

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

The Development of Motion-tracking Strategies for Cineradiographic Images

A thesis is presented in partial fulfilment of the requirements for
the degree of Doctor of Philosophy in Technology at Massey
University.

by

Wyatt H. Page

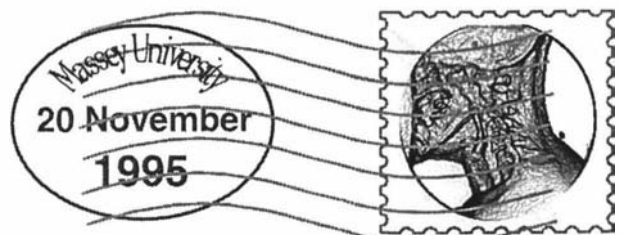
The Development of Motion-tracking Strategies for Cineradiographic Images

Wyatt H. Page

Department of Production Technology

Massey University

NEW ZEALAND



Abstract

This research describes the development of motion-tracking algorithms for a new dynamic spinal motion analysis system. This system utilises digital image processing techniques to extract motion parameters from video cineradiographic sequences of the human spine. The automated tracking of vertebral motion results in accurate assessment of translational and rotational displacement. This has been verified by extensive testing on prescribed motion sequences generated by a digital image warping based algorithm. The use of the motion measurement system provides a new tool for spinal health care professionals in the diagnosis of spinal dysfunction.

Key words -

cineradiography, image processing, motion-tracking, computer graphics, spinal kinematics.

Contents

<i>Abstract</i>	iv
<i>Preface</i>	xi
<i>Dedication</i>	xii
<i>Acknowledgments</i>	xiii
1 Introduction.....	1
1.1 Background.....	1
1.2 Thesis Overview	2
1.3 Contents by Chapter	3
1.4 References.....	5
2 X-ray Radiography	7
2.1 Radiographic Images - An Overview of The Physics.....	7
2.1.1 X-ray Generation.....	7
2.1.2 X-ray Absorption.....	11
2.2 Recording of X-ray Radiation	11
2.2.1 Static Radiography.....	12
2.2.2 Dynamic Radiography	17
2.3 Radiographic Noise.....	20
2.3.1 Radiographic Characteristics: Frequency domain description.....	21
2.4 Radiographic Information.....	25
2.5 Application of Radiography.....	26
2.5.1 Spinal Motion Analysis.....	27
2.6 References.....	27
3 Detecting Edges In Cineradiographic Images.....	29
3.1 Image Segmentation.....	29
Pixel-based methods.....	29
Region-based methods	30
Edge-based methods.....	31
3.2 Low Level Edge Detection Operators	32

3.2.1	Gradient Operators	33
3.2.2	Compass Operators	34
3.2.3	Laplace Operators.....	35
3.2.4	Stochastic Gradients	36
3.2.5	Performance of Low Level Edge Detection Operators	37
	Performance Measures	37
	Low-level Edge Detection Results.....	38
3.3	High-level Edge Detection	40
3.3.1	Human Visual System Based Edge Detection	41
3.3.2	Optimal Edge Detector	44
3.3.3	Rule-based Edge-detection	49
3.3.4	Combining Fragmentary Edge information.....	49
3.3.5	Performance of High Level Edge Detectors	52
3.4.	An Alternative HVS Inspired Approach To Edge Detection.....	55
3.4.1	Weber's Law and Profile Based Edge Detection	57
3.4.2.	Target Based Computation Of Threshold Profiles	59
3.4.3.	Direct Thresholding Using ES-LAI Edge-Template Statistics.....	66
3.5.	Summary of Edge Detection In Cineradiographic Images.....	69
3.6	References.....	72
4	Motion-tracking: An Overview	76
4.1	Visual Motion Perception and Description.....	76
4.1.1	The Aperture Problem	79
4.2	Matching Approaches	80
4.2.1	Matching In The Spatial Domain.....	81
	Moment Invariants.....	82
	Surface fitting	83
	Preprocessing	84
4.2.2	Matching In The Transform Domain	84
	Energy Compaction and Decorrelation.....	85
	Matching	86
	The Fourier Transform	86
	Real Transforms	87
4.2.3	Search Strategies	88
	Two-dimensional logarithmic search.....	89
	Sequential search.....	90

	Hierarchical search	90
4.2.4	Motion Prediction	90
4.3	Optical Flow Methods.....	91
4.3.1	Differential Approaches.....	92
4.3.2	Matching Approaches.....	93
4.3.3	Contour Approaches	94
4.3.4	Energy Based Approaches.....	94
4.3.5	Phase based Approaches	96
4.3.6	Adapting Optical Flow For Radiographic Images	97
	Continuity Equation	97
4.4	A Suitable Tracking Method for Cineradiographic Images	99
4.5	References.....	100
5	Match Statistic Performance.....	102
5.1	Template Matching	102
5.1.1	Match Statistics	103
5.1.2	A Match Statistic Sharpness and Selectivity Measure	104
5.2	Salient Features and Optimising Match Selectivity	106
5.3	Match Statistic Evaluation.....	108
5.3.1	Auto-frame Feature Point Matching	108
5.3.2	Interframe Feature Point Matching	113
	Selectivity and Positional Stability	114
	Interframe Match Summary	117
	Interframe Selectivity Constancy	118
5.3.3	Global Image Match Selectivity.....	120
5.4	Match Statistic Conclusions	122
5.5	References.....	123
6	Prescribed Motion Cineradiographic Sequences	126
6.1	Introduction	126
6.2	The Stop-and-shoot Method.....	127
6.3	Articulated Equivalent Phantom.....	128
6.4	Artificially Animated Cineradiographic Sequences	129
6.4.1	Digital Image Warping	129
	Two-Pass Transforms	131
	Two-Pass Mesh Warping	131

6.4.2	Image Rotation	133
	Rotation Evaluation	135
6.5	Motion Sequence Generation	138
6.5.1	Extended Image Warping.....	139
	Application to Cineradiographic Sequences	140
6.5.2	Initial Implementation	141
6.5.3	Enhanced Implementation.....	144
6.5.4	Image Warping Reassessed.....	148
	A Surface Fitting Paradigm for Geometric Correction	149
6.5.5	Final Implementation	151
	Image Noise.....	154
6.6	Motion Test Sequences	158
6.7	References.....	159
7	Motion-tracking Cineradiographic Images	161
7.1	Introduction	162
7.2	Practical Implementation Issues	162
7.2.1	Defining Objects Of Interest.....	162
7.2.2	Vertebral Displacement and Orientation.....	164
	Orientation by Moments	165
	Orientation by Fold-and-Match, a new method	166
7.2.3	Deriving Suitable Motion Measures	168
7.2.4	Displaying Motion Data Effectively.....	171
	Motion Visualisation	171
	Operator Interaction	173
7.3	The Motion-tracking Algorithm	175
7.3.1	Theoretical Tracking Constraints	178
7.3.2	Practical Algorithmic Implementation	180
7.3.3	Image Noise.....	182
7.4	Tracking Performance	183
7.4.1	Single-Cycle Linear Motion: Tracking Results	184
	Single-cycle Linear Translation.....	186
	Single-cycle Linear Rotation	189
	Single-cycle Composite Motion.....	191
7.4.2	Double-Cycle Composite Motion: Tracking Results.....	192
	Double-cycle Composite Motion	192

Double-cycle, Extended length, Composite Motion	193
7.4.3 Full Motion with Vertebral Interaction: Tracking Results.....	194
Realistic Motion.....	195
7.5 Summary of Motion-tracking Performance	199
7.6 References.....	200
8 Discussion and Conclusions	202
8.1 Introduction.....	202
8.2 Algorithm Enhancements	203
8.2.1 Adaptive Enhancements	205
8.3 Future Work.....	206
8.4 References.....	207
Bibliography	208
Appendix A - Gradient Mask Derivation.....	A-1
A.1 Introduction	A-1
A.2 Curve Fitting Problem.....	A-1
A.2.1 Curve Fitting Formulation	A-1
A.2.2 Application to Gradient Masks	A-2
Alternative Orthogonal Formulation	A-3
A.2.3 Gradient Mask Results	A-3
A.3 References.....	A-4
Appendix B - Computational Cost Calculations	B-1
B.1 Introduction	B-1
B.2 Match Statistic Calculations	B-1
B.3 Motion-tracking Algorithm Calculations	B-4
B.4 References.....	B-5
Appendix C - Pascal Source Code	C-1
C.1 Gradient Filters	C-2
C.2 Canny Edge-detector	C-13
C.3 Edge Comparison Functions	C-21
C.4 Weber's Law based Edge-detection	C-23
C.5 Match-statistic Performance Support	C-33

C.6	Image Rotation	C-49
C.7	Two-pass Mesh-warp Image Functions	C-52
C.8	Optimal Triangulation	C-63
C.8	Prescribed Motion by Image Warping.....	C-67
C.10	Motion-Tracking	C-82

Preface

In the beginning (assuming there was a beginning) there was.... whatever there was!

And much, much later I came along, courtesy of my father, a man who inspired me with a talent for almost anything requiring mental skill, creativity and mental dexterity. Nurtured by my mother, a very talented person in her own right, whose greatest talent was her love for life and almost everything in it, I was exposed to many marvellous things and in particular electrical devices and electronics caught my fancy.

My father (the *mad* experimenter) introduced me to the joys of electrocution either by design in the form of a Tesla coil or Windhurst machine or by accident in being reminded that the anode voltage on a UX-807 (an old tetrode indirectly heated vacuum tube of the 1935 era) in an audio amplifier is more than enough to give you a lift in life. It was probably this 'lift in life' that I received on a not too infrequent basis that gave me a head start on many of my friends. At age nine I had not only built many a crystal set but had begun to design better ones, covering a variety of frequencies from longwave to shortwave. One of my first real joys was building a single-triode super-regenerative receiver (one of Colonel Armstrong's many genius ideas). It has such economy of parts and performance that I could scarcely believe at the time plus it could pickup both AM and FM signals. Through this sort of endeavour from receivers to transmitters and lots of audio amplifiers in between, I acquired (guided by my father) the *old* language of electronics and its handicraft.

It will be, in all probability, almost 10 years to the day that I received my first degree (BE Electrical - first class) that I will be receiving my second degree for the work contained in this volume. When I completed my first degree I had had enough of University and yearned for a job and some money (not an unusual feeling, I'm sure!). This lead me to work for four-and-a-half years for a small but diverse electronics manufacturing and design company in which my industrial skills were honed and I met a lot of nice people. Later I joined the Production Technology Department at Massey University where I completed this work and met even more nice people!

It was on 19 July 1989 that I began this work (serendipity had its way) and early the following year I attended the wedding of my best school friend who had also chosen to study and work in electronics. Tragically only three months later he was dead with cancer. So before I continue I will remember Ross Winton Jamieson, '*Rosco*' to his friends, a very talented person who on days like this is badly missed. May the morphic resonance of your being *ring* somewhere in the universe.

So in closing, I hope that the reader (may there be many of you in the future) finds the story that unfolds is done right and done well.

Wyatt Page 14 November 1995



Dedication

A philosopher being asked what was the first
thing necessary to win the love of a woman
answered: "Opportunity."

Thomas Moore

To opportunity, may I always know you
when we meet.

WHP



Acknowledgments

I wish to thank the following people

My supervisor Bill Monteith who was HOD when I started this research and Dean by the time I finished! Your enthusiasm and ideas (very diverse at times), have guided me well through this research.

My second supervisor Bob Hodgson who had just started as the first Professor of Information Engineering in the Department when I started this research and was HOD by the time I finished. Your gentle guidance and help particularly at the very end was greatly appreciated thiee

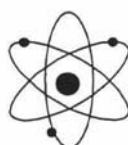
Dr Les Whitehead of the Whitehead Chiropractic Clinic, thank you for access to images from your cineradiographic system.

My very good friend Nicci Coffey who has in the last weeks proofread my work with precision, I thank you greatly.

Brent Foster helped me out of those tricky Windows programming problems.

My great friend Bruce Rapley who I came to visit one day on my way to Australia and ended up doing a PhD and becoming a lecturer all at the same time. Thank you for your enthusiasm to set me on this path.

My family - now you will see me more often.



Introduction

1.1 Background

It will be 100 years this year, 1995, since Professor W. C. Roentgen first observed by accident, what he later called X-rays. Within weeks of the discovery, dentists and doctors were using X-rays in private practice, such was the rapid uptake of this new technology. In 1938, forty years after Roentgen's results were published [1.1], the first application of movie radiography or cineradiography was described by Reynolds [1.2]. Unfortunately using the technology of the time, high-energy continuous X-ray exposure was required, making the method unsafe for human subjects. A mere ten years later, the first X-ray image intensifiers were commercially available. These enabled the total X-ray dose to be reduced by many orders of magnitude. A complete cineradiographic system with video camera and VDU monitor was not to become commercially available, however for another 20 years. A further five years on from this, the first video recorders were available, and a practical system in which the motion of a section of the human spine could be recorded and replayed for visual examination became reality.

One of the single most common medical conditions in modern society concerns problems with the spine, the so called 'back-problems'. Static radiography has been used for almost a century by spinal health professionals in determining the presence of vertebral misalignment and the extent of degeneration in the human spine. Traditionally the practitioner manually extracts measurements from a radiograph by drawing lines directly onto the film through well defined and repeatable points commonly known as '*landmarks*'. A ruler and protractor are then used to measure the displacement and angle of each vertebra based on the reference lines. Typically as many as fifty measurements are taken from a single full spine radiograph, requiring up to an hour to determine. These measurements can be generic such as vertebral disk height, angle and body rotation [1.3]. They can be local, for example the Atlas/Axis angle, or regional as in lumbar curvature and radius. They can also be global, as is the weight bearing and stress lines [1.3].

In order to interpret the significance of these measurements an understanding of the biomechanics of the spine is required. This understanding has been developed from quantitative measurements made on so-called standard *normal spine radiographs* [1.4] and from mechanical

modelling of the spinal column. The difficulty in making a functional diagnosis based on this information is that it is all derived from a stationary spine, usually in only a single position. Predicting what will happen when the spine moves based only on the starting geometry and some basic mechanics, is very difficult. Video cineradiography potentially provides a means of resolving this problem since it can record the spine in normal, unconstrained motion. However, even a short sequence may contain many hundreds of images. Attempting to manually process this large number of images and extract the necessary measurements is totally impractical. Apart from being a very laborious process, repeatability and accuracy is likely to be poor due to operator fatigue.

Previously several authors [1.5] have used cineradiography to infer the behaviour of the spine in motion. But to make the processing practical they have discarded all the images other than a centre image in the neutral position, and two images at the extremes of the motion. These “*motion sequences*” (consisting of only three images) have then been magnified and transferred to film or more recently [1.6] digitised using a computer, and lines manually drawn so that measurements can be made. This approach is inherently unsatisfactory from both a mechanical and a signal processing point of view (see section 6.2 of chapter 6). There are only three *snap-shots* of the true motion and questionable assumptions have to be made about the motion in between in order to draw useful conclusions. Clearly there is a requirement for a system that can automatically and reliably extract motion information from cineradiographic sequences of the human spine. Such a system has the potential of greatly improving the understanding of spinal motion and the diagnosis of spinal dysfunction.

1.2 Thesis Overview

This thesis describes research into the development of suitable algorithms and systems to track the motion of vertebra in cineradiographic sequences of the human spine. Such a system should have the following properties:

- **High accuracy and repeatability** - Within the resolution and noise limitations of video cineradiographic systems the measurements should be accurate and repeatable.
- **Automatic motion-tracking and measurement** - Operator interaction should be minimised in this stage.
- **Provide an opportunity for operator interaction** so that derived measurements can be readily related back to what can be seen visually in a sequence.
- **Be easy to use** so that researchers (and potentially practitioners) will be encouraged to use such a system.
- **Run on standard hardware.**

The last of these requirements, for the system to run on standard hardware, is very much a moving target. The personal computer will be celebrating only its fifteenth birthday next year by which time it will have world wide sales nearly equalling that of the motor car. Performance is currently two and a half orders of magnitude (500 times) better than the first PC, while costing less than half the price of the original machine. Just as importantly, it will come with a standard Graphical User Interface (GUI) and have enough memory, disk space and video performance to make video capture and editing common place. Thus the development described in this thesis is poised at an ideal time to exploit this technology.

Knowledge from a range of diverse research disciplines is required in order to tackle the problem of the development of a system for spinal motion measurement. The first requirement is a fundamental understanding of X-ray radiography with a particular emphasis on the properties of video cineradiographic systems. Converting the video signal (images) to digital form requires a basic understanding of data acquisition. Once images are in digital form, a variety of disciplines need to come together to realise such a system. These include digital image and signal processing for developing motion-tracking algorithms, and computer graphics to enable data visualisation and understanding. Lastly, all of this cannot be achieved without significant skill in software programming. These were the challenges to be met by the author.

If successful, the computer analysis of the image sequences to establish dynamic motion characteristics will be of significant importance to spinal health-care specialists of all types.

1.3 Contents by Chapter

A book style structure has been chosen for this thesis, each chapter beginning with an introductory section that leads to the presentation of the core material, and finally a summary to provide connectivity to the following chapter. Each chapter builds on the previous chapters as the thesis develops. Each chapter is self-contained with its own set of references, all of which are brought together in a comprehensive bibliography at the end of the thesis.

A short paragraph is presented below on each chapter to enable orientation of the material that will be subsequently detailed.

Chapter 1: Provides background to the research and an overview of the thesis including a chapter-by-chapter breakdown.

Chapter 2: Begins with an overview of the physics of radiographic imaging. Next the details of recording X-ray information for both static and dynamic radiography are presented. This leads to the issues of radiographic noise and information content, and finally the idea of the

application of radiography to spinal motion analysis and cineradiographic image segmentation are discussed.

Chapter 3: This expansive chapter investigates approaches to detecting edges in cineradiographic images of the human spine. It is broadly divided into three sections. The first section introduces low-level approaches to edge detection and then evaluates their performance (using a developed quantitative measure) on cervical spine cineradiographic images. This approach is then mirrored in the second section for high-level edge detectors. In the last section an alternative high-level method is developed and evaluated. The chapter concludes with a summary of results/difficulties in edge detection and points towards an alternative solution.

Chapter 4: An extensive overview of approaches to motion-tracking beginning from the idea of visual perception, then introducing fundamental definitions and theoretical constraints. The two main themes covered are matching approaches followed by optical flow techniques and how they can be adapted to radiographic imagery. Finally, based on theoretical and practical considerations a match based approach to motion-tracking is chosen.

Chapter 5: At the core of template-based matching approaches to motion-tracking is a *best-match* measure. This chapter develops a feature-point selectivity measure and then uses it to evaluate the performance of the four match statistics in both intra- and interframe matching of cineradiographic images. Based on feature selectivity consistency, match stability and computational cost considerations, the best match statistic is chosen.

Chapter 6: This chapter considers the issue of how to test a motion-tracking algorithm on cineradiographic sequences of the human spine when the motion to be measured is unknown. It covers the development of a digital image warping based animation system for the generation of test sequences with prescribed motion.

Chapter 7: In this, the penultimate chapter, practical, implementation issues of a motion measurement system are considered and a number of solutions proposed and implemented. A full description of the motion-tracking algorithm is presented and theoretical performance constraints developed. The algorithm is then extensively tested on a wide range of prescribed motion cervical spine sequences.

