
6.2	Another face rotated . . . . .	89
7.1	The coffee-mug and the generic model . . . . .	92
7.2	Specular Reflection . . . . .	95
7.3	Reconstructed views of the coffee-mug . . . . .	98
8.1	A composite of a mother and her daughter. . . . .	99
8.2	The average of 2 faces. . . . .	100
8.3	The average of 30 faces . . . . .	100
8.4	Original images and models . . . . .	101

8.5	Is this a real person? . . . . .	103
8.6	Changing faces . . . . .	104
8.7	Eigenfaces . . . . .	108
B.1	Three views of Candide . . . . .	122

# Acknowledgements

My first and biggest thanks must go to my supervisor Ian Crow. A PhD. supervisor is expected to be many things: teacher, advisor, critic and colleague. Some even take their students skiing and sailing (we'll do it yet, Ian !). Ian, thanks for believing in me, especially when I didn't.

To David Tock for sparing time to teach me the Ways of Unix. To Alan Bennett who understood my work well enough to make useful suggestions, if not usually well enough to make them before me. To Roly Lishman of the Computing Science Department for seeing things from a different angle.

To British Telecom for giving me experience of the world outside academia, and the means of acquiring a sports car. To Bill Welsh and Mark Shackleton for looking after me on my visits to Martlesham.

To Vicki Bruce and Mike Burton of the Nottingham Psychology department, and Ann Coombes of University College Hospital, London, for the 3D models. A special thanks to Vicki for having her head measured.

To Tony Glendinning for testing my software, and creating the Nottingham images (figure 2.2) with it.

To Peter Cameron for his Principal Component Analysis images in figure 8.7.

To Phil Benson for sharing his code for the centroid-edge scan with me (even though I do nothing but criticize it in public), and for his comments on chapter 2.

To all the people whose faces I have photographed in the name of science.

To Mum and Dad, especially for letting me stay at home and keeping me away

from the distractions of cooking and washing while I was “writing up”.

To Wallace Anderson for introducing me to Ian Craw and for fixing my car. To Kate Anderson for acting as mother when home was too far away. It is nice to have an extra set of parents—even better when they are ‘on site’.

To Mary-Claire van Leunen for *A Handbook for Scholars*, with all its advice on writing an accurate and interesting text. I hope she’ll forgive me for writing *one* section which is of very little interest to the readers of this volume.

A PhD is a severe strain on one’s sanity. Michael Mayer thinks I preserved his, and in turn I’d like to thank him for sterling efforts to save mine. After Michael left I turned to Scottish Country Dancing, and I’d like to thank my many partners: Jessie, Helen, Isla, Sarah, Virginia, Jackie, Pippa, Alison, Michelle, Lynn, Marianne, Carola, Elaine, Morag, Anna, Ruth, Kirsteen, Janet, Gillian, Sonia, Lisa, Heather, Fiona and of course Elliot. I’ve danced with lots of other ladies too, none of whom have trod on my toes. Yet.

Susan Ward for being with me on that trip to Loch Ness and the Black Isle, and other great days in my life.

# Chapter 1

## Introduction

We describe work which allows a computer to create new and realistic views of a person from a single photograph. Additional photographs can be accommodated if available. Each synthetic image shows the person from a new viewpoint, and possibly with a different expression. Previous work which generates images of individuals by computer uses a 3-dimensional model of the individual. We use the same *generic* head model for everyone, and distort it so that it is the same shape as the individual in each photograph.

### 1.1 Why create synthetic images of faces ?

It is easy to take a picture with a camera, so why use a computer ? If the individual is available it would indeed be easy to take another photograph from the desired viewpoint. Often however the individual cannot be rephotographed.

Psychologists often wish to show subjects a collection of faces which for experimental reasons must all face in the same direction. Thus a three-quarter view of, say, the Prime Minister cannot be used in an experiment using full-face views. Partly for copyright reasons, good quality photographs of well-known people can be difficult to obtain, so our programme could allow psychologists a wider selection of faces with which to experiment.

Police forces around the country have mug-shot libraries. Most are full-face and profile, but in order to reduce the quantity of information, the Metropolitan Police use just one 45-degree view of each suspect. A system such as ours could be used to convert between the two systems. With the extension to multiple views we could use both views when converting a provincial photo-pair into a Metropolitan view.

**Facial Simulation For Video Telephones** There is much interest in video-telephones—telephones which allow the users to see each other while they talk—especially for teleconferencing. Prototypes have been marketed, but these require high bandwidth links—cables or other connections capable of carrying a lot of information very quickly. These links are expensive and have to be installed at each site, and it would be much simpler if a video telephone could just plug into an ordinary telephone socket. Unfortunately there isn't enough bandwidth, or information handling capacity, in an ordinary telephone line to carry two full television pictures.

A digitised television picture is made up of small rectangular regions known as pixels. A single frame may have three or four hundred thousand pixels and to give the illusion of smooth movement there must be at least a dozen frames per second, preferably twice as many. This means that, without compression, we need to send several million pixels per second. Existing phone lines can only carry a maximum of 14400 pixels per second, and this limit can drop by a third or more on a poor line. Thus the video information must be compressed massively to fit onto the telephone line. Unfortunately conventional data-compression techniques just aren't good enough.

In the early 1980s, Parke, amongst others, recognised that the data could be compressed massively if a computer at one end could examine each frame picture and extract various parameter values describing how the face has changed since the previous frame. If the video telephone at the other end of the line has access to a single image of the person it should be possible to reconstruct the image from

these parameter values.

Over the last twenty years, there have been many attempts to generate pictures of faces by computer. A single image is all very well, but real heads are hardly ever still, so there was a natural desire to put several images together to create a moving sequence. Much of the early work was concerned with animation, which as Frederic I. Parke stated [31], is not the same as simulation:

The goal of animation is to communicate an idea, story, mood, etc; the goal of simulation is to exactly model a process, theory or system. Animation allows and often requires artistic licence, i.e. deviation from exact modeling.

Because of this licence animation is easier than the simulation of real faces.

## 1.2 Previous work: Animation

**Parke (1972)** Parke [29] appears to have been the first to attempt computerised facial animation. He used a polygonal model with about 250 vertices and 400 polygons. The model was built by painting polygons onto half of his assistant's face. When he was satisfied that the polygons correctly modeled the face in a number of different expressions he took orthogonal pairs of photographs, one of the face from directly in front and the other from the side. He found the 3 dimensional positions of the corners of each polygon from their positions in the two views. This calculation is explained in a later paper [30].

To distinguish lips from skin from hair each polygon of the model was given a colour, and the model was displayed on a raster display. For a realistic effect the model was smooth shaded using Gouraud shading (see the glossary, appendix D for a brief description).

He calculated the 3D positions of the polygon corners from several pairs of photographs of his assistant, each pair showing a different expression. By using a cosine interpolation scheme he calculated the position of the face showing

intermediate expressions which he put together to create animated sequences of faces.

Although this seems to have been the earliest attempt at computerized facial animation, Parke had a lot of advanced equipment available, and the final results were considerably better than several later attempts. The images still seem impressive today.

**Facial Action Coding System for Expressions** Paul Ekman and Wallace Friesen's Facial Action Coding System (FACS) [17] is the standard system for representing visible changes in facial expression. Every facial expression can be broken down into distinct *action units* (AUs). There are about fifty of these AUs: each represents a minimal change in facial posture, and is caused by the contraction or relaxation of a single muscle or muscle group. They may be activated wholly or only partially. The expression of a face can thus be represented as a list of Action Units and their activation strengths.

Although designed as a way in which psychologists studying non-verbal communication could encode facial expressions, FACS lends itself very well to computer graphics, and most computer face models in which the expression can be varied have attempted to incorporate it.

Since expressions are rarely frozen, it is often the changes in the AUs which are of interest, and FACS is usually applied to dynamic face models.

**Rydfalk: Candide** Mikael Rydfalk [37] gives details of a parameterised face, known as 'Candide', designed for real time animation with modest hardware support. For Rydfalk in 1987 this meant that the model must be made from not more than 100 triangular patches - a major restriction. Appendix B describes the model in detail.

Rydfalk found that for an active 3-dimensional model, realism meant two things: static realism, in which the stationary model should look natural from

any angle; and dynamic realism, which meant that the movements and changes in expression should appear natural.

To keep the number of triangles down, *Candide* is just a mask, with a face and a forehead, but no ears or neck and no back or sides of the head. However, static realism demanded that the forehead was modelled accurately, using around 25 triangles. This limited the dynamic realism because the lips and cheeks were too rough. In addition *Candide* was unable to wrinkle her skin. Rydfalk used the Action Units of FACS to give expression to *Candide*, but because of these limitations she could not exhibit every AU.

The real-time graphic device used for display is named (RETINA), but no description is given and no output is shown. Full details of *Candide* were given in the report and *Candide* has been adapted by other researchers for their own purposes.

**Platt and Badler** Stephen Platt and Norman Badler [36] modelled a face as a collection of skin patches attached by muscle fibres to an underlying bone structure. Like Rydfalk's *Candide*, their model is a mask with just a face and a forehead. It has over 400 skin patches - mostly rectangles - so is considerably more detailed than *Candide*.

Each edge of the model has a natural length and an elasticity. When a muscle contracts it pulls the skin attached to it, setting up tensions which deform the model. Since FACS notation is closely related to the muscles, they were able to convert each AU into the contraction strength of each muscle. This enabled them to display different expressions quite successfully. They noted several deficiencies of the model however. Since the bone was modelled as distinct fixed points rather than a solid mass, muscles had a tendency to flow *through* rather than *around* the bone. This was a particular problem at the eyebrows. They made no attempt to model the jaw action, but believe that this could be included quite easily by allowing the muscles to rotate the jaw around an appropriate axis.

They used a vector-oriented graphic display device which meant that their display was limited to wire frame line drawings of the facial mask. A single wire frame image is not very realistic, but animating the mask by showing a sequence of slightly different masks one after another produced a more convincing effect.

**Parke (1982)** In his second attempt at facial animation [31] Parke again used a polygonal model made from shaded polygons. There were over 400 polygons in the main model, plus submodels for the teeth and for each eyeball. The Gouraud shading used to display his previous model was replaced with Phong shading. Although more expensive to calculate, this allowed the model to show highlights.

Parke built many parameters into this model, allowing him considerable control over the head. These parameters fell into two types: conformation parameters and expression parameters. Conformation parameters control attributes which vary between individuals but which are the same for different views of the same person. Parke included conformation parameters to control skin colour, height to width ratio of the face, colour of features (such as eyebrows, iris and lips), a parameter to represent facial growth and many parameters controlling the relative sizes of different parts of the face. Expression parameters control attributes which vary between views of a person, often changing continuously. Apart from parameters such as eyelid opening, eyebrow position and shape, pupil dilation, jaw rotation, and position of the lips and the corner of the mouth, all of which alter the expression, Parke counted the orientation of the head in the image as an expression parameter.

A well designed parameterization makes it easy to change the model in sensible, realistic ways. In order to change a model without a parameterization we would need to find new positions for many points manually. If we liked the change but wished to exaggerate it further we would have to move all these points again. Parameterizing a model requires some initial effort, but thereafter many sensible alterations can be made simply by changing the value of a single parameter.

Parke was the first to point out that a parameterised face would in principle allow face images to be transmitted very efficiently, as we showed on page 2. He also noted that observers became more critical of the displayed model as it became more realistic.

**Waters** Keith Waters [47, 48, 49] has developed Platt and Badler’s idea of a muscle-based model, and displayed it on a raster display. In addition to the linear muscles which Platt and Badler used, this model has sphincter muscles. These are rings of muscles running around the eyes and mouth which are not attached to the bone.

In its latest manifestation the model has four layers: the epidermis, the dermis, a subcutaneous layer and the bones, and the muscles are modelled by non-Hookian springs. As muscles contract, appropriate folds and furrows appear automatically in the skin, such is the accuracy with which stretching is modelled.

The images generated by his models show clear emotions, and can articulate speech clearly enough to be lip-read as well as if not better than many human speakers.

**Patel and Willis** Like Waters’ work, the Facial Animation, Construction and Editing System (FACES) of Manjula Patel and Philip Willis [32] uses FACS to animate a muscle-based model head. This is an interactive system which allows the user to create and animate images of a face. The shape of the model can be altered interactively to match that of a specific individual.

### 1.3 Previous work: Simulation

While much of the work described above produces images of faces which act in realistic ways—they have what Rydfalk called dynamic realism—none of them can generate a convincing image of a specific person, *static* realism.

One technique often used to enhance realism in computer graphics is texture mapping. Essentially this involves pasting a picture onto the model to add surface detail.

**British Telecom** Bill Welsh and others at British Telecom, Martlesham have added texture from a picture of a real person to a model based on *Candide*, and made the mouth move to follow spoken input in real time (although with a short delay). This work has not however been published.

**Yau and Duffy** John Yau and Neil Duffy [16, 54, 55, 56] in their picture-phone coding work have a system capable of modelling the face of a specific individual. They start by accurately measuring the individual with a laser range-finder, and build a 3-dimensional wire-frame model with around 400 triangles from these measurements (we describe how they built the model from raw measurements on page 43). They find each of the model's vertices in a digitised photograph of the individual. As the polygons of the model are displayed, the colour and brightness from the original picture are copied onto the new image.

Two different methods are used to show expressions. In their first paper Duffy and Yau use several sub-pictures of the mouth and eyes, taken at the same time as the main picture, but showing different expressions. They superimpose the appropriate sub-picture onto the main image, using a cross-fading function to ensure picture continuity across the edge. This image was used as the texture source for the final synthetic image (which is likely to be viewed from a different viewpoint).

Yau quickly realized that this "picture-book" approach to displaying expression was not enough, and in subsequent work the face is deformed to match the changes in expression, although in a less sophisticated manner than Platt and Badler.

**MBASIC** The Model Based Analysis Synthesis Image Coding System developed by Kiyoharu Aizawa, Hiroshi Harashima and Takahiro Saito [1, 2, 20] attempts to encode and decode a typical videophone sequence showing a head and shoulders view of a person.

Like Yau and Duffy, they texture map a photograph onto a wireframe face model. To show expression they use both a “clip-and-paste” method equivalent to Yau’s picture-book method, and deform the model using FACS. They prefer to deform the model claiming that to clip and paste would mean sending up to ten times as much data.

Yau used a new model of each individual, but the model used in MBASIC is the same for each person. This model is of a general face and appears to have been hand-built. It is composed of about 400 triangles. Since each person’s face is a different shape the model is deformed to match the face in the photograph. Their method is described at greater length in chapter 5.4.

## 1.4 The Work of this Thesis

The work of this thesis is in some ways similar to the MBASIC work, which we were unaware of at the time (some of it had not been published).

We wished to create novel views of a person from one or more photographs. Rather than measuring each person in order to build a 3D model of their head, we wished to use a *generic* head model, the same for everyone. We would then deform this model to match the head in the photograph and texture-map the pixel intensity values from the photograph onto the head, just as Yau had done.

Before we describe this further we describe some related work in two dimensions.

## Chapter 2

# Distorting Pictures in 2 Dimensions

In this chapter we will describe 2 dimensional distortions of pictures of human faces. Although this was intended as a preliminary to the 3-dimensional work which follows, many of the techniques are common, and the 2D work has found applications in its own right.

The 2D distortions arose out of a collaborative venture with a number of psychologists.<sup>1</sup> In 1988 psychologists at the University of Nottingham [10] were investigating human memory of faces; they were particularly interested in the importance of the configuration of the features—the relative positions of the eyes, the nose, the mouth, etc. Their stimuli were created using *Mac-a-Mug Pro*<sup>TM</sup>, a software package for the Macintosh Plus computer, which allowed them to create faces from different combinations of hairstyles, eyes, noses, mouths, chins and other, optional, features. Several versions of each face were created, with the same features positioned slightly differently in each version. Although these faces (shown in figure 2.1) were fairly realistic they were just black and white, with

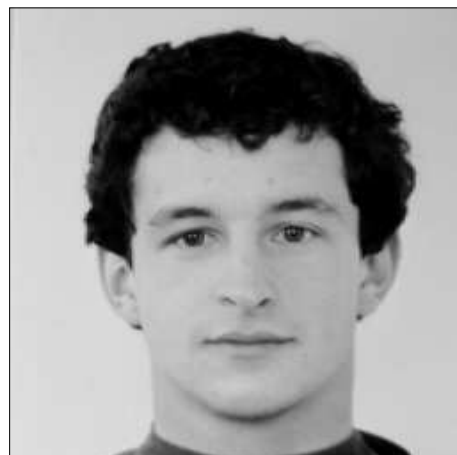
---

<sup>1</sup>A multicentred investigation into face recognition funded by an ESRC programme award to Vicki Bruce (Nottingham University), Ian Craw (Aberdeen University), Hadyn Ellis (University of Wales, Cardiff), Andy Ellis (York University), Andy Young (Durham University) and Dave Perrett (St Andrews University).

<sup>TM</sup>Mac-a-Mug Pro is a trademark of Shaherazam



**Figure 2.1:** *Three Mac-a-Mug faces, as used by the Nottingham group.*



**Figure 2.2:** *Three faces created by our method.*

no levels of grey, and no colour. It was suggested that the experiment would be improved if the features could be moved around on photographs of real people. With this possible goal in mind, we set out to distort images by computer.

## 2.1 Distorting Images

If we wish to distort pictures by computer we must have a way of storing a picture inside the machine, and ways of getting pictures in and out. The pictures were taken with an Hitachi HV-720K CCTV monochrome video camera and the output from this was digitized by an Imaging Technology FG-100-V Image Processor connected to the VME bus of a Sun 3/160 computer. The digitized image is an array of, usually  $512 \times 512$ , picture elements or *pixels*. Each pixel is an integer in the range [0–255], representing the grey level intensity of a small square or rectangle of the image. If we had used a colour camera we could represent colour images, by storing the intensities of the red, the green and the blue channels of the video signal at each pixel. Although the work described in this thesis was done in monochrome the changes necessary to use and generate colour images would be trivial. The images can be displayed on the computer display, showing the full range of colour and grey level intensities. It is possible to take a photograph of the screen, but paper hardcopies of the images are more usually printed on an Apple LaserWriter II PostScript laser printer. This prints black dots on plain paper, so the levels of grey are simulated by halftoning, printing more dots where the image is darker, and fewer where it is lighter, rather like a newspaper.<sup>2</sup> From a distance this gives the impression of many, but not the full range of levels of grey.

A digitized image can be manipulated in many ways by changing the pixel values or moving them around the array, according to specified rules—tasks computers are well suited to. A system was written to distort digitized photographs, the distortion being controlled by the user. The photographs could be of faces

---

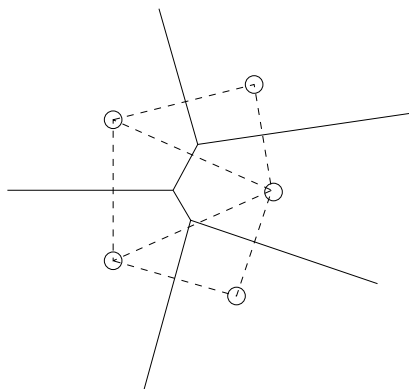
<sup>2</sup>In a newspaper the size of the dots can be varied, but the dots of a laser printer must all be the same size.

or anything else; the distortion is loosely modelled on what would happen to a picture painted on a rubber sheet if the sheet were stretched as it were pinned to a baseboard.

## 2.2 The Delaunay Triangulation of a Set of Points

As we will see in section 2.3, we distort an image by first dividing it into non-overlapping triangles whose corners lie where we would stick the pins into the rubber sheet. In the absence of any more specific information the Delaunay triangulation is the most appropriate division into triangles.

**The Voronoi Tessellation and the Delaunay Triangulation** The *Voronoi tessellation* of a set of data points in the plane is a partition of the plane into disjoint tiles. Each tile contains one of the data points and is composed of all points of the plane which are closer to that data point than to any other data point. It is also known as the Dirichlet or Thiessen tessellation. If we join the data points corresponding to adjacent tiles we form the dual graph of the Voronoi tessellation. Usually only three tiles of the tessellation meet at any point, thus



**Figure 2.3:** A Voronoi tessellation (solid lines) and its dual (dotted)

most regions of the dual graph will have 3 edges. The dual graph is therefore known as the Delaunay *triangulation* of the data points.

Occasionally four or more data points will be concyclic, more than three tiles will meet at a point, and the dual graph will contain a region with more than 3 sides. In this case we divide such regions up into triangles to form a Delaunay triangulation. There will be several ways of dividing the region into triangles, thus the Delaunay triangulation is no longer unique.

Lawson [24] and Sibson [38] both showed that the Delaunay triangulation is *locally equiangular*. This means that the triangles are as close to being equilateral as possible in the sense that every convex quadrilateral formed by two adjacent triangles, satisfies the **max-min angle** criterion:

The minimum of the six angles is greater than it would have been if the alternate diagonal had been drawn and the other pair of triangles chosen.

If the choice of diagonal does not change the size of the smallest angle, then the quadrilateral is cyclic, and we have two alternative Delaunay triangulations.

Our implementation of an algorithm to generate the Delaunay triangulation of a set of points in the plane is based on those of Sloan [40] and Correc and Chapuis [13]. We shall explain the differences as we come to them.

The algorithm is incremental—we create the Delaunay triangulation  $\mathcal{T}_n$  of the  $n$  points  $\mathbf{v}_1, \dots, \mathbf{v}_n$  by adding the point  $\mathbf{v}_n$  to the Delaunay triangulation  $\mathcal{T}_{n-1}$  of the  $n - 1$  points  $\mathbf{v}_1, \dots, \mathbf{v}_{n-1}$ . We start with a large triangle  $\mathcal{T}_0$ , big enough to surround all the given points and leave a reasonable gap. The given points are added to this trivial triangulation, one at a time. At each stage we find the triangle which the new point lies in, and create a new triangulation by joining the new point to each corner of this triangle. The resulting triangulation may not be locally equiangular - in which case we rearrange it so that it is once more a Delaunay triangulation.

By starting with a large triangle which surrounds all the given points we ensure that at each stage the new point lies inside a triangle of the current triangulation. This single triangle  $\mathcal{T}_0$  is clearly the Delaunay triangulation of three points. If we

make  $\mathcal{T}_0$  big enough to leave a reasonably large gap around the convex hull, when we remove the corners of  $\mathcal{T}_0$  we will be left with the Delaunay triangulation of the given points.

**Data Structures** The triangulation is stored in 2 arrays: an array of vertices and an array of triangles. For each vertex we store its position in the plane. For each triangle we store its three corners and the three adjacent triangles. We use a special code to indicate that there is no triangle adjacent on a particular edge. The three corners  $\mathbf{v}_1$ ,  $\mathbf{v}_2$  and  $\mathbf{v}_3$  and the three adjacent triangles  $T_1$ ,  $T_2$  and  $T_3$  are stored in order, with  $T_1$  between  $\mathbf{v}_1$  and  $\mathbf{v}_2$ . As we will see, to find the triangle containing a given point quickly it is helpful if all the triangles are defined with the corners going the same way, either all clockwise or all anti-clockwise around the triangle. For definiteness this implementation assumes they are all defined clockwise.<sup>3</sup>

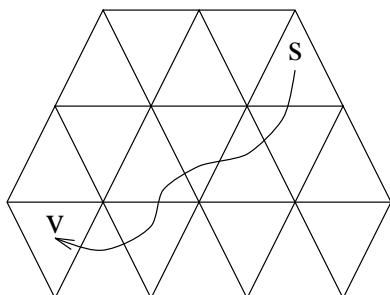
**Lawson's Stepping Algorithm** In order to add a point to the current triangulation we must find the triangle that it currently lies in. It would be very time consuming to test every triangle. Fortunately we don't have to. Lawson [24] devised a walking algorithm which avoids having to check each triangle to see whether it contains the new point. This walking search usually starts at a triangle which has the previous vertex as a corner. It then repeatedly steps into an adjacent triangle nearer to the point, until the containing triangle is reached. This is particularly effective if the points are arranged so that each point is close to the one inserted immediately before it. There will often be some sort of order or structure to the list of points, given as the data was captured—in almost any scanning technique successive points will be close together, except possibly where the scan has to fly back ready to scan another line. If the list of points has no particular order, or if optimum performance is necessary, the points can

---

<sup>3</sup>The notion “clockwise” depends on the coordinate axis system. If the positive  $x$ -axis points to the right, and the positive  $y$ -axis points down, then our triangles are indeed clockwise. This is the most common but not the only coordinate system we use.

be pre-sorted. Sloan uses a bin-sort which divides the area containing the  $N$  points into approximately  $N^{\frac{1}{2}}$  rectangular bins. However we are only interested in triangulations with small numbers of points. The two dimensional distortions require triangulations of 20–30 points, and the Delaunay triangulations used to create specific models in chapter 5 currently have at most 37 points. We have not therefore found the routine slow enough to warrant pre-sorting of the vertices, although we could easily add it if it became necessary.

A typical walk is shown in figure 2.4. If the current triangle and the desired vertex are on opposite sides of any of the lines formed by extending the edges of the triangle indefinitely, step over the line to the triangle on the other side. (If there are two such lines, pick either triangle.) Keep on stepping over lines until the desired vertex and the current triangle are on the same side of (or the vertex is actually on) each directed line. We have now found the triangle we were searching for.



**Figure 2.4:** A possible walk from triangle  $s$  to the triangle containing point  $v$

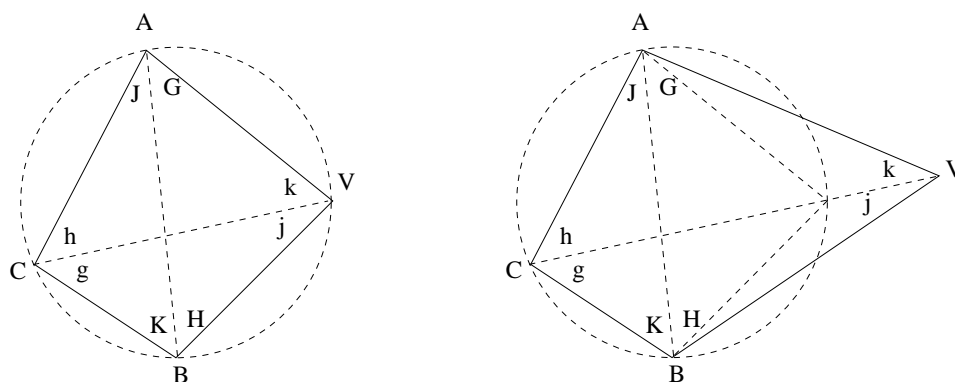
The walk fails if the current triangle and the desired vertex are on opposite sides of an edge of the triangle but there is no triangle adjacent on this edge to step into. Since the union of the triangles is its own convex hull this can only happen if the vertex doesn't lie on any of the triangles.

Since every triangle is defined with the corners going clockwise we can implement the stepping test in a particularly efficient way. Consider a vertex  $v$  and a triangle  $\mathbf{ABC}$ .  $v$  is on the same side of edge  $\mathbf{AB}$  as corner  $\mathbf{C}$  if and only if it is to the right of the *directed* line  $\mathbf{AB}$ . In a left-handed coordinate system this happens

if and only if  $(\mathbf{v}_y - \mathbf{A}_y) * (\mathbf{B}_x - \mathbf{A}_x) < (\mathbf{v}_x - \mathbf{A}_x) * (\mathbf{B}_y - \mathbf{A}_y)$ .