Chapter 8: This chapter concludes the research with a discussion of the motion-tracking algorithm developed and how it can be improved by the addition of adaptive elements based on material developed in the early chapters of this thesis. The chapter finally looks to the future by considering how a measurement system could be used to diagnose and characterise spinal dysfunction.

In the later chapters of this thesis a number of figures have been produced from a software programme called CineMetriX. The author was responsible for the supervision of the development of this programme and in particular the design of the user interface and motion tracking algorithm. CineMetriX was written in conjunction with the Whitehead Chiropractic and Research Clinic, Palmerston North, New Zealand. This programme is not currently available in commercial release.

An enhanced version of CineMetriX was developed by the author during the writing of this thesis that incorporates many elements not in the original programme.

Appendices: Not normally considered to be a core part of a thesis but in this case they have provided the author with an opportunity (within the constraints of printed material) to give the reader an experience of two prescribed motion cineradiographic sequences. Turn to the first page of appendix C (marked by the single coloured page at the back of the thesis) for reader interaction instructions.

1.4 References

- [1.1] **Roentgen, W.C.** *Annus Physics Leipzig*, Vol. 64, No. 1, 1898.
- [1.2] **Reynolds R.J.** *Cineradiography by the Indirect Method*. Journal of Radiography, Vol. 31, pp 177..182, 1938.
- [1.3] **Johnson B.M.** *Measurements in Skeletal Radiography*. Wiley, pp 180..195, 1977.
- [1.4] **Arnold C.F. and Green J.** *Standard Radiography*, Medical Press, 1978.
- [1.5] **Dimnet J. Pasquet M.H., Krag M.H. and Panjabi M.M.** *Cervical spine motion in the sagittal plane: Kinematics and geometric parameters*. Journal of Biomechanics, Vol. 15, pp 959..969, 1982.
- [1.6] **Fielding J.W.** *Normal and selected abnormal motion of the cervical spine from the second cervical vertebra to the seventh second cervical vertebra based on cineroentgenography*. Journal of Bone Joint and Surgery, Vol. 47a, pp 1779..1781, 1964.

The following is a list of papers by the author that have previously been published based on elements of the research presented in this thesis.

- [1.7] **Page W.H and Monteith W.** *Edge Detection Based on Weber's Law Revisited*. Proceedings of the Image and Vision Computing Conference of New Zealand (IVCNZ '94), pp 2.3.1-2.3.5 August 16-17, 1994.

- [1.8] **Page W.H, Monteith W. and Whitehead L.** *Dynamic Spinal Analysis: Fact or Fiction?* Journal of the Australian Chiropractic Association (JACA), pp 82..85, September 1993.
- [1.9] **Page W.H. and Monteith W.** *Creating Deterministic Motion Using Digital Image Warping.* Proceedings of Digital Image Computing: Techniques and Applications Conference (DICTA'93), pp 485..492, Sydney Australia, 1993.
- [1.10] **Page W.H. and Hoogeeven R.M.** *An Analogue Video Preprocessor.* Proceedings of the Image and Vision Computing Conference of New Zealand (IVCNZ'93), pp 415..421, August 1993.
- [1.11] **Page W.H and Monteith W.** *Bone movement analysis from computer processing of X-ray cinematic video images.* Proceedings of the 4th IEE International Conference on Image Processing and Its Application. pp 381..384, April 7-9, Maastricht, The Netherlands. IEE London, 1992, ISBN 0 85296 543 5.
- [1.12] **Page W.H and Monteith W.** *Weber's Law and its Application to Dynamic Thresholding in Edge Detection.* Proceedings of the 6th New Zealand Image Processing Workshop, pp 71..76, August 1991.
- [1.13] **Page W.H and Monteith W.** *A system for the Analysis of Bone Motion in Cinematic X-ray Images.* Proceedings of the 28th National Electronics Conference (NELCON 91), pp. 75..80, August 1991.

X-ray Radiography

2.1 Radiographic Images - An Overview of The Physics

X-rays are a form of ionising electromagnetic radiation having wavelengths in the range of 0.01 to 1 nano metres (nm). Artificially generated X-rays are generally produced by bombarding a metal target with a stream of high energy electrons. X-rays travel in straight lines, are not affected by electromagnetic fields, have the ability to penetrate matter opaque to visible light, and have similar action to light on photographic media. The penetrating ability of an X-ray beam depends on its wavelength and the density of the matter through which it passes.

According to quantum theory electromagnetic radiation is not continuous but occurs in small 'packets' called *quanta*. The energy of the smallest quantity associated with a given phenomenon is called a *quantum* and is described by equation 2.1,

$$E = h\nu = \frac{hc}{\lambda} \quad \text{eq. 2.1}$$

where h is Planck's constant, ν is the frequency of the radiation, c is the speed of light and λ is the wavelength of the radiation. For light the term *photon* is used in place of quantum.

The wavelength of X-rays is commonly quoted in centimetres, nanometres, or Angstrom units (\AA), where $1\text{\AA} = 10^{-10}\text{m}$. The energy of X-rays is usually quoted in terms of the energy of the electrons (measured in *electron-volts* or *eV*) producing the X-rays. One electron-volt is the energy acquired by an electron when it is accelerated through a potential difference of one volt.

2.1.1 X-ray Generation

The method of artificially generating X-rays has changed little since they were first observed by Professor W. C. Roentgen in 1895 [2.1]. Within a year of his discovery that these rays could penetrate a range of opaque materials, a remarkable number of applications had been described. Roentgen's X-ray tube was a gaseous high-voltage discharge tube in which electrons emitted from a hot cathode were accelerated towards an anode which they struck at high velocity. If an electron starts from zero velocity at the surface of the cathode and is accelerated towards an

anode across a potential difference of V volts, on arrival at the anode it will have gained kinetic energy given by:

$$\frac{1}{2}mv^2 = eV \times 10^7 \text{ J} \quad \text{eq. 2.2}$$

where m is the mass of an electron, v is the electron velocity assumed to be small relative to the speed of light and e is the charge on an electron.

When an electron hits the anode and loses energy, X-rays can be produced by two different mechanisms [2.1]. The first involves the sudden deceleration of the negatively charged electrons as they interact with the strong positive electric field of the nucleus. The energy lost due to this interaction results in *general* or *Bremsstrahlung* radiation. The minimum wavelength of the resulting quantum of radiation is:

$$\lambda_{\min} = \frac{hc}{eV} \approx \frac{123951}{V} \text{ nm} \quad \text{eq. 2.3}$$

Usually the electron gives up only a small amount of its kinetic energy during the interaction so that a continuous spectrum of energies is produced. The majority of the electron energy does not produce x-rays and is converted directly into heat.

The second mechanism for X-ray production occurs when an electron with sufficient velocity hits a target atom and knocks out an orbiting electron leaving the atom in an unstable *excited* state. An electron from another near-by orbital will tend to jump into the vacant orbital to restore stability. The energy difference between the two orbitals is emitted as a quantum of X-rays. Quantum physics [2.2] shows that the energy levels of orbitals are discrete with the highest energy being in the K-shell, closest to the nucleus, where it is most tightly bound. X-rays emitted due to the difference in orbital energies is called *characteristic* or *line radiation*. Most of the X-ray radiation emitted from the interaction of an electron beam with a target anode is of the low energy general radiation type.

The quality and quantity of X-rays produced by an X-ray tube is defined by two factors. The first is the number of electrons flowing across the anode-cathode gap. This is determined by the tube current usually specified in *milliamperes* (mA). The value of this current effectively determines the number of electrons interacting with the target anode and hence the quantity of X-ray production. The maximum velocity the electrons obtain on collision with the target anode determines the maximum energy of the produced X-ray quantum. This is controlled by the applied potential difference between the cathode and anode and is usually specified as the kV_p or kilovoltage potential.

The X-ray beam produced by an X-ray tube contains photons with a wide range of energies. Many of these photons are of low energy and cannot escape the X-ray tube and housing. Also

the generated X-ray photons are not directed in a single direction. This has the consequence that only a small fraction can pass through the X-ray tubes window to produce a useful X-ray beam outside. As the X-ray photons pass through the tube window the mean energy of the beam increases. Low-energy photons are more likely to be attenuated or absorbed by the window material, in effect selectively removing a high percentage of the low-energy photons. This process is referred to as *hardening* of the X-ray beam. Hardening produces a higher quality, less polychromatic, X-ray beam with a more tightly defined range of energies.

X-ray tubes come in two basic configurations. Figure 2.1 shows these two forms.

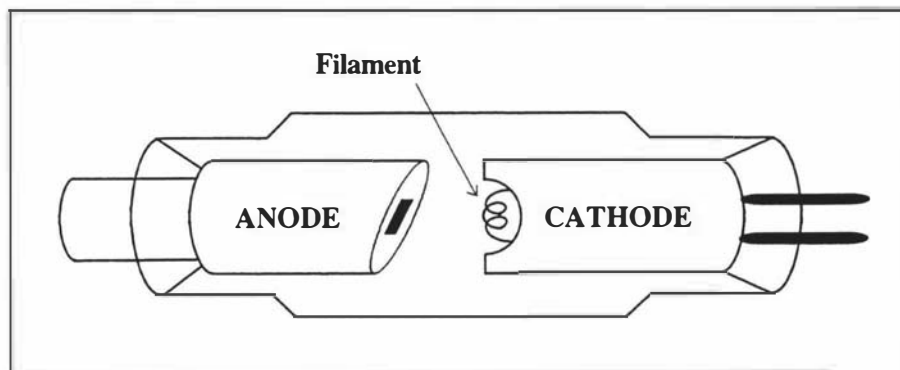


Figure 2.1a: Stationary anode X-ray tube

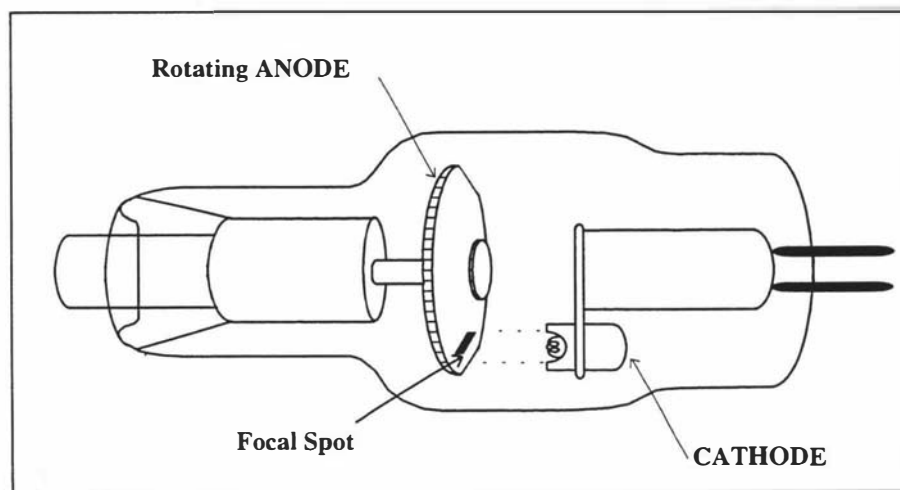


Figure 2.1b: Rotary anode X-ray tube

The only fundamental difference between the two constructions is that in the second form the anode is not stationary but consists of a rotating disk. Typically, in excess of 99% of the energy of the electron beam is converted into heat. This heat must be conducted away from the X-ray tube otherwise it will overheat and become damaged. In the stationary anode configuration of figure 2.1a, the anode consists of a tungsten block bonded to the end of a copper cylinder. The copper is required to ensure rapid heat transfer away from the tungsten block. Maximum output

from an X-ray tube is limited by the dissipation of heat. Stationary anode tubes can dissipate relatively little heat without incurring damage, thus they are usually found only in portable or low-power units where the lack of any moving parts is a significant advantage. The second tube configuration containing a rotating disk anode enables the heat produced by the electron beam to be distributed across the surface of the disk as it rotates thereby substantially reducing hot spots on the anode surface. Heat still has to be removed, but not as rapidly as in the static configuration since the active anode area is continuously being changed, so allowing time for the unexposed surface to recover. The overall effect of having a rotating disk anode is that higher currents and voltages can be applied to the tube for longer periods producing higher output X-ray flux.

In most X-ray systems additional filtration is used to further harden the X-ray beam and hence increase the effective energy of the X-rays. This is achieved by the introduction of a thin sheet of aluminium in the path of the beam. Filtration is usually measured in millimetres (mm) of aluminium equivalents. Most manufacturers recommend a total filtration equivalent to 2.5 mm. This total filtration includes the equivalent thickness due to the X-ray tube window, housing and face plate. After filtration typically only 2.4% of the original beam remains. The net benefit of filtration is that the total X-ray dose to a patient is reduced by almost 80% compared to the non-filtrated case.

A device that controls the area exposed to the primary X-ray beam is referred to as a *collimator*. Beam collimation decreases the X-ray exposure to the patient and operator while improving radiographic quality by decreasing the amount of X-ray scatter. They are available in several different forms but the most flexible and widely used collimator consists of adjustable lead shutters. Usually the horizontal and vertical shutters can be moved independently of each other, enabling non-square areas to be covered. Figure 2.2 shows a box type four-shutter collimator.

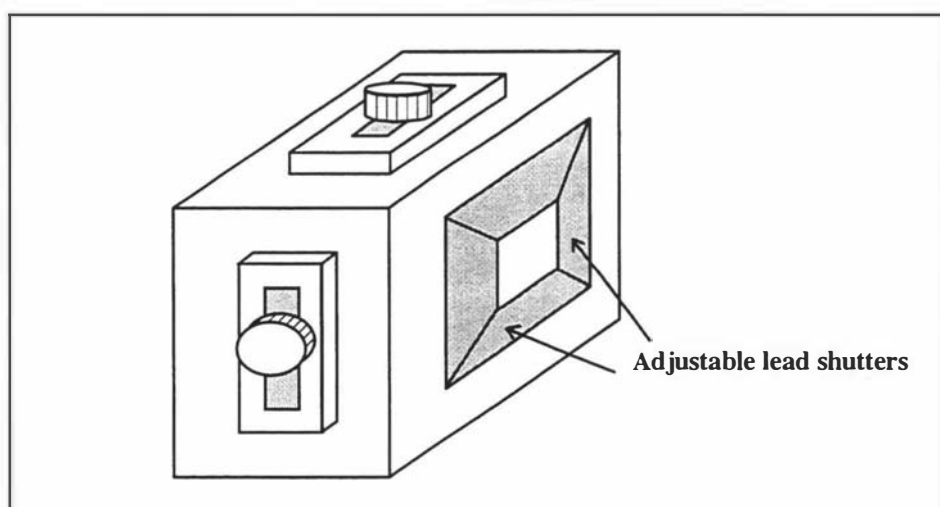


Figure 2.2: Box, four-shutter collimator

2.1.2 X-ray Absorption

As an X-ray beam passes through matter a portion of the energy of the X-ray beam is absorbed. This absorption process is not simple but is made up of a number of mechanisms. The process may result in the production of secondary radiation of variable energies that may not be in the same direction as the primary X-ray beam. In practical radiography both X-ray absorption and scattering must be considered.

Four main types of X-ray absorption occur in general radiography, they are:

- Photoelectric scattering
- Rayleigh scattering
- Compton scattering
- Pair production

In medical diagnostic radiology where X-ray energy levels are typically 100 kV_p, only the first three types of absorption occur. The general law governing the absorption of ionising radiation as it passes through matter can be stated as: *'the fraction of radiation absorbed in passing through a thin layer of material is proportional to the thickness of the layer and an absorption coefficient'*. This is mathematically described in equation 2.4.

$$\frac{I_0 - I}{I_0} = \frac{\Delta I}{I_0} = \mu \Delta x \quad \text{eq. 2.4}$$

where I_0 is the incident intensity, I is the emergent intensity, Δx is the layer of material thickness and μ is the absorption coefficient.

To find the total absorption in passing through a homogenous material of thickness x , the above equation is integrated to give:

$$I = I_0 \exp(-\mu x) \quad \text{eq. 2.5}$$

This is the standard X-ray absorption equation in which the absorption is exponentially proportional to the thickness of the material. The thicker the material the greater the absorption. Due to the four different types of absorption that may occur, the value of the absorption coefficient μ , is dependent in a non-linear way on both the X-ray energy and the nature of the absorber.

2.2 Recording of X-ray Radiation

If an X-ray beam passes through an object that may contain internal cavities and other inhomogeneities there will be local variations in the spatial intensity of the emergent beam. The

emergent beam will be modulated by these internal properties and hence contain information about the internal structure. However X-rays are not directly perceivable by the human senses thus the information contained in the X-ray beam must be converted to a form that can be readily observed. A number of detecting mediums are available, they all depend on a secondary interaction of the X-rays with matter, to render the information.

The first and most widely used method for X-ray detection is through the *photographic effect*. In this method the information is recorded directly as a variation in silver deposition on a processed film. The film is full size and provides a permanent record, the radiograph. However in general this will contain only a record of the relative spatial variation of the X-ray intensity. The radiographs greatest advantage is that it is an integrating device. If sufficient exposure time is allowed the radiograph can record very low radiation fluxes. The object must be essentially stationary for the exposure period, however for biological material this means that the all important peak radiation dose can be substantially reduced.

The second method of X-ray detection is to observe the visible fluorescent light emitted by a particular substance when exposed to an X-ray source. The substance is usually formed into a thin screen that is placed at the same location as a photographic film. The two disadvantages of fluorescent screens are that they do not provide a permanent record of the information and they do not integrate the exposure. The response of a screen is dependent purely on the X-ray intensity and not the overall quantity. However the lack of integration can be an advantage as it allows an object to be X-rayed when in motion and the results viewed.

The third method is to use the fluorescent screen as a primary converter (X-rays to light) and use a video camera to image the screen. The output of the camera can then be displayed on a video monitor (CRT) allowing the operator to be out of the line of the primary X-ray beam. If the video signal from the camera is taken through an analogue-to-digital (A/D) converter, the information can be stored permanently in digital form in a computer.

Other methods enable X-ray radiation to be detected and measured, however most of these methods only allow point-by-point measurement to be made across the X-ray beam. An exception is the semiconductor detector in a matrix array [2.3]. Each element in the semiconductor array produces an electrical signal proportional to the integrated X-ray intensity at its location. These signals can be digitised by an A/D converter in a computer and viewed as an image. The digital image may then be stored on disk for later processing.

2.2.1 Static Radiography

An X-ray film consists of a thin base of acetate or polyester, which is transparent and flexible [2.4]. This is coated with a radiation-sensitive emulsion, usually a silver halide. The emulsion is attached to the base by a thin adhesive layer and the whole surface coated with a layer of gelatine

as a protective '*supercoat*' to minimise abrasion. In order to obtain a useful photoelectric effect from X-ray radiation, the emulsion is much thicker than for light photography, and an emulsion is applied to both sides of the base, as shown in figure 2.3.

Only a very small fraction of the X-ray radiation falling on a film is absorbed. Most of it passes through the film without interaction. However if a metal foil such as lead is placed in contact with the back emulsion of the film, then electrons ejected from the foil due to the X-ray interaction can enter the emulsion and assist in the formation of the latent image. This is the principle of metal intensifying screens [2.5].

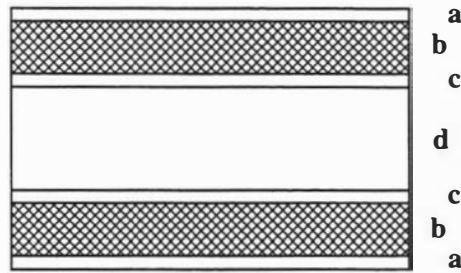


Figure 2.3: Radiographic film construction

a, emulsion protection layer; b, emulsion; c, adhesive substrate; d, base layer - polyester.

If a film of uniform blackness is illuminated by a light source of intensity I_0 and the transmitted intensity through the film is I_t , then the photographic density D of the film is defined by:

$$D = \log_{10} \left(\frac{I_0}{I_t} \right) \quad \text{eq. 2.6}$$

where the ratio $\frac{I_0}{I_t}$ is the optical opacity and its inverse is the transmittance of the film.

Practical measurement of D depends on the grain size and distribution of the silver deposition on the film, and on the incident light. If D is plotted against the logarithm of the exposure (integrated intensity) to radiation (light or X-rays), then a characteristic curve for the film is produced. The characteristic curve of a film has three well-defined regions as illustrated in figure 2.4. With no exposure a small density is produced on development. This density is the fog level of figure 2.4. The fog density is produced by two factors; the inherent density of the base film and a chemical fog density due to the fact that some of the silver grains are capable of being developed even without exposure.

As the exposure is increased from zero, the density slowly increases from the fog level, eventually reaching a region in the middle of the curve where the response is approximately

linearly proportional to a logarithmic change in exposure. A further increase in exposure causes the rate of increase in response to roll off as the chemical reaction nears completion. For very large exposures a maximum density D_{\max} is reached.

In the linear region the curve equation is usually given as:

$$D = \gamma [\log_{10}(E) - \log_{10}(i)] \quad \text{eq. 2.7}$$

where γ is the slope of the approximately straight portion, and $\log_{10} i$ is the intercept of this portion extrapolated to the fog density level.

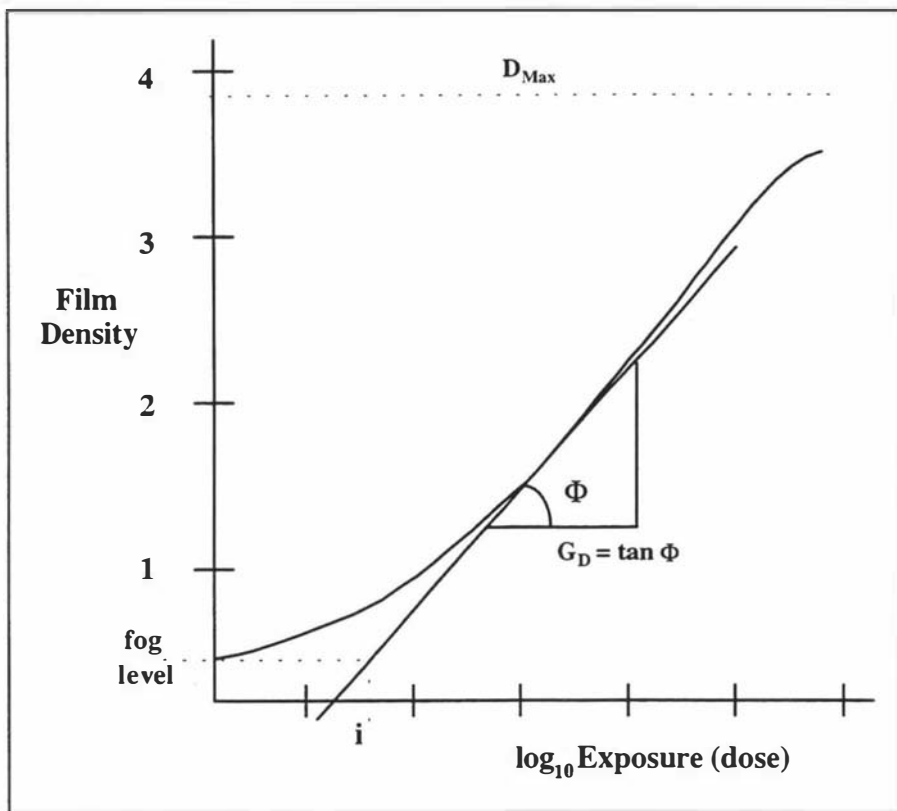


Figure 2.4: Film characteristic curve for X-ray exposure

The slope or gradient G of a tangent line to any part of the curve is often termed the film contrast. Measured at a density D , this gradient is given by:

$$G_D = \frac{d(D)}{d(\log_{10} E)} \quad \text{eq. 2.8}$$

In reality the contrast gradient varies with both exposure E , and film density D . The value G_D is a fundamental parameter in film radiography as it provides a measure of the available film contrast for a specific density. The characteristic curve of a film is also greatly affected by the

development process. A typical system for the development of radiographic films is optimised to give maximum speed and contrast with a minimum of increase in fog density. For certain types of film emulsion the linear portion of the curve can cover a range as large as four densities, equivalent to a four-decade range in exposure.

The finest detail that can be recorded on a film is dependent on a number of factors. One of the most important is the size of the silver grains in the emulsion. Generally, if the grains are smaller then finer detail can be resolved. Prior to exposure the halide grains that form the emulsion are well defined in shape and too small to be visible to the naked eye. Once exposed and developed, the resulting silver metal grains are filamentary in structure and in all orientations. The effect, when viewed, is that the silver deposit is non-uniform. This subjective impression is called *graininess* and is due to the statistical variation in density of grains per unit area and the overlapping of conglomerated grains. Graininess is affected by the developer composition and for a given emulsion increases with radiation energy.

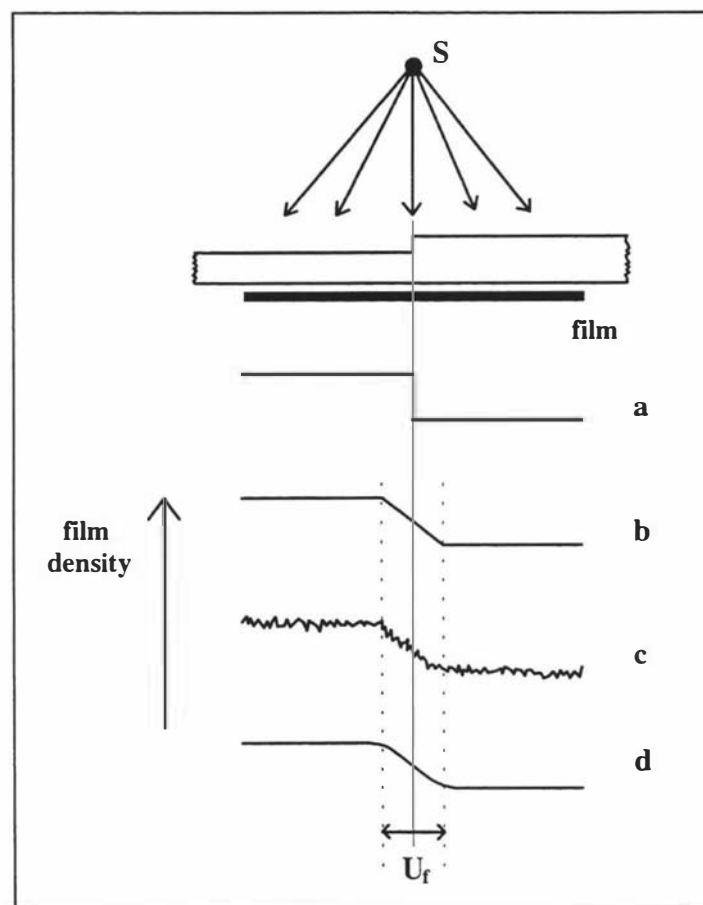


Figure 2.5: Inherent film unsharpness, U_f

The second factor that limits image resolution due to X-ray exposure is the generation of secondary electrons. A silver grain on the absorption of a primary X-ray photon may release a

new or secondary photon of X-ray energy. This secondary emission then sensitises adjacent grains that may then be rendered on development into small string of gains. The two-dimensional effect is to blur a point image into a disk. This type of loss of resolution is termed inherent unsharpness, U_f . At the 100 kV_p X-ray energy level typically used for biological diagnostic radiography, about five grains are made developable for each photon absorbed in the emulsion. Below 33 kV_p no appreciable secondary photon emission occurs.

Theoretically the image of a sharp edge in an object would be as shown in figure 2.5a. Practically however the density variation across the image of a real edge will be of the general shape of figure 2.5(b,c,d), where the width of the band of density change is a measure of the film unsharpness. At low film densities the measured unsharpness is similar to that of figure 2.5d, a smoothed version of figure 2.5c. At this level the slow rolloff at the ends of the transition makes it difficult to make a precise measurement of unsharpness. For 100 kV_p diagnostic radiography, the value of unsharpness is approximately 0.05mm, rising to 0.09 mm at 200 kV_p.

Two further causes of radiographic unsharpness are

Geometric unsharpness, U_g :

arising from the finite size of the X-ray source, which is never a true point.

Movement unsharpness, U_m ;

due to the relative movement between the specimen and the film.

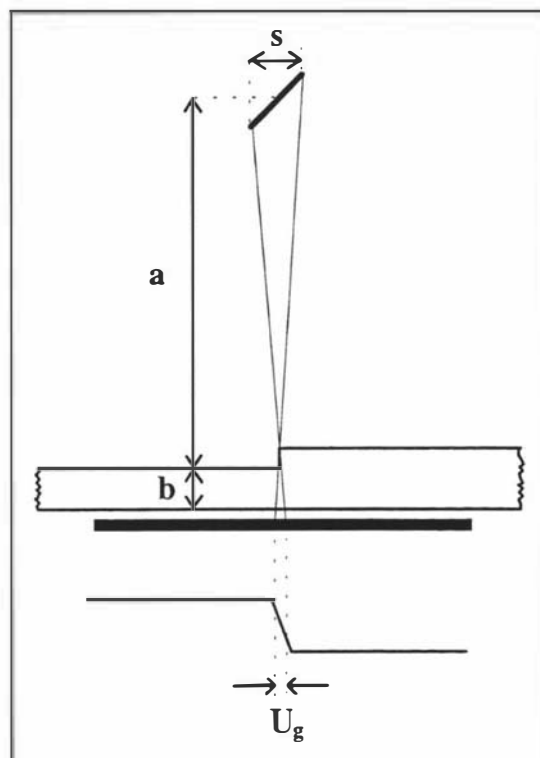


Figure 2.6: Geometric unsharpness

The value of the geometric unsharpness can be specified from the simple geometry of figure 2.6 as:

$$U_g = \frac{sb}{a} \quad \text{eq. 2.9}$$

where s is the width of the radiation source seen at the film, b is the distance from the object to the film and a is the distance from the source to the object.

Assuming that the radiation source is uniform in intensity over its area, the latent image on the film will become progressively wider as the source size increases.

Figure 2.7 shows the effect of increasing the source width from an ideal point source to a width much larger than the defect under inspection. Image contrast C decreases substantially from the reference level C_0 for a point source with increase source width. For diagnostic radiography at 100 kV_p and with a typical source width of 5mm, the value of the geometric unsharpness is approximately the same as the film unsharpness, or about 0.05 mm.

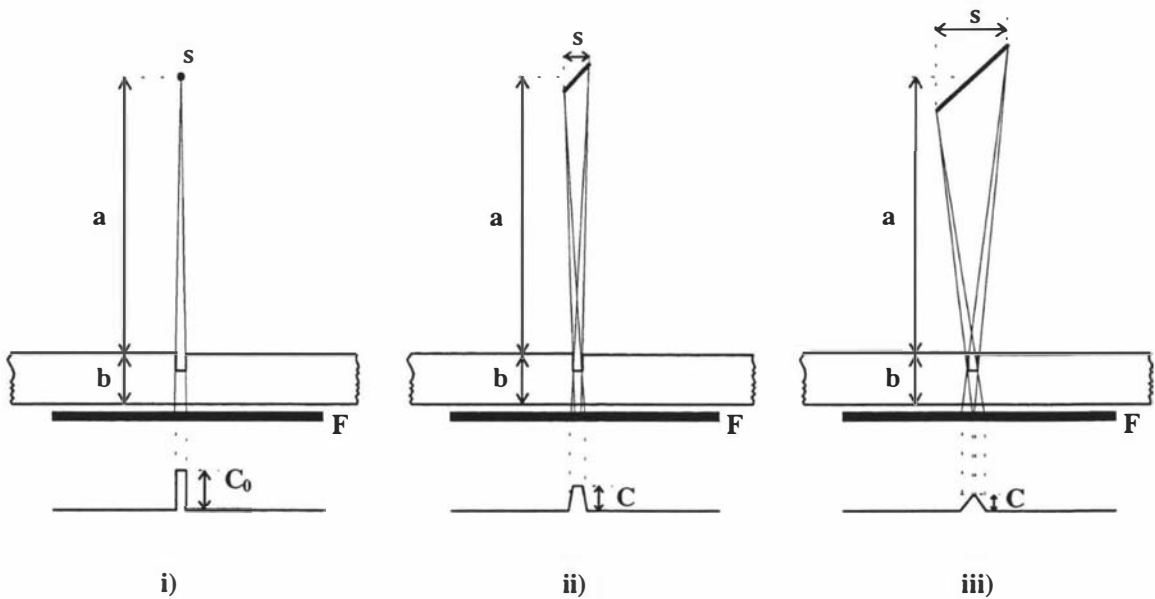


Figure 2.7: Geometric unsharpness for increasing source width - contrast C is reduced

2.2.2 Dynamic Radiography

The principle advantage of film radiography, exposure integration, becomes a major disadvantage when it is desired to radiograph an object in motion. The earliest successful work on motion radiography or *Cineradiography*, was performed by Reynolds in 1921 [2.6]. In his system a 35mm movie film was used to record the moving X-ray images. In order to minimise motion blur while still enabling an adequate image to be recorded, a very high X-ray intensity was required compared to normal radiograph levels. The net effect of this high X-ray level was that the total integrated exposure for even a few seconds of motion, was extremely high. The

detrimental effects of a high X-ray dose on biological material were well established by this time, hence the technique was not used in general medical practice.

However visualisation (viewed by a human eye, but not recorded) of X-ray motion sequences of biological material could be performed with relative safety using fluoroscopy. Fluoroscopic methods rely on the replacement of the photographic film with a thin screen of material that emits visible light (*fluoresces*) when exposed to an X-ray source. Thinner, less-absorbent parts of an object under inspection are seen as brighter areas on the screen, so the normal tonal range is reversed compared to a film radiograph. The fluoroscopic screen does not integrate the radiation, so no motion blur occurs. Apart from the obvious limitation of not producing a permanent record, the major disadvantage of fluoroscopy over film radiography is poor image detail and sensitivity. There are three main reasons for this loss of quality:

- 1) The image produced by a fluoroscopic screen is very dim compared to the brightness of a film viewed on a light box. Available luminance levels are typically 100 times lower ($0.003\text{--}0.3\text{ cd m}^{-2}$). At these very low levels, the human eye even when fully dark-adapted, cannot perceive the small contrast or fine detail that is discernible on a film radiograph.
- 2) Fluoroscopic screens are designed to produce the brightest possible images. The resulting image is generally very coarse-grained and incapable of resolving fine detail.
- 3) Fluoroscopic screens have a unity contrast gradient, whereas film has a contrast gradient of 4-6 at normal densities. The effect of this is that in film recording small differences in X-ray intensity across an object are enhanced by this factor.

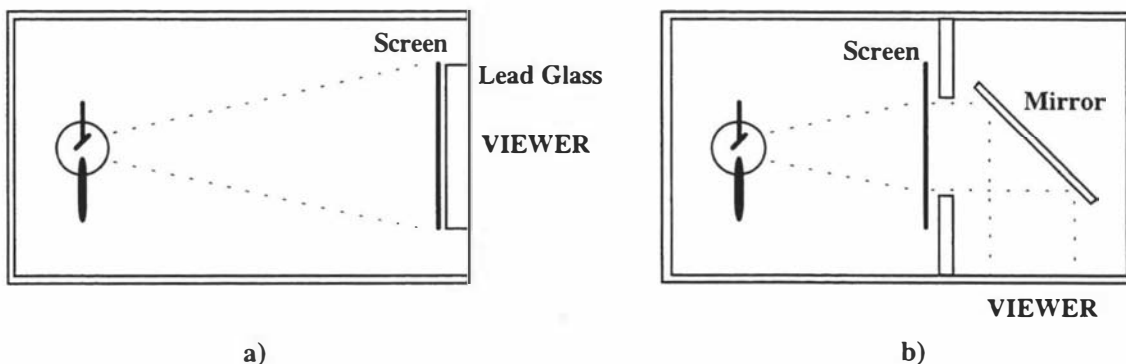


Figure 2.8: Typical fluoroscopic configurations

Only a small fraction of the X-ray photons are converted into light on passing through a fluoroscopic screen. Thus viewing the screen directly can result in significant X-ray exposure to the operator. Two standard ways to minimise this exposure are shown in figures 2.8 a) and b).

Both systems are encapsulated in a lead box. The first system has a lead glass sheet placed in front of the fluoroscopic screen. At the typical 100 kV_p energy levels used with fluoroscopy, a 30mm sheet of lead glass provides effective attenuation while still providing adequate transparency. The second system of figure 2.8b uses a silver-surfaced mirror of approximately the same size as the screen.

Although the use of fluoroscopic screens enables the X-ray visualisation of moving objects, the total X-ray dose is still very large because of the need for a high output (large mA) X-ray tube in order to produce an adequately bright image. A number of methods have been devised to amplify the light output of a screen. The first developments were electronic X-ray intensifier tubes. In these devices an image from the output of a fluorescent screen positioned at one end of a tube is converted successively to light and then electrons. These electrons are then focused onto a smaller screen at the other end of the tube where the image is reconverted back into light. Figure 2.9 shows such a system, where all elements are encapsulated in a vacuum envelope. The process of reduction in size and electron acceleration produces a significant increase in image brightness. The output image can be between 300 and 1000 times brighter than a simple fluoroscopic image. The first of these tubes was made in 1948, both in Holland [2.7] and the United States [2.8], with a primary or input screen diameter of 5 inches (approx. 127mm). Tubes today are available with diameters up to 325mm (13 inches), enabling relatively large objects or sections of an object to be imaged.

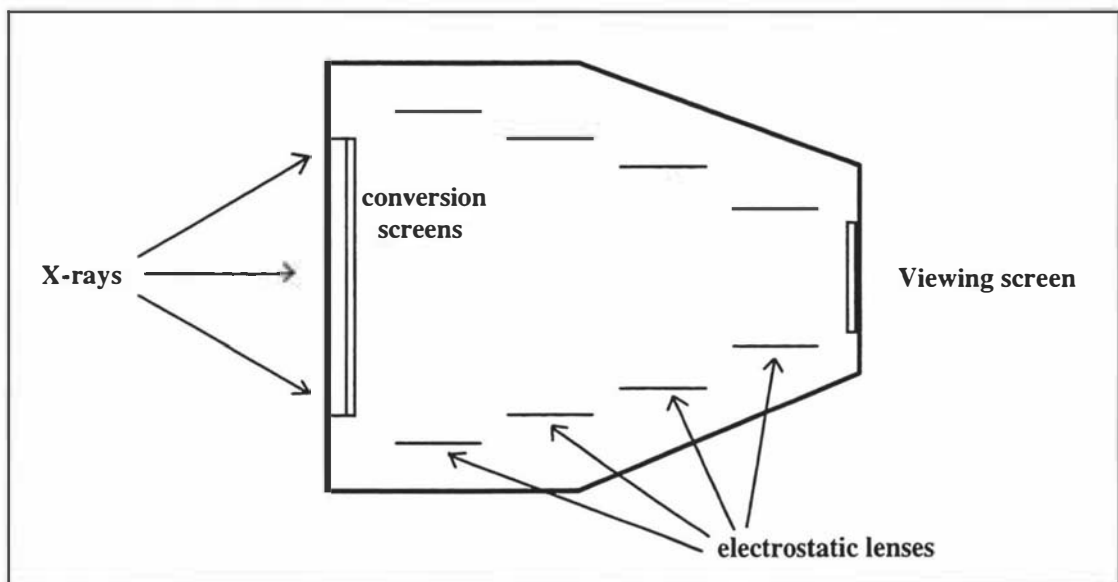


Figure 2.9: Simplified X-ray image intensifier tube

The final output or viewing screen is typically only 60mm in diameter. The image produced at this screen is bright enough to be viewed under ordinary lighting conditions. However because of its small size, almost all practical systems use a video camera to view the image and display

the results on a video monitor (cathode ray tube, CRT). This has the added advantage that the operator can be well out of the line of the primary X-ray beam.