**The Circumcircle Test** We have already stated (page 14) that the Delaunay triangulation is locally equiangular; that is every convex quadrilateral formed by two adjacent triangles satisfies the max-min angle criterion. We will find it useful to consider an equivalent formulation known as the circumcircle test.<sup>4</sup>

Let us consider the case when the vertices of triangles  $\mathbf{ACB}$  and  $\mathbf{ABV}$  lie on the circumference of a common circle (figure 2.5, left). The centre of this circle is equally far from all 4 vertices so the dual of the Voronoi tessellation contains the quadrilateral  $\mathbf{ABCV}$  and the Delaunay triangulation could equally have contained triangles  $\mathbf{VAC}$  and  $\mathbf{VCB}$ . Since the angle subtended by the arc



**Figure 2.5:** *Equivalence of the max-min angle criterion and the circumcircle test.*

$\mathbf{AV}$  is the same at any point on the remainder of the circumference, angles  $\mathbf{ACV}$  and  $\mathbf{ABV}$  (marked ‘ $\mathbf{h}$ ’ and ‘ $\mathbf{H}$ ’) are the same, and similarly the three pairs of angles marked  $\mathbf{g-G}$ ,  $\mathbf{j-J}$  and  $\mathbf{k-K}$  are equal. Thus the smallest angle in the pair of triangles  $\mathbf{VAC}$ ,  $\mathbf{VCB}$  equals the smallest angle in the pair of the triangles  $\mathbf{ACB}$ ,  $\mathbf{ABV}$ .  $\mathbf{V}$  is at the critical point for the Delaunay triangulation and for the max-min angle criterion.

Consider what happens if instead  $\mathbf{V}$  lies outside the circumcircle of  $\mathbf{ABC}$  (figure 2.5, right). Now  $\mathbf{G} > \mathbf{g}$ ,  $\mathbf{H} > \mathbf{h}$ ,  $\mathbf{j} < \mathbf{J}$  and  $\mathbf{k} < \mathbf{K}$ . The angles of the

<sup>4</sup>See Lawson [24] for a complete and formal analysis of the relationship between the max-min angle criterion, the circumcircle test and the dual of the Voronoi tessellation.

triangles  $\mathbf{VAC}$ ,  $\mathbf{VCB}$  are  $\mathbf{g}$ ,  $\mathbf{h}$ ,  $\mathbf{j}$ ,  $\mathbf{k}$ ,  $\mathbf{H+K}$  and  $\mathbf{J+G}$ . Since the smallest of these cannot be  $\mathbf{H+K}$  or  $\mathbf{J+G}$ , the smallest must be smaller than all the angles of the triangles  $\mathbf{ACB}$ ,  $\mathbf{ABV}$  (which are  $\mathbf{G}$ ,  $\mathbf{H}$ ,  $\mathbf{J}$ ,  $\mathbf{K}$ ,  $\mathbf{k+j}$  and  $\mathbf{g+h}$ ) and the max-min angle criterion is satisfied by the triangle pair  $\mathbf{ACB}$ ,  $\mathbf{ABV}$ .

When  $\mathbf{V}$  lies inside the circumcircle the angle inequalities are reversed. Two triangles  $\mathbf{ACB}$ ,  $\mathbf{ABV}$  satisfy the max-min angle criterion if and only if the point  $\mathbf{V}$  lies outside the circumcircle of triangle  $\mathbf{ACB}$ .

Thus the max-min angle criterion is equivalent to applying the **circumcircle test**—does the new point  $\mathbf{V}$  lie inside the circumcircle of the triangle  $\mathbf{ABC}$  (figure 2.6)?

The following piece of pseudo-code is a numerically stable and efficient implementation of the circumcircle test, given by Sloan who credits it to Cline and Renka [12]. It returns TRUE if point  $\mathbf{V}$  is inside the circumcircle of triangle  $\mathbf{ABC}$ , and FALSE otherwise.

```

AC = A - C
BC = B - C
AV = A - V
BV = B - V
COSL = ACx·BCx + ACy·BCy           = AC·BC
COSM = BVx·AVx + BVy·AVy           = BV·AV

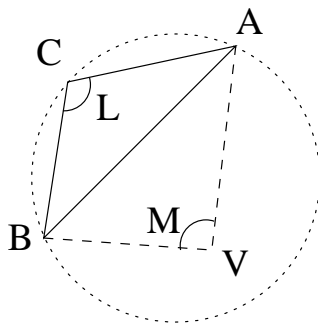
if (COSL ≥ 0 and COSM ≥ 0)
    return(FALSE)
else if (COSL < 0 and COSM < 0)
    return(TRUE)

SINL = ACx·BCy - BCx·ACy           = || AC × BC ||
SINM = BVx·AVy - AVx·BVy           = || BV × AV ||
SINLM = SINL · COSM + SINM · COSL

if (SINLM < 0)
    return(TRUE)
else
    return(FALSE)

```

□



**Figure 2.6:** *The circumcircle test*

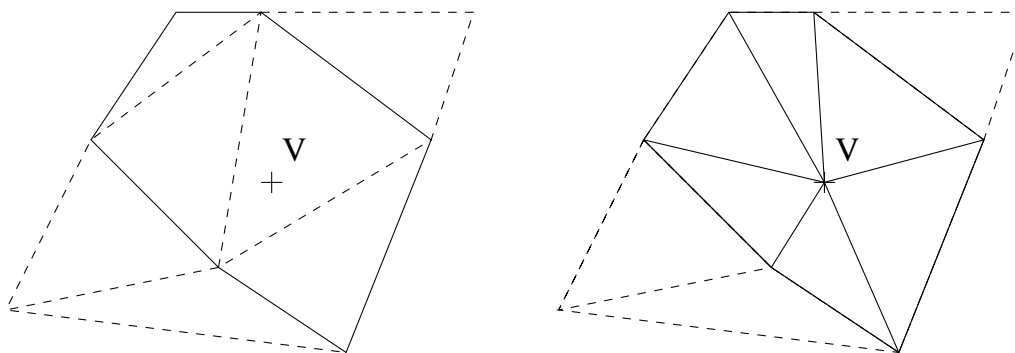
### Rearranging the triangulation to ensure that it is locally equiangular

Once we have found the triangle of  $\mathcal{T}_{n-1}$  which contains the new vertex  $\mathbf{v}_n$  we must add the new vertex and re-arrange the new triangulation to form the new Delaunay triangulation  $\mathcal{T}_n$ . Sloan [40] adds the new vertex to the triangulation immediately, splitting the containing triangle into three.

However, like Correc and Chapuis [13], we first consider the *insertion polygon*. We wish to find all the triangles of  $\mathcal{T}_{n-1}$  which do not appear in  $\mathcal{T}_n$ . We might expect that these are the triangles which when connected to  $\mathbf{v}_n$  form convex quadrilaterals which fail the max-min angle criterion, and indeed this is the case. As we have shown, these are the triangles whose circumcircles contain the new vertex. It can also be shown that the union of these triangles is well-connected; this union is the insertion polygon. To find the insertion polygon we start with the triangle containing  $\mathbf{v}_n$  and, recursively, consider adjacent triangles. If  $\mathbf{v}_n$  lies inside the circumcircle of an adjacent triangle this triangle is added to the insertion polygon and its adjacent triangles are tested too.

Just as a pair of triangles which fail the max-min angle criterion can be made to satisfy it by “swapping the diagonal”, we form the new Delaunay triangulation  $\mathcal{T}_n$  by joining each vertex of the insertion polygon to the new vertex  $\mathbf{v}_n$  as shown in figure 2.7.

Once we have inserted all the vertices we are almost, but not quite, finished. We still have a large triangle surrounding the control points, which we would like



**Figure 2.7:** *The new vertex placed in the insertion polygon.*

to remove from the final triangulation. This is simply a matter of deleting the triangles which share a corner with the large triangle. Occasionally this could leave a triangulation which did not cover the convex hull of the control points, but by centring the large triangle on the convex hull and making it several times larger, we can in practice ensure that the Delaunay triangulation is correct.<sup>5</sup>

We can now give the complete algorithm to find the Delaunay Triangulation of a set of control points:

Find large triangle  $\mathcal{T}_0$  which fits around all the control points.

For each control point  $\mathbf{v}$ :

    Find the triangle  $\mathbf{T}$  which contains  $\mathbf{v}$ .

    Let the insertion polygon  $\mathbf{P}$  equal  $\mathbf{T}$ .

    Push the triangles adjacent to  $\mathbf{T}$  onto a stack.

    While (stack not empty) do

        Pull triangle  $\mathbf{E}$ .

        If ( $\mathbf{v}$  inside circumcircle of  $\mathbf{E}$ ) then

            Add  $\mathbf{E}$  to  $\mathbf{P}$ .

            Push the two new adjacent triangles of  $\mathbf{E}$  onto stack.

            (We have already visited one of them.)

        endif

    endwhile

    We have found the insertion polygon  $\mathbf{P}$ .

    Destroy the triangles making up  $\mathbf{P}$ .

    Make new triangles by joining  $\mathbf{v}$  to each vertex of  $\mathbf{P}$ .

endfor

Remove the corners of  $\mathcal{T}_0$ , and all the triangles which have these as corners. □

---

<sup>5</sup>Unfortunately, however many times larger we make the triangle, it *is* possible to construct a set of control points which would fail.

## 2.3 Modelling the Distortion

We can now describe the distortion, by means of an analogy. Imagine that a photograph is printed on a stretched rubber sheet. We distort the photograph by sticking pins through the rubber sheet, moving the pins and anchoring them into a baseboard beneath. In this manner we can distort the photograph in many ways.

We describe this distortion in another way. Each pin has a *control point* associated with it. Each control point has a position in the original photograph, corresponding to the position of the pin in the rubber sheet, and a position in the distorted image, corresponding to the position of the pin in the baseboard.

Mapping the position of each control point in the photograph to its position in the distorted image gives us a function from a finite set of points in the original photograph into the distorted image. We will extend this function to form a function from the convex hull of the original points. This is the *distortion mapping*.

The distortion will occur within the convex hull of the control points, but the picture may extend outside this region. We shall assume that the remainder of the picture is unchanged by the distortion. If any part of the boundary of the convex hull moves during the distortion this may mean that parts of the original photograph are missing or duplicated in the distorted image. If this is not wanted more control points should be added so that either the distorted convex hull fills the image, or the control points in the boundary do not move.

The original function can be extended to a graph embedded in the plane of the picture in such a way that the nodes are the control points and the arcs are all straight lines. This extension can be defined as follows. If the point  $\mathbf{P}$  lies on the arc between control points  $\mathbf{A}$  and  $\mathbf{B}$ , since the arc is a straight line, we have

$$\mathbf{P} = a\mathbf{A} + (1 - a)\mathbf{B}. \quad (2.1)$$

for some  $a$ .

The image of  $\mathbf{P}$  under the function, which we shall call  $f$ , is defined in the now

obvious way:

$$f(\mathbf{P}) = af(\mathbf{A}) + (1 - a)f(\mathbf{B}). \quad (2.2)$$

This ensures that equal distances along each arc map to equal distances along the image arc.

The control point nearest to a given point in the picture is likely to be a corner of the Delaunay triangle in which it lies,<sup>6</sup> since the Delaunay triangulation is the dual of the Voronoi tessellation. There are many possible graphs with suitable embeddings in the plane of the picture, and as we will see other triangulations can also produce useful distortions, but the Delaunay triangulation gives a good default.

Once the mapping has been defined on the edges, we use *barycentric coordinates* to extend the mapping to the interior of each triangle, and hence the whole of the convex hull of the control points. If the point  $\mathbf{P}$  lies anywhere in the plane of the triangle  $ABC$ , there is a unique triple  $(a \ b \ c)$  such that:

$$\mathbf{P} = a\mathbf{A} + b\mathbf{B} + c\mathbf{C} \quad (2.3)$$

and

$$a + b + c = 1 \quad (2.4)$$

The triple  $(a \ b \ c)$  is independent of the coordinate system used. We say that the point  $\mathbf{P}$  has barycentric coordinates  $(a \ b \ c)$  with respect to the triangle  $ABC$ . Note that  $c = 0$  when  $\mathbf{P}$  lies on the line  $AB$ : thus equations 2.3 and 2.4 simplify to equation 2.1.

If  $\mathbf{P}$  lies within the triangle  $ABC$  we extend the function  $f$  so that  $f(\mathbf{P})$  has the same barycentric coordinates with respect to the distorted triangle as  $\mathbf{P}$  has with respect to the original triangle, ie:

$$f(\mathbf{P}) = af(\mathbf{A}) + bf(\mathbf{B}) + cf(\mathbf{C}). \quad (2.5)$$

This is a generalization of equation (2.2). A point on the arc  $AB$  has barycentric coordinate  $c = 0$ , so both equations map the point to the same image. The

---

<sup>6</sup>Likely, but not guaranteed.

function is continuous across each arc and hence continuous on the convex hull of the control points. We note that on each triangle of the Delaunay triangulation, this function is affine.

This function is much simpler than a physically accurate model of the way a rubber sheet would distort. Although it might be interesting to compare it with a true elastic mapping, people who have seen the distorted images of faces we have generated (eg. figure 2.2) have found them entirely believable. In fact in many cases unless the image is compared with the original, it is not obvious that there is a distortion at all, much less where the edges are.

## 2.4 Generating the Distorted Picture

Digitised images are, as we described in section 2.1, made up of pixels. We generate the distorted picture by copying the pixels from the photograph using a discretized version of the distortion mapping.

For each pixel in the distorted image we invert the mapping to find the preimage of the pixel in the original photograph, and copy the colour and intensity values from the preimage into this new pixel. This is known as screen-space *texture-mapping*. See Heckbert [21] for a recent survey of general texture mapping techniques. For us to be able to invert the distortion mapping it must map onto the whole image and no two points can map to the same point—it must be onto and 1:1.

We can pick extra control points explicitly to ensure that the distortion mapping is onto - specifying the preimage of each corner of the distorted image would be sufficient. Frequently we only wish to distort part of the photograph, leaving everything outside a *bounding polygon* as it was. Going back to the rubber sheet analogy, we frequently want to draw a closed curve around part of the picture. We would like to glue the rubber sheet onto the baseboard along this outline, so that when we move the pins we only distort the part of the picture inside the

glued outline. The most convenient way of doing this is to assume this outline is polygonal and have the vertices of this polygon as control points. The polygon will form the outline of the convex hull of the control points, and the distorted convex hull will be the same as the original. We can now extend the mapping to the whole plane by defining it to be the identity mapping on the exterior of the convex hull. This ensures that the mapping is onto.

This means that if it is not invertible it cannot be 1:1—there must be two distinct points with the same image. If these points lie on the same triangle then the image of the triangle must be squashed to a line or a point, and have zero area. If the two points lie on different triangles, these must overlap. Since the mapping is continuous this can only happen if part of the picture is folded over onto itself, reversing some of the triangles. As we have already mentioned (page 15) we specify the triangles by giving the corners in a clockwise order. Thus if a triangle is reversed its vertices will now be in anti-clockwise order. When part of the picture is squashed to nothing, at least one of the distorted triangles will have zero area. These facts enable us to determine when the distortion mapping is not invertible. If the distortion is specified interactively we can alert the user that they have specified a fold, and allow them to remove it.

Since we are working with a discrete form of the image, strange effects may occur when a low spatial frequency in the photograph is distorted to a frequency too high for the discretization. This frequency will be sub-sampled resulting in *aliasing*. Many filtering techniques have been used to over-come this problem, some at great computational expense. Again Heckbert [21] has an interesting summary. As Yau pointed out, pictures of faces are not particularly prone to this problem, and we have not yet found it necessary to implement any sort of antialiasing.

## 2.5 Implementation of the Distortion Algorithm

We now describe our implementation of the distortion algorithm. We first triangulate the control points using the Delaunay triangulation algorithm described in section 2.2. We actually use the Delaunay triangulation of the *distorted*, rather than the original, positions of the control points, although there is frequently no difference between the two triangulations and hence between the resultant mappings. The reason for this choice is that early implementations of the algorithm sometimes missed pixels at the edges of triangles. Taking the Delaunay triangulation of the distorted points ensures that the distorted triangulation is locally equiangular, minimizing the lengths of triangle edges in the final image, and hence the chance of these ‘dropouts’ occurring.

Generally the distorted image is the same size as the original, so we copy the original to the distorted image. This implements the identity mapping for the exterior of the convex hull of the control points in a simple manner. It would be possible to calculate the convex hull and thus avoid filling the interior using the incorrect identity mapping, but in most cases the efficiency gain would be too small to justify the increased complexity of the code.

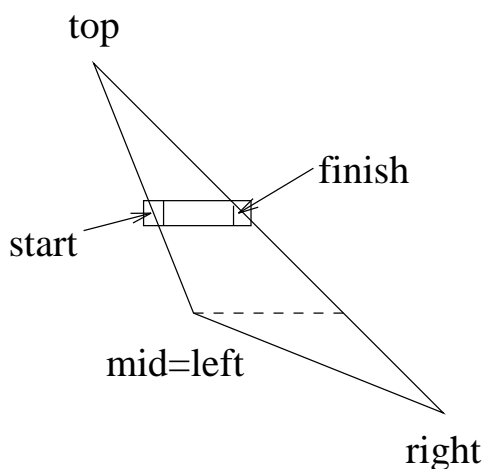
We then overwrite the pixels making up the interior of the hull by copying the picture one triangle at a time. We copy a single triangle by copying texture into each of the pixels which make up the triangle, and only to these pixels. These pixels are found by scanning the “after” position of the triangle, one row at a time, starting at the top and working down, but we must be careful which pixels to choose, so that the dropouts mentioned above cannot occur.

These dropouts appear when pixels on the boundary of two triangles are not texture mapped as part of either triangle. They were not noticed at first because the distortion was performed on top of a copy of the original picture. This meant that most of the time the correct texture still appeared even if a dropout occurred.

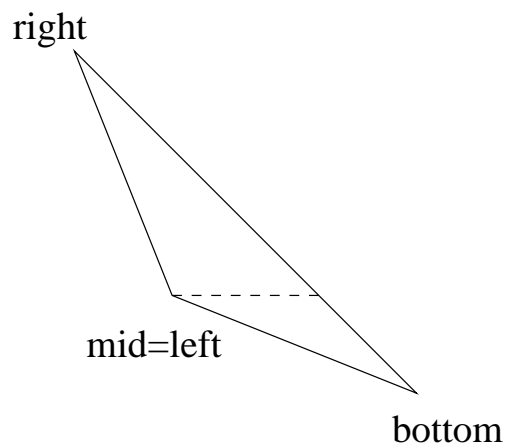
The simplest solution was to extend the texture mapping a pixel to the left or right beyond each edge. This removed most of the dropouts although there was a tendency to lose the continuity of any pattern across the edge. However some dropouts remained along edges that were nearly horizontal, and we found it necessary to implement the scan much more carefully.

The root problem is that the pixellated triangle is discrete and scanning it using floating point variables which are then rounded is very prone to round-off errors. By rounding each corner of the triangle to the nearest pixel and scanning using integer coordinates this round-off can be avoided, and by always scanning down and to the right we calculate the same edge pixels for two adjacent triangles.

Each triangle is scanned in two halves, from the top down to the middle corner, and from the middle corner to the bottom. When scanning the top half (figure 2.8) we step along the edges **top-left** and **top-right** from **top** to **mid** one row at a time, and across each row from **start** to **finish**.



**Figure 2.8:** *Scanning a triangle: top half.*



**Figure 2.9:** *Scanning a triangle: bottom half.*

We use a simplified version of Bresenham's line algorithm [19, pp 433–6] to find **start** and **finish** for each row. Bresenham's line algorithm calculates the pixels of a line incrementally. When the line is taller than it is wide (the gradient is greater than 1) standard Bresenham gives one pixel for each row, but when the

### Pseudo-code for the scan-line algorithm.

To texture map a triangle with corners **after**[1–3],  
using the texture from a triangle with corners **before**[1–3]:

```

b12 = before[2] - before[1]           b13 = before[3] - before[1]
a12 = after[2] - after[1]           a13 = after[3] - after[1]
if abs( det(a12 | a13) ) < 2 then area of triangle is less than one pixel
    exit

```

Calculate **RowStepBefore** and **ColStepBefore** using:

$$(\mathbf{RowStepBefore} \mid \mathbf{ColStepBefore}) = (\mathbf{b12} \mid \mathbf{b13}) (\mathbf{a12} \mid \mathbf{a13})^{-1}$$

Pick distinct labels *top*, *mid* and *bottom* so that:

$$\mathbf{after}[top]_y \leq \mathbf{after}[mid]_y \leq \mathbf{after}[bottom]_y$$

Pick labels *left*, *right* so that edge *top* → *left* is to left of *top* → *right*, as follows:

```

if (after[mid]x - after[top]x) * (after[bottom]y - after[top]y) <
    (after[bottom]x - after[top]x) * (after[mid]y - after[top]y) then
    left = mid           right = bottom

```

else

$$*left* = *bottom* *right* = *mid*$$

Calculate constants for “steep Bresenham” lines *top* → *left* and *top* → *right*

$$\mathbf{left}_{col} = \mathbf{right}_{col} = \text{ROUND}(\mathbf{after}[top]_x)$$

$$\mathbf{left}_{row} = \mathbf{right}_{row} = \text{ROUND}(\mathbf{after}[top]_y)$$

$$\mathbf{ScanLeftBefore} = \mathbf{before}[top]$$

for **Scan<sub>row</sub>** from  $\text{ROUND}(\mathbf{after}[top]_y)$  to  $\text{ROUND}(\mathbf{after}[bottom]_y)$  do

if **Scan<sub>row</sub>** =  $\text{ROUND}(\mathbf{after}[mid]_y)$  then we are half way

change labels *left*, *right* so that edge *left* → *bottom* is to left of *right* → *bottom*

calculate constants for “steep Bresenham” algorithm for these edges

$$\mathbf{ScanBefore} = \mathbf{ScanLeftBefore}$$

for **Scan<sub>col</sub>** from **left<sub>col</sub>** to **right<sub>col</sub>** do

copy texture from **ScanBefore** to **Scan**

add **ColStepBefore** to **ScanBefore**

endfor

apply one “steep Bresenham” step to **right**

apply one “steep Bresenham” step to **left**

if **left<sub>col</sub>** changed then

update **ScanLeftBefore** by  $\pm \mathbf{ColStepBefore}$

add **RowStepBefore** to **ScanLeftBefore**

endfor

□

line is short and wide this would leave gaps between pixels, so Bresenham works the other way, calculating one pixel for each column. We want **start** and **finish** to move down exactly one row at each step, so we use the steep part of Bresenham's algorithm whatever the gradient of the line.

When we reach the middle corner we have finished scanning the top half of the triangle and must re-label the corners before scanning the bottom half. The new labels are shown in figure 2.9. Note that the corner **left** is actually to the right of corner **right**—the labels are chosen so that the edge **left-bottom** is to the left of **right-bottom**. One of these edges (**right-bottom** in this case) is a continuation of an edge from the top half, but the other edge is new; new constants must be calculated so that the simplified Bresenham steps along the new edge.

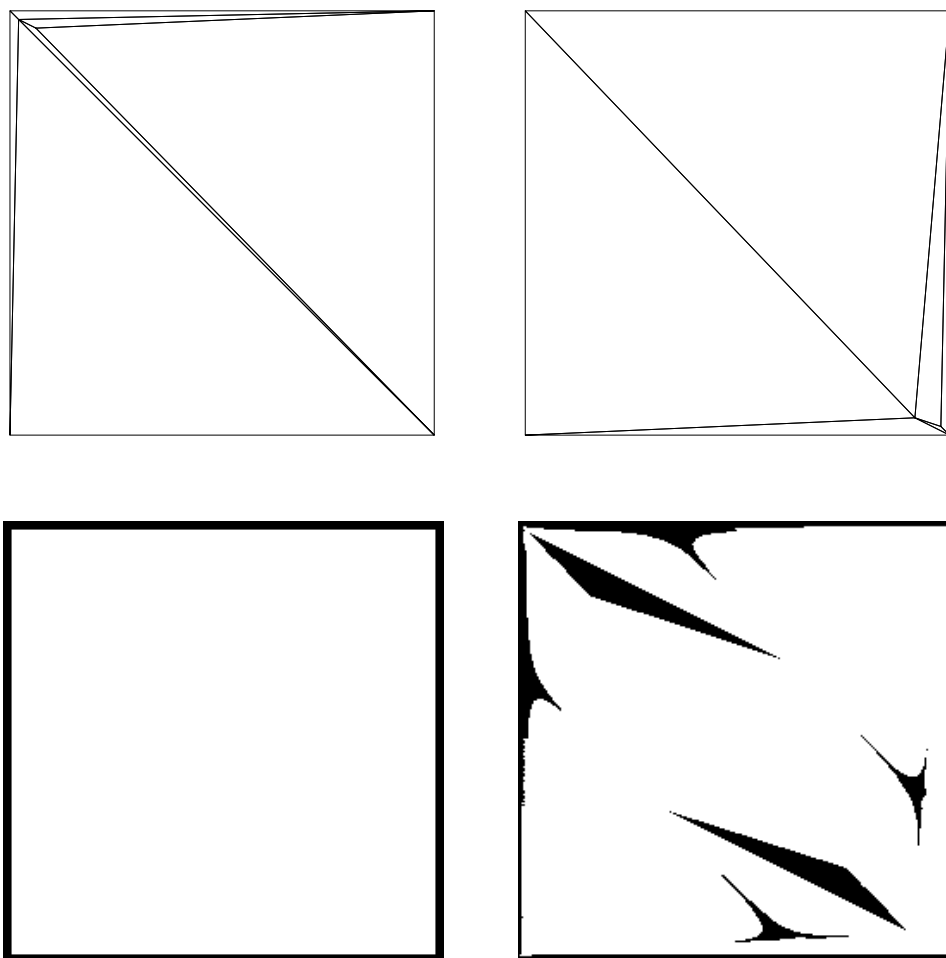
While we are scanning the triangle we find the preimage of each pixel. Since the distortion mapping on a single triangle is affine and we are scanning the “after” triangle in a coherent manner, it is not necessary to explicitly invert the mapping to find the preimage of each pixel. The preimages of two horizontally or vertically adjacent pixels differ in position by a constant displacement, so we merely need to add the appropriate displacement as we step to adjacent pixels of the “after” triangle. In general the displacement will have non-integer components, so we calculate the preimage positions using floating-point variables.

Running on a Sun SPARCstation ELC, the algorithm takes about 1 second to distort the  $256 \times 256$  pixel image as shown in figure 2.10.

**Texture-mapping a single pixel.** Strictly speaking a pixel has area so its preimage is a region of the texture image. If we implemented anti-aliasing we would consider this region properly, but currently we just round the preimage position to the nearest pixel and copy a point sample from the texture image into the pixel currently being scanned.

**Centroid-edge scan distortion** Phil Benson claims [6] that he encountered dropouts and other problems when mapping texture from one triangle to another

using horizontal or vertical scans similar to the one described in this section, and uses a centroid-edge scan instead. This works by scanning lines from the centroid of the triangle to each pixel on each edge. Phil was kind enough to give us his code for this algorithm, but we have found that although it doesn't cause dropouts under small distortions (when the triangles change very little) it can leave very large dropouts under more extreme distortions—see figure 2.10. These



**Figure 2.10:** A white square distorted with the row-column scan algorithm (bottom left) and the centroid-edge scan algorithm (bottom right). Top left shows the “before”, and top right the “after” positions of the control points. The result should be a white square inside a black border.

dropouts appear to occur because the scan along each edge of the triangle and the scan from the centroid to each edge pixel both finish when a test based on

the *source* positions fails. This test fails when the scan reaches the edge pixel of the source triangle, even though it may be several pixels away from the edge of the destination triangle. By contrast no dropouts have been found when using our implementation of the algorithm given in section 2.5. Our conclusion is that all algorithms for texture mapping triangles are heavily dependent on the precise implementation used.

## 2.6 Applications

We began this chapter with the Nottingham work on the configuration of faces using Mac-a-Mug images. We are now ready to describe how our picture distorting system was used to create photographic stimuli for the Nottingham group. Figure 2.2 shows the final results. In order to distort an image we must first choose the control points and specify both their positions in the original image and the positions we wish to distort them to.

It is important that the outline of the face doesn't change when we distort the image, so we define a bounding polygon which doesn't move (see section 2.4). We wish to move either a single feature or a group of features around the face, leaving the picture of the feature unaltered. In the rubber sheet analogy we wish to make the rubber underneath the feature rigid by gluing it onto a shaped piece of cardboard which we then displace. This is done by defining a second polygon and moving each of its vertices by the same amount. The vertices of these 2 polygons are the control points we need.

Since the Nottingham group wanted stimuli created from 21 different faces, and several versions of each face, each with the features moved by different controlled amounts, we wrote a program which displayed the image on the computer screen and allowed the user to specify the positions of the control points interactively by using a mouse. The user is able to specify how many pixels the feature should be moved in each direction, and the program can calculate the distorted positions,

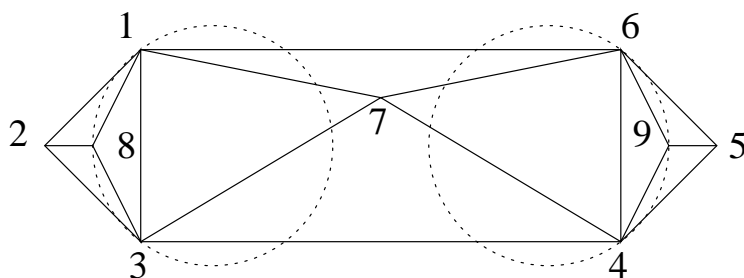
saving all the control points in a file which could then be used as input to the distortion program. It can also triangulate the triangulation of the control points, highlighting any triangles which are reversed by the distortion, recognising these because the corners are given in anti-clockwise order. This alerts the user to folds in the distortion mapping, making it easy for corrections to be made.

Unfortunately the Delaunay triangulation of these points may include triangles which cross the inner polygon, so are half inside the feature we are shifting and half outside it. Especially when the feature starts or finishes close to the outer polygon, this can distort the feature considerably.

This cannot happen if each edge of the inner polygon lies on the common edge of two triangles. We modify the triangulation to ensure this. The circumcircle test of the Delaunay triangulation routine tests whether two adjacent triangles should be changed into two others by swapping the common edge. We alter this test, so that if the common edge lies on the inner polygon we never swap it, and always swap the common edge when this would make it lie on the polygon. Figure 2.11 shows a set of control points that the algorithm would triangulate without ever testing all the edges of the inner polygon, giving a final triangulation that still had triangles which crossed the inner polygon. This has never happened in practice, but the correct remedy would be to allow the user to modify the triangulation directly, rather than changing the algorithm again.

This technique was used to create 12 distorted versions each of 21 faces (figure 2.2 shows some examples), and these were used in further experiments by the Nottingham group, who concluded that the effects found using the Mac-a-Mug images were as strong, if not stronger, using the texture-mapped images [9, 8].

**Other possible applications** While we were working on this technique Phil Benson of the University of St. Andrews was developing very similar methods to create photographic quality caricatures of individuals [6]. He first created a set of 186 control points and a hand built triangulation. The positions of these points



**Figure 2.11:** A set of control points which are mis-triangulated by the new algorithm. Vertices 1–6 form the outer polygon and vertices 7–9 the inner polygon. When triangulated the result should include triangle 789, but if the vertices are inserted in numerical order, we get the triangulation shown.

were found for an average face, and also found manually on a picture of the individual he wished to caricature. By exaggerating the differences between the positions of the control points on the individual and the average face, and distorting the photograph so that the control points moved to these exaggerated positions he was able to generate a photographic caricature in which the distinguishing features of the individual were enhanced. He also conducted experiments [5] which showed that such caricatures of famous people were more quickly identified, and said by subjects to be better likenesses, than the original photograph.

The 3-dimensional merges of two faces which we will describe in chapter 8 can also be done in 2D. This was first shown in the newspaper coverage of the 1990 Conservative Party leadership elections [4] in which Benson transformed a picture of the defending candidate, Margaret Thatcher, into a picture of the challenger, Michael Heseltine.

## Chapter 3

# Generating Novel 3-Dimensional Views

Many useful images and effects can be created by distorting pictures of a face in 2 dimensions, as described in the previous chapter. However heads are 3-dimensional objects; many manipulations of the image are achieved more effectively if we take account of this.

In our introduction we mentioned 2 reasons for wishing to rotate a picture of a head. We could rotate a head using a 2D distortion if we identified suitable “before” and “after” positions for a set of control points by hand. These positions are the projections of 3D points; if we knew these we could generate the “after” positions directly.

Moreover, it is quite likely that, in one or other view, some of these points would be hidden behind other parts of the face. This means that some of the triangles are likely to overlap, making the 2D texture mapping impossible, as we cannot tell which triangle is visible and which obscured. If we used a 3D distortion with an implicit understanding of the shape of the head the 3D positions would indicate which triangles are hidden behind others, removing the ambiguity of the 2D texture mapping.

We noted on page 8 that Yau and Duffy combined a photograph and a 3D

model of the same individual to create their novel views. They used a laser scanning system to make 3D measurements of the individual in order to build the model. Often however we may have a photograph but neither the individual nor the laser scanner. One solution—the main subject of this thesis—is to use a 3D model of the *generic* human head. This model will be used for each photograph we wish to rotate. Since this will not be exactly like the head in the photograph we will distort it to form a new, *specific* model head which is the same shape as the head of the individual in the photograph. This distortion will be described in chapter 5. This specific model can easily be rotated to give the impression of viewing from a new viewpoint.

If we wished, we could hand-build a 2D distortion which changed a smile into a frown, or any other change of expression, but again this is really a change of 3D positions projected onto the plane of the image. Platt and Badler used the standard facial expression description system FACS to give expression to a 3D line-drawn model of a face. If such a model was identified with the photograph, when we changed the expression on the model its projection could be used to give the “after” positions of the control points.

We then texture map the head model, using the original photograph to provide the colour and texture information, so that the head looks like the person in the photograph, not like a chicken-wire sculpture.

To recap: to generate a novel 3D view from a photograph of an individual we

- build the generic head model,
  - create the specific head model by distorting the generic head,
  - transform the model to create the novel view,
- and
- display the specific model with the colour and texture from the photograph.



**Figure 3.1:** *Candide texture-mapped*

**What Limits the Quality of the Novel View ?** Ideally the novel views of the subject would be indistinguishable from images we could take using a camera.

Although a digitised image can be as good as, or better quality than, a photograph or a still from a video camera, our digitised images are not as good as a good photograph, because they do not resolve such fine detail or have as many possible variations in brightness.

Figure 3.1 shows an image generated using Candide as a simple specific model. Because the model is not very detailed the image has sharp corners and is very angular. Since we obtain the specific head by distorting the generic model they

both have the same level of detail. Thus the final image is affected by the level of detail in the generic model.

When the specific head is an accurate model of the subject, the final image depends on the accuracy with which the vertices of the model are located in the photograph. If these positions are accurate our method is the essentially the same as the one described by Yau, and we would expect the same quality of results.

Thus the quality of our images also depends upon the accuracy of the specific model and how well the vertices are located in the original image.

### 3.1 Representing the Generic Model

As mentioned above the generic head model is intended to be a 3D description of a general human head, not just the head of the individual in the photograph. It is possible to describe a 3D object using a (very large) number of small cubes or *voxels*. However it is really the *surface* of the head that interests us.

If a surface is relatively flat and it is possible to see the whole of it from a suitable viewpoint, we can represent it with a *depth map*. This is a 2D array of the heights of the surface above a projection plane, recorded at the intersection points of a grid on the plane. We would like to model the whole surface of the head, right around the back, so a single depth map will not do. We could model the surface by splitting it into a number of *patches*, representing the shape of each one by a depth map, but the rectangular grid restricts the ways we can manipulate a depth map, and makes it difficult to join them together. Instead we use a simple function to define the shape of each patch.

Planar patches have been used to model heads [29, 37, 54, 2, 48], and other 3D surfaces are frequently modelled using patches made with Bezier and other bicubic splines (see Foley and van Dam [19, chapter 13]) and from superquadrics [34].

More planar patches are needed to model a surface to a given precision than are needed with the other types of patch, but we have chosen to use planar surface-

patches, because we will need to distort the model to match the photograph. In order to distort a model made with planar patches we just move the corners of each patch, but to distort a model made with bicubic splines or ellipsoid patches we would also need to make adjustments to the curvature parameters of each patch. It is not clear how to determine these adjustments so that the model best matches the photograph. Since we can achieve the same precision by modelling the surface by using more planar patches, we do not believe that other types of patch justify the effort needed to determine the best curvature parameters, at least at present. We have thus chosen to stick with a model made with the simpler and neater planar patches. The resulting description of the surface of the head is just a list of the polygons which mark the edges of each patch and a list of the corners or *vertices* of these polygons. Together these lists form a *wire-frame model* of the head. We simplify our representation of a surface one stage further, by using *triangulations*—wireframe models in which every polygon is a triangle.

## 3.2 Data for the Generic Head

We have two candidate wire frame model heads available. Let us consider these.

Details of Rydfalk’s wireframe face *Candide* [37] are repeated in appendix B. When Rydfalk designed this model he was limited to 100 triangles, and he was not concerned that the face should look like a person. We experimented by texture-mapping a real face onto *Candide* (figure 3.1) and concluded that to make the images realistic we would require a more detailed model. The biggest complaint was the very pointed nose.

At the other extreme, we have a triangulation of the head of Vicki Bruce, one of the Nottingham psychologists mentioned earlier. The model is known after her. It was constructed by Alf Linney [27] at University College Hospital, London by vertically scanning a laser range-finder up and down Vicki’s head while she was seated on a rotating platform. It is made up from nearly 20 000 triangles

and 10 000 vertices. Even without the colour, shade and texture of a real face this model is highly convincing, but displaying it with even a simple shading model takes several minutes on our Sun 3/160. The finest details of the model are likely to be specific to Vicki and not found on every human head. Neither the generic→specific transformation (which we will describe in chapter 5) nor the texture-mapping of the display routine (chapter 6) will readily preserve this fine detail. Thus we desire a generic head somewhere between these two, more detailed than Candide, but not as detailed as Vicki. When creating wire-frame models of individuals for picture-phone coding, Yau [53] and Aizawa [1] both found that around 400 polygons (mostly triangles) provided a sufficiently detailed model face for texture mapping. We thus propose that the generic model of a full head should be made from around 1000 triangles. This gives a sensible compromise between detail and drawing time of the final image.

In the next chapter we will consider how to approximate one triangulation with another, less detailed triangulation. We would like, initially, to find a triangulation with around 1000 triangles which approximates Vicki. This may remove much of the detail which was specific to Vicki, while preserving the information which distinguishes a human head from, say, a rugby ball. We were aware however that it might not be possible to use a single generic head for everyone—it might be better to have a number of different heads, representing people of different ages, racial types and perhaps even for men and women. Since the start of this work the UCL group have measured the heads of many other people, and produced even more detailed models, including average male and female heads, each made from around 70 000 rectangles. These average models may give better results for a wider range of faces than a model, such as Vicki, derived from a single person.

Approximating one triangulation with another may have other advantages too. Benson has found that the number and position of the points around the eyes is critical for his photorealistic caricatures. On the other hand the position of points around the back of the head are relatively unimportant because hair is relatively uniform at coarse resolutions. Hair also confuses the laser range-finder,

so measurements of the back of the head are not as good as those of the face. We may therefore find that the best model approximates different parts of the head to different accuracies. The ability to cut the surface into pieces and approximate each piece separately would allow us to retain the detail of the face without having too much false precision in the back of the head.

## Chapter 4

# Building Generic Head Models

As explained in chapter 3, we would like to construct a triangulated wireframe model head made up of around 1000 triangles, to used as the generic head model. We wish to construct this triangulation by finding a suitable approximation to the range-finder data describing the head of our associate Vicki. This data was used to generate figure 4.1; it came to us as a triangulation with nearly 20 000 triangles, and we would like a method of approximating it with another triangulation which has fewer triangles.

### A Naïve Approximation

The range-finder data was obtained by taking many vertical sweeps down the head, rotating the subject between each sweep. This pattern is preserved in the data for the model. As a result it is relatively easy to keep every  $n$ -th vertex, throwing the  $n-1$  intervening vertices away. As  $n$  becomes large the triangles will become long and thin since the distance between successive vertices increases but the distance between sweeps remains constant. We can keep the shape of the triangles more or less the same if we throw out whole sweeps as well.

As figure 4.2 shows, the resulting model is a good approximation where the surface changes very little, but in places, such as the mouth, where the surface



**Figure 4.1:** *Vicki Bruce's head, reconstructed by Phong-shading the original triangulated data.*



**Figure 4.2:** A naïve approximation of the original head (figure 4.1). The model has 2498 vertices and 4885 triangles.

gradient changes more rapidly the approximation is less good.

The triangles of the this model (and of the original mesh) are all about the same size. What we really want is triangles which approximate the surface equally well, or equally badly. This would give us small triangles where the surface gradient changes rapidly and larger triangles where the surface is flatter.

We are looking for a triangulation which approximates the original to within some tolerance but whose error exceeds this tolerance if any vertex is removed. By trying several different tolerances it should be possible to find a triangulation with around 1000 triangles. Since it is not the tolerance that concerns us but the approximate number of triangles in the triangulation, it doesn't matter if a few of the vertices can in fact be removed without exceeding the tolerance.

## 4.1 Approximating a Triangulation

The best approximation with a given number of triangles would probably have vertices not lying in the surface, but slightly off to one side or the other. Finding the best approximation would involve finding the best position for each vertex of the resulting triangulation. It is not clear how to do this, but it appears to require a search in a space with a very large number of degrees of freedom.

To simplify the search we restrict the problem to that of finding an approximating triangulation whose vertices are vertices of the original triangulation.

**$2\frac{1}{2}$ D methods** There is a considerable body of work concerned with approximating a given set of points with a triangulation.

Yau [53] describes an algorithm which generates a wireframe model of a face from a depth map image. If the depth interpolated between two points on the depth map approximates the depth map sufficiently well at each intermediate point, then the two points are the end points of an acceptable edge of the triangulation. Yau first approximates the occluding boundary of the head using edges

as big as possible. Starting at the top of the head and working down, he builds isosceles triangles onto each edge. At first each triangle is very short, but it is allowed to grow in height, until the edges cannot be made any longer. More triangles are then grown from the new edges and this is repeated until the whole face has been triangulated.

Another method is based on the *Delaunay triangulation* (see chapter 2) of the  $x$ - and  $y$ -coordinates of the points. By choosing a suitable subset of the points the  $z$ -coordinate of the remaining points can be approximated by interpolating the  $x$ - and  $y$ -coordinates across the appropriate triangle. Many authors have considered this method, but arguably the best solution has been proposed by Floriani [15], who developed an efficient algorithm which produces a pyramid of successively more detailed, more accurate Delaunay triangulations approximating a given set of points. Each triangulation is a refinement of the previous one, giving a multi-resolution representation of the surface. The whole pyramid can be stored and searched about as efficiently as a single triangulation.

Recently Terzopoulos has used a spring model to approximate Alf Linney's range-finder data with a 50x25 mesh. It would be easy to convert such a mesh into a triangulation. No description of this work is yet available, but Terzopoulos and Vasilescu [42] have described a spring mesh method which approximates a depth-map. Like Yau's method and the Delaunay-triangulation method, this approximates a surface by sampling the depth map at a number of locations.

These locations are chosen by first placing a regular rectangular mesh over the depth-map. The nodes of this mesh are given the same constant mass, and are joined by springs with adjustable stiffnesses. The stiffness of each spring is controlled by the average of some property<sup>1</sup> of the depth-map at each of its endpoints. Thus the nodes where the gradient is changing most are attached to the strongest springs. The spring-mass system is allowed to move towards equilibrium, with the spring stiffnesses changing as the nodes move. At equilibrium the nodes

---

<sup>1</sup>The property is  $\|\nabla\|\nabla d\|$ , the magnitude of the gradient of the magnitude of the gradient of the depth at that endpoint.

(and hence sample points) are close together in regions where the depth is changing rapidly, but fewer nodes where the surface is flatter.

These algorithms all require sets of data points defining comparatively flat surface which project onto the  $xy$  plane under parallel projection without overlap. It seems reasonable therefore to call them  $2\frac{1}{2}$ D methods. Unfortunately the vertices of Vicki do not fulfill this requirement as they are spaced right around the head, covering the back as well as the front. By projecting the points onto a sphere or an ellipsoid and dividing this into a number of almost planar patches it would, in principle, be possible to apply one of these algorithms.

**3D methods** Two other truly 3 dimensional methods suggest themselves. The first is due to Faugeras et al. [18]. They suggest picking three vertices spaced around the model, then repeatedly adding the vertex which is furthest from the approximation, until all the original vertices are sufficiently close to the approximating triangulation.

Alternatively we could start with the full triangulation, and remove vertices wherever this is possible without reducing the accuracy below the given threshold.

## 4.2 Faugeras' Point Insertion Algorithm

Faugeras et al. [18] describe an algorithm which generates a triangulation approximation to a given wireframe model of a surface of a 3-dimensional object. Faugeras points out, however, that the algorithm will only work for surfaces of genus 0, thus it would not work for a doughnut or a coffee cup. This also means that we would have to extend the algorithm—since our model head is a surface which stops at the neck it has a hole at the the bottom.

**Outline** The algorithm starts with a trivial triangulation—two triangles back-to-back. Each triangle is the approximation for one half of the given wireframe. We consider each piece of the wireframe in turn. If every vertex is sufficiently close

to the approximating triangle, then the approximating triangle is good enough. If not we split the triangle into 3 by adding the vertex furthest from the triangle. We split the piece of wireframe into 3 smaller pieces, one piece approximated by each new triangle.

The new edges of the triangles may not be very good approximations, so we refine them by adding the vertex furthest from, but approximated by, the edge. This splits a pair of triangles into four.

We now repeat the process, until every vertex of the original is sufficiently close to a triangle.

### 4.2.1 The Algorithm Described by Faugeras

Faugeras represents the original wireframe as a graph. The reason for this can best be seen by considering the correspondence between the graph and an approximating triangulation. Since every vertex of the approximation is a vertex of the wireframe, the corners of the triangles are nodes of the graph. The edge of each triangle approximates a path in the graph between the end-points. The interior of each triangle approximates a connected component (CC) of the graph. Every node of the graph lies on a CC or on one or more paths, but no node lies on a path and a CC.

**The initial approximation.** In Faugeras's implementation the 3 points which form the zero-order approximation are chosen by the user. The resulting approximation is sensitive to this choice; they suggest that for best results the points should be chosen in the first plane of inertia.

The next step is to find the shortest, most planar cycle containing the three points. This gives the three paths between the vertices, and splits the graph into two subgraphs—the connected component approximated by each of the triangles.

**Split** We consider each approximating triangle  $T$ , with vertices  $P$ ,  $Q$  and  $R$  in turn. Let  $C(T)$  be the connected component approximated by  $T$ . If every vertex of  $C(T)$  is sufficiently close to  $T$  then it is good enough and will appear in the final triangulation—we add  $T$  to the set of definitive triangles and consider it no more.

Otherwise find the vertex which is furthest away—call it  $M$ . In Faugeras' implementation  $M$  is actually the point which maximizes  $D - k.D'$  where  $D$  is the distance between  $M$  and the plane of the triangle, and  $D'$  is the distance between  $M$  and the barycentre of the triangle. Faugeras claims that this gives better results “because it reduces the disparities of the sizes of the new [planar] faces”, but gives no indication of what value of  $k$  is typically used, or of how to determine it.

We replace triangle  $T$  in the approximating triangulation with 3 new triangles— $PQM$ ,  $QRM$ , &  $RPM$ . We must also find paths from  $M$  to each of  $P$ ,  $Q$  &  $R$ , and the connected components approximated by each new triangle.

Faugeras chooses the path from  $M$  to  $P$  so that it is close to the bisector plane of  $MPQ$  and  $MRP$ , and the paths from  $M$  to  $Q$  and  $R$  so that they are close to the appropriate bisector planes. This is intended to keep the paths away from each other and from the paths  $PQ$ ,  $QR$  and  $RP$  already chosen around the edge of triangle  $T$ . Faugeras uses the “shortest” or rather cheapest path, where the cost of each path is the sum of the cost of the nodes which lie on it, and the cost of each node is the distance between it and the bisector plane in which we would like the path to lie.

To find the cheapest path efficiently, he uses a best-first search implemented using a binary tournament. This is a binary tree structure in which the value at each node is smaller than the value at the parent node, and allows value it be inserted into a list, or the smallest value to be determined in  $O(\log n)$  time. The entire shortest path search takes  $O(n \cdot \log n)$  time where  $n$  is the number of vertices in the CC.

To find the new connected components, Faugeras first finds a point in each

new CC, then *explores* the region around this point until the bounding paths are reached. Points in each connected component are found by searching the neighbours of appropriate paths for a point which lies in  $C(T)$ . For example to find  $C(PQM)$  we search the neighbours of points on the path PQ, but excluding points on the paths QR and RP, to avoid points which should belong to either  $C(QRM)$  or  $C(RPM)$ . Once we have found a point on the new CC, we mark it as such and, recursively, consider its neighbours. Each neighbour which also lies in  $C(T)$  also becomes a member of the new CC.

**Refine** Since splitting a triangle into three can create an edge which is a poor approximation to the path in the wire frame, the new triangles are refined after each step. We split each adjacent pair of triangles into four by adding the point on the path approximated by the common edge which is furthest from this edge. These new triangles are added to the new approximation, ready to be split into three again. As before this means that we must find new paths approximated by the new edges, and update the CCs.

**Data structures** The original triangulation is represented by the array of its nodes. For each node they store 4 things: the position of the node, the list of neighbouring nodes, which CC approximates the node (if any), and the list of the paths which pass through the node. The list of paths is empty if, and only if, the node is approximated by a CC.

The current triangulation is represented by the array of its triangles. For each triangle they store the 3 corner vertices and the name of the CC which is approximated by the triangle.

## 4.2.2 Faugeras' Pseudo-code for the Complete Algorithm

### Initialization

The sets NEW, ACTIVE and DEFINITIVE of [planar] faces are initially empty.  
 Choose three points P, Q, R in the input graph.  
 Find a cycle that contains the three points.  
 Split the two CCs separated by the cycle.  
 Add the six faces to the set ACTIVE.

### Loop:

NEW = empty set.

#### Step 1 (Split):

For each face of ACTIVE do:  
 Remove T from ACTIVE.  
 Find the point M in C(T) that maximizes  
 the distance to the face T.  
 If the distance is lower than S then  
 Add T to DEFINITIVE.  
 else  
 Split T into three faces T1, T2, T3 and  
 add the new faces to NEW.

#### Step 2 (Refine):

For each pair (T1, T2) in NEW such that  
 T1 and T2 have a common edge do:  
 Remove T1 and T2 from NEW.  
 Break the common edge and add the new triangles to ACTIVE.

### Condition

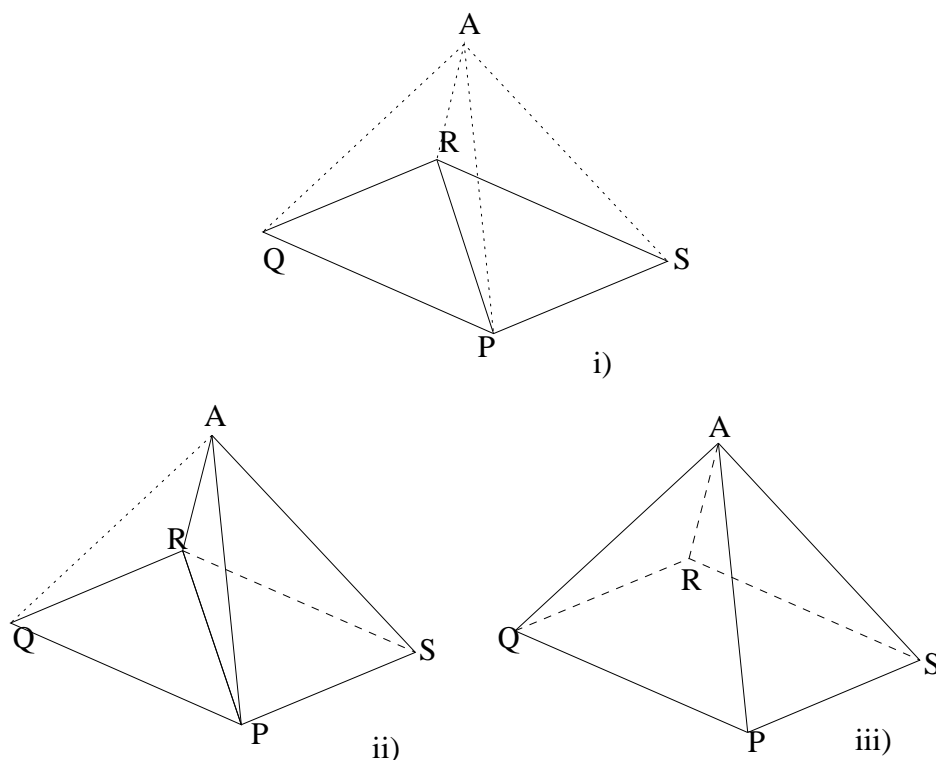
ACTIVE = ACTIVE  $\cup$  NEW.  
 If ACTIVE is empty then STOP,  
 else goto LOOP.

□

## 4.2.3 Criticisms of Faugeras's Method

A good approximation should be close to the original points and to the implied surface. The following illustration (figure 4.3) shows that it is the *Refine* step of algorithm which ensures this. We would like a very accurate approximation to a pyramid without a base. The initial approximation, diagram i), consists of the triangles  $PQR$  and  $RSP$ , and the tolerance is such that we must add the apex  $A$ .

Diagram ii) shows a new approximation which includes the apex, by replacing



**Figure 4.3:** *Approximating a pyramid. i) Initial triangulation. ii) Triangulation which matches vertices. iii) Triangulation which matches vertices and edges.*

triangle  $RSP$  with the 3 triangles  $RAS$ ,  $SAP$  and  $PAR$ . This is the result of applying the *split* routine to triangle  $RSP$  of i). This includes every vertex of the pyramid so it must be a good (perfect) approximation to the given set of points. However the edge  $QA$  is not well approximated by this approximation.

Diagram iii) shows a third approximation—the four triangles which make up the top of the true pyramid. This is clearly a perfect approximation everywhere. This is the result of applying the *refine* routine to the edge  $PR$  in i).