The primary screen of the image intensifier tube consists of two layers back-to-back. The first layer converts the X-ray image into light in the blue and ultra-violet part of the spectrum. This matches the characteristics of photoelectric layer which performs the light-to-electron conversion. The electrons are then accelerated by a high potential (typically 30kV) and focused electrostatically onto the viewing screen. The viewing screen is a small disc on which a very thin layer of very fine grain phosphor is deposited. This results in a high conversion efficiency and good resolution.

Image sensitivity and resolution attainable with modern X-ray image intensifier systems are in many cases better than direct fluoroscopy. Using high-sensitivity video cameras the net gain in intensity can be as large as 10,000. With this high gain, the total X-ray dose for a ten-second viewing of a section of the human spine in motion, can be as little as one quarter the dose required for a standard chest radiograph [2.9]. If the video signal from the camera is fed to a computer equipped with an analogue-to-digital (A/D) converter, it can be converted into digital form and permanently stored for later retrieval, viewing, and analysis.

2.3 Radiographic Noise

This overall gain obtained by using a video fluoroscopic system does however incur a number of penalties. The relatively simple film radiographic system is replaced by a complex system with many conversion stages. Apart from the high cost of such a system, the increase in resolution has been gained at the expense of reduced performance on low-contrast detail. Each stage of the process adds noise to the original signal and any attempts to produce a sharper image will enhance any noise present. Thus the problem of image quality is reduced to establishing the minimum signal that can be detected in the presence of noise.

The quality of the resultant image will be fundamentally limited by the quantum fluctuations in the number of X-ray quanta utilised in forming the image. Each stage amplifies the previous stages output quanta, adding its own quantum fluctuation. Basic quantum physics shows that the nature of radiation emission (light, X-rays, electrons), absorption and conversion, and the associated natural fluctuations in quanta, is related to the square root of the average number of quanta involved at each stage. The source of statistical fluctuation in each stage of a video fluoroscopic system can be treated as being independent. The average statistical fluctuations at the end of a multistage process is given by:

$$s^2 = s_1^2 + s_2^2 + s_3^2 + s_4^2 + \dots \quad \text{eq. 2.10}$$

where $s_1^2, s_2^2, s_3^2, s_4^2$, etc., are the average fluctuations introduced at each stage referred to the value at the output. In a first approximation it can be shown that:

$$s = g N_a^2 \quad \text{eq. 2.11}$$

where N_a is the smallest number of quanta (photon or electrons) which are utilised at any stage of the complete process, and g is the gain of this minimum quanta stage referred to the output. If N quanta form the final output image then:

$$N = g N_a \quad \text{eq. 2.12}$$

A model derived by Sturm and Morgan [2.10] to describe the overall video fluoroscopic system is:

$$d = \frac{2K}{C} \left[\frac{g}{\pi t N_0} \right]^{0.5} \times 100 \quad \text{eq. 2.13}$$

where d is the diameter of the circular image of the fluorescent screen, C is the image contrast, N_0 is the number of visible quanta leaving the screen per square millimetre per second, t is the storage time of the device viewing the screen, and K is a constant between 3 and 5.

This equation specifies the smallest diameter image of contrast C in terms of the number of utilised quanta. If detail sensitivity is the major concern, it can be shown that for an image intensifier system the optimal overall amplification occurs when the number of quanta utilised at the output device equal to the number of X-ray quanta absorbed and utilised by the primary input fluorescent screen.

2.3.1 Radiographic Characteristics: Frequency domain description

The complete cineradiographic system from source to detector and display can also be modelled as an image transfer process using an imperfect recording system. At each stage of the transfer process the original image is modified due to the limitations in that stage. The recorded or output image for each stage is given by:

$$Signal_{Recorded} = Signal_{In} [M(f)^2 T(f)^2] \quad \text{eq. 2.14}$$

where $M(f)$ represents the transfer characteristics of the radiographic system, principally *geometric unsharpness*; and $T(f)$ represents the recorder characteristics, unsharpness and noise both expressed in terms of spatial frequency f , in lines per millimetre.

Figure 2.10 shows the spatial frequency transfer curve for an ideal and practical radiographic system. In the ideal system all frequencies are transmitted unattenuated. However for the

practical system the response has dropped to half its original value once the frequency has increased to approximately 15 lines/mm. Frequencies above this are rapidly attenuated to the extent that all frequencies above 20 lines/mm are not transmitted.

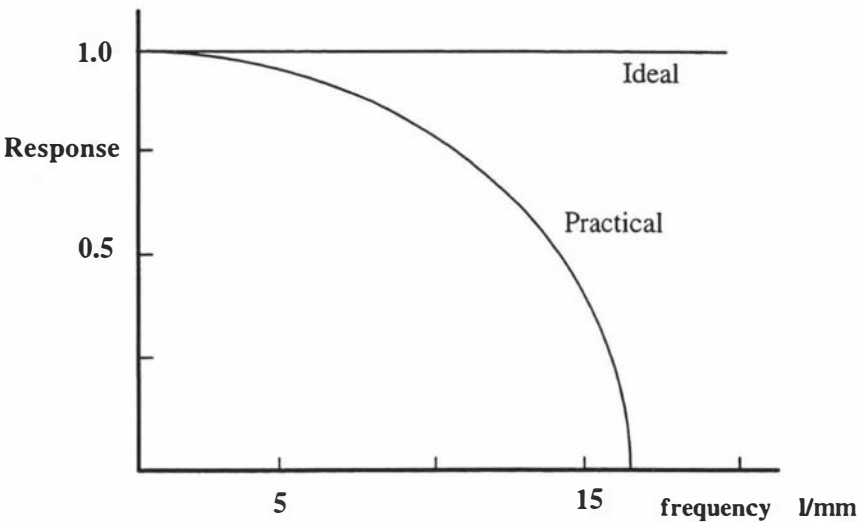


Figure 2.10: Spatial frequency transfer functions - ideal and practical

The basic parameters of radiographic imaging in terms spatial frequency theory are the Line Spread Function (LSF) and the Modulation Transfer Function (MTF). If an X-ray beam passes through a very narrow metal slit and the resulting image is recorded on film, then the film density will be as shown in figure 2.11. This is the line spread function for the setup and characterises the available image definition in a more formal way than was specified in section 2.2.1, relating to unsharpness.

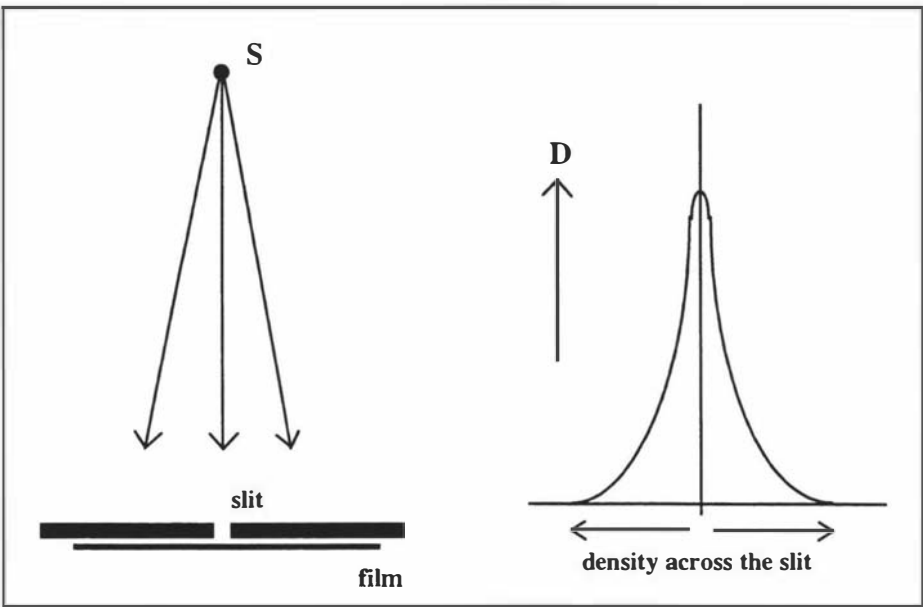


Figure 2.11: Line spread function of a radiographic film

If an object consisting of a series of close regularly-spaced bars is radiographed, for a perfect system the resulting image should be that of figure 2.12a. For practical systems however the image density distribution will be more like figure 2.12b where the edges of the bars have become blurred and the contrast reduced. As the object bar and spacing is reduced, a stage will be reached when the pattern is no longer resolved. The difference in density between the bar and the space on the film is the image response, R , or modulation. The fineness of the pattern is the spatial frequency, f . A plot of the magnitude of R against f is defined as the Modulation Transfer Function, MTF. Strictly, the intensity distribution used to evaluate the MTF should be sinusoidal rather than the square-wave produced by the bar/space test pattern. However practically generating an object with this type of pattern is difficult. If the effects of phase are taken into account by plotting the actual response, not just the magnitude, then the resulting curve is referred to as the Optical Transfer Function (OTF) [2.11].

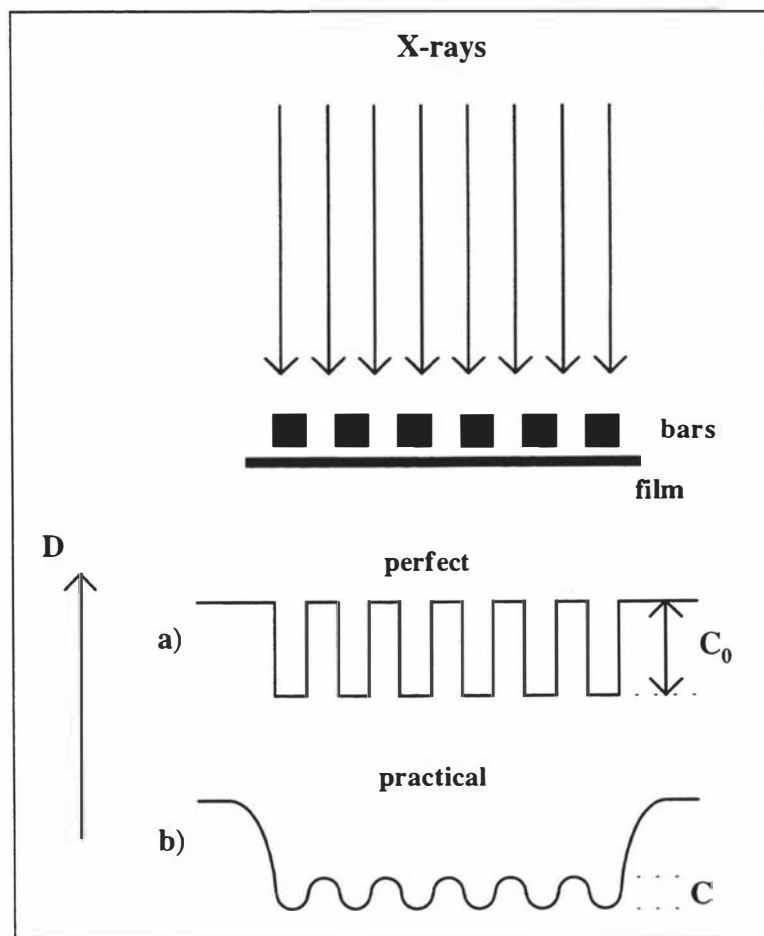


Figure 2.12: Modulation transfer function

Practically in radiography it is easier to determine the MTF or OTF mathematically from the experimental LSF curves by convolution with an appropriate sinusoidal function. If the line spread function $S(x)$, is symmetrical, then the modulation transfer function is given by:

$$R(f) = \frac{\int (S(x) \cos(2\pi f x)) dx}{\int S(x) dx} \quad \text{eq. 2.15}$$

Multiple spread functions in space can be combined together by convolution to produce a joint or overall spread function. If $f(x, y)$ represents the LSF due to geometric unsharpness and $g(x, y)$ the LSF due to film unsharpness, then the joint spread function $h(x, y)$ is:

$$h(x, y) = f(x, y) \otimes g(x, y) \quad \text{eq. 2.16}$$

where \otimes represents the convolution operator.

In the frequency domain the overall spread function can be simply found by multiplying the individual spread functions together. If F, G are the Fourier transforms of the line spread functions due to geometric and film unsharpness, and H is the combined result, all in terms of spatial frequencies u, v , then:

$$H(u, v) = G(u, v) \times F(u, v) \quad \text{eq. 2.17}$$

There is a close relationship between the LSF and the unsharpness curves of section 2.2.1. Plotting the slope of the unsharpness curves against x , gives the LSF curve directly. Conversely integrating the LSF curve will give the unsharpness curve.

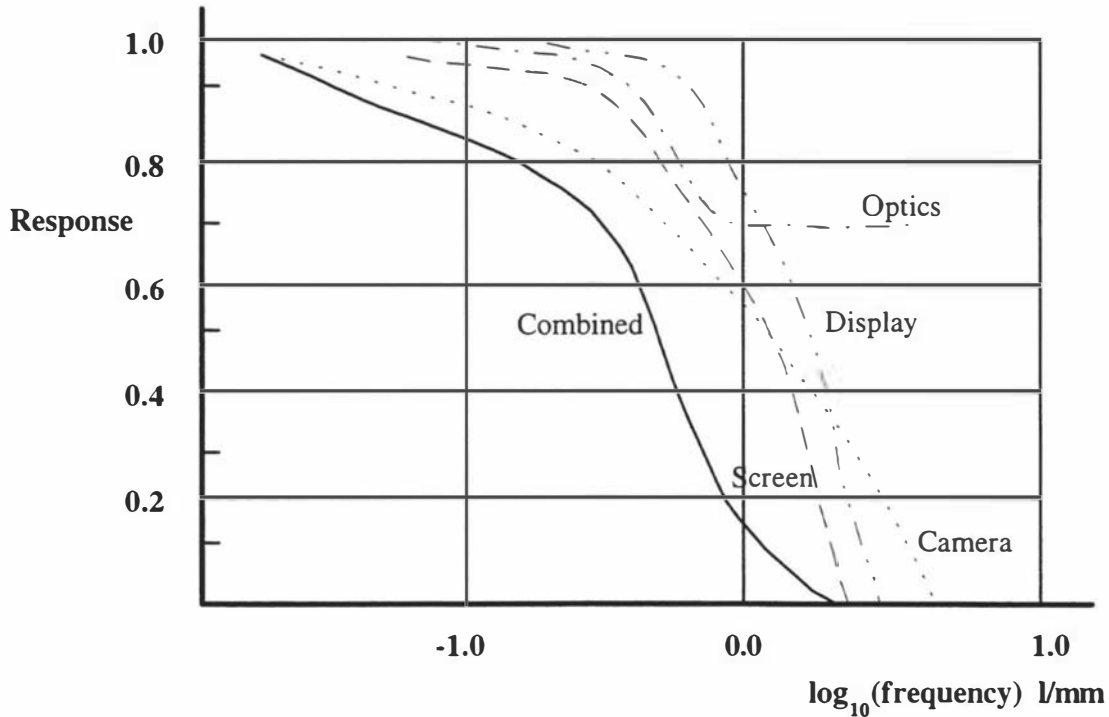


Figure 2.13: Modulation transfer curves for video-fluoroscopic systems

Figure 2.13 shows the modulation transfer function curves for each stage of a complete video radiographic system [2.12]. It is clear from the curves that the limiting component in the system is the primary conversion screen (X-rays-to-light), the optical system has much better performance than is necessary. The curve for the camera is for an older Vidicon unit. Modern charge-coupled devices (CCD) have significantly better modulation transfer functions.

Work by Schade [2.13] attempted to reduce the MTF for a system into a single value 'quality-criterion'. He introduced the concept of *noise equivalent pass-band* (N_e), defined as:

$$N_e = (MFT)^2 df \quad \text{eq. 2.18}$$

and showed that this measure correlated very well with the subjective mean observer scores (MOS) of image sharpness.

2.4 Radiographic Information

Static radiographs potentially contain a large amount of information. The information capacity of a media or communications system can be computed from basic information theory [2.14] by considering the signal-to-noise ratio (SNR) of the system. Several authors [2.15, 2.16] have applied this approach to the characterisation of the information capacity of photographic film. The mean information content, I recorded in an area, A of an emulsion is given by:

$$I = A \iint \left(1 + \frac{S(u, v)}{N(u, v)} \right) du dv \quad \text{bits} \quad \text{eq. 2.19}$$

where $S(u, v)$ and $N(u, v)$ are the signal and noise functions, respectively.

Noise in a photographic emulsion can be characterised qualitatively in terms of granularity, as discussed in section 2.2.1. A quantitative spatial frequency description of the noise properties of photographic film is given by the Wiener spectrum. The Wiener spectrum, $W(f)$, is analogous to the power spectrum of electrical noise in electric circuit theory.

Because photographic emulsions are isotropic, measurements in only one direction are sufficient to completely describe the mean information content:

$$I = 2\pi \int \log_2 \left(1 + \frac{R_0(f)^2 P(f)}{W(f)} \right) f df \quad \text{bits per unit area} \quad \text{eq. 2.20}$$

where $P(f)$ is the input signal to the film, $W(f)$ is the film Wiener spectrum and $R_0(f)$ is the normalised MTF of the system.

This equation is an over-simplification of the process as it does not account for the fact that the input signal consists of a finite number of X-ray quanta with their own inherent signal-to-noise ratio.

A useful description of the upper limit of the mean information content of a film can be made by considering the limiting case where the input signal tends to zero, as would be the case for a low-contrast image. This results in the following description:

$$I \approx 4.36 \int \left(\frac{R_0(f)^2}{W(f)} \right) f df \quad \text{bits} \quad \text{eq. 2.21}$$

Using this equation, the information content of a high quality 5 x 7 inch film is 5×10^7 and for a single video frame 3×10^4 , bits per square millimetre. Practically however, visual inspection has shown that these figures highly over-estimate the available information content. At $70 kV_p$ on high-resolution film the information content is about 1000 bits / mm^2 , while for the highest resolution video fluoroscopic systems the value is about a third of this figure.

2.5 Application of Radiography

In static diagnostic radiography of human or animal subjects, a practitioner is generally looking for both large and small scale abnormalities with varying ranges of contrast. Large scale dysfunction information may take the form of severe vertebral misalignment or a major breakage in a limb. Small scale information may be of the form of a hair-line fracture or a tumour in soft tissue. These diverse requirements greatly reduce the information content of importance in a radiograph however the extraction of this useful or desired information requires a trained radiologist.

A great deal of work has been performed concerning the automatic detection of specific abnormalities in static film radiographs. Much of this effort has been directed at the detection of cancerous tumours. The radiograph is first digitised to a very high spatial and amplitude resolution and then algorithms proceed in the digital domain to locate a specific type of abnormality that may exist. Any located abnormalities are then high-lighted automatically, and a radiologist can visually check to ensure that the located item is in fact a true abnormality. These systems can detect up to 95% of abnormalities, with near 100% repeatability [2.17].

In video cineradiography both spatial and amplitude resolution are substantially lower than static film radiography, thus this method is unsuitable for the detection of small low contrast features. However resolution is sufficiently high to enable the imaging of larger scale information.

2.5.1 Spinal Motion Analysis

Static radiography has been used for almost a century by health professionals in determining vertebral misalignment and degeneration in the human spine. A practitioner manually extracts measurements by drawing lines directly onto the radiograph through well-defined and repeatable points commonly known as 'landmarks'. Typically, as many as fifty measurements are produced from a single full spine radiograph, taking typically up to an hour to perform. Further to this, static radiography of the human spine provides no direct information concerning the kinematics of bone motion. The spine is a structure designed to move during normal operation, hence information concerning how vertebrae interact is vitally important to an accurate understanding and diagnosis of spinal dysfunction. Cineradiography provides a means of recording the spine in normal motion, however even a short sequence may contain many hundreds of images. Attempting to process this large number of images manually and to extract the necessary measurements is totally impractical. Apart from being a very laborious process, repeatability and accuracy is unlikely to be very high due to operator fatigue. Clearly there is a requirement for a system that can automatically and reliably extract the information of interest from each image.

2.6 References

- [2.1] **Roentgen W.C.** *Annus Physics Leipzig*, Vol. 64(1), 1898.
- [2.2] **Goodwin P.N., Quirby E.H. and Morgan R.H.** *Physical Foundation of Radiology*, 4th Edition, Harper and Row Publisher Inc, pp 24..30, 1970.
- [2.3] **Melissinos A.C.** *Experiments in Modern Physics*, Academic Press, pp 28..32, 1966.
- [2.4] **Gorbunov V.I. and Melikhov V.S.** *Defektoscopia*, No. 2, (A summary of semiconductor X-ray sensors), pp 28..30, 1978.
- [2.5] **Juhl J.H. and Gummy A.B.** *Essentials of Radiologic Imaging*, 6th Edition. J.B. Lippincott Company, Philadelphia, pp 10..12, 1993.
- [2.6] **Halmshaw R.** *Industrial Radiology - theory and practice*, Applied Science Publishers, ISBN 0-85334-105-2, pp 122..125, 1982.
- [2.7] **Reynolds R.J.** *Cineradiography by the Indirect Method*. *Journal of Radiography*, vol. 31, pp 177..182, 1938.
- [2.8] **Teves M.C. and Tol, T.** *Philips Technical Review*, No. 14, pp 33, 1952.
- [2.9] **Coltman J.W.** *Journal of Radiology*, No. 51, pp 539, 1948.
- [2.10] Internal Report, *New Zealand Radiation Laboratory*, 1986. Prepared for the Whitehead Research and Chiropractic Clinic.

- [2.11] **Sturm R.E., and Morgan, R.H.** *American Journal of Roentgenography*, No. 62, pp 617..622, 1949.
- [2.12] **Jain A.K.** *Fundamentals of Digital Image Processing*. Prentice-Hall Information and System Science Series, pp 21..22, 1989.
- [2.13] **Halmshaw R.** *Industrial Radiology - theory and practice*, Applied Science Publishers, ISBN 0-85334-105-2, p 297, 1982.
- [2.14] **Schade O.** *Journal of the Society of Motion Pictures, Television and Engineering*, Vol. 73, No. 2, pp 81..84, 1964.
- [2.15] **Shannon C.E.** *Journal of Bell System Techniques*, No. 27, p 623, 1948.
- [2.16] **Fellgett P.** *Journal of Photographic Science*, Vol 11. No. 1, p 31, 1962.
- [2.17] **Kanamori H.** *Japanese Journal of Applied Physics*, Vol 7, No. 4, p 414, 1968.
- [2.18] **Young A.A. and Axel L.** *Automatic Spot Detection in Chest Radiographs*. IEEE Transaction on Medical Imaging, Vol. 12, No. 4, pp 505..515, 1994.

Detecting Edges In Cineradiographic Images

3.1 Image Segmentation

The aim of image segmentation is to decompose a scene into its components. At a high level, these components are ideally the objects that the human observer perceives in the scene. In the case of radiographs of the human spine, the main scene components are the vertebrae (including the skull and the pelvis). Soft tissue such as skin and cartilage may also be perceived, but are generally only of interest when looking for abnormalities such as tumours. For most scenes, the recognition of the objects that form the scene is implicit in the high level segmentation process. Image segmentation at a slightly lower level may involve the decomposition of the scene into planar, convex and concave surfaces. These surfaces are also likely to contain different colours, reflectance properties, and texture. The recognition of different spatial properties and relationships enables a human observer to readily segment the scene on this basis. In many cases this partitioning process can be considered as the identification of local intensity discontinuities. These discontinuities form the boundaries of the objects.

There are three basic approaches to image segmentation:

- *Pixel-based methods*
- *Region-based methods*
- *Edge-based methods*

Pixel-based methods use only the value of the individual pixels. Region-based methods analyse the pixel values in larger areas. While edge-based methods attempt to detect edges and follow them to form the boundaries of objects.

Pixel-based methods

Pixel or point-based segmentation methods use only the value of a pixel to determine whether or not it belongs to an object. In order to do this, it is assumed that a range of intensity values will uniquely characterise the object of interest. This is generally only the case for simple, well

structured scenes in which there is a clearly defined background (generally not of interest) and a foreground, containing the objects of interest. Most pixel-based methods begin by computing a histogram of the pixel values. If the histogram shows a *bi-modal distribution* with two distinct maxima, then ideally a zone will exist between the two maxima where no features exist. A threshold can be set in the middle of this zone to yield perfect separation of the objects and the background. In practice however this is rarely the case, the two distributions will usually overlap, resulting in a range of values that include both foreground and background pixels. Choosing an optimal threshold in this situation (one that minimises the number of false contributions) will usually involve a *trial-and-error* approach from a human operator. The histogram may also contain multiple modes due to uneven illumination or slight differences in the reflectance properties of some of the foreground objects. No single threshold will give satisfactory separation of foreground objects from the background. One way to tackle this situation directly but still maintain the simple point-based methods, is to determine the probability distributions of the foreground objects and the background. Statistical analysis can then be applied to the threshold decision process to minimise the number of erroneous inclusions [3.1]. However if the probability distributions have significant overlap, point-based segmentation will not produce acceptable results.

Region-based methods

Region-based methods classify pixels using context. This can be effective because an important characteristic of an object is its *connectivity*.

All the common approaches to region-based segmentation can be grouped under the common heading of region-growing. The goal of region-growing methods is to use object characteristics to map individual pixels into sets of pixels with similar local properties. Most commonly, geometric characteristics of regions are considered to be connected two-dimensional areas. A wide range of classes of connectivity can be specified, including disconnectivity, non-simple connectivity (holes are allowed) and boundary smoothness. This connectivity may not be restricted to a single spatial resolution. The image may be decomposed into a pyramid structure via a set of rules and regions determined by scanning up through the pyramid [3.2]. In all methods the nature of the rules depends on the goals of the particular technique. A region may be an entire object or simply a part of an object. Higher level rules can be applied to part object regions to combine them to form an entire object. A wide range of region-growing algorithms have been described in the literature [3.3], but a variation on an early method called the *split-and-merge* algorithm [3.4] is often used.

Edge-based methods

Early experiments by Attneave [3.5] on visual perception illustrated the extreme importance of boundaries to image recognition. Many objects may be easily recognised from crude (binary) outline sketches. Devising algorithms to find the boundaries of objects directly from their grey-level values proves to be a difficult task when the shape of the boundaries are complicated. Better results are obtained by first finding all the intensity discontinuities or edges in the image and then through appropriate rules, composing these edges into the actual objects boundaries. This intermediate representation approach serves to take boundaries that are highly model-dependent and decompose them into a series of local edges that are highly model-independent. However if the local edges are too fragmented, then the recomposition back into the actual object boundaries becomes a difficult task.

Many different edge operators have been described in the literature. The reason for the large variety is that different edge operators perform best in different situations. The unifying feature of most of these operators is that they compute an orientation or *direction*, usually aligned with the direction of maximal grey-scale change, and a *magnitude* describing the size of the change.

Edge operators fall into three categories:

- *operators that approximate the mathematical gradient*
- *operators that use multiple masks with different orientations*
- *operators that fit local intensities with parametric edge models*

In general all these methods will involve the setting of some form of threshold or decision parameter in order to determine the presence of an edge.

If in a cineradiographic sequence of the human spine the vertebrae can be identified in each frame by tracing their outlines or boundaries, then the problem of determining their motion is greatly simplified. The difference in position and orientation of each vertebral boundary through the sequence will describe the motion. If this motion is the result of in-plane movement (parallel to the image plane), then as the vertebrae do not deform, their boundaries will not change shape. Determining the difference in position and orientation of two boundaries (closed outlines) with the same shape is relatively straightforward. However if the orientation of each vertebra is to be meaningful with respect to the alignment of the spine as a whole, then the reference orientation for each vertebra should be consistent with the orientation perceived by a spinal-care practitioner. Section 8.2 of chapter 8, details some of the difficulties in computing *natural* or perceived orientation.

The following sections contain a thorough investigation of edge operators with respect to the detection of vertebral boundaries in cineradiographic images of the human spine.

3.2 Low-level Edge Detection Operators

The boundaries of objects are often indicated by local intensity discontinuities. In order to detect these intensity discontinuities a filter is required that emphasises changes in intensity and suppresses areas with constant intensity. Figure 3.1 illustrates that derivative operators perform such an operation. The first derivative shows an extremum at the edge, while the second derivative crosses zero where the edge is at its steepest. Both these criteria can be used in the detection of edges.

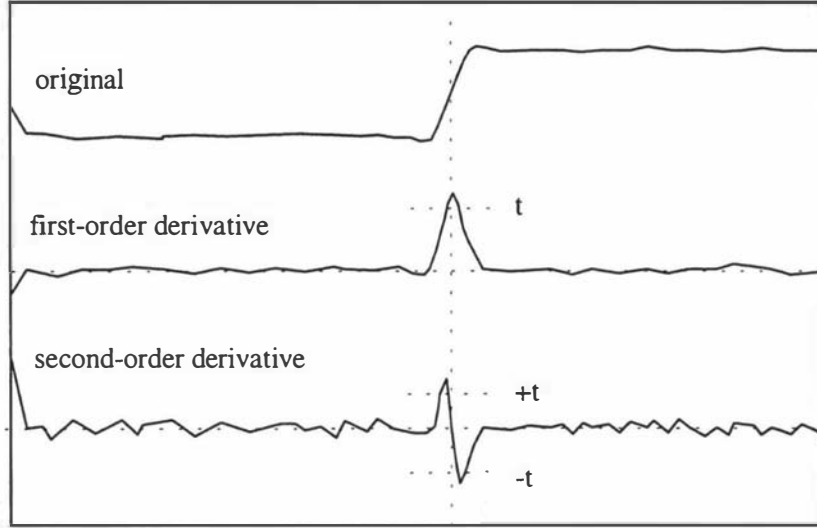


Figure 3.1: Noisy 1-D edge and first and second derivatives

For a continuous image $I(x, y)$, containing a simple intensity edge, the first derivative (or gradient) will reach a local maximum in the direction of the edge. If the gradient of I is measured along r in the direction θ , then this can be used to detect the presence of an edge (see figure 3.2).

Mathematically this gradient is:

$$\frac{\partial I}{\partial r} = \frac{\partial I}{\partial x} \frac{\partial x}{\partial r} + \frac{\partial I}{\partial y} \frac{\partial y}{\partial r} = I_x \cos \theta + I_y \sin \theta \quad \text{eq. 3.1}$$

The gradient takes on a maximum value when, $\partial/\partial\theta(\partial I/\partial r) = 0$, corresponding to the direction n , normal to the edge.

Applying this result to equation 3.1 produces the direction and magnitude equations,

$$\theta_g = \tan^{-1} \left(\frac{I_x}{I_y} \right) \quad \text{eq. 3.2}$$

$$\left(\frac{\partial I}{\partial r}\right)_{\max} = \sqrt{I_x^2 + I_y^2} \quad \text{eq. 3.3}$$

Where θ_g is the direction of the edge.

From this result two types of edge detection operators can be defined: *gradient operators* that use finite differences to approximate the orthogonal gradients I_x, I_y and *compass operators* that approximate the directional gradient $\partial f / \partial r$.

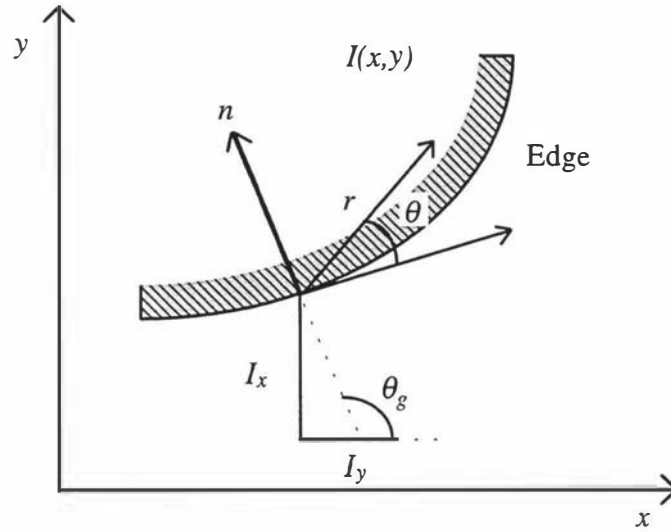


Figure 3.2: Gradient of $I(x, y)$ along direction r

Let \mathbf{H} denote a $p \times p$ mask, then the inner product of this mask and an arbitrary image \mathbf{I} , at location (m, n) is defined as:

$$\langle \mathbf{I}, \mathbf{H} \rangle_{m,n} = \sum_{x=1}^p \sum_{y=1}^p h(x, y) i(x+m, y+n) \quad \text{eq. 3.4}$$

3.2.1 Gradient Operators

Gradient operators are represented by a pair of masks or templates $\mathbf{H}_1, \mathbf{H}_2$ which measure the gradient of an image $I(x, y)$ in two orthogonal directions. At each location in the image the masks are convolved with the local region (inner product) and the results combined to give the gradient vector magnitude.

Defining the two bi-directional gradients $g_1 = \langle \mathbf{I}, \mathbf{H}_1 \rangle_{m,n}$, $g_2 = \langle \mathbf{I}, \mathbf{H}_2 \rangle_{m,n}$ then from equation 3.3, the directional gradient magnitude is given by:

$$g(m, n) = \sqrt{g_1^2(m, n) + g_2^2(m, n)} \quad \text{eq. 3.5}$$

In this situation a pixel location (m,n) is declared an edge pixel if $g(m,n)$ exceeds some specified threshold.

Table 3.1 contains three of the common gradient operators. The origin denoted by $\begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$ is located at the top right-hand corner for the Roberts [3.6] operator, and in the centre for both Prewitt [3.7] and Sobel [3.8] operators.

Operator	H_1	H_2
Roberts	$\begin{bmatrix} [0] & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} [1] & 0 \\ 0 & -1 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & [0] & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & [0] & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Table 3.1: Some common gradient operators

3.2.2 Compass Operators

Compass operators use multiple gradient masks orientated in a selected number of directions to measure the local gradient. If $g_k(m,n)$ denotes the gradient computed in the direction θ_k then the gradient at position (m,n) is given by:

$$g(m,n) = \max_k (|g_k(m,n)|) \quad \text{eq. 3.6}$$

A pixel location (m,n) is once more declared an edge pixel if $g(m,n)$ exceeds some specified threshold.

Kirsch [3.9] was one of the first people to study the use of compass operators when working with biological images. Table 3.2 contains four north-facing compass gradient masks. The mask for the other eight compass directions are obtained by a circular shift of each element about the centre position. Each shift corresponds to a rotation of $\pi/4$ or 45° .

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & [2] & 1 \\ -1 & -1 & -1 \end{bmatrix}$ <p>a)</p>	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & [0] & 0 \\ -1 & -1 & -1 \end{bmatrix}$ <p>b)</p>
$\begin{bmatrix} 5 & 5 & 5 \\ -3 & [0] & -3 \\ -3 & -3 & -3 \end{bmatrix} \text{ (Kirsch)}$ <p>c)</p>	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & [0] & 0 \\ -1 & -2 & -1 \end{bmatrix}$ <p>d)</p>

Table 3.2: Compass Gradients: North ↑ direction

3.2.3 Laplace Operators

Let the *gradient* of a 2-d function f , be defined as:

$$\nabla f = \frac{\partial f}{\partial x} \underline{n}_x + \frac{\partial f}{\partial y} \underline{n}_y \quad eq. 3.7$$

where ∇ denotes the gradient operator and $\underline{n}_x, \underline{n}_y$ are unit vectors in the direction of x and y respectively.

Then the *Laplacian* of f is defined as:

$$\nabla^2 f = \nabla f \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad eq. 3.8$$

where '.' denotes the vector dot-product operation

From equation 3.7 it is clear that the Laplacian is a second-order non-directional (*isotropic*) linear operator that can be formed by a double application of the gradient operator [3.10].

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & [4] & -1 \\ 0 & -1 & 0 \end{bmatrix}$ <p>a)</p>	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & [8] & -1 \\ -1 & -1 & -1 \end{bmatrix}$ <p>b)</p>	$\begin{bmatrix} 1 & -2 & 1 \\ -2 & [4] & -2 \\ 1 & -2 & 1 \end{bmatrix}$ <p>c)</p>
---	---	---

Table 3.3: Discrete Laplace Operators

Because the *Laplace operator* involves second-order derivatives and hence double differentiation, it is more sensitive to noise than the first-order *gradient operator*. Also, direct thresholding of $\nabla^2 f$ produces a double edge (see figure 3.1). For this reason the *zero-crossings*

of the Laplacian are better at detecting edge locations. On a 2-d grid, a zero-crossing is declared if there is a zero-crossing in at least one direction.

Table 3.3 contains three standard 3x3 masks that approximate the Laplacian.

3.2.4 Stochastic Gradients

The previous two edge operators can be shown to perform poorly in the presence of noise [3.14]. Prefiltering the image by some form of low-pass filter can improve their performance, however a better approach is to design the edge extraction masks to take into account the presence of noise in a controlled fashion. Masks designed in this way are called *stochastic gradient* [3.11] masks. Their calculation requires the postulation of an edge and noise model.

Consider an edge model in which the edge transition is only one pixel wide (see figure 3.4). To detect the presence of this type of edge at position (p, q) the horizontal gradient could be computed as:

$$g_1(p, q) = \hat{I}_f(p, q-1) - \hat{I}_b(p, q+1) \quad \text{eq. 3.9}$$

Where $\hat{I}_f(p, q)$ and $\hat{I}_b(p, q)$ are the optimal forward and backward estimates of $I(p, q)$ based on the noisy data over some local region W .

If the noise is assumed to be additive and Gaussian, then the best linear mean-square *semicausal* (*causal* in the n direction) FIR (Finite Impulse Response) estimate [3.12] for $\hat{I}_f(p, q)$ can be computed in the local neighbourhood from the observed values to the left of position (p, q) . Similarly the best estimate for $\hat{I}_b(p, q)$ can be computed from the observed values the right of position (p, q) . All that is required to perform this calculation is a specification of the expected *signal-to-noise ratio* (SNR) local to the edge.