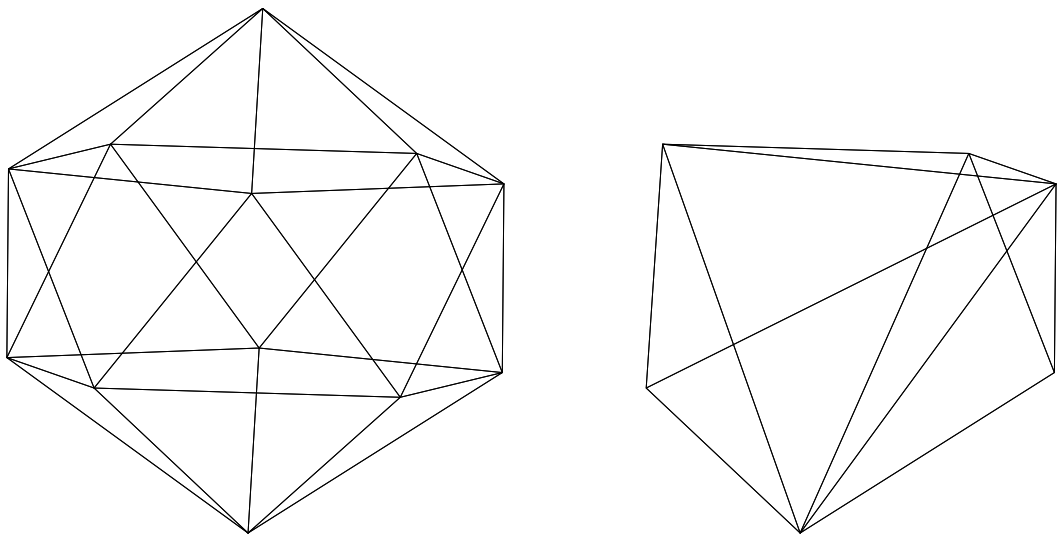
Since the edge-refining step is so important to the final result, some errors in Faugeras' pseudo-code must be pointed out:

- The edges of the first six triangles should be refined before the triangles are split.
- A triangular face is accepted as definitive if all the vertices in its connected

component are sufficiently close, whether or not the vertices approximated by the edge are close enough.

- Faugeras makes no explicit choice about which pairs of new triangles are refined, and no triangle is refined more than once in a single iteration of the algorithm. As many of the newly created triangles will have new triangles adjacent on 2 or even 3 sides this means that some of the edges will not be refined, at least not until the triangles have been split once more.

Figure 4.4 shows how these errors can affect the final approximation.



**Figure 4.4:** *A regular icosahedron and Faugeras' approximation. The tolerance was so small that the approximation should have been the same as the original.*

After spending some time trying to fix these problems and extend the algorithm to cope with holes (so that we can approximate a head model which stops at the neck), without success, we felt that it was necessary to try another approach.

### 4.3 A Point Removal Algorithm

Faugeras' algorithm (section 4.2) starts with a very simple triangulation and adds vertices until this is a sufficiently good approximation to the vertices of the given

triangulation  $\mathcal{T}$ . We have seen that it may fail to generate good approximations even for quite simple triangulations.

We shall now consider an alternative approach, starting with the triangulation  $\mathcal{T}$  and removing vertices until we reach the point where removing any other vertex would make the error in approximating the *surface* of  $\mathcal{T}$  exceed some given threshold  $\epsilon$ . The error in approximating a vertex  $V$  which lies on a boundary of the given triangulation is the distance from  $V$  to the nearest point on a boundary of the approximating triangulation, and the error in approximating a vertex  $V$  which does not lie on a boundary of the given triangulation is the distance from  $V$  to the nearest point on a triangle of the approximating triangulation.

### 4.3.1 Removing a Vertex is a Local Operation

It would take too long to calculate the error in approximating each of possibly ten thousand vertices of  $\mathcal{T}$  to determine whether we may remove a single one; fortunately we don't need to. Removing a single vertex  $\mathbf{v}$  from a triangulation  $T$  is a local operation—the new, reduced, triangulation will only differ from  $T$  in a neighbourhood of  $\mathbf{v}$ . The triangulation of this neighbourhood is a subtriangulation of  $T$ —call it the *neighbourhood triangulation*  $N$ . We split  $T$  into two: the neighbourhood triangulation and the *complementary triangulation*  $C$  - the complement of  $N$  in  $T$ . We now remove the vertex  $\mathbf{v}$  from  $N$ , to form the *reduced neighbourhood triangulation*  $R$ . Finally we form the *reduced triangulation*  $T^*$  by taking the union  $R \cup C$ . This is the triangulation  $T$  with the vertex  $\mathbf{v}$  deleted.

If  $T$  is an acceptable approximation to the given triangulation  $\mathcal{T}$ , then  $T^*$  will be acceptable if it is a sufficiently good approximation to all the vertices of  $\mathcal{T}$  which are best approximated in  $T$  by a triangle in  $N$ . This will certainly be true if the error in approximating each of these vertices by  $R$  is sufficiently small. Thus we can determine whether removing the vertex  $\mathbf{v}$  will result in an acceptable approximation to  $\mathcal{T}$ , by calculating the approximation errors just for the vertex  $\mathbf{v}$  and those vertices which are approximated by triangles in  $N$ . We don't even

need to calculate the union  $T^*$  if the test fails.

If the surface folds back on itself it is possible for a vertex to be close, in space, to some triangle, but a long way away on the surface. If this triangle was used to approximate the vertex, the result would be a triangulation which was a good approximation to the vertices, but a poor approximation to the surface. Such triangles are automatically excluded from the improved search.

### 4.3.2 Remembering Removed Vertices

When we remove a vertex we cannot totally ignore it. If we did so we could easily start removing triangles which were approximating vertices which had already been removed. In this manner it would be possible to remove all but one triangle from a mesh if the desired accuracy  $\epsilon$  was greater than the mesh size. We must record some information about the vertices that have been removed.

One way would be to store a list of vertices with each triangle of the current triangulation. This list would contain the vertices approximated by this triangle. Whenever a vertex was removed it would be added to the list of the triangle which now approximates it. This would usually be the nearest triangle, but when the surface folds back on itself the nearest triangle in 3 dimensions might not be the closest on the surface. Each time a triangle was destroyed the vertices would be passed on to new triangles. When implemented this method was found to be very slow, because whenever a triangle was destroyed a new home had to be found for all the vertices which it was approximating. We store information about the removed vertices in a different way.

In this new method each triangle has a half-thickness and approximates the volume of space which is within a distance equal to this half-thickness of the triangle. The triangles of the original triangulation have zero thickness. Whenever we remove a vertex we calculate the thickness of all the new triangles, making them fat enough to approximate the old vertex and the volume of space approximated by the triangles which have been removed. If the thickness of any of these

new triangles is greater than the tolerance  $\epsilon$ , the new triangulation will be too inaccurate an approximation to the original surface, and we cannot remove the vertex.

### 4.3.3 Outline of the algorithm

To find a triangulation  $T$  which approximates triangulation  $\mathcal{T}$  such that the error in approximating each vertex of  $\mathcal{T}$  is at most  $\epsilon$ , and  $T$  has as few vertices (and hence triangles) as possible:

```

Set  $T_0 = \mathcal{T}$ 
Each triangle in  $T_0$  has zero thickness.
for  $n = 1, \dots$ , number of vertices in  $\mathcal{T}$  do
    Choose a vertex  $\mathbf{v}_n$  of  $T_{n-1}$  that we haven't chosen before.
    Let  $N_n =$  neighbourhood triangulation of  $\mathbf{v}_n$  in  $T_{n-1}$ .
    Remove  $\mathbf{v}_n$  from  $N_n$  to give the reduced neighbourhood triangulation  $R_n$ .
    if  $\mathbf{v}_n$  lies on the boundary
        edge_dist = distance from  $\mathbf{v}_n$  to the new edge of  $R_n$ 
        next_for_loop if edge_dist  $> \epsilon$ 
    endif

    if  $R_n$  has triangles
        dist = distance from  $\mathbf{v}_n$  to nearest triangle in  $R_n$ 
    else
        dist = edge_dist
    endif

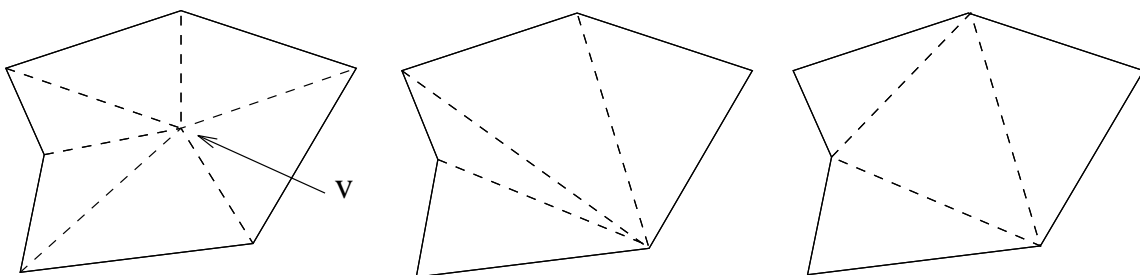
    worst_thickness = max thickness of triangles of  $N_n$ .
    if dist + worst_thickness  $< \epsilon$  then
        calculate thickness for each triangle in  $R_n$ 
         $T_n = (T_{n-1} - N_n) \cup R_n$ 
    else
         $T_n = T_{n-1}$ 
    endif
endfor
 $T_n$  is the desired approximation  $\square$ 

```

### 4.3.4 Removing a single vertex from the neighbourhood triangulation

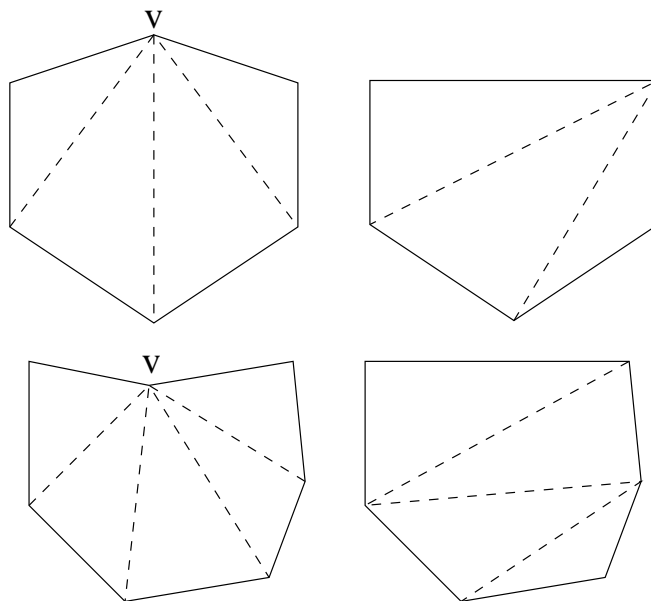
We introduced the neighbourhood triangulation of a vertex  $\mathbf{v}$  in section 4.3.1, defining it as the part of the triangulation which changes when  $\mathbf{v}$  is removed. This subtriangulation is in fact just the union of the triangles of which  $\mathbf{v}$  is a corner.

As the outline of the algorithm shows, we wish to remove  $\mathbf{v}$  from the neighbourhood triangulation ( $NT$ ) leaving the reduced neighbourhood triangulation ( $RNT$ ). There are in fact many possible reduced neighbourhood triangulations. Each is a triangulation with boundary whose vertices are the vertices of the  $NT$ , but not including the central vertex  $\mathbf{v}$ . Each vertex lies on the boundary of the  $RNT$ . Consider the boundaries of the neighbourhood triangulation and each  $RNT$  as polygons in 3-space. If  $\mathbf{v}$  lies on the boundary of the  $NT$  then the boundary polygon has vertices  $\mathbf{v}\mathbf{v}_2\dots\mathbf{v}_n$  in order. Remove this vertex to create the new polygon  $\mathbf{v}_2\mathbf{v}_3\dots\mathbf{v}_n$ . This is the boundary of every  $RNT$ . If  $\mathbf{v}$  does not lie on the boundary of the  $NT$  then each  $RNT$  has the same boundary as the  $NT$ . Figures 4.5 and 4.6 show some neighbourhood triangulations and possible  $RNT$ s.

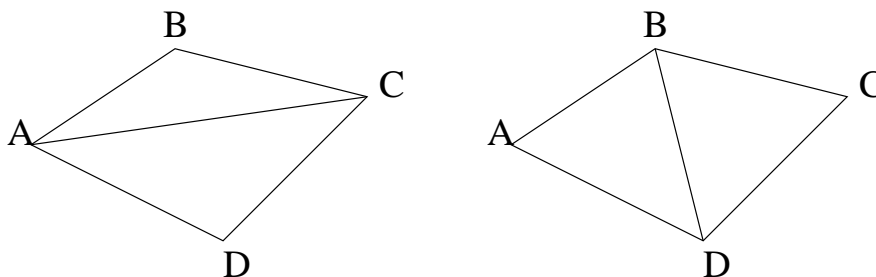


**Figure 4.5:** A neighbourhood triangulation (left) and two possible reduced neighbourhood triangulations.

We can obtain any  $RNT$  from any other by *swapping the diagonals* on pairs of triangles which have a common edge as follows (see figure 4.7). If  $ABC$  and  $CDA$  are two adjacent triangles we can form 2 new triangles  $ABD$  and  $BCD$ . If the 4 vertices are coplanar then the union of each pairs of triangles is the quadrilateral



**Figure 4.6:** When the central vertex  $v$  lies on the boundary of the neighbourhood triangulation, the reduced triangulation may cover a larger or a smaller area.



**Figure 4.7:** Swapping the Diagonals

$ABCD$ . The common edge of the original pair of triangles is one diagonal of this quadrilateral, and the common edge of the new pair is the other diagonal.

The simplest, most naïve RNTs are formed by picking a vertex  $v_1$  from the  $n$  vertices of the boundary polygon. Number the remaining vertices in order around the polygon. We create the  $n - 2$  triangles with vertices  $v_1v_2v_3$ ,  $v_1v_3v_4$ ,  $\dots$ ,  $v_1v_{n-1}v_n$ . The union of these triangles will be a triangulation of the bounding polygon, provided that no two triangles intersect except along an edge or at a vertex. If two triangles do have such a non-trivial intersection then the view of one edge of the polygon from  $v_1$  is obscured by another edge. This is the *Art*

*Gallery Problem* and is discussed at length in [28]. For our purposes it is sufficient to note that this cannot happen unless the polygon is concave and at least four of the vertices are coplanar. It is unlikely that the boundary polygon will be of this form. If a naïve triangulation of the polygon has triangles with non-trivial intersections then choosing a different vertex as  $\mathbf{v}_1$  may solve the problem. If not, swapping the diagonals, perhaps several times, will produce a true reduced neighbourhood triangulation.

The best RNT is the one which minimizes the maximum error in approximating  $\mathbf{v}_n$  and each of the vertices approximated by triangles in the neighbourhood triangulation, that is minimizes *WorstError* above. We could generate every possible RNT and calculate *WorstError* for each. This is clearly inefficient. If we start with any simple RNT and swap the diagonals whenever this reduces *WorstError* until swapping diagonals no longer helps, we will find an RNT which is locally minimum. By finding a suitable swapping strategy it may even be possible to ensure that this is the global minimum.

### 4.3.5 Calculating the thickness of a triangle

When we remove the vertex  $\mathbf{v}_n$  from its neighbourhood triangulation  $N_n$  we obtain a reduced neighbourhood triangulation  $R_n$ . If  $\mathbf{v}_n$  is close enough to  $R_n$  we remove the triangles of  $N_n$  from the previous triangulation and replace them with the triangles of  $R_n$ . Each of these new triangles must be given a thickness. Since we didn't record the exact positions of the vertices which have already been removed, we choose the thicknesses so that the volume of space approximated by the new triangles contains the volume approximated by the triangles of  $N_n$ .

The simplest way is to set the half-thickness of each triangle to be the distance from  $\mathbf{v}_n$  to the triangle plus the half-thickness of the fattest triangle in  $N_n$ .

In figure 4.8 we have accepted absurdly large errors in the approximation—the tolerance  $\epsilon$  is 2cm on a head 25cm tall ! Not surprisingly the result is not a good approximation to a head. Even so there are more than six times as many triangles

as we would like.



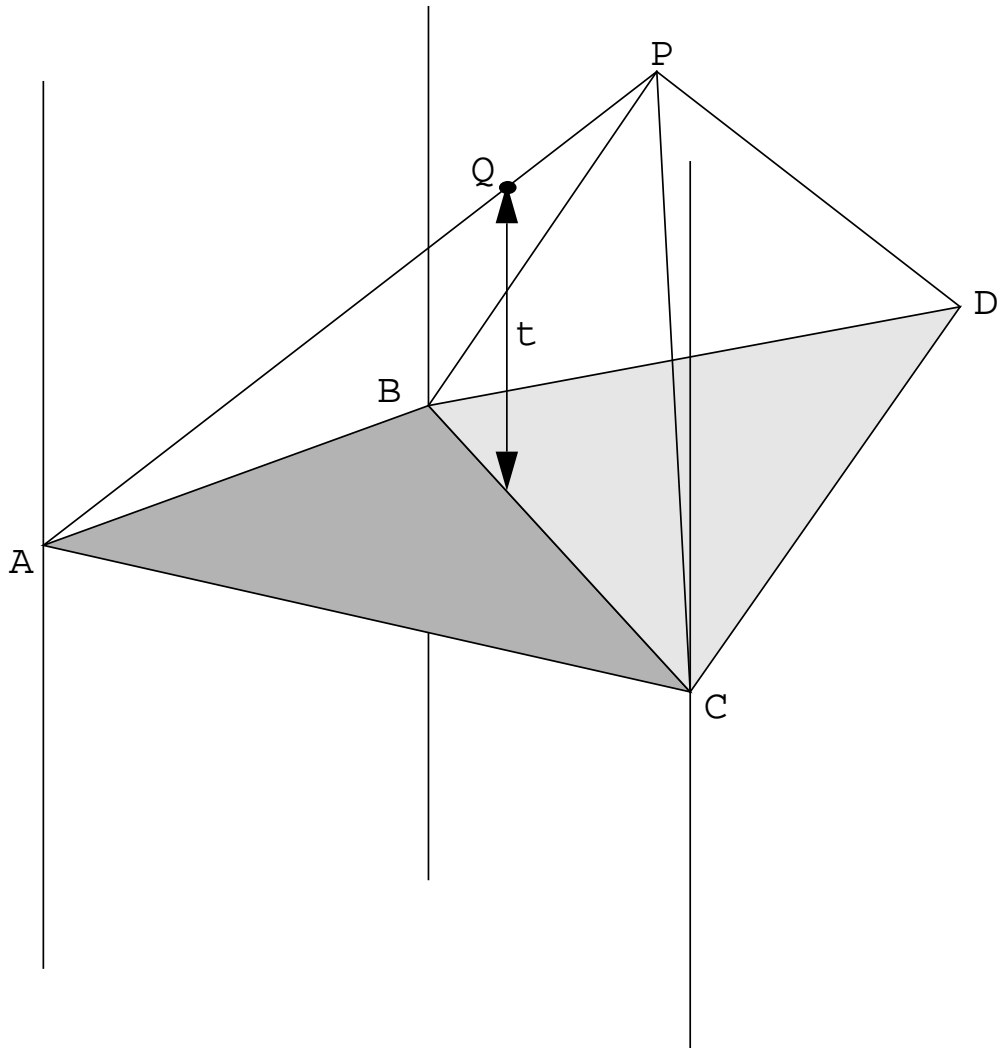
**Figure 4.8:** *Vicki approximated by the simple thickness calculation.  $\epsilon = 2\text{cm}$ , 3149 vertices and 6213 triangles.*



**Figure 4.9:** *Vicki approximated by the sophisticated thickness calculation.  $\epsilon = 2\text{cm}$ , 2979 vertices and 5861 triangles.*

We therefore use a more sophisticated method of calculating the thickness of each triangle. We imagine vertical walls along the perimeter of the new triangle, extending infinitely far above and below the triangle (see figure 4.10). We now consider the interior edges of  $N_n$ , the edges connecting  $\mathbf{v}_n$  to the other vertices. Whenever one of these edges pierces a wall we calculate the distance from the intersection point to the triangle, and add the thickness of the edge. If  $\mathbf{v}_n$  lies inside the walls we also calculate its height above (or below) the triangle, adding the thickness of the fattest triangle in  $N_n$ . The thickness of the new triangle is the maximum of this set of distance-plus-thicknesses.

As implemented, this method assumes that where two new triangles meet, each is only responsible for approximating the part of the volume directly above



**Figure 4.10:** *The sophisticated triangle thickness calculation. We wish to remove vertex  $P$ , leaving triangles  $ABC$  and  $BDC$ .  $Q$  lies on the edge  $AP$ , and in the plane through  $BC$  and perpendicular to the triangle  $ABC$ . The thickness of  $ABC$  equals the thickness of edge  $AP$  plus the distance ‘ $t$ ’ from  $Q$  to the triangle.*

or beneath it. Although not always strictly true, this assumption is almost true. We could however avoid this assumption by using walls which were not vertical, but which bisected the angle between adjacent triangles of  $R_n$ .

Comparing figures 4.8 and 4.9 shows that, for the same  $\epsilon$ , the sophisticated calculation uses fewer triangles and approximates the head better, than the simple thickness calculation. The strange triangles around Vicki's right eye in figure 4.9 are in fact correct. Although they lie very close to the surface, the triangles in this region face in different directions, and these different directions are enough for the shading routine to colour adjacent triangles very differently.

### 4.3.6 In what order do we remove the vertices ?

So far we have considered the vertices in the order in which they were collected—successive sweeps down the face. Since the final triangulation depends upon the order in which the vertices are chosen and tested for removal, this may not be the best method. As table 4.1 (page 63) shows, different choice strategies produce triangulations with very different numbers of triangles for the same threshold  $\epsilon$ . Two opposing strategies particularly suggest themselves.

- Attempt to remove a vertex close to the previous vertex, thus making big changes locally before making changes throughout the whole surface.

We have implemented this by removing the vertices in the order in which they were originally found—ordered into vertical sweeps down the head.

- Keep the removed vertices far apart, making lots of small changes right across the surface before making bigger changes anywhere.

Without detailed knowledge of the shape of the original model we cannot explicitly keep the vertices far apart—instead we shuffle the vertices and attempt to remove them in this random order.

Figure 4.11 shows a much better approximation than figure 4.9, but using 40% fewer triangles.

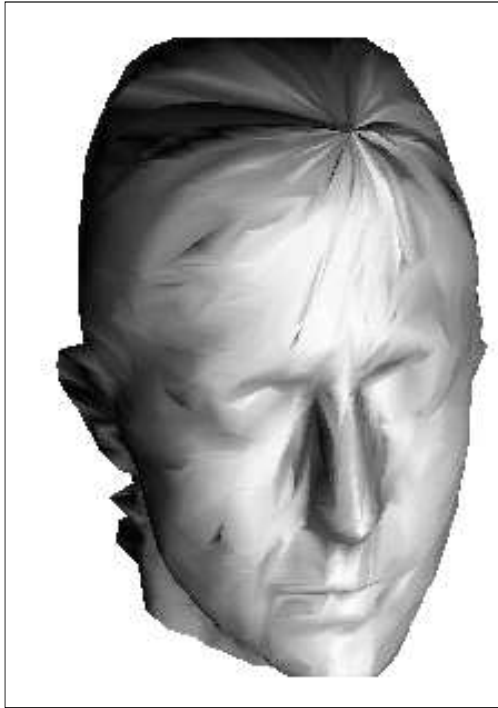


**Figure 4.11:** *Sophisticated, random approximation.  $\epsilon = 0.5$ , 1741 vertices and 3391 triangles*

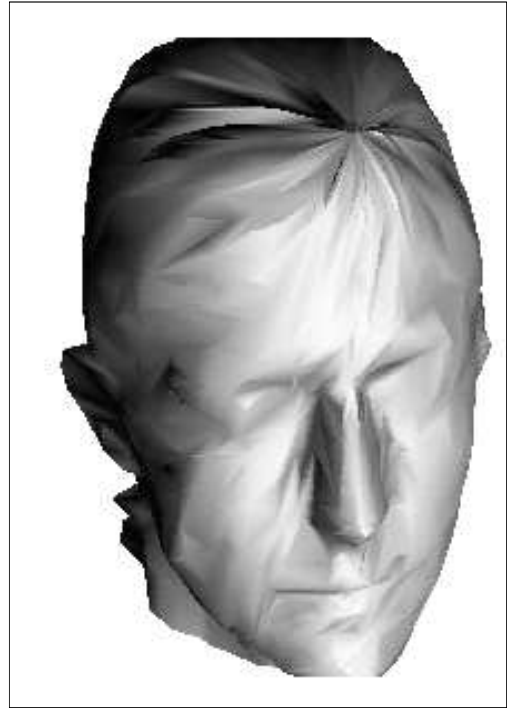
**Approximating the approximation** We can run the program again, using the output triangulation as the input data, approximating the approximation to Vicki. This gives an approximation which cannot be worse than the sum of the  $\epsilon$  used in each pass. Figure 4.13 shows two heads, one better, but more detailed than figure 4.11, and the other not as good but less detailed. This suggests that several passes with a close tolerance give about as good an approximation as a single, more generous pass, if the results have the same number of triangles.

**Our Generic Model.** Trying out all these methods of reducing triangulations took time, and we wanted a generic model to use for the remainder of the work, while we determined which was the best way of approximating a triangulation. In order to proceed with this we actually used the naïve approximation shown in figure 4.2 as the generic model for the examples in the remainder of this thesis.

This model has 2498 vertices and 4885 triangles—rather more than the 1000 triangles we would desire, but as we got access to faster computers this didn't cause many problems.



**Figure 4.12:** 2 passes of the sophisticated, random approximation.  $\epsilon=0.1\text{cm}$ , 2152 vertices and 4171 triangles



**Figure 4.13:** 3 passes of the sophisticated, random approximation.  $\epsilon=0.1\text{cm}$ , 1473 vertices and 2831 triangles

$\epsilon$ (cm)	Simple		Sophisticated			
	Ordered		Ordered		Random	
	vertices	triangles	vertices	triangles	vertices	triangles
0.1			4560	8932	3712	7223
0.2			4056	7953	2788	5441
0.5			3580	7043	1741	3391
1.0	4110	8076	3313	6515		
2.0	3149	6213	2979	5861		
5.0	1978	3898				
10.0	1461	2879				
0.1 twice					2152	4171
0.1 3 times					1473	2831
0.1 4 times					1116	2126
0.1 5 times					921	1739

**Table 4.1:** Comparison of the number of vertices and triangles in the approximations produced for different threshold values ( $\epsilon$ ), using the simple and sophisticated thickness calculations, with the vertices considered in order or at random. The last few rows show the results of using the sophisticated, random algorithm to approximate the approximation, several times.

## Chapter 5

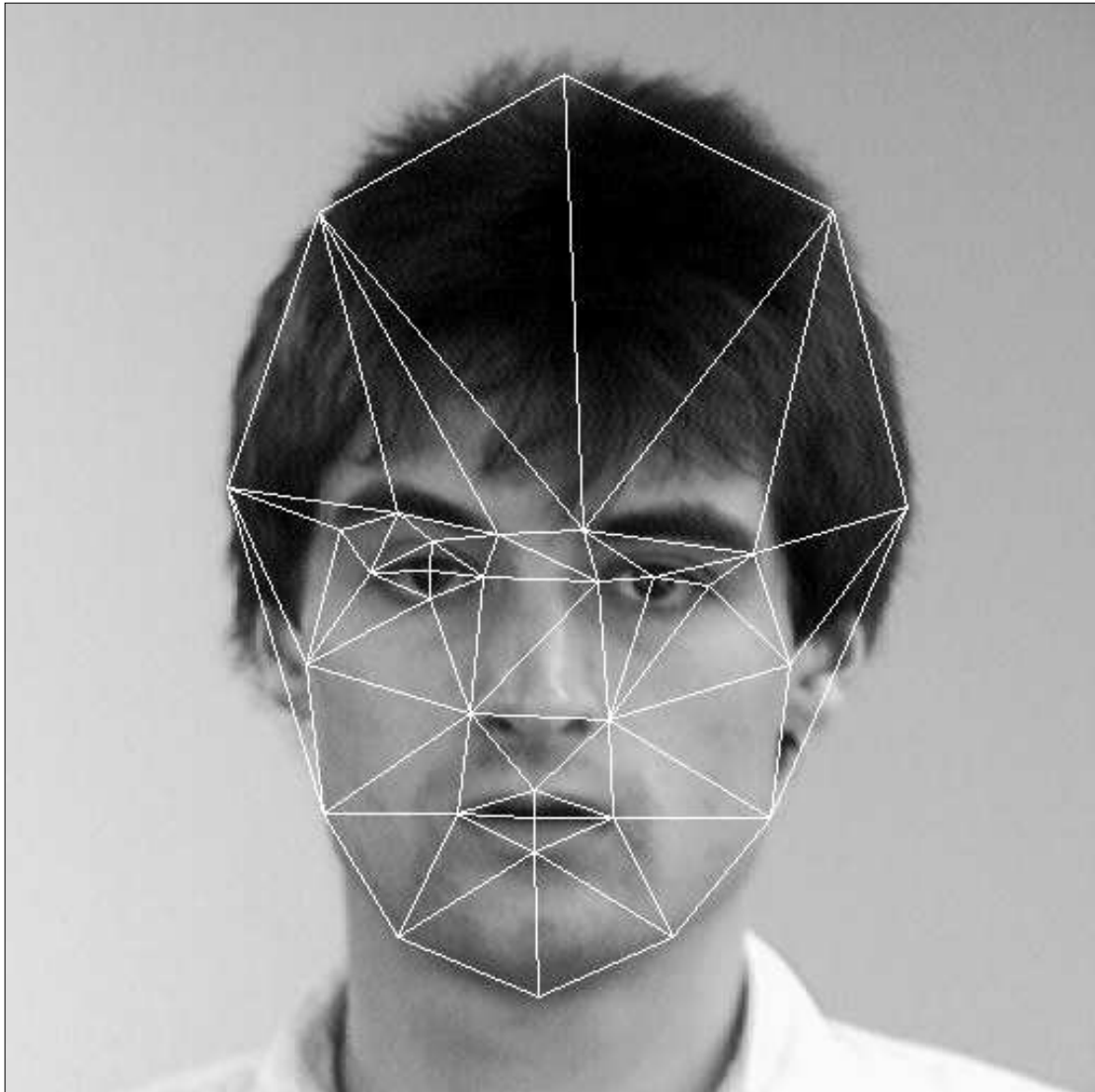
# The Specific Head Model

The generic head provides a 3D description of the shape of a human head, but we would like a description of the head of the specific individual in the photograph—a specific head model. We create this by moving the vertices of the generic triangulation appropriately. Each vertex of the model represents a point on the surface of a human head, and should be moved so that it matches this point when the model is projected onto the plane of the photograph.

Once we have found the positions of some of the points by examining the photograph directly, we build the specific model in three stages. We use these positions to calculate a global transformation which roughly matches the generic model to the photograph. We derive values for the depth of each of these vertices from the transformed model. Finally we interpolate locally to improve the positions of the remaining vertices.

**Finding Points in the Photograph** Each vertex of the model represents a point on the surface of a human head. Some vertices represent visually obvious points such as a corner of the mouth; others are less obvious, but have anatomical significance such as those defining the line of the cheek-bone; and a third group of vertices contribute to the overall shape of the model even though it would be almost impossible to find their exact positions on a real face.

It would be impractical to examine the photograph to find the thousand or more points on the photograph which the vertices of the model represent. Instead we find a smaller collection of the most important or *major* points. The remaining points are not found from the photograph, but by interpolating between the major points.



**Figure 5.1:** A face, showing the Delaunay triangulation of the major points.

Software has been developed which allows a human operator to locate each of the major points on the digitized photograph displayed on a computer screen, using a mouse (a pointing device). Figure 5.1 shows a set of major points found

on a picture of a face, using this software.

Alternatively, and preferably, the major points could be found by a feature-finding system such as those developed by researchers at Aberdeen [43, 3].

## 5.1 Globally Matching the Generic Model to the Photograph

The generic model gives us “world knowledge” of the shape of a human head and the positions of the major points relate the model to a head in a photograph. The specific model is built by combining these.

We would like each vertex of the distorted wire-frame to match the point it represents, when it is projected onto the plane of the photograph.

We apply two affine transformations to the generic model so that it globally matches the head in the photograph, first rotating it so that it is viewed from the same camera position as the head in the photograph, and then scaling and translating the model so that when we project it onto the photograph it lies on top of the head.

**Matching the viewpoint of the camera and the model** Like any vector in 3-space, the vertices of the model have 3 components. If we rotate the model so that it is viewed from the direction in which the camera saw the head, we convert to a new coordinate system. In this coordinate system the  $x$  and  $y$ -axes of the model are aligned with the  $x$  and  $y$ -axes of the photograph, and the  $z$ -axis (the *depth* of the model) is perpendicular to the plane of the photograph.

At present the user specifies the direction of the camera axis and we rotate the model so that this points along the negative  $z$ -axis. For convenience this can be given as the latitude and longitude of the camera, where the top of the head is the “north pole”, and the centre-line of the face is the line of zero longitude. The camera axis does not need to be specified very accurately—a description such

as *full face on*, *three-quarter right view* or *left profile* would allow us to create a realistic model, although it is better to rotate to the exact camera position if this is known.

It may be possible to determine the direction of the camera axis automatically. We will consider a spring-frame model as a way of locally improving the specific head in section 5.3.2. We speculate that the elastic energy remaining in this model at equilibrium may depend on how well we estimate the camera axis direction.

Once the viewpoint of the model and the camera have been aligned the  $x$  and  $y$ -coordinates of each vertex of the model are also the  $x$  and  $y$ -coordinates of its projection onto the photograph.

**Global Transformation in the Plane of the Photograph** We now transform the model in the plane of the photograph so that it projects onto the head in the picture.

We already know the true positions of the major points—found by examining the photograph (as described on page 64). These are the projections of some of the vertices of the model—we shall call these the *major vertices*, and the remainder of the vertices the *minor vertices*.<sup>1</sup> We apply an affine transformation to the  $x$  and  $y$ -coordinates of the model, scaling, translating and possibly rotating it in the plane of the photograph,<sup>2</sup> so that the major vertices project, as nearly as possible, onto the major points. We shall use different scale factors in each direction since some people have longer, thinner faces than others.

If  $\{\mathbf{p}_i = (p_i \ q_i \ 1)^T\}$  and  $\{\mathbf{v}_i = (v_i \ w_i \ 1)^T\}$  are the augmented 2D position vectors of  $n$  major points and  $n$  major vertices, respectively, then the error in

---

<sup>1</sup>Sometimes it will not be possible to find all the major points in the photograph—depending on the hairstyle it can be difficult to find the top of the head, for instance. In our implementation we use a special data value in the input data to mark a major point “not found”. Whenever a major point cannot be found we demote the corresponding major vertex to a minor vertex. In fact the best definition of a major vertex in the implementation is “a vertex for which we can find the position on the photograph”.

<sup>2</sup>A completely general 2D affine transformation would also allow us to *shear* the head, but real people don’t have sheared heads!

approximating the major points by the projections of the major vertices, transformed by a 2D affine transformation  $\mathbf{A}$ , is

$$\epsilon = \sum_{i=1}^n \|\mathbf{p}_i - \mathbf{A} \mathbf{v}_i\|^2 \quad (5.1)$$

To find the parameters for the transformation which gives the least-squares best fit to the major points we minimize  $\epsilon$ .

In normal use the camera and head are always the same way up. This means that we need only to scale and translate the model to match the photograph; in this case the transformation matrix is

$$\mathbf{A} = \begin{pmatrix} s_1 & 0 & t_1 \\ 0 & s_2 & t_2 \\ 0 & 0 & 1 \end{pmatrix}, \quad (5.2)$$

and we find the minimum of  $\epsilon$  for  $s_1$ ,  $t_1$ ,  $s_2$  and  $t_2$  from

$$\frac{\partial \epsilon}{\partial s_1} = 0, \quad \frac{\partial \epsilon}{\partial t_1} = 0, \quad \frac{\partial \epsilon}{\partial s_2} = 0 \quad \text{and} \quad \frac{\partial \epsilon}{\partial t_2} = 0.$$

This gives two independent pairs of equations, with solutions

$$\begin{aligned} s_1 &= \frac{n \sum pv - \sum p \sum v}{n \sum v^2 - (\sum v)^2} & t_1 &= \frac{\sum p - s_1 \sum v}{n} \\ s_2 &= \frac{n \sum qw - \sum q \sum w}{n \sum w^2 - (\sum w)^2} & t_2 &= \frac{\sum q - s_2 \sum w}{n}. \end{aligned}$$

This is just 1D linear regression along each axis.

As already mentioned, in normal use the camera and the head have always been the same way up. If they are not we should also rotate the model in the plane of the photograph. The best rotation is found by using the matrix

$$\mathbf{A} = \begin{pmatrix} s_1 \cos \theta & -s_1 \sin \theta & t_1 \\ s_2 \sin \theta & s_2 \cos \theta & t_2 \\ 0 & 0 & 1 \end{pmatrix}, \quad (5.3)$$

in equation 5.1, in place of the value in equation 5.2.

Now if we try to minimize  $\epsilon$  for  $\theta$  as well as  $s_1$ ,  $t_1$ ,  $s_2$  and  $t_2$  we have to solve five non-linear equations in five unknowns. One convenient way is to minimize for many *fixed* values of  $\theta$ , finding the best value of  $\theta$  by an iterative search.<sup>3</sup>

**Scaling** The parameters  $s_1$  and  $s_2$  in the transformation are the horizontal and vertical scale factors used to enlarge (or shrink) the model to fit the photograph. The depth of the model should be scaled so that the three directions of the model remains in proportion. The scale factor for the depth should be the geometric mean  $\sqrt{|s_1 s_2|}$  of the other two factors.

The 3 dimensional rotation to match the viewpoints, the rotation, scale and translating in the plane of the photograph and the depth scaling can all be combined into a single 3D affine transformation given by a  $4 \times 4$  matrix. This transformation globally matches the generic model to the points found in the photograph, so that the 3 components of each vertex are its horizontal and vertical coordinates in the picture and its depth—the distance from the vertex to the plane of the photo. Figure 5.2 shows the generic head globally transformed to match the major points shown figure 5.1.

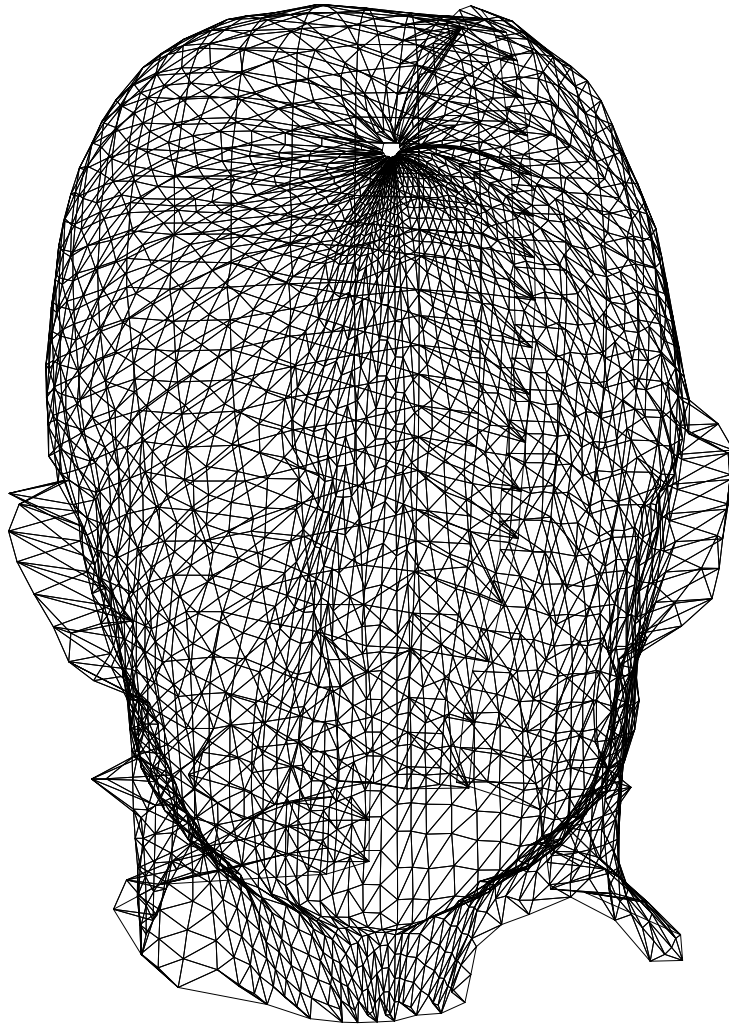
We are now ready to move the vertices locally to improve the fit. The major vertices are moved, parallel to the plane of the photograph, to the major points—the positions we found for them by examining the photograph. We next consider how best to find the depth components of the vertices.

## 5.2 Finding the Depth Components of the Model Vertices

**Depth using the Generic Model** We have found the major points, and hence the horizontal and vertical positions of the major vertices by examining the photograph and will interpolate between these to find the horizontal and vertical

---

<sup>3</sup>I would like to thank David Tock for suggesting this method.



**Figure 5.2:** *The generic head globally transformed to match the major points shown in figure 5.1.*

positions of the minor vertices. We must also find the depth components of all the vertices.

Once the global transformation has been applied to the generic model it is roughly the same size and shape as the head in the photograph. The generic head was introduced especially to give the system knowledge of the shape of human heads. This knowledge will always be available and gives good default values for the depth components of the vertices.

It is likely to fail only when the individual is markedly different from the norm represented by the generic head. For example a front view of a person will give little indication of whether they have an extremely large or small nose. Such a failure would not be catastrophic - instead of modelling John Smith we would end up modelling someone who looked like John Smith, but didn't have such a small nose.

Depth or shape information can be obtained directly from images in two ways:

- shading information in the photograph; and
- binocular or stereo information, if we have more than one photograph of the subject.

It may be possible to use these techniques to improve the depth estimates from the generic model.

**Depth using Shading Information** Pentland[33, 35], Horn[22] and others have shown that it is possible to obtain depth information from the light intensity information in a digitised photograph. Appendix A describes one depth from shading algorithm.

The output from current depth from shading algorithms is recognizable as a depth-map of the scene shown in the intensity image, but it is often of poor quality. Interpreted carefully the output can give qualitatively useful information

about depths in the scene. However the information from the depth map is not accurate enough to improve on the depth estimates given by the generic model.

**Depth using Stereo data** As we will see in chapter 7, two or more images of the head, taken from different viewpoints can improve the final image.

More than one photograph of the person, either a true stereo pair, or just several views from different (known) positions could also be used to derive feature positions in three dimensions. Consider a head in a Cartesian coordinate system with the y-axis upwards and the z-axis pointing in the direction the head is facing. A front view would give the x- and y-coordinates of each point, and a profile would give the y- and z-coordinates of the same points. Combining the two gives the complete position of the point in three dimensions, once we have resolved or made use of any discrepancy in the repeated coordinate. Parke [30] describes how to find these coordinates, allowing for the effect of perspective in each photograph. Such a technique would enable the system to spot deviations from the norm that a single image could not show, such as John Smith's abnormally small nose.

Systems which calculate depth information from stereograms and other carefully calibrated sets of images place a severe restriction on the photographs we could use. They are often very complex, requiring a considerable degree of scene recognition before the images can be matched. Even then I believe the results would add little depth information to that which we could derive from the generic model. It would only be worth the effort if the points were already accurately located on multiple images for the texture information as suggested in chapter 7.

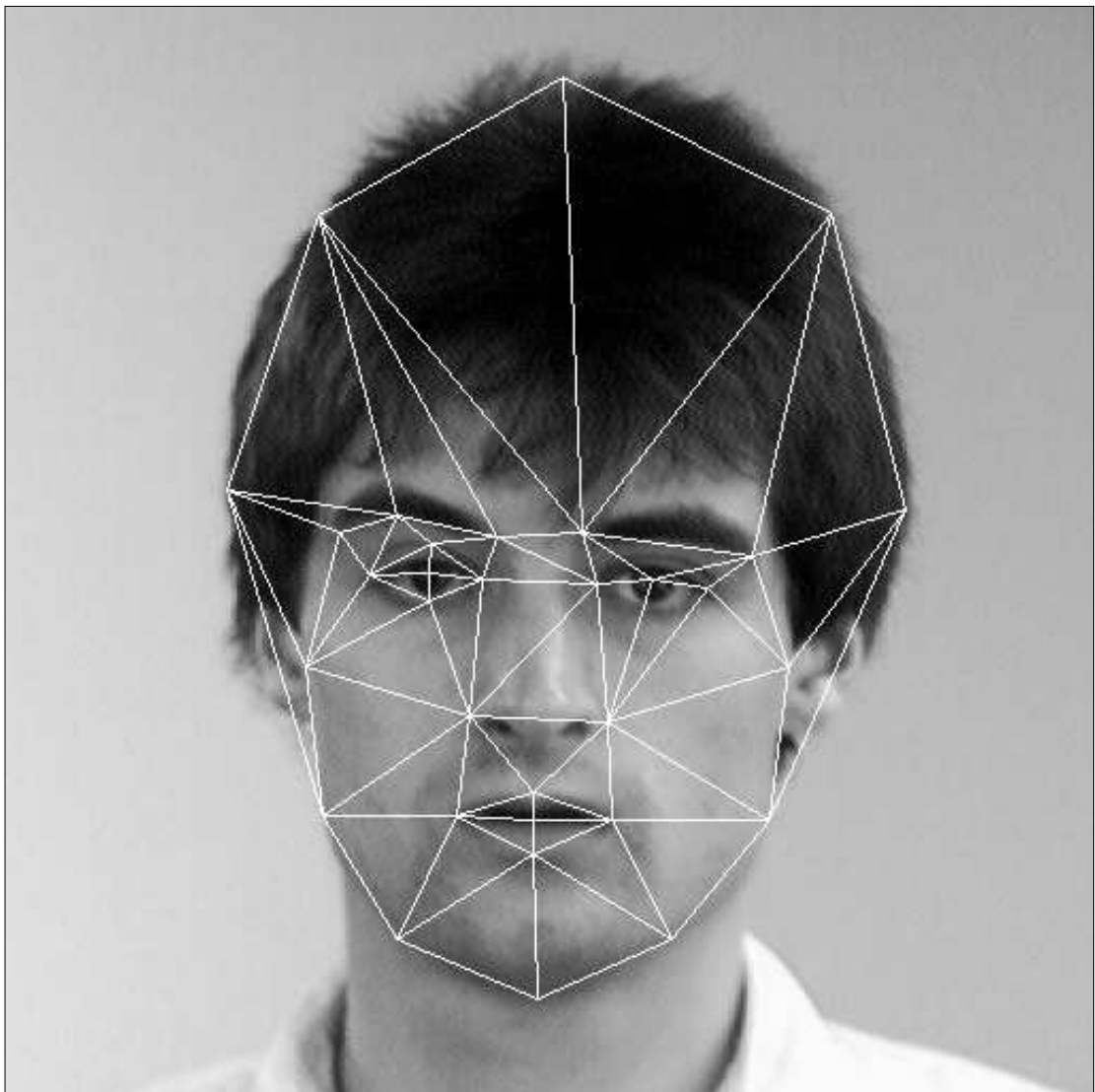
### 5.3 Locally Improving the Match

Once we have found the major vertices of the specific triangulation we can interpolate the positions of the minor vertices.

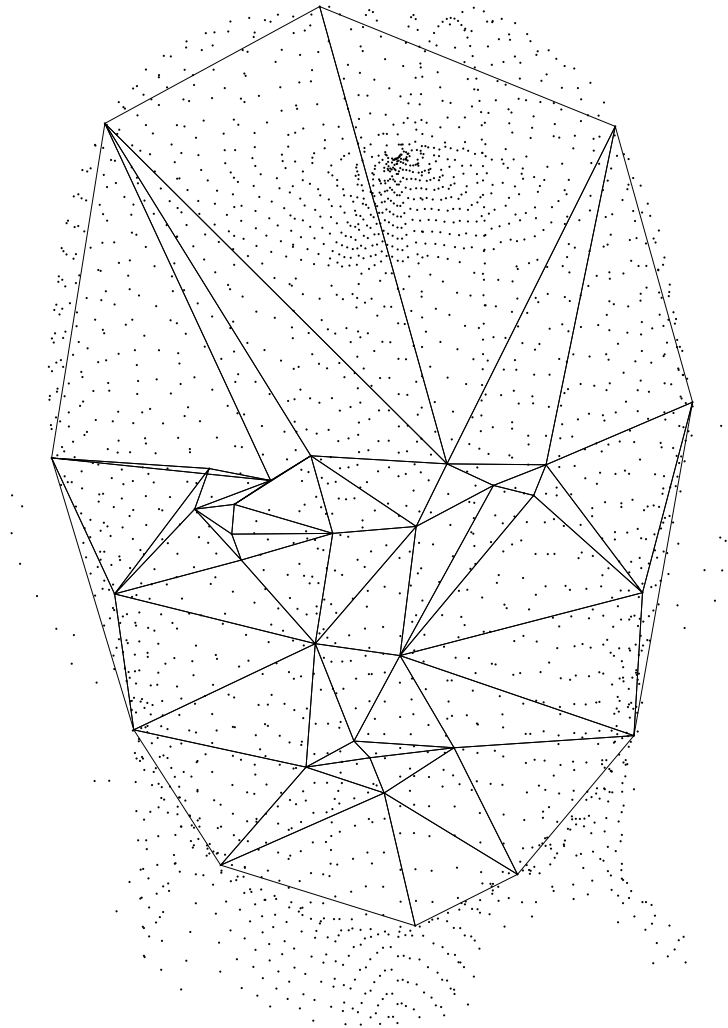
We propose 2 methods of interpolation. The first uses a 2D Delaunay triangu-

lation of the major vertices, and is based on the distortions of chapter 2. Section 5.3.2 describes a second method, which simulates the distortion of the wireframe model as if it were made from springs. The recent work on deformable models, especially by Terzopoulos [41], may also suggest possible approaches.

### 5.3.1 Finding the Minor Vertices: Delaunay Triangulation Method



**Figure 5.3:** *The Delaunay triangulation of the major points. These are the “after” positions of the distortion.*



**Figure 5.4:** *The points of the generic model globally transformed to fit the major points of figure 5.3. These are the “before” positions of the distortion.*

If we use the 2D projections of the globally transformed major vertices (figure 5.4) and their positions on the photograph (figure 5.3) as the before and after positions of a 2D distortion as described in chapter 2, then the distorted positions of the globally transformed minor vertices will be sensible 2D positions for the minor vertices of the specific model. The depth component of each minor vertex is taken directly from the globally transformed model.

If we project the vertices of the model onto the plane of the photograph they divide into three groups:

1. the major vertices
2. the minor vertices lying within the convex hull of the major vertices
3. the minor vertices lying outside the convex hull of the major vertices

The major vertices we have already found.

For the minor vertices within the convex hull (2) we take the depth from the model as before, and find the other two components by interpolating between the major vertices, in the same way that the distortion mapping (chapter 2.3) interpolated the position of each point in the distorted picture. This is done as follows: first find the Delaunay triangulation of the major points (see chapter 2.2). This divides the convex-hull of the major points into triangles whose corners are the major vertices, in such a way that the triangulation is *locally equiangular*. Replacing the major points with the projections of the major vertices gives us another triangulation (which may not be locally equiangular) with triangles corresponding to the triangles of the Delaunay triangulation. Each of the vertices in (2) above projects into at least one of these triangles, and the position of this projection can be uniquely given in terms of the corners of the triangle using *barycentric coordinates*. There is a point on the photograph with the same barycentric coordinates within the corresponding Delaunay triangle. This point gives us the horizontal and vertical components of the desired vertex of the specific model.

We mentioned in chapter 2.4 that the distortion mapping for a 2D distortion must be 1:1 and that all the triangles must be the same way around in the “before” and “after” positions. This is also true when interpolating the positions of the minor vertices—if we tried to move a vertex which lay in a flipped triangle it would move in the wrong direction. Since the “after” triangulation is locally equiangular we might hope that flipped triangles are rare. Unfortunately, the major points found by the feature-finding system result in Delaunay triangulations with long thin triangles. This means that a point does not have to move very far in order to flip a triangle. The triangle above the right eye (left of picture) in figure 5.4 has flipped because the lower edge of the eyebrow is concave upwards on the model, but concave downwards in the photograph (figure 5.3). Also above the right eye of figure 5.4 there is a triangle which has been squashed flat and has almost no area. This is because the middle of the eyebrow is nearer the middle of the face than the top of the eye in the model, but nearer the edge of the face in the photograph. Any sufficiently detailed set of major vertices will include vertices which are to the right of or above another vertex in one face, but to the left of or below in another.

We cannot interpolate the positions of the minor vertices lying outside of the convex hull in this manner because they do not lie in any of the triangles of the Delaunay triangulation, but must rely on the global transformation to give a reasonable estimate of their positions.

### 5.3.2 Finding the Minor Vertices Using a Spring-Frame Triangulation

Section 5.3.1 describes a way of interpolating the positions of the minor vertices based on a two-dimensional Delaunay triangulation of the major vertices. This has problems with flipped triangles, and cannot extrapolate positions for minor vertices outside the convex hull of the major points, so we consider an alternative method. We would like to distort the generic head triangulation into a new shape

which fits the photograph better. We shall assume that the global transformation has been applied to the generic head, so that only local changes remain to be made. The depth coordinate of each vertex is found from the generic model as before, so we only wish to distort the model parallel to the plane of the photograph.

Imagine that the model is made out of springs, so that three springs form the boundary of each triangle. The springs all work in tension and in compression and have the same stiffness  $e$  but each has its own rest length. These rest lengths are chosen so that if no external forces are applied the spring model takes up the shape of the globally transformed model.

We hold the spring model by the major vertices so that they project onto the corresponding major points. Since we do not wish to distort the model perpendicular to the photograph we imagine a constraining force on each minor vertex which opposes any change in its depth. The model is then allowed to reach equilibrium. This least-energy position is the specific model—our final model for the head of the person in the photograph.

We use an iterative, finite element method to simulate the springs as they expand or contract to reach equilibrium. For each vertex  $\mathbf{v}_i$  in turn we use Hook's law to calculate the force  $\mathbf{F}_{ij}$  which each adjacent spring  $s_{ij}$  exerts on it,

$$\mathbf{F}_{ij} = e \cdot \left(1 - \frac{L_{ij}}{\|\mathbf{v}_j - \mathbf{v}_i\|}\right) \cdot (\mathbf{v}_j - \mathbf{v}_i),$$

where  $L_{ij}$  is the rest length of spring  $s_{ij}$  which connects  $\mathbf{v}_i$  to adjacent vertex  $\mathbf{v}_j$ . Since the depth of each vertex is fixed we set the depth component of this force to zero. If  $\mathbf{v}_i$  is a minor vertex we displace it in proportion to the resultant force (the sum of the forces of all the adjacent springs). If it is a major vertex we do not displace it at all. Once we have displaced each minor vertex, we repeat this until the sum of the magnitudes of the displacements in one complete cycle is zero, or close enough to zero to be ignored.