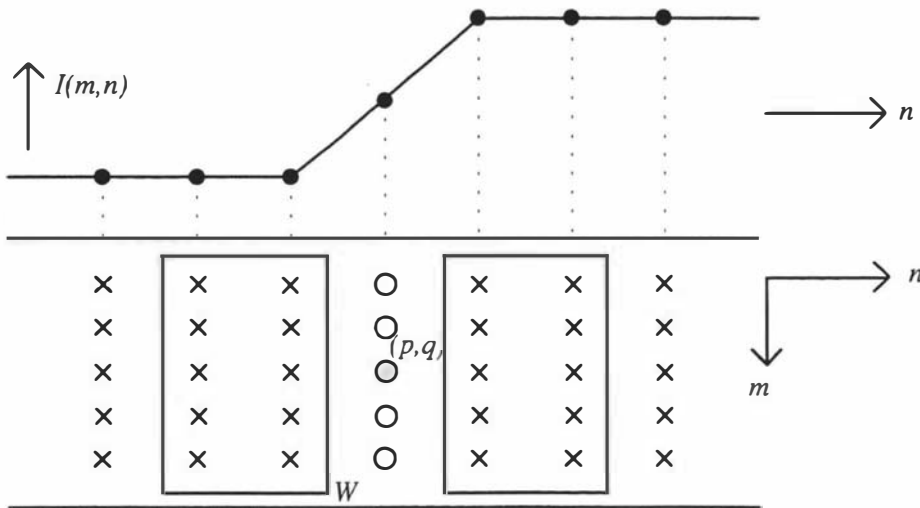


Figure 3.3: One pixel wide edge transition model

Using the linear mean-square FIR estimation, the \mathbf{H}_1 normalised stochastic masks for a low and high SNR are shown in table 3.4. The complementary orthogonal mask \mathbf{H}_2 , is given simply by \mathbf{H}_1^T . For increasing SNR the mask weights decay rapidly, eventually leading to an edge estimate based only on the two values immediately adjacent to the centre position.

Size	SNR=1	SNR=9
3x3	$\begin{bmatrix} 0.97 & 0 & -0.97 \\ 1.00 & [0] & -1.00 \\ 0.97 & 0 & -0.97 \end{bmatrix}$	$\begin{bmatrix} 0.77 & 0 & -0.77 \\ 1.00 & [0] & -1.00 \\ 0.77 & 0 & -0.77 \end{bmatrix}$
5x5	$\begin{bmatrix} 0.80 & 0.84 & 0 & -0.84 & -0.80 \\ 0.85 & 0.90 & 0 & -0.90 & -0.85 \\ 0.87 & 1.00 & [0] & -1.00 & -0.87 \\ 0.85 & 0.90 & 0 & -0.90 & -0.85 \\ 0.80 & 0.84 & 0 & -0.84 & -0.80 \end{bmatrix}$	$\begin{bmatrix} 0.27 & 0.36 & 0 & -0.36 & -0.27 \\ 0.37 & 0.56 & 0 & -0.56 & -0.37 \\ 0.46 & 1.00 & [0] & -1.00 & -0.46 \\ 0.37 & 0.56 & 0 & -0.56 & -0.37 \\ 0.27 & 0.36 & 0 & -0.36 & -0.27 \end{bmatrix}$

Table 3.4: Stochastic Gradient Masks

3.2.5 Performance of Low-level Edge Detection Operators

Comparing the performance of edge detectors is a very difficult task. Some operators may find all edges but respond to noise, while others may be insensitive to noise but miss some important edges. A number of measures have been devised to quantitatively compare the performance of edge detectors in the presence of noise.

Performance Measures

A simple performance measure for edge detection in the presence of noise can be computed from the ratio of the number of declared edges pixels, n_d , to the number of new or missed edge pixels, n_e . So long as n_d is constant for both the noiseless and noisy images, then the edge detection error rate is :

$$P_e = \frac{n_e}{n_d} \quad \text{eq. 3.10}$$

This error measure is too simplistic for realistically comparing edge operators as it does not account for the location of a new edge pixel relative to the nearest true edge pixel. Pratt's [3.13] figure of merit attempts to take into account the effects of true and falsely detected edge points and their relative distance apart. This figure of merit is defined by :

$$P \equiv \frac{1}{\max(N_D, N_I)} \sum_{i=1}^{N_A} \left(\frac{1}{1 + ad_i^2} \right) \quad \text{eq. 3.11}$$

where N_D and N_I represent the number of detected and ideal edge points respectively, a is a scaling parameter (often set to unity), and d is the distance from each detected edge point to the nearest ideal edge point.

Using this performance measure, a review of edge operators [3.14] showed that stochastic operators performed best in the presence of additive Gaussian noise, followed by Prewitt, Sobel and Kirsch operators scoring similarly, and Roberts with the poorest performance.

Pratt's performance measure does not compare well different edge operators when the detected edge segments are highly fragmented. This limitation prompted Kitchen and Rosenfeld [3.15] to develop a measure that provides information about the thinness and continuity of the edge segments. However their measure has not been widely used due to its poor discrimination. This poor discrimination has been illustrated by several authors [3.16], where a fairly high score for the continuation measure was recorded for unrelated edge images.

Low-level Edge Detection Results

Figure 3.4a contains a single frame from a cineradiographic sequence of the cervical section of the human spine. A vertical section of this image centred about the vertebrae and covering from the bottom of the skull to fourth vertebra, is shown enlarged in figure 3.4b. This sub-image was used to test the performance of all the low level edge operators.

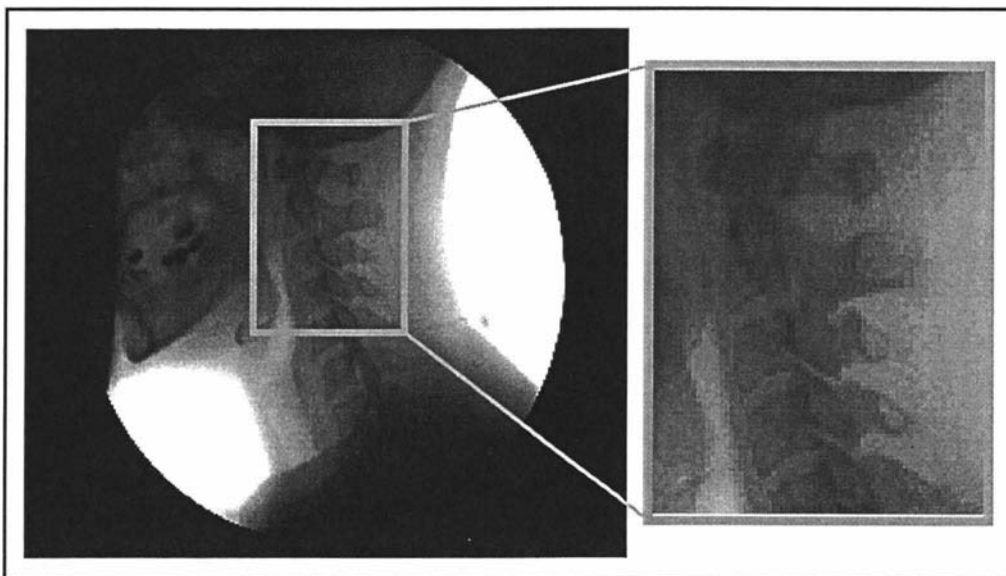


Figure 3.4: a) Original cervical image

b) Enlarged test section

Table 3.5 contains the results of applying the low-level edge operators to the small test image. The gradient images are the raw output of each operator scaled to fit the 0-255 byte range. The optimally thresholded images are the result of applying a global threshold to the gradient image chosen to maximise Pratt's figure of merit, shown at the bottom of each edge image.















Table 3.5	Roberts	Prewitt	Sobel	Kirsch	Laplacian	Stochastic 3x3 SNR = 9	Stochastic 5x5 SNR = 9
Gradient (scaled)							
Optimally thresholded	 P=42.8%	 P=45.3%	 P=42.7%	 P=44.7%	 P=24.3%	 P=44.4%	 P=43.3%

Table 3.5: Low-level edge operators applied to cervical test image

The figure of merit was computed with respect to a reference edge image (see table 3.6) generated manually, by tracing the vertebral outlines. The scaling parameter a , is set to unity.

It is clear visually that none of the edge operators extract the vertebral edges satisfactorily. The edges are thick and very fragmented, with many detected pixels not related to the vertebral boundaries. For the second order Laplacian case (mask **b** from table 3.2), the optimally thresholded image is the result of zero-crossing detection. A zero-crossing was declared if a change of sign in the gradient was detected in a 3x3 neighbourhood in any of the four directions about the middle pixel (see section 3.3). To try and reduce the number of falsely detected edge pixels, the plus and minus deviation from zero had to exceed a threshold before a zero-crossing was declared. Although this improved the results, the thresholded Laplacian image in table 3.5 is the best that could be obtained. It is extremely difficult to make out any of the vertebral edges in this image as they are extremely fragmented and severely contaminated with noise.

The last pair of images in table 3.5 were produced using the larger 5x5 stochastic masks computed for a signal-to-noise ratio of 9 (see table 3.3). This larger operator produces greater smoothing, as is evident in the gradient image. Once thresholded, the edge image shows reduced noise, but this is at the expense of thicker edges and the non-detection of weak vertebral edges.

Overall, the operators (with the exception of the Laplacian) score similar figures of merit even though the computational cost between operators is significantly different. The Prewitt gradient operator scored the highest with 45.3%.

3.3 High-level Edge Detection

The low-level edge operators of the previous section are principally characterised by the measurement of a local derivative to indicate the presence of an edge. This simplistic approach at first appears to make no assumptions about the nature of the intensity discontinuities (edges) or the underlining noise characteristic of the signal. However, implicit in this measurement is the assumption that the local derivative exists and can be reliably estimated from a small number of values in a local neighbourhood in the discrete image. This estimation problem is an ill-conditioned problem in the Hadamard [3.17] sense.

Most high-level edge detection schemes attempt to incorporate directly, assumptions about the nature of an edge, and its relative strength with respect to local background noise. There are three basic approaches:

- *Human visual theory computational approach*
- *Optimal edge detector computational approach*
- *Rule-based approaches*

The basis for the first computational approach to edge detection is from the framework of the theory of human vision proposed by Marr and Poggio [3.18] with reference to Logan's [3.19] theorem of early vision. From this work Marr and Hildreth [3.20] proposed a theory for edge detection in which oriented primitives called zero-crossing segments are computed over a wide range of spatial scales. Rules are then applied to combine the oriented zero-crossing primitives to produce a description of the image called a primal sketch.

Canny laid the foundations for the second of these three approaches with his theory for an *optimal* edge detector [3.21]. Methods that follow this approach proceed by postulating an edge and noise model (as for Stochastic gradients) and then compute an optimal operator for these proposed models using Canny's criteria.

The third approach to high-level edge detection usually begins with edge strength and direction maps computed by low-level gradient based operators. Rules are then applied to these data to enable the linking and thinning of edge pixels and the removal of isolated non-edge pixels. Typical rules include such properties as smoothness of orientation and neighbourhood connectivity [3.22].

3.3.1 Human Visual System (HVS) Based Edge Detection

The theory of edge detection proposed by Marr and Hildreth in 1980 was derived from physiological models of early human vision. Their theory produced results that were consistent with many of the findings of psychophysical experiments. The underlying premise for the Marr-Hildreth edge operator is that intensity changes in natural images occur over a wide range of spatial scales and thus should be detected separately at each scale. To realise this result the image must be locally smoothed by an optimal filter at various resolutions and then intensity changes detected at each resolution. Such an optimal smoothing filter should be approximately band limited to constrain the scale over which intensity changes take place. This filter should be smooth and localised in the spatial domain, and in particular its spatial variance should be small. The smoothing filter proposed by Marr and Hildreth, the Gaussian, is optimal in both its spatial and frequency localisation [3.23]. For two-dimensional continuous signals the Gaussian is given by:

$$G(r) = \frac{1}{2\pi\sigma^2} e^{\left(\frac{-r^2}{2\sigma^2}\right)} \quad \text{eq. 3.12}$$

where r is the radial distance from the origin and σ is the filter variance.

The maximum size of r constrains the size of the neighbourhood in which smoothing occurs. The filter variance σ governs the rate at which the filter decays. A small σ produces a rapid

decay and hence a sharp localised response. While a large σ produces a slow decay with increased smoothing over a larger area (see figure 3.5).

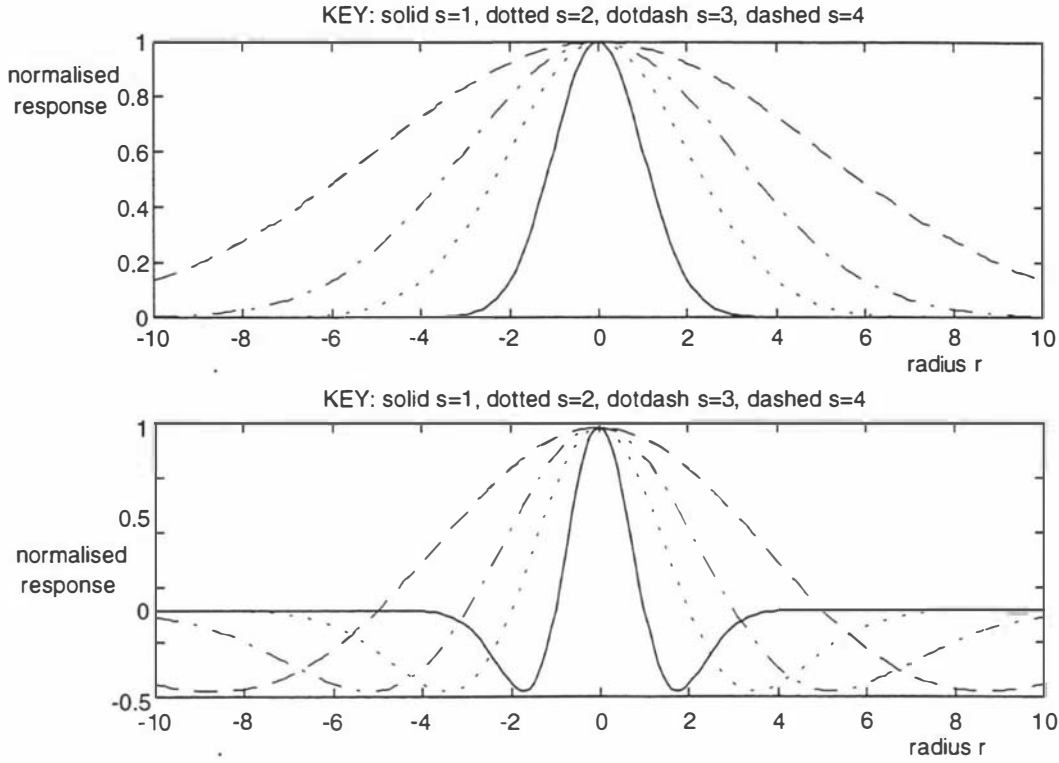


Figure 3.5: Normalised Gaussian and Laplacian of a Gaussian (LOG)

In order to detect intensity changes at each resolution Marr-Hildreth proposed the use of the non-directional second derivative operator the Laplacian (see section 3.2.3). The Laplacian operator can be combined with the Gaussian filter to produce a single edge enhancement operator, the Laplacian of a Gaussian (LOG, see equation 3.12). This is approximately a bandpass operator with a half power bandwidth of about 1.2 octaves. The use of this operator is consistent with the physiological models of simple visual cells proposed by Marr *et al* [3.24].

$$\nabla^2 G(r) = \frac{-1}{\pi\sigma^4} \left(1 - \frac{r^2}{2\sigma^2}\right) e^{\left(\frac{-r^2}{2\sigma^2}\right)} \quad \text{eq. 3.13}$$

The third part of the Marr-Hildreth edge detector concerns the combination of zero-crossing segment information from different resolutions. Marr and Hildreth introduced the *spatial coincidence assumption* from which three rules were derived to combine data. This assumption can be briefly stated as:

If the zero-crossing segments of independent LOG signals appear over a contiguous range of resolutions with consistent position and orientation at each resolution, then

the set of zero-crossing segments can be taken to indicate an intensity change due to a single physical phenomena.

The three rules deal with the specific cases of isolated edges, spatially close parallel edges (bars), blobs and terminations.

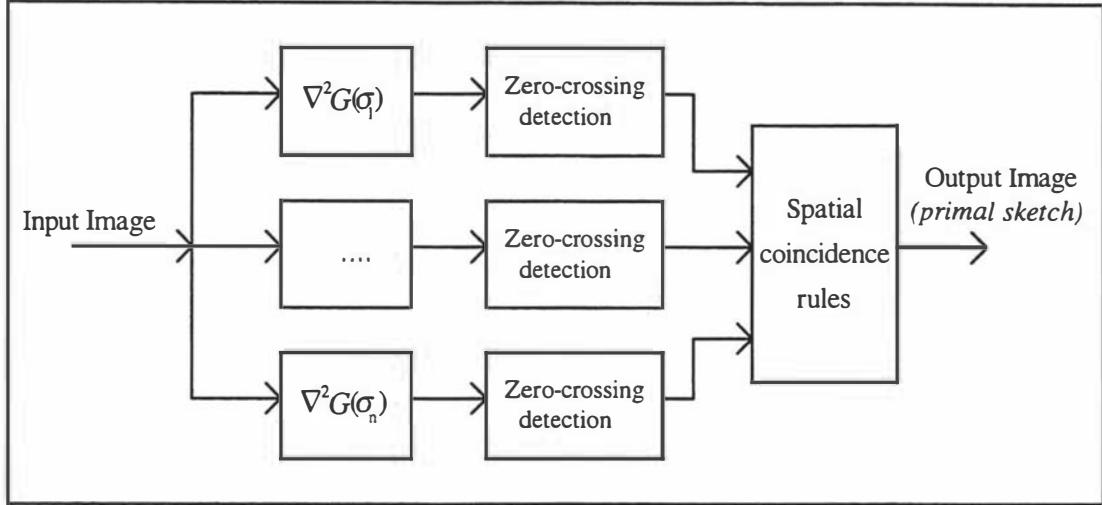


Figure 3.6: Marr-Hildreth Edge Detector

Figure 3.6 shows a block diagram of the complete Marr-Hildreth edge detector. The input image is filtered by a bank of n , LOG filters, covering a range of spatial variances, σ_1 to σ_n . The zero-crossings of the filter outputs are then detected and combined using the spatial coincidence rules to form the output primal edge image (sketch).

A number of practical issues arise in the implementation of the Marr-Hildreth edge detector. The most important issue concerns the choice of the bank of LOG filters. The spacing of the filters has to be wide enough such that the channels are relatively independent. The half power bandwidth of a Gaussian is approximately 1.2 octaves and it drops to less than 10% power at two octaves. Thus a practical filter spacing should be greater than 1.2 octaves. Also to ensure that filter error (due to the discrete implementation) is constant at all scales, the size of the filter radius must increase in proportion to σ . Furthermore, the range of resolutions (number of filters) needed to detect all intensity changes in an image is directly dependent on the content of the image, which in general is unknown. Marr and Hildreth used the results of psychophysical experiments [3.25] to infer the range of resolutions present in the human visual system. Attempting to implement this range of filters on a machine vision system results in reduced filter channel independence due to the relatively low spatial resolution of most machine vision systems in comparison to the human eye. This reduced channel independence may greatly reduce the effectiveness of the simple rules proposed by Marr and Hildreth for combining the zero-crossing segments from the output of each channel.