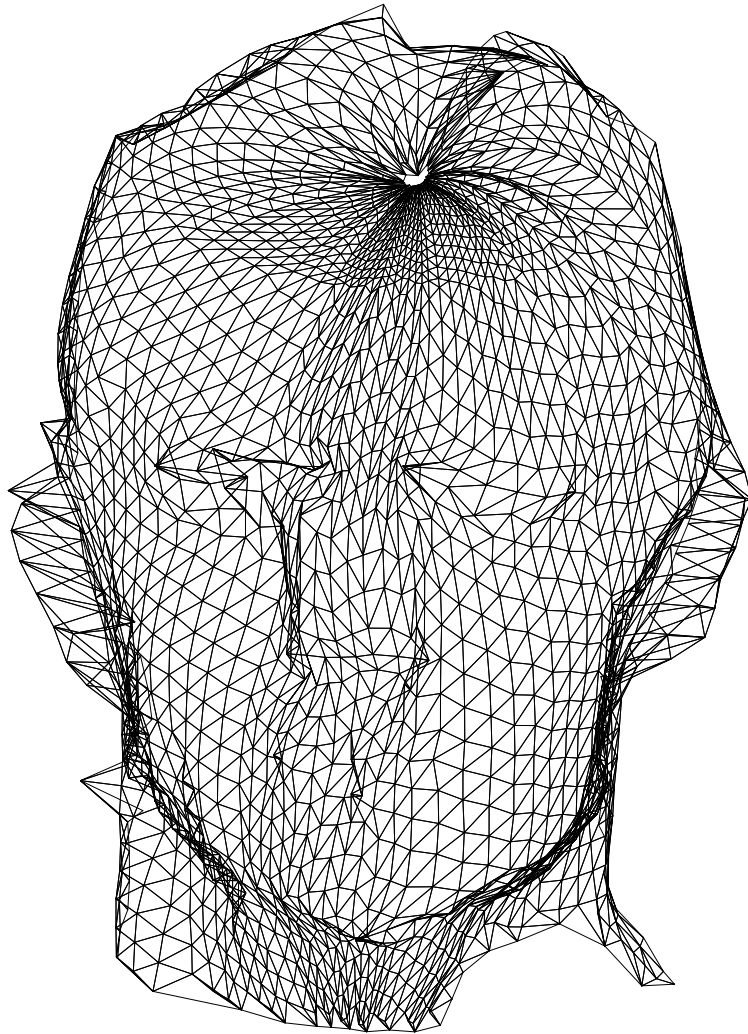
The equilibrium position of the spring model is found using the following iterative algorithm, and figure 5.5 shows the result.

Apply global transformation to model.  
 Calculate rest length of each spring.  
   Spring with ends at vertices  $i$  and  $j$  has rest length  $L_{ij}$   
   where  $\mathbf{v}_i$  is position of vertex  $i$ .  
 Displace major vertices to desired positions on photo.  
**repeat**  
   **for** each minor vertex  $i$   
     Force at  $\mathbf{v}_i$  towards  $\mathbf{v}_j$  caused by connecting spring:  
     
$$\mathbf{F}_{ij} = e \cdot \left(1 - \frac{L_{ij}}{\|\mathbf{v}_j - \mathbf{v}_i\|}\right) \cdot (\mathbf{v}_j - \mathbf{v}_i).$$
  
     Displace vertex  $\mathbf{v}_i$  in proportion to the *total* force acting upon it.  
   **endfor**  
**until** total displacement in one step is sufficiently small.

**Convergence** In a real spring model all the vertices would move at once, but in our simulation they move one at a time. The convergence of the simulation depends on the order in which we move the vertices. Initially only the springs connected to the major vertices are under stress, so we move the vertices adjacent to the major vertices first, then their adjacent vertices, and so on, propagating the stress as we go. For subsequent passes we should start with the springs under greatest load and propagate from here, but this has not been implemented yet.

The current algorithm only converges slowly, requiring many thousand iterations in the worst case. It may be possible to improve convergence by modelling the dynamics of the spring system. At present we move each vertex according to the sum of the forces acting upon it, but in a physical model the forces would change the velocities of various parts of the model. In order to model this we would have to assign mass to each part of the system and use Newton's 2nd Law to accelerate and decelerate each one, iterating until the force and velocity at each part is (sufficiently close to) zero.

As an extension, instead of fixing a major vertex to a particular location we could allow it to travel along a fixed wire. This might be appropriate when we know the *outline* of the head in the photograph but do not know exact positions of the major vertices which lie along it.



**Figure 5.5:** *The equilibrium position of the spring model figure 5.2, transformed to match figure 5.3.*

## 5.4 MBASIC

Since the work of this thesis we have become aware of Japanese work on a Model Based Analysis Synthesis Image Coding System [2]. Aizawa, Harashima and Saito were also deforming a generic face model to fit a photograph of a person. Their generic model appears to have been hand built, and all its vertices lie in equidistant horizontal planes. They determined the global transformation from four points of the face: the bottom of the chin, the skull outline each side of the eyes, and the midpoint of the face between the eyebrows. Like us, they also scaled the depth to keep it in proportion with the transformed face.

They find the outline of the lower face (the chin and jaw) by searching for the greatest intensity changes in each row of the picture. The vertices on the lower outline of the model are moved so that they lie on this outline. The other vertices of the lower face are moved towards or away from the central axis, in proportion to the displacements of the outline vertices on the same horizontal plane.

## 5.5 Conclusion

We can now transform the generic head into a specific head model with vertices which project correctly onto the photograph. To do this we first find the positions on the photograph of a small number of the vertices. We then apply an affine transformation to the generic head, so that it globally matches these positions. We then improve the fit by moving the major vertices to these known positions. The remaining vertices move with these, as if the edges of the model were springs.

# Chapter 6

## Displaying the Head

We have a photograph of an individual and a triangulated wireframe model of their head (the specific model, created from the generic model, as described in chapter 5). We would like to combine these to create a new and novel view of the person.

### 6.1 Expression

We might wish to change the facial expression in the new view—in fact an animated *sequence* of views would not be realistic unless the expression changed, however slightly, between each frame.

Yau and Duffy and Aizawa, Harashima and Saito (the MBASIC group) both considered a clip and paste method which combines subpictures of the same face with different expressions. This is described at greater length on page 8.

Yau and Duffy and the MBASIC group both concluded that a better idea would be to use the facial action coding system FACS, described in chapter 1.2. This has been incorporated into computer models of heads (without texture-mapping) by Platt and Badler, and by Keith Waters.

To do this we must define the result of applying each action unit from FACS

to each vertex of the model. Any facial expression can be uniquely described by specifying the activation strength of each AU. Thus we create the desired expression on the model by activating each AU by the specified amount.

Since the generic and specific models have the same vertices we would only need to define the AUs for the generic model—they would be transformed automatically as part of the generic→specific transformation. In this way we could vary the expression on any face, in contrast to previous work, which has to be re-implemented for each new face. This would allow us to create views with novel and realistic expressions. Of course, since the new image is created from a single photograph and a set of averages, it is possible that someone who knows the individual well will say “But Johnny never smiles as much as that”. To solve this one we would need information on what expressions were characteristic of the individual.

If we are to use FACS we must have a neutral starting point. It is very unlikely that the photograph will be of a neutral face, so we must remove the original expression from the specific model before adding the new one. This is trivial *if* we can determine the original expression. Platt and Badler [36] believed that determining the expression would be straight forward, but ten years later no success has been reported. One way would be to find the facial features and analyze their relative positions. As we have seen in chapter 5, feature location is beginning to be reliable, so I don’t believe that determining the expression would be impossible.

## 6.2 Display

In chapter 5 we created a head model which was designed to project nicely onto the photograph, so that each feature projected onto the picture of it. Logically we reverse this and project the photograph back onto the 3-dimensional specific model, which we have covered with a rubber sheet to act as a projection screen.

The colouring and shading of the surface—the *texture*—of the head in the photograph now appear on the model. Once we have distorted the model to change its expression as described in section 6.1, we rotate the model and project it onto the viewing plane to create the final image.

We simulate this double projection by copying the texture one triangle of the head model at a time, just as we did when we distorted an image in two dimensions (chapter 2.5).

The “before” positions of the corners of the triangles are the positions of the vertices on the photograph, so they are the horizontal and vertical components of the positions of the vertices of the specific model.

We apply a viewing transformation to the specific model (with its expression changed, if desired) to view it from the desired viewpoint. This viewing transformation is implemented by (pre-)multiplying each vertex of the model by a matrix. The horizontal and vertical components of these transformed positions are the “after” positions of the corners of the triangles. We now have two triangles, one on the photograph and one on the viewing plane. In principle we can now use the triangle distortion method from chapter 2.5.

However the head has a back and a front, and is not convex, so whatever viewpoint we take, some part of the surface will be obscured by some other part. Whole triangles around the far side and perhaps in other places too, will be hidden. We wish only to texture-map (or at least appear only to texture-map) that part of the surface which will be visible in the final image, avoiding texturing the hidden surfaces. This is the hidden surface problem, and is described by Foley and van Dam [19, chapter 15]. Equally, although we can project vertices of the model back onto the photograph to find the position of any part of the model’s surface, the photograph only shows the surface nearest to the camera.

**Hidden surface removal** Much of the hidden surface points away from the viewer—if the obscuring surface was removed we would see the surface from the

inside. If each triangle was defined in a clockwise manner, as viewed from the outside of the head, we could immediately exclude all the triangles which appear anti-clockwise in the final view, as we must be looking at them from the inside. We determine which side of the triangle we are looking at as follows. If  $\mathbf{c}_0$ ,  $\mathbf{c}_1$  and  $\mathbf{c}_2$  are the position vectors of the corners of the triangle in a right-handed coordinate system, then  $\mathbf{n} = (\mathbf{c}_2 - \mathbf{c}_0) \times (\mathbf{c}_1 - \mathbf{c}_0)$  is a vector normal to the triangle, pointing towards an observer who sees the triangle  $\mathbf{c}_0\mathbf{c}_1\mathbf{c}_2$  as clockwise. If the dot product of  $\mathbf{n}$  with the viewing direction vector  $\mathbf{v}$  is zero then we are looking at the triangle edge on, if  $\mathbf{n} \cdot \mathbf{v} > 0$  we are looking at the triangle from the inside, and if  $\mathbf{n} \cdot \mathbf{v} < 0$  we are looking at it from the outside.

This has not actually been implemented because we have attempted to retain compatibility with data for two existing display packages with different conventions for the coordinate axes—one is right handed and the other left handed. This means that it is impossible to know which triangles are the right way around. By omitting this test we draw twice as many triangles, but the main hidden surface removal algorithm, which we will describe next, ensures that these triangles are not visible in the final image.

Since a face is concave there will often be triangles which do face the right way but are obscured, either fully or partially, by nearer triangles. We wish to remove these hidden surfaces. Early hidden surface removal was constrained by the limited amount of memory available—often it was only possible to store a few lines of the image being created, the rest being relatively inaccessible, either on disk or on a separate display device. Much effort was spent sorting the triangles by their depths so that only the nearest triangle was drawn, or at least ensuring that it was drawn last. Yau used Watkins scan-line algorithm [50] to remove the hidden surfaces of his faces. This considers the image a line at a time, finding the triangles which appear in the line and sorting them to find out which one is visible in each pixel. Since the same triangle is usually visible in adjacent pixels the process can be made quite efficient, but the full algorithm is complex.

We have no such restrictions on memory usage, so have used a much simpler technique which uses a z-buffer. This is a depth-map in which we store the depth of the surface as each pixel is written. Before we texture-map each pixel we compare the depth of the current triangle with the depth already stored in the z-buffer, and only re-colour the pixel if the current triangle is nearer.

The time taken by the z-buffer is proportional to the total area of the triangles, whereas the time taken by the scan-line algorithm increases with the number of triangles. Foley and van Dam [19, table 15.1] report work which suggests that the z-buffer takes more than twice as long as the scan-line algorithm to draw 2500 triangles, but only half as long to draw 60 000 triangles. Our original aim was a model with 1000 triangles, chosen as a reasonable compromise between picture quality and speed of drawing on a Sun 3/160c. Three years later we have a Sun SPARCstation 2 which creates images about sixteen times as quickly, taking around 29 seconds to texture map a model with 4885 triangles, where the old machine took almost eight minutes. It has proved difficult to reduce the detail in model heads, so that they have only 1000 triangles, and as measuring technology improves model heads with ever more detail are becoming available—the latest heads from UCL have over a hundred thousand triangles. While more detailed models cannot improve the texture mapped image unless extra points can be found on the photograph, it seems that models will continue to grow more detailed as the technology allows. We believe therefore that the speed advantage is likely to shift from the scan-line algorithm to the z-buffer.

**Hidden texture** When the surface we wish to draw is hidden from the view of the camera, we will have no texture to map onto the new image. Currently a neutral colour is assumed for the texture of such pixels, but our intention is that by using two or more photographs we will be able to find texture for each part of the new view. Chapter 7 explores this idea.

Just as we ensured that the texture in the final image always came from the nearest triangle, we must ensure that the texture we wish to copy onto a triangle

is indeed visible to the camera and not blocked from view. This is a new problem. To the best of my knowledge, in all previous work not only is the relationship between the texture and the surface of the model defined in advance by human, but also texture is associated with every part of the model.

Just as we used a depth map of the model projected onto the viewing plane (the z-buffer) to solve the hidden surface problem, we shall use a depth map of the model projected onto the plane of the photograph to resolve the problem: “can we see this part of the surface in the photograph ?” This requires a new preliminary step: we draw the model as it would appear from the camera view-point, a triangle at a time, but instead of calculating a colour or texture for each pixel we just fill in the z-buffer. This gives us a second, *texture* depth map, with each cell giving the depth of the head at the corresponding pixel of the photograph.

As before, we scan each pixel in the new image of each triangle, but now we interpolate between the corners of the triangle to give us the depth component of the pixel. We find the pre-image of the pixel by considering barycentric coordinates, but now there is a depth component too. Just like the 2D positions, the depth components can be calculated incrementally, adding a constant displacement each time we step to an adjacent pixel. We are now ready to texture map the head in 3D.

If the new pixel is further away than the pixel already in the depth map we can ignore it since it is a hidden pixel, but if it is nearer then it is visible in the final image and we must draw it.

If the pre-image depth is comparable with the texture depth map the camera could see the pixel and we can use the texture. The pre-image of a pixel may lie between pixels of the photograph and, since the triangle may be tilted at a steep angle to the camera, the depth may change considerably between adjacent pixels. We therefore compare the pre-image depth with a 3x3 grid of adjacent cells in the texture depth map, rather than a single value.

If the depth map is much closer than the current triangle the desired texture is

hidden, so we fill the pixel with a neutral intensity value which does not stand out from those around it. Occasionally it may be possible to replace this neutral value by interpolating between nearby pixels which were visible to the camera, perhaps as a post-processing stage. An alternative where there are large hidden areas is to use texture from a second picture, as described in chapter 7.

It is also possible for the new pixel to be much nearer than the texture—this is an artifact of discretisation. When we created the depth map we used the pixels of each triangle, but when we come to texture map, we use the pre-images of the pixels in the new triangle. Sometimes the pre-image of a pixel at the edge of a new triangle lies just outside the pre-image of the triangle. When we look at the pre-image of the pixel in the depth map the depth will come from some other triangle, or none at all, so the depth may be very different from the interpolated value.

**Illumination** When the positions of an object, a viewer and a light source change relative to one another the observed illumination will change. We will discuss this further in chapter 7, where we have more than one texture image.

### 6.3 Results

We are now ready to show the texture mapped specific model. Figures 6.1 and 6.2 show faces texture mapped onto specific head models built from the generic head model of figure 5.2 and the positions of the 32 major vertices shown in figure 5.3.

Several parts of one or both heads have been coloured using a neutral tone, since they were out of view of the camera. These include the back half of the head, the inside of the ear, the underside of the chin, and the part of the face obscured by flared nostrils. The side of the cheek and the bridge on the nose are (close to) perpendicular to the camera, so they too are obscured in places.

Some of the background has mistakenly been used as texture, since the outline

of the specific model doesn't match the outline in the photograph everywhere.

Note that both nostrils are visible even when the head is rotated through  $45^\circ$ . Because the three parts of the base of Vicki's nose are equally prominent, rather than the central septum protruding lower than the sides, as is often the case, a parallel projection shows Vicki's further nostril. Because we have no other information, the specific models have inherited this characteristic of Vicki's nose.



**Figure 6.1:** *The author rotated 4 times. The face in figure 5.3 has been mapped onto the specific model of figure 5.5. The painted model is display as if viewed from the original camera position and rotated through  $15^\circ$ ,  $30^\circ$  and  $45^\circ$ .*



**Figure 6.2:** *The face shown on page 35, rotated using the detailed model. Once the entire system was built, this picture was created from scratch in a single afternoon, even allowing the spring frame method 10 thousand iterations to build the specific model.*

## Chapter 7

# Using Texture From Several Images

In chapter 6 I described how to texture map a single photograph of a person onto the specific model to create the new image. The original picture which provides the texture information, and the image we wish to create are two views of the same head. Unless the viewpoints are the same there will be parts of the surface which are visible in the created image but not in the original. When we try to texture map these parts of the surface there will be no texture information to use. In chapter 6, I mentioned ways of determining suitable texture information by interpolating or by using neutral default values. There is however an alternative.

Consider what we could do if we had a second picture, taken from a different viewpoint from the first. This picture will show part of the head which was previously hidden, giving us texture information about more of the surface. Now when we generate an image from a viewpoint between these two we do have texture information for the whole of the visible surface, so we can texture map every pixel. We can go further. Several pictures will give the texture information from even more of the surface and we can texture map complete images from a whole range of viewpoints.

## 7.1 Experiments Using More Than One Picture of an Object

I wanted to know what sort of difficulties would occur when I used texture information from more than one picture, so I experimented. At the time I didn't have a suitable model of a head, and wanted to use an object that wouldn't move, so that I could take pictures at different times without worrying that the object might have changed. I chose a glazed, patterned coffee mug which was lying around the office, thinking that the cylindrical body would be simple to model and the handle could be hidden round the back, or modelled approximately.

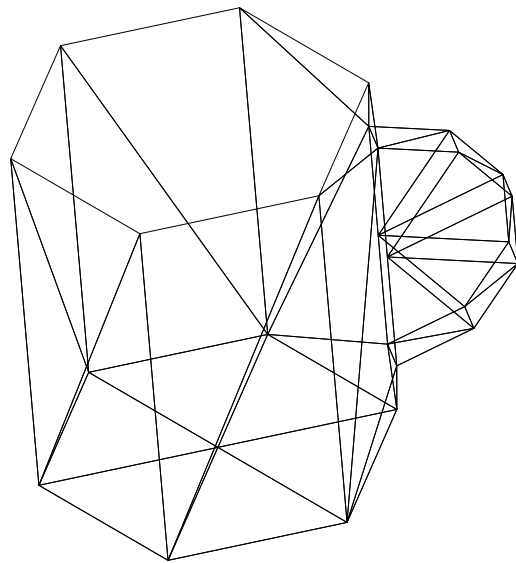
Using a simple object for this work highlighted some of the pitfalls and allowed me to discover solutions at a time when I had no suitable head model.

## 7.2 The Coffee Mug Model

We need a model of the mug to map the texture onto. Just like the heads, I built a generic model coffee mug and transformed it to form a specific model that projected correctly onto one of the images.

As figure 7.1 shows, the generic model was a hexagonal cylinder, a base and a simple handle. The faces are all triangles since the texture mapper was not designed to work with more complex polygons. As any schoolboy topologist will tell you, a coffee mug is like a doughnut with a hole in the middle. There is no similar hole in a head so I felt that it wasn't necessary to model the hole in the coffee mug. I just pretended that the handle was solid.

Since the specific mug was created using just one of the images, when we recreate the view of the mug as seen in any of the other images, it does not appear exactly the same shape as in the true image.



**Figure 7.1:** *The generic coffee mug model*

### 7.3 Taking the Pictures

The digitizer described on page 12 was able to digitise and store an image from each of three video inputs one after another. I had access to two cameras and intended to use both, but found that the lenses were very different. In particular the ‘pin-cushion’ effect was very noticeable on one of them. Rather than trying to use software to compensate for the differences between lenses I settled for using just one camera.

Taking pictures of an object from several viewpoints with only one camera means either moving the camera or rotating the object. I wanted to be able easily to return the camera and object to the positions of previous images for repeatability. The simplest way of doing this was to fix the camera and mount the mug on a rotating platform.<sup>1</sup> Another way might be to arrange a mirror so that the two views appear within the field of view.

---

<sup>1</sup>A cake icing stand

## 7.4 Combining Several Sources of Texture

Texture mapping from a single image is described in chapter 6, but a brief outline is given here as a reminder. The new image is created by considering each triangle of the model in turn, and each pixel within each triangle. Each pixel represents a small region on the surface of the object and, provided that the triangle is visible in the texture image, each of these regions corresponds to a group of pixels in the texture image. The intensity of each new pixel is set to the average value of the corresponding group of pixels in the texture image.<sup>2</sup>

When we have more than one texture image many of the triangles will appear in more than one texture image, giving us several possible values for the pixel intensities - one from each texture image. Some images will have better views of the region than others. If we could determine which was the best view we could use it for texture.

A region of the object will be best seen in an image which was taken when the camera was pointing at right angles to the surface. A camera close to the region but pointing at an angle to it may give a better view than one looking straight at the region from a distance. These two factors can be combined; assuming that the images are equally clear, the best image of a region is the one in which it appears largest. Different regions of the surface point in different directions, so different images will give the best views of different regions.

Since the texture mapping uses a linear transformation we effectively assume that each triangle of the model represents a flat face of the object. If this is so, the best view of each subregion of the triangle is the same, and we only need to find the best view for each triangle—it is the same for all the pixels (provided that the whole of the triangle is visible in this image).

For each triangle we use the texture from the image with the best view. It would also be possible to combine the texture information for each pixel by weight-

---

<sup>2</sup>This assumes that anti-aliasing is implemented. If not, the value is taken to be the value of just one of the group of pixels.

ing the texture information according to how good each view of the triangle is.

### 7.4.1 Matching Points

The major vertices must be located in each image, so that we know where the texture from each triangle comes from. When two adjacent triangles are texture mapped using different images it is important that the vertices are located on exactly the same part of the object in each image, otherwise the pattern may be broken across the triangles, rather like a badly papered wall. This was particularly visible on the lettering on the mug, and can be seen on the bottom ‘k’ in the bottom left picture of figure 7.3. even though the vertices have been relocated manually several times in an attempt to remove the discontinuity. Pattern breaks are unlikely to be a problem on areas of a face such as the cheeks where there are no sharp colour changes, but could be a problem in other areas, especially around the eyes.

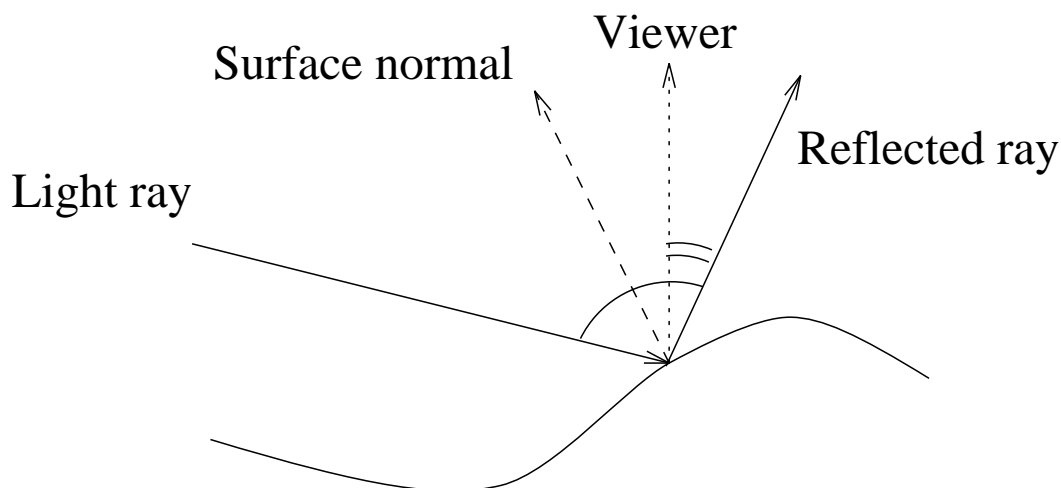
## 7.5 Illumination

In the bottom left picture of figure 7.3 the mug becomes slightly lighter about two third of the way from the left edge, where the texture source changes. When we rotated the mug to take the second picture, this part of the mug faced more directly towards the light, allowing more light to fall onto it. If we had kept the mug and the lights still, but moved the camera to get the second view this would not have happened. Another lighting problem which the experiment with the mug brought to my attention was that of highlighting. This cannot be solved as simply as the previous problem.

If we take two pictures with the object and camera in exactly the same place but different lighting then the pictures will be slightly different. Several models of illumination are described in some detail in appendix A. We will consider only one here.

### 7.5.1 Specular Reflection and Highlights

Specular Reflection is often seen in objects such as apples and billiard balls, and occurs in all objects which have a shiny surface. It is similar to the true reflection of a mirror, but if the shiny surfaces are not completely smooth some of the light is scattered, and reflections of objects appear expanded and blurred so that the reflected object is no longer recognizable. Because much of the light is scattered, only the reflections of bright objects such as lights are noticeable. Such reflections are called *highlights*.



**Figure 7.2:** *Specular Reflection*

The model of specular reflection proposed by Bui Tong Phong [11] is widely used in computer graphics. In this model specular reflection due to a point light source at infinity gives a contribution  $I_p$  to the the observed light intensity at a point on a surface, where:

$$I_p = W(\theta) \cdot \cos^n \alpha \quad (7.1)$$

$\theta$  is the angle of incidence, and  $\alpha$  is the angle between the the viewing direction and the reflected ray. The constant  $n$  depends on the surface; values are typically in the range 1–200; for a perfect reflector  $n$  is infinite. The function  $W$  also depends on the surface—in practise it is often set to a constant chosen to produce pleasing results. Although empirically derived, this model gives realistic images

under normal viewing conditions.<sup>3</sup>

Under normal conditions white faces don't show highlights, but a bald head may be shiny. Pictures of pop stars and sportsmen, especially boxers, in action often show considerable highlighting. Lee [25, 26] found that the reflectance of skin was comparable to that of semi-gloss paper.

Unlike other illumination effects, the position of highlights depends upon the position of the observer—the highlight is strongest at points on the surface where the ray from the light source is reflected into the camera. This means that even if two images of an object are taken from different positions by identical cameras at the same time, the observed intensities at certain points on the object will be different in the two views. Since the texture includes any highlights present, two texture images taken from different viewpoints will give conflicting highlights. In fact a single texture image will give the wrong highlight information for an image viewed from another position. The obvious solution is to remove the highlights from each image, texture map and then add the correct highlights for the new image. Provided that we know the positions of the light sources in the original images we can predict the highlights in these images and remove them to give us texture sources without highlights.

## 7.6 Conclusions

The images of a coffee mug created from a generic model and two images, are not good enough to convince a sceptical observer, principally because the specific models were not mug shaped. Since a coffee mug is basically cylindrical it would be far better to model it using a cylinder instead of a many-sided prism. However the images are good enough to make an observer critical, invoking the effect Parke

---

<sup>3</sup>It is not correct under all conditions however. When the angle of incidence is large (around or above 70°) the observed highlight is strongest at a slightly different angle [19, pp 577–580]. The last word on lighting models appears to be the Torrance–Sparrow illumination model [45, 44], which considers the surface to be made up of many perfectly reflecting microfacets pointing in different directions, and allows for self shadowing and other details.

noted (the better the image the more critical the observer). We are, however, interested in the method, not the coffee mugs, and the following points should be learnt from the experiment.

It is important to match the locations the major vertices in the same positions in each image, to avoid pattern discontinuities in the texture.

Ideally, all texture images should be taken under identical illumination. In particular it is better to move the camera than the object. Even then, highlights may have to be removed from each texture image, and added back into the synthesised image.



**Figure 7.3:** Two views of the coffee-mug and reconstructions from the intermediate and the second viewpoint.

## Chapter 8

### Some Applications

We have shown how we can rotate faces in photographs, to create novel views of the subjects, but there are other uses for 3D texture-mapped heads.



**Figure 8.1:** *A composite of a mother (top) and her daughter (bottom).*

We hinted in chapter 2 that it is possible to merge pictures of two different

people. The simplest merges are created by cutting out part of one face and pasting it onto another. With some skill and ingenuity it is possible to make the join almost invisible, as figure 8.1 (which was created with the aid of our software) shows.

## 8.1 Average Faces and Face Merges

Another sort of composite image dates back to the early days of photography. If two people are photographed on the same plate, taking care that the faces are the same size and in the same position, the result is a somewhat blurred picture of a face. Two or more digital images can be combined in this fashion, by taking the mean of the intensity values at each pixel. Figures 8.2 & 8.3 show examples.



**Figure 8.2:** *The average of 2 faces. The originals are shown in figure 8.4.*



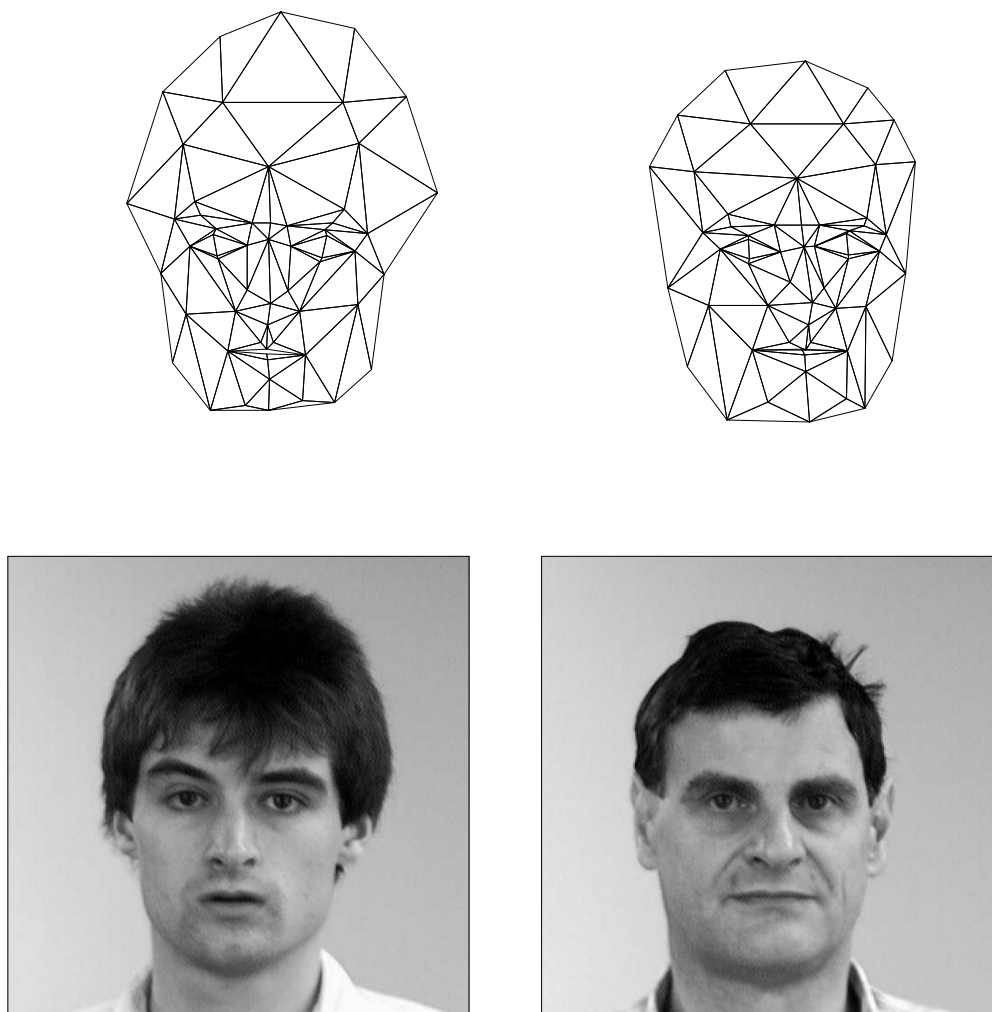
**Figure 8.3:** *The average of 30 faces. The eyes are clear because the images were taken with the eyes in the same place on each picture.*

Such images are blurred because, although the faces have been aligned, the individual features have not.

Since the significant points of each feature are vertices of the generic model,

we can use the 3D model to align the features of each face.

We will now describe how to make a better average of the two faces in figure 8.4, by texture-mapping each image onto the average of the models then averaging the two images. This gives an image of a face whose shape and shading are each the average of the given faces.



**Figure 8.4:** *Original images and models*

We use *Candide* as the generic model, but the process would work equally with a larger generic model. Since *Candide* has only 76 vertices it is not unreasonably laborious to find all of the vertices in each image by hand. This simplifies the generic→specific transformation, since there are no minor vertices, but it is not

an essential step.

We then build the specific models, one for each face, noting that the right hand image is not full-on, but rotated slightly to the camera's right. We allow for this by rotating candid into the same position before applying the generic  $\rightarrow$  specific transformation and then rotating it back to full-on afterwards. This ensures that the two models are the same way up and looking in the same direction, essential since we wish to find the average head shape.

Since the specific models for the two images are derived from the same generic model they have the same topology and we can calculate their mean simply by taking the mean position of each vertex in the two models. Since the models are the same way up and looking in the same direction, this mean model is a sensible average of the two heads. This may seem obvious, but consider the mean of two models which are identical except that one has been rotated through  $180^\circ$ —every vertex lies on the axis of rotation !

Once we have formed the average model we can map the texture from each image onto it. We could use the texture from both sources simultaneously, but instead we create two images of the average model from the same viewpoint, one for each texture source. The faces in these images have the texture of one of the originals and the *shape* of the average head. Since the features are now aligned we can average the two images to get a picture which shows a clear, non-blurred, picture (shown in figure 8.5) of a theoretical person whose shape and texture are the average of the two originals.

The better a picture of a face the more critical of it we are, but there is a sense in which figure 8.5 is a real face and figure 8.2 is not — we would expect an unsuspecting person to claim that figure 8.5 was a photograph of a real person.

Now that we have a picture of the mean of two faces, why not create a sequence of images which starts as person A and ends as person B? The second image could be of a face which is 10% B and 90% A, the third 20% B and 70% A, and so on until the last two images which are 90% and 100% B. Figure 8.6 shows such a

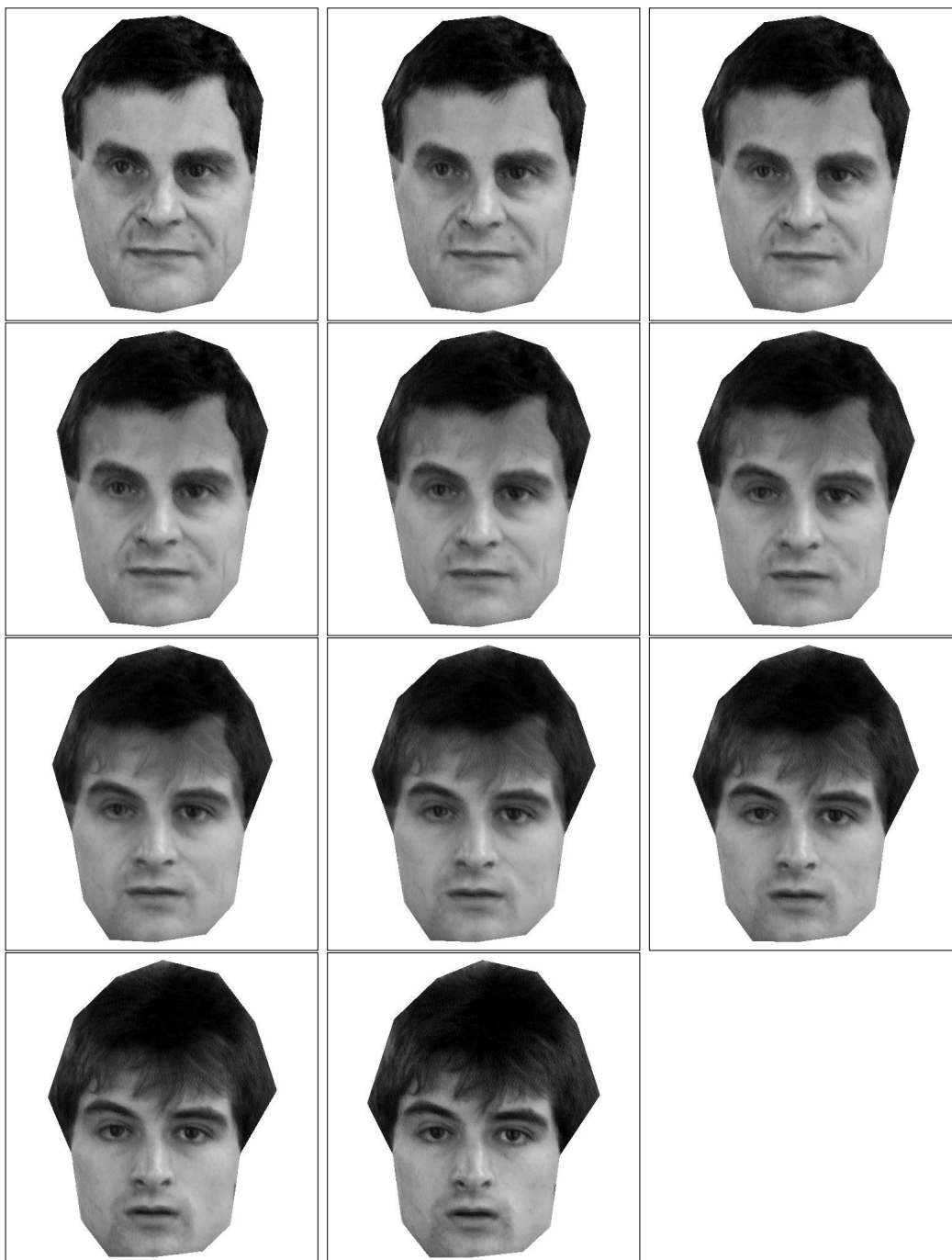


**Figure 8.5:** *Is this a real person?*

sequence. When these images are shown one after the other the effect is of one person changing into another.

This effect has been attempted many times in science fiction films and television programmes, for example whenever a new actor takes over the title role in the long-running BBC television series *Dr. Who* [52, 51]. However I have never before seen a transformation in which the intermediate images appear to be photographs of real people.

This sequence suggested that it may be possible to linearize the space of all



**Figure 8.6:** *Changing faces*

pictures of faces.

## 8.2 Standardized Faces for Principal Component Analysis

Principal component analysis (PCA) has been used to reconstruct and to recognise images of faces, with some limited success. Recently with Craw and Cameron [14] we have improved the results of PCA by mapping the images onto a 3D head model as a preprocessing step.

### 8.2.1 Principal component analysis of face images

In the natural sciences it is common to make many measurements of a set of objects and then seek to reduce the data. Often the measurements are highly correlated and it is possible to reduce the data to a small number of variables which still encapsulate the full variability of the original objects. PCA models this process by considering each object as a point in an  $N$ -dimensional linear space. If we measure  $k$  objects these points will span a subspace of this linear space; this subspace is known as object space. If the  $N$  measurements are independent then the object space will have dimension  $\min\{N,k\}$ . If, however, the measurements are redundant, with some combination of variables predicting the value of others, then the dimension of object space may be reduced.

PCA seeks to capture this redundancy, producing a set of axes—transformed variables—for object space. Measurement errors mean that the dimensionality is not reduced, instead the variability is concentrated in certain directions, and negligible in others. PCA returns the axes in order of explanatory power, the greatest first, finishing with the axes which have so little explanatory power that it is lost in noise. These axes are the ( $N$ -dimensional) eigenvectors of the covariance matrix of the measurements, and the magnitude of the eigenvalues give the

corresponding explanatory powers. The eigenvalue problem can also be solved for  $k \times k$  matrices, a valuable saving if as described below we have a small number ( $k \sim 100$ ) of large data sets ( $N \sim 16384$ ).

We apply PCA to images by considering each pixel as a separate variable, thus each image is represented by, for example, a 16384-tuple of intensities. If we chose 100 images at random they will span a subspace of dimension 100, but if the images are real images which are related in some way, perhaps all faces, then the dimension of this subspace will be less than 100, and the axes of this “face subspace” will capture the essence of a “face”.

Sirovich and Kirby [39, 23] used this technique to study the dimension of this “face-space”. Working with an ensemble of 100 faces and extracting a central cameo of approximately 4000 pixels they found that the first 50 axes, or eigenfaces accounted for 95% of the variability. In addition they used the axes to represent faces not in the original ensemble, using very little information. This technique has also been used by Turk and Pentland [46] to recognize faces of people in the initial ensemble.

The model underlying PCA is a linear space model—it assumes that objects lie in a lower dimensional subspace of the space of measurements. The theory of PCA is only applicable if the objects of interest (in our case faces) form a linear subspace of images. While it is reasonable to argue that the sum of two images is another image, as figure 8.2 shows the sum (or average) of two face images is not in general a face image; at best it will be fuzzy, and it will be much worse than that if the two faces combined are of noticeably different sizes or positions.

This means that in published work faces have been very carefully normalised before being subjected to PCA. This ensures (or attempts to ensure) that the sum of two faces *is* still a face, so that the theory is useful. Sirovich and Kirby arranged their images so that the axis of symmetry and the horizontal line through the eyes were always aligned. The field depth was adjusted to keep a constant facial width and a correction was applied to simulate uniform illumination. Turk and Pentland

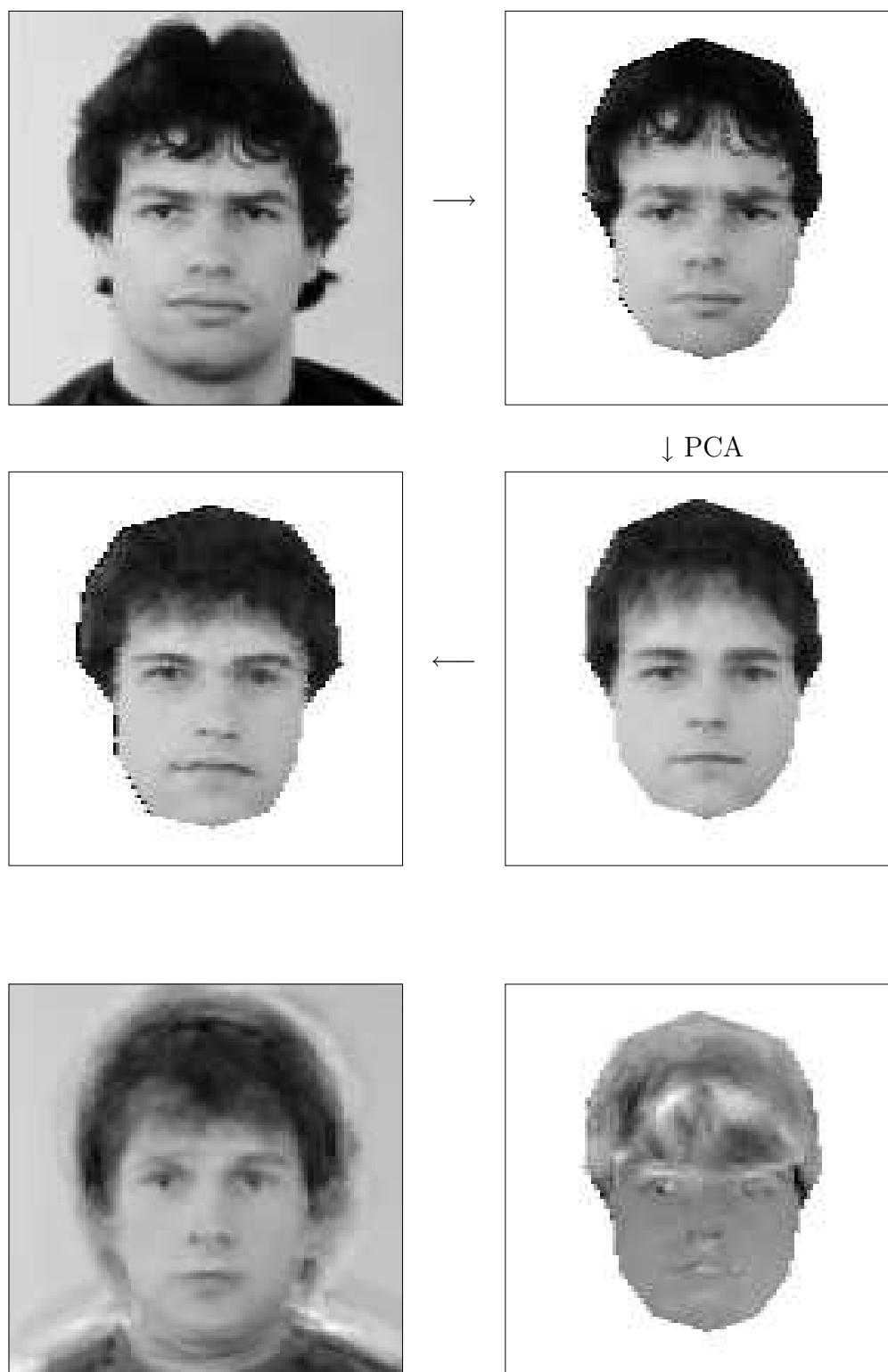
took similar precautions, insisting that the unknown face was centred and the same size as both the training images and the eigenfaces. They also eliminated background information by masking the face image with a fixed Gaussian centred on the face.

### 8.2.2 Improving PCA

Sirovich and Kirby, and Turk and Pentland both normalized their images, implicitly mapping them into an almost linear subspace, in order to use PCA.

We improved the average of two faces in section 8.1 by first mapping them onto a common 3D model, thus incorporating knowledge of faces into the average. In exactly the same way we can improve PCA if we first linearize the images by mapping them onto a common head model. To reconstruct the images we apply the reverse mapping to the PCA representation. This has been done by Craw and Cameron [14].

Of course we need the positions of the major vertices on each image to carry out this mapping. If we extend the image vectors to include these positions, as well as the pixel intensities, before applying PCA then the eigenfaces will capture the variation in the shape of faces as well as the variation in intensity.



**Figure 8.7:** Top row: a face and its mapping onto the common model. Middle right: the PCA approximation. Middle left: its mapping back onto the original shape. Bottom left: the PCA approximation to the original face, using Sirovich and Kirby's method. Bottom right one of our eigenfaces.

# Chapter 9

## Summary

Realistic images of faces of individual people have been generated by computer by texture mapping a photograph of the individual onto a wireframe model of their head. In previous work each individual had to be measured in order to build this model.

We use a *generic* head model to describe the shape of a human head. This model gives us “world knowledge” of the shape of a head; to create the new images we distort this to fit the face in the photograph, creating a new, *specific*, head model. This specific model can be distorted further to change the expression before it is projected onto a viewing plane. We then map the texture from the photograph of the person, one triangle of the model at a time, onto the viewing plane to create a novel view of the individual. Texture from more than one image may be used, by selecting the most appropriate texture source for each triangle of the model.

The generic  $\rightarrow$  specific transformation is implemented by making each edge of the model into a spring, displacing some of the vertices to fit the photograph and then allowing the spring-frame model to relax to a position of least energy.

**2D Distortions** Psychologists studying how people remember images of faces were interested in having several photographs of the same person, but with the

facial features repositioned in each image.

The features of the face are repositioned by specifying the “before” and “after” positions of a number *control points*. We calculate the Delaunay triangulation of the control points. This gives us a graph with nodes at the control points, straight line arcs and triangular regions. We use barycentric coordinates to extend the mapping between the before and after graphs into a mapping between the original and desired images. This distortion mapping is then used to copy colour and shading information (texture) onto the desired image, one triangle at a time.

This *texture mapping* is implemented by scanning the rows of pixels which make up each triangle. This ensures that every pixel of the image is coloured in correctly. Benson [6] gives an alternative way of texture mapping triangles. This is described and compared with our implementation. We find that under certain conditions Benson’s implementation can leave significant parts of a triangle unfilled.

We show how this method of distorting faces has been used to create stimuli for the Nottingham psychologists. Chapter 2 concludes by mentioning that this method can also be used to merge the faces of two different people in a convincing way (a 3D version of this is described later), and that Benson has used it to caricature faces.

**Generic Heads** 2D distortions are useful but a head is a 3D object, and many manipulations of the image would be achieved more effectively if we took account of this.

Impressive as these 2D distortions of faces are, they take no account of the fact that faces are 3D objects; for example if we know the 3D shape of the head it would be easy to rotate the image to create a picture of the person as viewed from another camera position. Yau and Duffy [16] used a laser range-finder to find the 3D shape of each head that they wished to create images of, but often the person and the range-finder are unavailable. If we had a generic head model—a model of a human head, but not necessarily a model of any individual—and could distort

it to fit the face in a photograph we could use this model to rotate the photo.

There are several possible ways of representing the 3D shape of an object; we have chosen perhaps the simplest—a wireframe model composed entirely of triangles, since this is easy to distort to fit a photograph and because it requires very little change to apply the 2D texture mapping algorithm to this model.

We tried using Rydfalk’s *Candide* [37] as a generic model, but found that 100 triangles does not give sufficient detail for realistic rotations. We had available a laser-scanned model of a human head—“Vicki” with around 20 000 triangles, but displaying this took too long, so we decided to compromise by using a generic model with around 1000 triangles. This model would be built by finding a suitable approximation to Vicki.

A naïve approximation would sub-sample the triangulation, keeping one vertex in every  $n$  in each direction. This means that the approximation consists of triangles all about the same size. This gives a much better approximation where the surface is flat than in “interesting” areas where the shape of the surface is changing.

Several methods of approximating a set of points to a uniform accuracy with a wireframe model have been published, but only Faugeras et al. [18] is appropriate if the model is to have a front and a back—the others are intended for points on an almost flat surface.

Faugeras’ algorithm starts with a trivial triangulation — two triangles back-to-back. Each triangle is the approximation for one half of the given wireframe. We consider each piece of the wireframe in turn. If every vertex is sufficiently close to the approximating triangle, then the approximating triangle is good enough. If not we split the triangle into 3 by adding the vertex furthest from the triangle. We split the piece of wireframe into 3 smaller pieces, one piece approximated by each new triangle. The new edges of the triangles may not be very good approximations, so we refine them by adding the vertex furthest from, but approximated by, the edge. This splits a pair of triangles into four. We now repeat the whole process,

splitting triangles into 3 and refining edges until every vertex of the original is sufficiently close to a triangle.

The algorithm can only produce correct approximations to a surface (as opposed to the vertices) if the edge refinement step is applied sufficiently often. We show that the algorithm as given by Faugeras does not refine every edge that it should by showing the result of approximating an icosahedron.

We describe an alternative method which removes vertices from a triangulation as long as the resulting triangulation remains a sufficiently good approximation to the original. We do this by giving each triangle a thickness and making it thick enough to approximate the parts of the triangulation which have been removed.

Although this algorithm does produce approximating triangulations, in practise the approximations are not as useful as those produced by the naïve method. The generic model used for the remainder of the work was therefore built by the naïve method.

**Generic→specific transformation** The generic head provides a 3D description of a human head, but it is not the head in the photograph. We distort the generic model so that when projected onto the plane of the photograph it matches the head in the picture.

We apply an affine transformation to the generic head, rotating and scaling it so that it roughly matches the face in the photo. A number of vertices (the *major* vertices) are then located in the picture, either manually or automatically, using software developed at Aberdeen University. Several methods of deriving the depth coordinates of these vertices from the image are discussed. We get a good estimate of the depth directly from the rotated generic model, since this gives us world knowledge of the shape of a human head.

Once we have the 3D positions of some vertices of the model we interpolate to find the remaining *minor* vertices. Two interpolation methods are compared. The first uses a 2D Delaunay triangulation of the major vertices and barycentric

coordinates to interpolate linearly within each triangle; this is similar to the distortion mapping we have used to distort 2D images. The method is not able to extrapolate the positions of the minor vertices which lie outside the convex hull of the major vertices, and it breaks down when the orientation of one or more triangles reverses between the generic model and the photograph.

The second method simulates the distortion of the rotated generic wireframe model as if it were made from springs. We clamp the the major vertices in the positions which match the photograph and allow the model to reach a position of least energy. This equilibrium position gives the interpolated positions for the minor vertices.

**Display** We discuss how it would be possible to change the expression of the face by distorting the model further, before going on to describe how it is texture-mapped and displayed.

We create a picture of the face viewed from a new viewpoint by first rotating the specific model and projecting the vertices onto the appropriate viewing plane. We then copy the shading information from the triangles of the original image to the triangles on this viewing plane, in much the same way as we did for the 2D distortions. Since the head is 3 dimensional, parts of the surface may be obscured behind other parts, both in the original image and in the new view. We use a z-buffer hidden-surface algorithm to ensure that we display the correct surface, and that we only use texture from parts of the surface visible in the original photograph.

**Several Images—Mugs** One image of an object will never show us the whole surface of an object, but a small number of images can show all of the surface visible from many viewpoints.

We experiment with multiple images, using a coffee mug and describe the pitfalls of taking several pictures of an object from different viewpoints.

If we have several images of an object much of the surface will be visible in more than one image, and we must decide which image to use as the texture source for each region of the surface. We see a surface best when it is close to and facing the camera so, provided that the images are all of (approximately) the same quality, each region will appear best in the view in which it is largest.

Since the texture mapping uses a linear transformation we effectively assume that each triangle of the model represents a flat face of the object. If this is so, then the best view of each subregion of the triangle is the same, and we only need to find the best view for each triangle - it is the same for all the pixels provided that the whole of the triangle is visible in this image. Thus the texture for each triangle of the model comes from the image in which the triangle appears largest.

*Specular* reflection often causes highlights in shiny objects. The position of these highlights depends upon the camera viewpoint, so combining two or more images by texture mapping can create a picture with unrealistic highlights. To correct this we must remove the highlights from each image before texture mapping and then add the highlights as they would appear in the final image.

**Applications** We describe how a picture of an “average person” can be constructed from images and wireframe models of two people. Unlike most face averages, this face is a *convincing picture of a non-existent person*. We create a sequence of weighted averages which convincingly changes from one face into another.

We describe how texture mapping many faces onto a standard shaped model improves the reconstruction and recognition of face images by Principal Component Analysis.

**Shape from shading** We consider Pentland’s fourier domain shape from shading algorithm [35] and show that the simplifying assumptions he makes imply a more restrictive illumination model than the one Pentland claims to have used.

# Appendix A

## Fourier-domain Depth from Shading

Given a digitised picture (an intensity image) of a scene, an ideal depth from shading algorithm will give us a depth-map of the scene. At each point  $(x,y)$  of the depth-map, the value stored represents the  $z$  coordinate of the point which was seen at  $(x,y)$  in the intensity image.

This is one example of shape from shading, the more general problem of finding the shape of objects in the scene.

Pentland [35] describes a depth from shading algorithm which works by applying a simple transformation to the Fourier transform of the intensity image.

Before describing Pentland's algorithm and my implementation, I shall briefly discuss lighting models and a peculiar interaction between Fourier transforms and one particular lighting model.