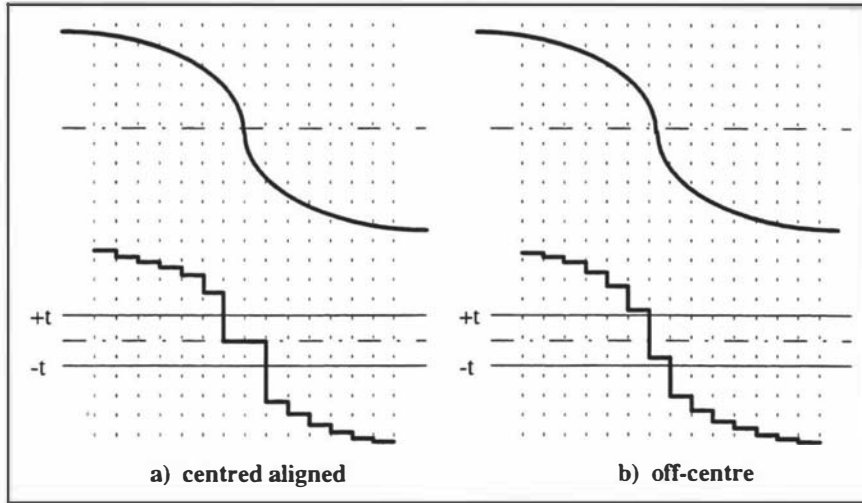


Figure 3.7: one-dimensional zero-crossing detection

The second practical issue of concern in the implementation of the Marr-Hildreth edge detector is zero-crossing detection. This issue is almost ignored in the literature, when it is addressed, it usually takes the form of a statement such as, "*in two-dimensional signals a zero-crossing is declared if a zero-crossing occurs in any direction*". The application of this statement appears at first to be a straight forward, as unlike most other methods for detecting intensity changes, no threshold is required. However, on closer examination, a number of difficulties arise due to the discrete nature of the signal [3.26]. A zero-crossing will be detected when there is a change of sign in any direction about a location. If the location is centred on the actual zero crossing as in figure 3.7a, then detection is straight forward as the magnitude of the positive and negative deviation will almost be symmetric. A relatively uncritical threshold t , can be set to determine the presence of the zero-crossing. However in practice, zero-crossings will not be centred on the pixel grid, resulting in an asymmetric magnitude deviation about the true location (see figure 3.7b). The choice of the threshold now becomes more critical to successful detection of the zero-crossing. A new but elegant solution to this problem is to produce a magnitude image where the pixel value is weighted by how far it is away from the nearest local zero-crossing [3.27]. Simple linear interpolation can be used to perform this task resulting in what looks like an anti-aliased zero-crossing image. A global or adaptive threshold can then be applied to produce the zero-crossing map required for the standard Marr-Hildreth edge detector. Alternatively the additional information contained in zero-crossing magnitude images could be used to directly assist in combining the output of each channel.

3.3.2 Optimal Edge Detector

An edge can be defined as a specific shape in the intensity verse distance space. Thus edge detection can be reduced to searching for the right shape in this space. From this computational

definition of edge detection, Canny [3.18] proposed a theory for *optimal edge detection*. Central to his theory is the optimisation of three criteria; good signal-to-noise ratio, good localisation, and maximum suppression of false responses. He derived quantitative measures for these three qualities which are represented by equations 3.13 to 3.15 respectively.

Define $f(x)$ as the optimal edge operator and $E(x)$ as the edge signature to be detected. If the signal noise is assumed to be additive, white and Gaussian with variance σ^2 , then the signal-to-noise ratio, S , at the edge location ($x = 0$) is :

$$S = \frac{\left| \int_{-\infty}^{+\infty} E(-x) f(x) dx \right|}{\sigma \sqrt{\int_{-\infty}^{+\infty} f^2(x) dx}} \quad \text{eq. 3.14}$$

Good edge locality, L , can be measured as being inversely proportional to the position variance of the maximum of the filter, about the supposed edge location.

$$L = \sigma^2 \frac{\left| \int_{-\infty}^{+\infty} E(-x) f'(x) dx \right|}{\sqrt{\int_{-\infty}^{+\infty} [f'(x)]^2 dx}} \quad \text{eq. 3.15}$$

where $f'(x)$ is the first derivative of the optimal filter and $E'(x)$ is the first derivative of the edge signature.

The third measure, suppression of false responses, C , can be defined as being proportional to the mean distance between the maxima of the filter response to Gaussian noise.

$$C = \frac{1}{2w} \sqrt{\frac{\int_{-w}^{+w} [f''(x)]^2 dx}{\int_{-w}^{+w} [f'(x)]^2 dx}} \quad \text{eq. 3.16}$$

where $f''(x)$ is second derivative of the optimal filter and w is the half width of the filter.

Canny found the optimal operator for a step edge signature by combining the first two measures to produce an edge performance measure P , where $P = (SL)^2$. He then maximised P subject to the third criterion of minimal number of false responses. The equations resulting from this optimisation were long and complex and difficult to implement. This lead Canny to proposed an approximation to his best operator, the derivative of a Gaussian, $\nabla G(r)$.

$$\nabla G(r) = \frac{-r}{2\pi\sigma^4} e^{\left(\frac{-r^2}{2\sigma^2}\right)} \quad \text{eq. 3.17}$$

The normalised response of this operator for a range of σ , is shown in figure 3.8. The operator is simple in form and its performance measure is 80% relative to Canny's best operator.

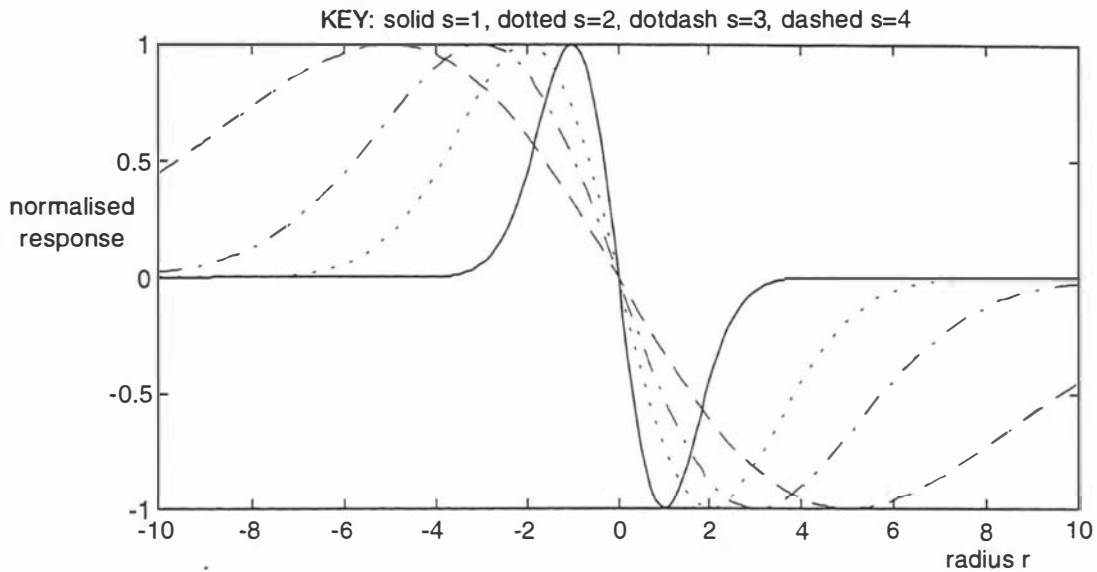


Figure 3.8: Normalised derivative of a Gaussian

Many other authors have extended Canny's work, notably Spacek [3.28] and Deriche [3.29]. Spacek formed a performance measure by combining all three of Canny's quantitative measures. In doing this he greatly simplified the differential equations whose solution is the optimal edge operator. The final filter contained six parameters, two of which Spacek fixed, and then found the other four by applying boundary conditions. Deriche followed the same approach as Canny but allowed his filter to have infinite extent. He then was able to show that filters of this kind can be implemented recursively and thus very efficiently. The final functional form of his filter contains only two parameters. These two parameters could be chosen freely so that the signal-to-noise ratio can be traded at the expense of good locality and vice versa.

The algorithm that has become commonly known as the Canny edge detector is shown in figure 3.9. The first stage is the linear *Canny filter*, the derivative of a Gaussian. The output of this filter is passed through two very non-linear stages, non-maximum suppression and hysteresis thresholding, finally producing the output edge image.

Non-maximum suppression had been used in other edge detectors [3.30] prior to Canny's work. The basic operation attempts to determine if at a candidate edge point, the gradient magnitude is a local maxima in the direction of the gradient. If the edge point is not aligned with the gradient direction, then the location is suppressed. This operation can be implemented in a number of different ways. Canny originally fitted a 2-D linear surface to a 2x2 neighbourhood in order to estimate the direction of the local gradient. A more common approach is to use a 3x3 neighbourhood containing the nearest eight neighbours. The centre value is a local maximum if

in the direction it points (to the nearest 45° increment) the two nearest neighbours have a lower gradient magnitude.

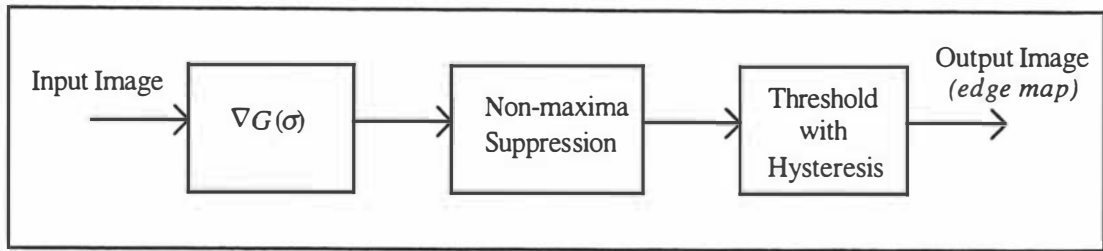


Figure 3.9: Canny Edge Detector

The combined effect of gradient filtering and non-maximum suppression is very similar to zero-crossing detection of the second derivative. However it has significant advantage in detecting local extrema when they are points of inflection. A point of inflection may occur as either a maximum or a minimum of the absolute value of the first derivative. Maximum points of inflection correspond to locations of sharp variation whereas minima points of inflection correspond to slow variations. Using a second derivative operator it is very difficult to distinguish between these two type of zero-crossings. In contrast, using a first-order derivative with non-maximum suppression, only points of sharp variation are chosen.

The third stage of Canny's edge detector is thresholding with hysteresis. Canny's proposed this heuristic strategy as a way to reduce spurious responses while preventing the fragmentation of the detected edges. In this strategy, two threshold limits are set on the output signal-to-noise ratio of the operator. The start of an edge segment is declared at a point where the gradient magnitude is above the upper threshold. The edge segment is grown by adding the nearest neighbour pixels that have a gradient magnitude that is above the lower threshold. A segment is terminated in the direction of growth when the nearest neighbour gradient magnitude drops below the lower threshold. Control can also be put on the length of the edge segments. Segments shorter than a specified length can be discarded.

In order to set the two threshold levels, an estimate has to be made of the image noise level. In Canny's original work the image model edges are assumed to be step transitions and the image is contaminated with additive white Gaussian noise. Applying a gradient of a Gaussian operator to such an image will produce an image whose histogram can be shown to be a Rayleigh distribution [3.31] everywhere, except at the edges. A Rayleigh distribution takes the form of equation 3.18. It is very similar to the gradient of a Gaussian equation, 3.17. If a Rayleigh distribution is fitted to the lowest $k\%$ of the gradient magnitude histogram then the lower threshold can be set at say the 90 percentile of this fitted distribution. The upper threshold can then be set to be some ratio of the lower threshold.

$$R(r) = \frac{r}{\sigma^2} e^{\left(\frac{-r^2}{2\sigma^2}\right)} \quad \text{eq. 3.18}$$

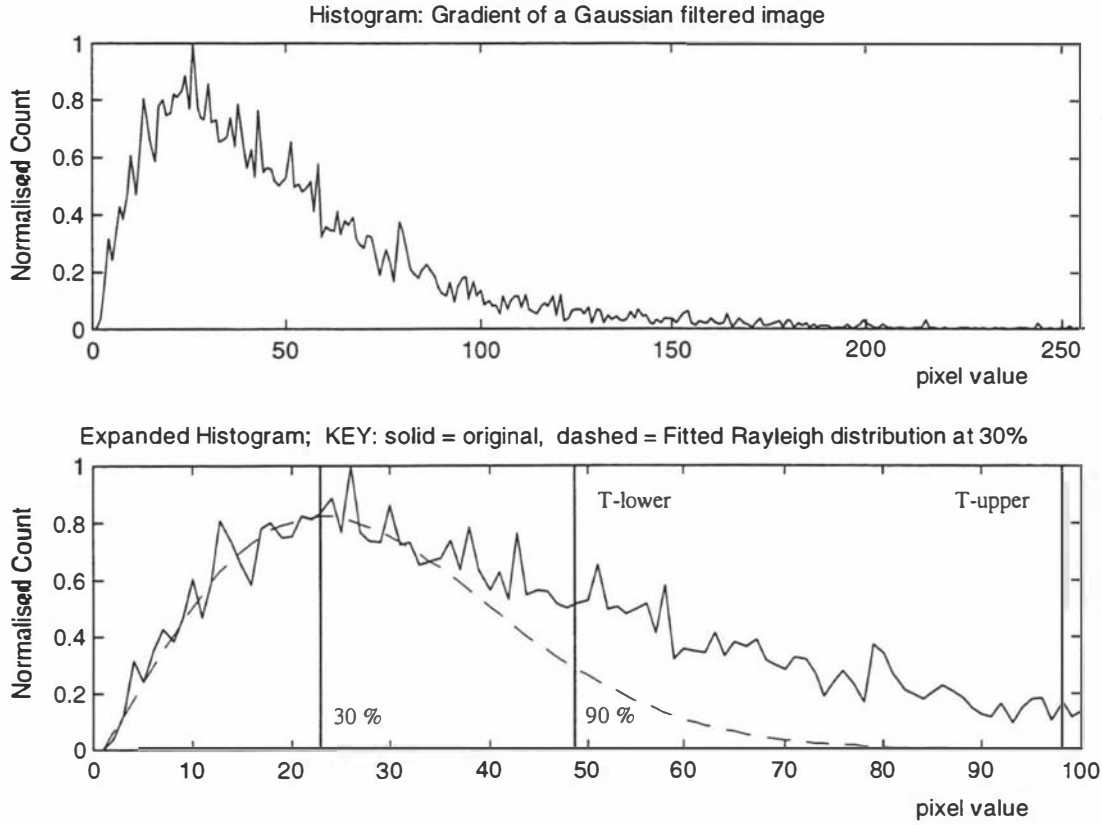


Figure 3.10: Thresholding with Hysteresis - threshold selection

Figure 3.10 illustrates the computation of the threshold levels from the histogram of the cervical section test image filtered with a gradient of a Gaussian with $\sigma = 0.8$. The top graph is the normalised histogram for the image, while the bottom graph is an expanded histogram with a Rayleigh distribution fitted to the first 30 percentile data. The lower threshold (T_{lower}) is computed at the 90 percentile level of the fitted distribution and the upper threshold (T_{upper}) is set to twice this lower threshold.

Although the three-stage process described above is commonly referred to as the Canny edge detector, Canny originally proposed [3.21] that it should be applied at a number of discrete resolutions and the results combined together using a set of predefined rules. Canny's *feature synthesis* stage performs the same basic operation as the last stage of the Marr-Hildreth edge detector, where *spatial coincidence* rules are applied to combine the zero-crossing maps. The underlying problem with both these methods is that there is no known optimal algorithm to integrate edge maps. In fact, integrating the edge maps is another ill-posed problem.

3.3.3 Rule-based Edge detection

Rule-based edge detection schemes generally begin with the magnitude and direction information produced by applying some form of gradient operator(s) to the image. A decision strategy then proceeds to use this information to determine appropriate candidates (pixel locations) for the start of significant edge segments and then begins to build each edge segment. Because the edge segments build up incrementally, algorithms of this type are often called sequential edge detectors. A typical sequential edge detection algorithm by Delp and Chu [3.16] is implemented as an edge tracing algorithm, and is described as follows:

"Search along each row (starting at the top left corner) for a pixel with an edge strength above a contour start threshold that is not already an edge point. This pixel is classified as an edge point and the algorithm begins to trace the edge segment.

Candidates for the next edge point are those pixels within a one pixel radius whose direction is within $\pm\Delta\theta$ of the direction of the current edge point. The procedure then starts from this new edge point and checks locally for possible edge points. If another edge point is not found, the edge segment is terminated. If the length of a contour at termination is below a minimum then it is discarded. If the newly-found contour connects to an existing contour then they are merged. The algorithm then returns to the neighbourhood to search for the next start point. Algorithm termination occurs when the bottom right corner of the image is reached."

The two critical elements of sequential edge detection algorithms are the *next-point* criteria and threshold selection. Approaches to choosing the next point based on the previous point include dynamic programming, graph-searching techniques [3.32], non-maximum suppression and connectivity measures such as that of Kitchen and Rosenfeld [3.15]. Similarly, a wide range of methods have been described for determining threshold selection. They include, simple fixed threshold, Frei-Chen classification rule [3.30], and a locally adaptive threshold [3.33].

With such a wide choice of methods at every stage of a rule-based edge detection algorithm (including the selection of the operator to compute the local gradient and direction information), almost all practical implementations have been assembled heuristically, usually on specific types of images. Algorithmic complexity and computational cost vary greatly between different implementations. Performance of such algorithms can be extremely good, with figures of merit (see equation 3.11) in excess of 90% for image signal-to-noise ratios greater than 15dB [3.16]. However for SNR's below 15dB, results are extremely variable and highly image dependent.

3.3.4 Combining Fragmentary Edge information

The result of applying an edge-detection algorithm to an image is the reduction of the grey-level information to binary information consisting of edge segments. These edge segments mark the

location of the significant local intensity changes in the original image. Often the most significant intensity changes occur at the boundaries of objects or geometrical features. Due to photometric effects, noise and limitations of edge detection algorithms, an object boundary may have been detected as a number discrete edge segments. If geometric features of an object such as size and orientation are to be computed, then it is useful to have complete object boundaries. The task of combining the fragmentary edge information can be approached in a number of ways. Popular methods include edge-linking [3.34], heuristic graph searching [3.35], dynamic programming [3.36], and the Hough transform [3.37].

The first three of these methods view a boundary as a path through a graph formed by linking edge segments together. The linkage rules form the procedure for connecting the edge segments. In the case of heuristic graph searching methods, a proposed path between two points A and B , is considered to consist of node locations $x_k, k = 1, 2, \dots$ (corresponding to the pixel coordinates). If given an *evaluation function* $\phi(x_k)$ which measures a value for the path constrained to go through node x_k then each successor of the start node is examined and the one selected that maximises $\phi(x)$. This selected node becomes the new start node and the process is repeated until B is reached. The node selection from A to B constitutes the boundary. The computational cost of such a procedure is highly dependent on the form of ϕ .