### A.1 Lighting Models

Consider a surface lit by several distant point sources, viewed from a distance. The light could be reflected off the surface in many different ways (see [22] and

[11] for example). We shall consider just two of these - the Lambertian and the Lunar reflectance functions.

### A.1.1 The Lambertian reflectance function

According to Horn [22]:

A Lambertian surface or diffuser looks equally bright from all directions; the amount of light reflected depends only upon the cosine of the incident angle.

If a surface  $z=z(x,y)$  is illuminated by a distant point source, then at each point  $(x,y)$  the observed intensity  $I(x,y)$  is

$$I(x, y) = (\alpha \mathbf{N}) \cdot (\lambda \mathbf{L}) \quad (\text{A.1})$$

where  $\alpha$  is the albedo or reflectance of the surface,  $\mathbf{N}$  is the unit surface normal vector,  $\lambda$  is the intensity of the source and  $\mathbf{L}$  is the unit vector in the direction of the light source.

If we have several such sources illuminating the surface (and there is no self-shadowing), then the intensities sum. Since the sum of dot-products equals the dot-product of the sum, the total intensity is the same as if we had just one source, in the mean illuminant position.  $\alpha$  and  $\lambda$  are constants across the surface, so we shall consider the normalized intensity obtained by setting these to unity. Thus we have the Lambertian reflectance function:

$$I(x, y) = \mathbf{N} \cdot \mathbf{L} \quad (\text{A.2})$$

### A.1.2 The Lunar reflectance function

Horn points out that for a dusty surface at a great distance, for example the surface of the Moon, or perhaps Mercury, viewed from Earth, the intensity is constant for constant values of  $\cos(i)/\cos(e)$  where  $i$  is the incident angle and  $e$  is

the emittance angle. A simple model of this is the lunar reflectance function:

$$I_{lunar}(x, y) = \frac{\mathbf{N} \cdot \mathbf{L}}{\mathbf{N} \cdot \mathbf{V}} \quad (\text{A.3})$$

where  $\mathbf{V}$  is the unit vector pointing towards the viewer.

## A.2 A Peculiar Interaction

The depth map of any surface can be considered as a (probably infinite) sum of simpler depth maps. These are simple in the sense that their Fourier transforms are delta functions - ie have just one non-zero component. Consider the depth map  $d(x,y)$  which has Fourier transform non-zero only at the point  $(u,v)$  in the Fourier domain. This depth map is a wave travelling in the direction  $(u,v)$  in the  $x$ - $y$  plane, with frequency  $\sqrt{u^2 + v^2}$ , and a sinusoidal cross-section. If this surface is lit by the lunar reflectance function with the illumination lying in the plane perpendicular to the direction of travel, ie such that

$$uL_x + vL_y = 0 \quad (\text{A.4})$$

then the intensity  $I(x,y)$  is constant over the whole image. Thus for a given lighting direction there is a whole family of invisible depth maps.

This family of invisible depth maps is significant to Pentland's algorithm, as we shall see.

## A.3 Pentland's algorithm

Pentland makes the following initial assumptions: Let  $z=z(x,y)$  be a surface, and let us assume that:

- (1) the surface is Lambertian,
- (2) the surface is illuminated by (possibly several) distant point sources,
- (3) the surface is not self-shadowing.

We will also take  $z < 0$  within the region of interest, and assume orthographic projection onto the  $x,y$  plane.

Thus the  $x,y$  plane is the viewing plane and the viewing vector  $\mathbf{V} = (0, 0, 1)$ . Representing the surface by a function of  $(x,y)$  ensures that the surface is not self-occluding.

Let  $\mathbf{L} = (L_x, L_y, L_z) = (\cos t \sin s, \sin t \sin s, \cos s)$  be the unit vector in the mean illuminant direction, where  $t$  is the tilt of the illuminant (the angle that the image-plane component of the illumination makes in the image-plane) and  $s$  is its slant (the angle the illumination makes to the image-plane normal).

Under these assumptions the normalized image intensity

$$I(x, y) = \frac{pL_x + qL_y + L_z}{\sqrt{p^2 + q^2 + 1}} \quad (\text{A.5})$$

where  $p, q$  are the partial derivatives of  $z$  in the  $x$  &  $y$  directions respectively, evaluated at the point  $(x,y)$ .

The vector  $\mathbf{N}^* = (p, q, 1)$  is perpendicular to the surface at  $(x,y)$  [22], so by equation (A.2), for a Lambertian surface the normalized image intensity is:

$$I(x, y) = \frac{\mathbf{N}^* \cdot \mathbf{L}}{\sqrt{\mathbf{N}^* \cdot \mathbf{N}^*}} = \frac{pL_x + qL_y + L_z}{\sqrt{p^2 + q^2 + 1}} \quad (\text{A.6})$$

We now take the Taylor series expansion of  $I(x,y)$  about  $p, q = 0$  up through the quadratic terms:

$$I(x, y) \approx L_z + pL_x + qL_y + \frac{1}{2}L_z(p^2 + q^2). \quad (\text{A.7})$$

When the partial derivatives  $p$  &  $q$  are small or the illumination is nearly perpendicular to the viewing vector, the quadratic term will be negligible. We assume this is the case, and approximate the intensity by the linear terms only:

$$I_{approx}(x, y) = L_z + pL_x + qL_y \quad (= \frac{\mathbf{N}^* \cdot \mathbf{L}}{\mathbf{N}^* \cdot \mathbf{V}}) \quad (\text{A.8})$$

Since  $\mathbf{N}$  and  $\mathbf{N}^*$  are both perpendicular to the surface they must be parallel, so as Pentland notes, our approximation is in fact the lunar reflectance function, eqn A.3.

If we take the Fourier transform of this and ignore the constant or D.C. term we get:

$$F_i(u, v) = 2\pi i(uL_x + vL_y)F_z(u, v) \quad (\text{A.9})$$

for each point  $(u, v)$  in the Fourier domain, where  $F_i$  is Fourier transform of the intensity and  $F_z$  is Fourier transform of the depth. As Pentland notes this can be recast, allowing us to recover most of the depth information, given the intensity and the direction of the light source:

$$F_z(u, v) = \frac{1}{2\pi i(uL_x + vL_y)} F_i(u, v) \quad (\text{A.10})$$

Consider what happens for values of  $u, v$  for which this is singular (ie values such that  $uL_x + vL_y = 0$ ). These are precisely the values of  $u, v$  for which the Fourier coefficient represents a member of the family of depth maps which have constant intensity when illuminated with the Lunar reflectance function. At these points the model (A.9) predicts that the intensity transform  $F_i(u, v)$  is zero (if it is not our model is only an approximation to the true lighting). Thus we have no information to enable us to determine the depth transform  $F_z(u, v)$ , and we shall assume it is zero. (This also avoids the problems of division by zero when we use A.10 to find  $F_z(u, v)$ .) Because of this assumption we cannot be certain that the calculated depth map is the true one, the best we can claim is that they differ only by some arbitrary sum of the depth maps which have constant intensity. This means that the calculated depth-map has bands of constant error in the direction of the projection of the illuminant vector onto the image plane.

## A.4 Application of the algorithm

A program was written in C to derive the depth-map for an input discretized image, using Pentland's algorithm. Pentland has proposed several methods of

calculating the mean illuminant direction, but none of these have been implemented, so it has not been possible to make a fair test of the algorithm on any real images. Our first tests were therefore made on synthesized images of simple objects - spheres and prolate spheroids - “eggs”.

The first test image was of an egg, illuminated by a distant point source at tilt=138.730369 degrees, slant=39.632571 degrees using the Lambertian reflectance model. The egg was visible in the resulting depth-map but this was dominated by strong sinusoidal bands. Experimenting with different slant and tilt convinced me that the bands were always perpendicular to the projection of the illumination vector, and led me to discover the peculiar interaction between the lunar reflectance function and the Fourier transform. This interaction suggested a way of partially removing the bands. Since the program works on a discrete form of the image and uses a discrete Fourier transform, Fourier coefficients near to the singularity will be inaccurate as well as those actually on it. We thus ignore coefficients which are close to the singularity, by setting  $F_z(u,v)=0$  whenever  $abs(uL_x + vL_y) < \epsilon$ .

$\epsilon$  can be set at run-time and after experimentation I have found that  $\epsilon=0.5$  removes the bands without removing too much of the test images I have tried. Running the algorithm on this test image again, this time with  $\epsilon = 0.5$  removed the bands, but there were slight peaks or ice-caps near to the ends of the egg, and the egg appeared to be sunk into the background - not a perfect depth map.

The second test image was the same egg, illuminated from the same direction, but illuminated by the lunar reflectance function. The third was a sphere illuminated using the Lambertian reflectance model. In both cases the results were very little different from the first.

The last image was a sphere illuminated from the same position and by the lunar function. The depth map with  $\epsilon = 0.5$  was reasonable although still suffering the same problems. However the depth map for  $\epsilon = 1e-10$  was very good, (although some slight banding was still visible).

## A.5 Conclusions

Pentland has produced a novel depth from shading algorithm. Unlike previous shape from shading methods the algorithm makes no smoothness assumption - this was a major short-coming of these early methods. Like previous shape from shading methods the illumination direction must be known in advance. Pentland has proposed several ways of finding this, but none of them have been implemented, and it is believed that finding the illuminant direction is hard. Without such a routine, the choice of test images has been severely limited and I have very few results on which to make my conclusions. The solution is not completely determined, and I have yet to find a way of finding the best of the possible solutions, except by trial and error. I understand that Pentland has addressed this problem in a more recent paper, but I have not seen the paper. It has been possible to adjust the band removal parameter  $\epsilon$  to produce reasonable looking depth maps from the test images, but slightly different images require different settings. The choice of reflectance function appears to be very important: the lunar rf produced much better depth maps than the Lambertian model. This is unfortunate as the Lambertian model is a much better approximation to everyday lighting conditions. Pentland has produced some images of a girl's face by rotating the depth-map calculated by the algorithm, and then shading it. A sympathetic viewer could agree that these look like the girl viewed from a new angle, but I have the suspicion that it may be possible to obtain similar results with a trivial depth algorithm, such as taking the intensity gradient. (Note that if the viewing and illuminant vectors coincide this is a perfect depth map anyway). In summary I believe that Pentland's algorithm is interesting but would take significant effort to make it work reliably without external assistance.

# Appendix B

## Candide: Rydfalk's Wireframe Model

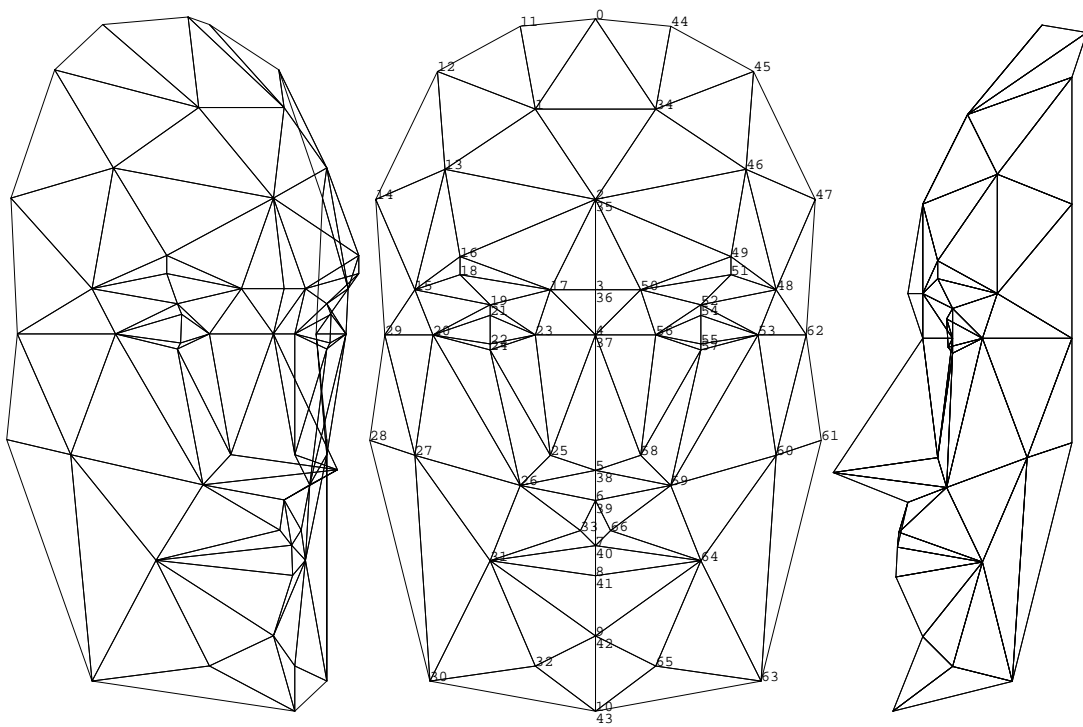


Figure B.1: Three views of Candide

	x	y	z		x	y	z	
0	0	250	40					
	Left half of face				Right half of face			
1	40	190	90		34	-40	190	90
2	0	130	120		35	0	130	120
3	0	70	130		36	0	70	130
4	0	40	120		37	0	40	120
5	0	-50	180		38	0	-50	180
6	0	-70	130		39	0	-70	130
7	0	-100	137	top lip	40	0	-100	137
8	0	-120	138		41	0	-120	138
9	0	-160	120		42	0	-160	120
10	0	-210	140		43	0	-210	140
11	50	245	10		44	-50	245	10
12	105	215	20		45	-105	215	20
13	100	150	70		46	-100	150	70
14	146	130	20		47	-146	130	20
15	120	70	70		48	-120	70	70
16	90	92	110		49	-90	92	110
17	30	70	120		50	-30	70	120
18	90	80	110		51	-90	80	110
19	70	60	100		52	-70	60	100
20	108	40	80		53	-108	40	80
21	70	53	104		54	-70	53	104
22	70	34	103		55	-70	34	103
23	40	40	100		56	-40	40	100
24	70	30	100		57	-70	30	100
25	30	-40	110		58	-30	-40	110
26	50	-60	104		59	-50	-60	104
27	120	-40	50		60	-120	-40	50
28	150	-30	20		61	-150	-30	20
29	140	40	20		62	-140	40	20
30	110	-190	60		63	-110	-190	60
31	70	-110	80		64	-70	-110	80
32	40	-180	100		65	-40	-180	100
33	10	-90	136		66	-10	-90	136

Points 35–39 and 41–43  
are the same as  
points 2–6 and 8–10

bottom lip

# Appendix C

## Source Code

### C.1 Generic $\rightarrow$ Specific Model Transformation

#### C.1.1 `vectors.h`

## **C.1.2 3d\_stuff.h**

### **C.1.3 delaunay.h**

### **C.1.4 specific.h**

### **C.1.5 specific\_delaunay.c**

### **C.1.6 specific.c**

## **C.2 Drawing the Model**

### **C.2.1 render.h**

## **C.2.2 render3d.c**

### **C.2.3 draw.h**

## C.2.4 draw.c

## **C.3 Triangulation Reduction: Faugeras' Method**

### **C.3.1 `vector_arithmetic.p`**

### **C.3.2 fauieras\_dist.p**

### **C.3.3 shortestPath.p**

### **C.3.4 fauieras.p**

## **C.4 Triangulation Reduction: Point Removing Method**

### **C.4.1 shuffle.p**

## **C.4.2** `ian_dist.p`

### **C.4.3 calculateThickness.p**

#### **C.4.4 triangulate\_polygon.p**

### **C.4.5 approx\_ian.p**

## **C.4.6 main program**

# Appendix D

## Glossary

**Aliasing** When we digitize a fast-changing pattern we must take several samples in the space of a single pattern cycle. If instead the pattern is under-sampled, by taking at most a single sample in each pattern cycle, the digitised image will contain lower frequency patterns which were not in the original pattern. These new patterns are *aliases*.

**Barycentre** The middle point of a triangle. It lies at the intersection of the lines which bisect the angles, and its barycentric coordinates are  $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ .

**Barycentric coordinates** Given a triangle ABC and a point P lying in the plane of the triangle, there is a unique triple (a b c) which describes the position of P as a linear combination of the positions of points A, B and C, independently of the coordinate system used. The details are given on page 22

**Convex hull** If a rubber band is put around a set of points in the plane and allowed to shrink to fit the points, it will lie on the boundary of the convex hull of the points. The convex hull of a set of 3D points can be formed in a similar manner by enclosing the points in a balloon and deflating it.

**Depth map** If a surface is relatively flat and it is possible to see the whole of it from a suitable viewpoint, we can project a rectangular grid onto the

surface, using a parallel projection, and record the height of the surface at each intersection point on the grid. This 2D array of heights is the depth map. Sometimes many entries in the array are empty—this gives a *sparse* depth map.

**Gouraud shading** A method of shading a 3D polygon in which the luminance of the surface is interpolated from the values at the corners. It is easier to calculate than Phong shading, but more likely to produce Mach banding.

**Mach banding** An optical illusion occurring where the intensity gradient of an image changes sharply, often causing light or dark bands in the image.

**Mouse** A device which allows the user to point to and select any part of the display on a computer screen.

**Phong lighting** An illumination model which allows for specular as well as diffuse reflection. See chapter 7.5.1 for further details. Often combined with Phong shading to simulate realistic illumination.

**Phong shading** A method of shading a 3D polygon in which the surface normal is interpolated across the surface. Often used in conjunction with Phong's lighting model, but any lighting model may be used. Both were first described by Bui Tuong Phong [11]. Compare with Gouraud shading.

**Pixel** A digitised image is discrete, made up of many small *picture elements* - squares or rectangles of uniform size. The colour and intensity can vary between pixels, but are constant within each pixel.

**Raster display device** A computer graphics display made up of a rectangular grid of pixels. Traditionally, like a conventional television, the picture is formed on a cathode ray tube (CRT) by adjusting the strength of the beam as it sweeps out parallel horizontal lines across the phosphor coated screen.

**Vector display device** A computer graphics display, now replaced by raster displays for almost all applications, in which the picture is formed by sweeping

out lines directly on the CRT. Vector displays don't require as much computer storage as raster displays, and are good at displaying pictures composed of straight lines in arbitrary orientations, but they are not good for displaying filled in regions.

# Bibliography

- [1] Kiyoharu Aizawa, Hiroshi Harashima, and Takahiro Saito. Model-based synthetic image coding system - construction of a 3-D model of a person's face. In *Picture Coding Symposium, PCS87*, pages 50–51, Stockholm, June 9-11 1987.
- [2] Kiyoharu Aizawa, Hiroshi Harashima, and Takahiro Saito. Model-based analysis synthesis image coding (MBASIC) system for a person's face. *Signal Processing: Image Communication*, 1(2):139–152, October 1989.
- [3] Alan Bennett and Ian Craw. Finding image features using deformable templates and detailed prior statistical knowledge. In BMVC91 [7]. Peter Mowforth, editor.
- [4] Philip J. Benson and David I. Perrett. Changing faces, changing places. *The Guardian*, November 21 1990.
- [5] Philip J. Benson and David I. Perrett. Perception and recognition of photographic quality facial caricatures: Implications for the recognition of natural images. *European Journal of Cognitive Psychology*, 3(1), 1991.
- [6] Philip J. Benson and David I. Perrett. Synthesising continuous-tone caricatures. *Image and Vision Computing*, 9(2):123–129, 1991.
- [7] British Machine Vision Association. *BMVC91 Proceedings of the British Machine Vision Conference*, Glasgow, 23–26 September 1991. Springer-Verlag. Peter Mowforth, editor.
- [8] Vicki Bruce. Personal communication.
- [9] Vicki Bruce, Tony Doyle, Neal Dench, and Mike Burton. Remembering facial configuration. In *European Society for Cognitive Psychology*, Como, Italy, September 1990. unpublished.
- [10] Vicki Bruce, Tony Doyle, Neal Dench, and Mike Burton. Remembering facial configuration. *Cognition*, 38(2):109–144, February 1991.
- [11] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the A.C.M.*, 18(6):311–317, June 1975.

- [12] A.K. Cline and R.L. Renka. A storage efficient method for construction of a Thiessen triangulation. *Rocky Mountain Journal of Mathematics*, 14, 1984. Cited in [40].
- [13] Y. Correc and E. Chapuis. Fast computation of Delaunay triangulations. *Advances in Engineering Software*, 9(2):77–83, April 1987.
- [14] Ian Craw and Peter Cameron. Parameterising images for recognition and reconstruction. In BMVC91 [7]. Peter Mowforth, editor.
- [15] Leila De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE: Computer Graphics and Applications*, pages 67–78, March 1989.
- [16] Neil D. Duffy and John F.S. Yau. Facial image reconstruction and manipulation from measurements obtained using a structured lighting technique. *Pattern Recognition Letters*, 7(4):239–243, April 1988.
- [17] P. Ekman and W. V. Friesen. *Manual for the Facial Action Coding System*. Consulting Psychology Press, Palo Alto, Calif., 1977.
- [18] O. D. Faugeras, M. Hebert, P. Mussi, and J. D. Boissonnat. Polyhedral approximation of 3-D objects without holes. *Computer Vision, Graphics and Image Processing*, 25:169–183, 1984.
- [19] James D. Foley and Andries van Dam. *Fundamentals of interactive computer graphics*. Addison-Wesley, 1982.
- [20] Hiroshi Harashima, Kiyoharu Aizawa, and Takahiro Saito. Model-based analysis synthesis coding of videotelephone images - conception and basic study of intelligent image coding. *The Transactions of the IEICE*, E 72(5):452–459, May 1989.
- [21] P. S. Heckbert. Survey of texture mapping in computer generated images. *IEEE: Computer Graphics and Applications*, pages 56–67, November 1986.
- [22] Berthold K. P. Horn. Understanding image intensities. *Artificial Intelligence*, 8(2):201–231, 1977. Copyright 1981, North-Holland.
- [23] M. Kirby and L. Sirovich. Application of the karhunen–loève procedure for the characterisation of human faces. *IEEE: Transactions on Pattern Analysis and Machine Intelligence*, pages 103–108, January 1990.
- [24] C. L. Lawson. Software for  $C^1$  interpolation. In J. Rice, editor, *Mathematical software III*, pages 161–194. Academic Press, New York, 1977.
- [25] K. Y. Lee. Texture mapping in model-based image coding with luminance compensation. Msc project report, University of Essex, Department of Electronic Systems Engineering, May 1991. Cited by Don Pearson, at the Royal

- Society Discussion Meeting *Processing the Facial Image*, London, 9th-10th July 1991.
- [26] K. Y. Lee and D. E. Pearson. Luminance compensation in model-based prediction. In *Picture Coding Symposium, PCS91*, Tokyo, 2-4th September 1991.
- [27] Alf D. Linney. The use of 3-D computer graphics for the simulation and prediction of facial surgery. In Vicki Bruce and Mike Burton, editors, *Processing Images of Faces*. Ablex, Norwood, N.J., 1991 ? To Appear.
- [28] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987. ISBN 0-19-503965-3.
- [29] Frederic I. Parke. Computer generated animation of faces. *Proceedings of the ACM*, 1:451-457, August 1972.
- [30] Frederic I. Parke. Measuring three-dimensional surfaces with a two-dimensional data tablet. In *Conference on Computer Graphics and Interactive Techniques*. ACM/SIGGRAPH, July 1974. reprinted in *Comput. and Graphics*, v1, pages 5-7, Pergammon Press 1975.
- [31] Frederic I. Parke. Parameterized models for facial animation. *IEEE CG&A*, pages 61-68, November 1982.
- [32] Manjula Patel and Philip J. Willis. Faces: Facial animation, construction and editing system. In *EUROGRAPHICS '91*, Vienna, September 1991.
- [33] Alex P. Pentland. Local shading analysis. *IEEE: Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(2), March 1984.
- [34] Alex P. Pentland. Perceptual organisation and the representation of natural form. *Artificial Intelligence*, 28:293-331, 1986.
- [35] Alex P. Pentland. Shape information from shading. In J. C. Simon, editor, *From Pixels to Features*, pages 103-113. Elsevier Science Publishers B.V. (North Holland), 1989.
- [36] Stephen M. Platt and Norman I. Badler. Animating facial expressions. *Computer Graphics*, 15(3):245-252, August 1981. Proc. ACM SIGGRAPH '81.
- [37] Mikael Rydfalk. Candide, a parameterised face. Technical Report Lith-ISY-I-0866, Linköping University, Department of Electrical Engineering Linköping University S-581-83, Linköping Sweden, October 1987.
- [38] R. Sibson. Locally equiangular triangles. *The Computer Journal*, 21(3):243-245, August 1978.

- [39] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterisation of human faces. *Journal (Optical Society of America) A: Optics and Image Science*, pages 519–524, March 1987.
- [40] S. W. Sloan. A fast algorithm for constructing Delaunay triangulations in the plane. *Advances in Engineering Software*, 9(1):34–55, January 1987.
- [41] Demetri Terzopoulos. Visual modeling. In BMVC91 [7]. Peter Mowforth, editor.
- [42] Demetri Terzopoulos and Manuela Vasilescu. Sampling and reconstruction with adaptive meshes. In *Proceedings of Computer Vision and Pattern Recognition Conference (CVPR-91)*, pages 70–75, 1991. Lahaina, Hawaii.
- [43] David Tock, Ian Craw, and Roly Lishman. A knowledge based system for measuring faces. In *BMVC90 Proceedings of the British Machine Vision Conference*, pages 401–407, University of Oxford, 24–27 September 1990. British Machine Vision Association.
- [44] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of the Optical Society of America*, 57(9):1105–1114, September 1967.
- [45] K. E. Torrance, E. M. Sparrow, and R. C. Birkebak. Polarization, direction, distribution, and off-specular peak phenomena in light reflected from roughened surfaces. *Journal of the Optical Society of America*, 56(7):916–925, July 1966.
- [46] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.
- [47] Keith Waters. Animating human heads. In *Online, Computer Graphics '87: proceedings of the conference held in London, October 1987*, pages 89–97, 1987.
- [48] Keith Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 21(4):17–24, 1987. Proc. ACM SIGGRAPH '87.
- [49] Keith Waters and Demetri Terzopoulos. The computer synthesis of expressive faces. *Philosophical Transactions of the Royal Society*, (to appear) January 1992. Presented at the Royal Society Discussion Meeting *Processing the Facial Image*, London, 9th-10th July 1991.
- [50] Watkins. *Principles of Interactive Computer Graphics*, pages 313–321, 537–552. McGraw-Hill, New York, 1st edition, 1973. Cited by Duffy and Yau [16].
- [51] Doctor Who. *Logopolis*. BBC TV program, by Christopher H. Bidmead, starring Tom Baker and Peter Davidson. First Broadcast 1981.

- [52] Doctor Who. *Planet of the Spiders*. BBC enterprises Ltd., 1991. Video, by Robert Sloman, starring Jon Pertwee. First broadcast 1974. BBCV 4491.
- [53] John F. S. Yau. An algorithm for automatic generation of an optimum polygon representation of a 3-D surface. Research Memorandum RM/87/3, Heriot-Watt University, 1987. This paper appears as a chapter of Yau's PhD thesis [54].
- [54] John F. S. Yau. *A Model-Based Approach to Picture-Phone Coding*. PhD thesis, Department of Electrical and Electronic Engineering, Heriot-Watt University, Edinburgh, July 1989.
- [55] John F.S. Yau and Neil D. Duffy. 3-D facial animation using image synthesis. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *New Trends in Computer Graphics*. Springer-Verlag, 1988.
- [56] John F.S. Yau and Neil D. Duffy. A texture mapping approach to 3-D facial image synthesis. *Computer Graphics Forum*, 7:129–134, 1988. Presented at 6th Annual EURO-GRAPHICS (UK) Conference, Sussex, April 6-8, 1988.