Unlike heuristic graph searching methods, dynamic programming aims to find the global optimum of a multi-stage process, given a particular *evaluation function*. The method is based on Bellman's *principal of optimality* [3.36], which states:

"The optimum path between two points is also the optimum between any two points lying on the path."

Thus if C is a point on the optimum path between A and B , then the path CB is the optimum path between C and B , *no matter how one arrives at C*. If there are N nodes to be considered for the path, then the dynamic programming procedure reduces the global optimisation problem to N stages of two variable optimisation. Furthermore, if ϕ can take on L possible values at each node, then the total number of search operations is $L^2(N-1) - L + N$. This will be significantly smaller than the $L^N - 1$ operations required for an exhaustive search when N and L are large.

The last of the popular methods for combining fragmentary edge information is the Hough transform [3.37]. This method was originally proposed for the detection and recognition of complex shapes. Unlike the previous three methods, this method requires the proposal of a geometric model for the object. It then proceeds to find all significant occurrences of objects (at any scale and orientation) fitting this model. The simplest object model that can be proposed is a straight line. If the perpendicular distance from a line to the origin is denoted s , and the angle it

forms with respect to the x -axis is denoted θ (figure 3.9a), then this line can be represented by the equation

$$s = x \cos \theta + y \sin \theta \quad \text{eq. 3.19}$$

Taking the Hough transform of this line results in just a point in the (s, θ) plane. In effect, all points on the line map into a single (s, θ) point (figure 3.9b). This result can be used to detect straight lines in a given a set of boundary (edge) points.

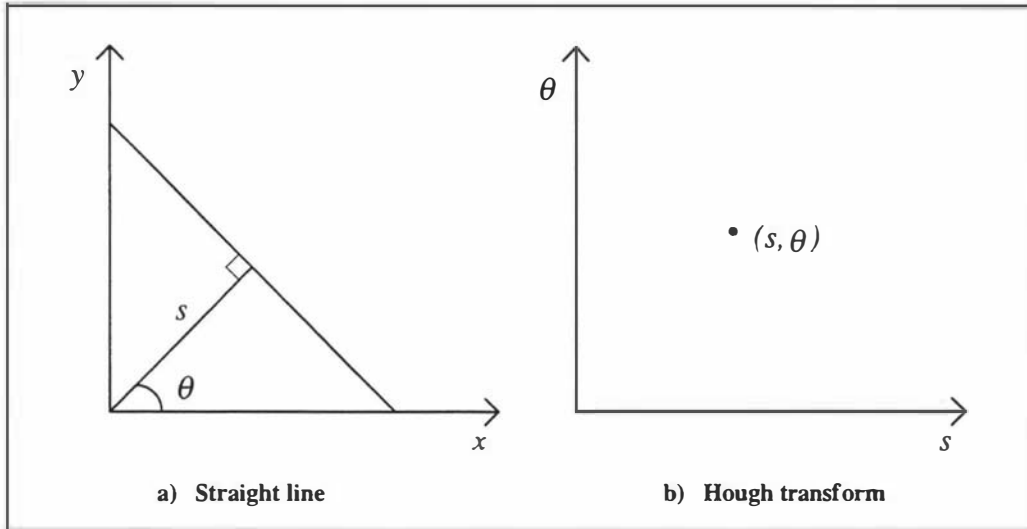


Figure 3.11: The Hough Transform

Practical implementations of the Hough transform proceed something like this. For a chosen quantisation in the value of the parameters s and θ map the coordinates, (x_i, y_j) , of the fragmentary edge points into the (s, θ) space. Count the number of edge points, $C(s, \theta)$ that map into each location in (s, θ) . Finding the local maxima of the count will give the s and θ values of different straight line segments through the edge points. In practice a relatively coarse quantisation for s and θ will be used in order to reduce the computational cost of searching for maxima in $C(s, \theta)$. The location of detected edge points will also usually be quantised to the pixel grid spacing. Thus a single line will no longer transform into a single point in (s, θ) space, but will be spread out over several quantisation levels. Searching for local maxima in $C(s, \theta)$ now constitutes searching for clusters and requires the use of clustering techniques [3.38].

Although the Hough transform can be generalised for the detection of arbitrary shapes [3.39], the dimensions of the parameter space that must be searched, become extremely large, even for simple curves. This has generally restricted the method to the detection of lines, circles, ellipses, and arcs. Although these shapes are important primitives for objects, a complete object will be made up of many of these primitives. Thus some form of geometric *feature synthesis* is required to interpret the detected primitives and combine them together to form a single geometric

description of an object. The Hough transform is a robust, but non-optimal, parametric curve fitting technique [3.40], equivalent to binary template matching with a top-hat function [3.41].

3.3.5 Performance of High-level Edge Detectors















The Marr-Hildreth and Canny edge detectors have become the default standard by which other edge detectors are gauged. Because of this, and in light of the impractical nature of implementing and assessing the performance of many of the other edge detectors described in the literature, only the Marr-Hildreth and Canny edge detectors were assessed. These two edge detectors were applied to the same cervical cineradiographic test image (figure 3.4b) used to evaluate the low-level edge operators. Results were assessed visually and also scored quantitatively using Pratt's figure of merit. Code for the implementation of these two edge detectors is contained in appendix C, page C-21.

In principle both the edge detectors required only a single control parameter, the standard deviation σ of the Gaussian filter, to be set by the user. For a range of σ , the resulting output edge maps are then combined using appropriate rules. Most comparisons using these two edge detectors are only performed for a single value of σ usually chosen visually to produce the *best* edges. In this assessment, the upper and lower value of σ were determined visually by experimentation. The lower value of σ was set such that the lowest contrast edges at the front of the third vertebra were detected. The upper value was set to a level where the high contrast edges were just beginning to lose their shape. Edge maps were generated for five values of σ between these limits, spaced at a ratio of approximately 1.3. These five edge maps were combined to produce a composite edge map. The criterion used to combine the edge maps for the Marr-Hildreth detector was that an edge pixel was declared if it occurred over at least three consecutive values of σ . This amounts to the simple implementation of Marr-Hildreth's *coincidence assumption* (section 3.31). For the Canny composite edge image, continuity of both location and direction, across at least three consecutive scales, was used. The tolerance on direction spread was set to 45° .

In practice the Canny edge detector has several additional user adjustable parameters. They are the internal parameters for the two non-linear stages. In the non-maximum suppression stage, a parameter can be set to control the decision on the acceptable local gradient alignment. In the hysteresis threshold stage, information is required in order to set the upper and lower thresholds. Again in the literature, there appears to be no standard approach to setting these parameters.

The way in which they are set usually depends on the particular implementation used. In this work, the 3x3 nearest neighbour approach was used for the non-maximum suppression stage. This approach has the advantage that it requires no additional computation in order to estimate the direction of the gradient plane.

Table 3.6: Marr-Hildreth and Canny edge detectors applied to cervical test image

Reference Images	Edge Dectector	Sigma = 0.80	Sigma = 1.00	Sigma = 1.30	Sigma = 1.70	Sigma = 2.30	Composite Image
 <p>Original</p>	Marr-Hildreth	 <p>P=24.7%</p>	 <p>P=25.5%</p>	 <p>P=27.2%</p>	 <p>P=31.7%</p>	 <p>P=31.8%</p>	 <p>P=28.1%</p>
 <p>Manually extracted edges</p>	Canny	 <p>P=44.1%</p>	 <p>P=42.6%</p>	 <p>P=38.1%</p>	 <p>P=34.6%</p>	 <p>P=34.5%</p>	 <p>P=31.3%</p>

For the hysteresis threshold stage, the Rayleigh distribution was fitted to the lower 30 percent of the gradient image histogram data. The lower threshold was computed at the 85 percent level of distribution and the upper threshold set to 1.8 times the lower value. All the internal parameters for the two non-linear stages were determined experimentally. The 30 percent level for fitting the Rayleigh distribution was found to be relatively non-critical. Results were somewhat more sensitive to the selection of the other two parameters.

Table 3.6 contains the results of applying both the Marr-Hildreth and Canny edge detection algorithms to the cervical test image of column one, over a range of filter resolutions. The second row of the first column also contains the manually generated reference edge image on which the figure of merit for each edge image was computed.

The first and most obvious difference between the output edge images of the Marr-Hildreth and Canny edge detectors is that the edges detected by the Marr-Hildreth detector are relatively thick compared to those of the Canny detector. The reason for this appears to be due to the inability of zero-crossing detection of the LOG to differentiate between maxima and minima points of inflection. This assumption was partially verified by removing the non-maxima suppression stage of the Canny detector and simply passing the gradient image directly to the hysteresis threshold stage. The detected edges were then of comparable thickness to those of Marr-Hildreth.

As the standard deviation of the smoothing filter increases across the table, the figure of merit improves for the Marr-Hildreth detector but declines for the Canny detector. The pattern of improvement in value for the Marr-Hildreth detector is due to the reduction in detection of non-edge pixels. The decline in value for the Canny detector occurs because of edge fragmentation due to the non-detection of low-strength edges.

The final column of table 3.6 contains the composite edge images. The figure of merit for the composite images is worse than the value for several of the single resolution images for both edge detectors. This result is a reflection of the non-optimality of the standard algorithms used to combine the edge maps.

Overall the Canny edge detector performed slightly better than Marr-Hildreth, however the results are still very poor, with the best figure of merit less than 50%. This is principally due to significant fragmentation of the detected vertebral edges and the non-detection of the front edges of the second vertebra. None of the high-level edge image figures of merit exceeded the value computed for the low level, computationally inexpensive, Kirsch compass operator of section 3.2.5. This result is a reflection on the difficulty in reliably detecting edges in cineradiographic images.

3.4. An Alternative HVS Inspired Approach to Edge Detection

When an image is presented to a human observer it is transformed by the human visual system (HVS) into information. This perceived information may be represented by attributes such as brightness, colour, edges and texture. Understanding this perception process is founded in the results of psycho-physical experiments into human visual perception. Many of the early developments in image processing were based on quantitatively representing this perceptual information.

One of the earliest studies into human visual perception concerned the phenomenon of simultaneous contrast. In figure 3.12a, the two smaller squares in the middle have equal luminance or *intensity* values, but the one to the left appears brighter. On the other hand in figure 3.12b, the two squares appear near equal in brightness although their luminance's are very different. The reason for the is that human visual perception is sensitive to luminance contrast rather than absolute luminance.

According to *Weber's Law* [3.42], if the intensity I_0 of an object is just noticeably different from the intensity I_s of its surround, then their difference ratio :

$$\frac{|I_s - I_0|}{I_0} = \text{constant} \quad \text{eq. 3.20}$$

Redefining $I_0 = I$ and $I_s = I + \Delta I$, where ΔI is small for just noticeably different intensities, equation 3.20 can be written as

$$\frac{\Delta I}{I} \approx \frac{d}{dI}(\log I) = \Delta c \quad (\text{constant}) \quad \text{eq. 3.21}$$

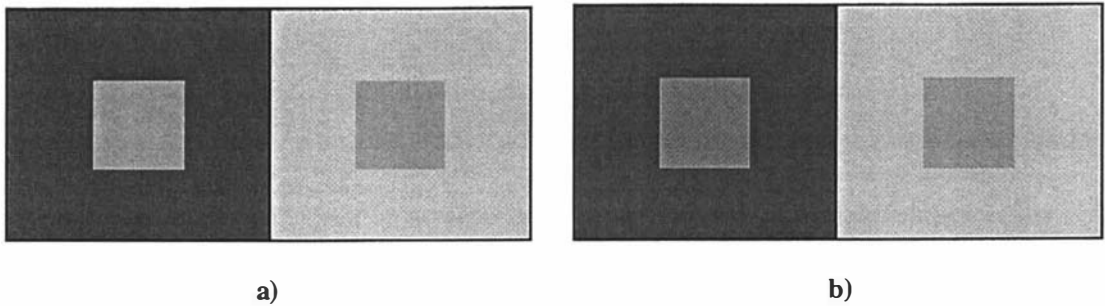


Figure 3.12: Simultaneous contrast, simple image

For simple images such as those in figure 3.12, the value of the contrast constant (called Weber's ratio) has been found to about 2% over a wide range of absolute intensity. At the extremes of the intensity range Weber's ratio increases rapidly to avoid sensor signal-to-noise limitations at low intensities and sensor saturation at high intensities (figure 3.13).

For more complex images, the human visual system adapts to the average intensity level, which is dependent on the characteristics of the scene. As the eye roams about a scene the instantaneous adaptation level fluctuates about this average value. The effect of this adaptation is that the measured Weber's ratio for small areas within an image is generally much higher than the 2% value obtained for a simple single target case. Also the range over which the ratio is a constant is much narrower.

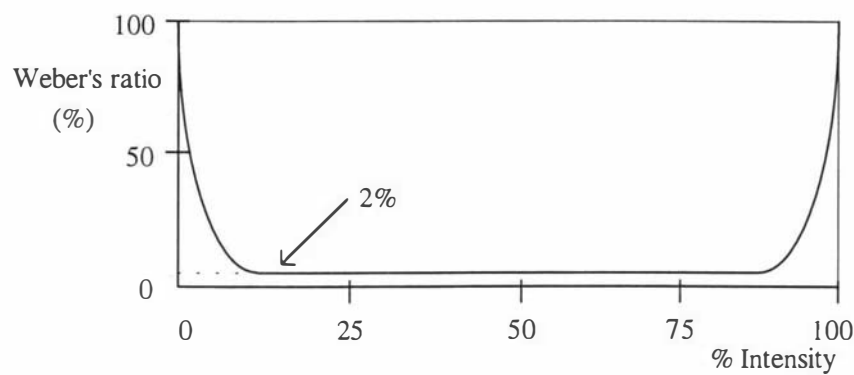


Figure 3.13: Weber's Law: Simple image

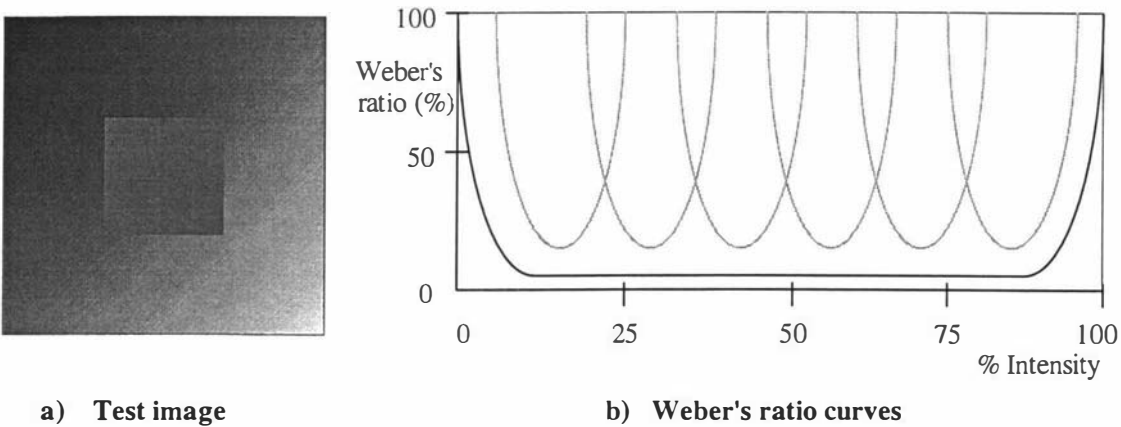


Figure 3.14: Weber's Law - complex image

Figure 3.14a) contains an image of a small square of constant intensity on a background with a linear intensity gradient diagonally from the top left hand corner to the bottom right hand corner. The small square appears to have the inverse intensity gradient to the background, darker in the bottom right-hand corner and lighter in the top left-hand corner, even though it is of constant intensity. Measuring Weber's ratio against the local average intensity at the edges of the small square results in figure 3.14b. Each of the 'U'-shaped curves represents Weber's law at a particular local average intensity. The range over which Weber's ratio is constant is narrow and its value is significantly higher than the 2% value for a constant intensity background.

3.4.1 Weber's Law and Profile Based Edge Detection

In section 3.2 it was established that boundaries of objects (edges) are often indicated by local intensity discontinuities. The magnitude of these local changes in intensity can be enhanced by applying a gradient filter or first derivative operator. A threshold can then be applied to the gradient enhanced image to produce an edge image. If a single global threshold is used to make the edge/no-edge decision then a number of difficulties arise. Setting a low threshold to allow low-strength edges to be registered will also register noise (both from photometric effects and random signal errors). Furthermore, high-strength edges will be registered as thick lines, often obscuring nearby low-strength edges. When the threshold is raised to improve the detection of a high-strength edges, low-strength edges will be lost. Thus using a global threshold will be a compromise, attempting to balance all three aspects above. Clearly if a threshold can be made dynamic, adapting to local image characteristics, then significantly better results should be produced.

Many different adaptive thresholding algorithms have been proposed in the literature. One common approach involves computing the cumulative histogram of the gradient in a local neighbourhood. A threshold is then chosen such that a specified percentage of the pixels with the largest gradient are declared as edge pixels. In the Canny edge detector (section 3.32) the upper and lower thresholds of the hysteresis threshold stage were computed by fitting a Rayleigh distribution to the low 30 percent of the gradient image histogram. A computationally inexpensive approach to adaptive thresholding was proposed by McIlroy *et al* [3.43] based on Weber's law. In their algorithm the local average intensity (*LAI*) was computed on a 2x2 neighbourhood, and the local change in intensity computed using the Roberts product (*RP*) gradient mask. Thresholding of the local gradient consisted of a simple comparison :

$$RP \geq f(LAI) \quad \text{eq. 3.22}$$

where f is a function which was specified by a look-up table.

The choice of the 2x2 neighbourhood enabled them to realise a hardware based real-time (at video rate) edge processor. The threshold or profile function f could be programmed *on-the-fly* to adapt to changes in the video signal. For scenes with a high signal-to-noise ratio their optimal experimental profile was found to be that shown in figure 3.15.

This threshold profile is implemented as a three section piece-wise linear function. It starts at a value of four (approximately 2% of the dynamic range) at zero LAI and increases slowly for low intensities. It then increases more rapidly for medium intensities and finally very rapidly at high intensities. This profile can be transformed to the Weber domain ($\Delta I / I$ vs I) by dividing each value by the corresponding LAI. This results in the Weber threshold profile of figure 3.16.

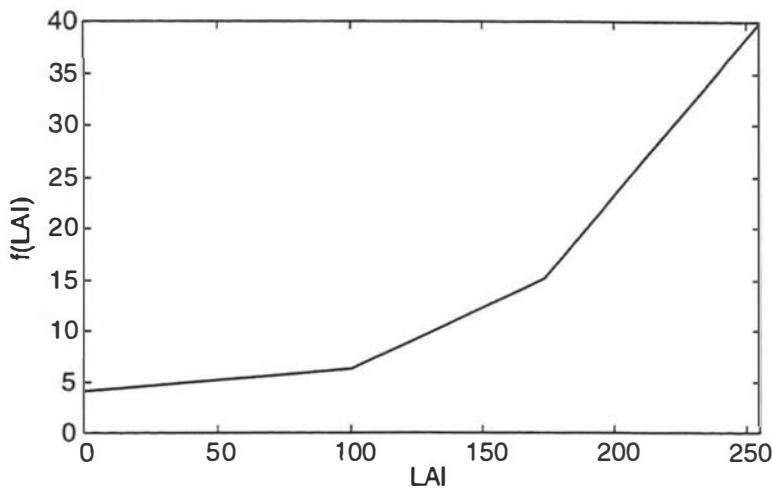


Figure 3.15: Threshold profile

For low values of LAI, the shape of this profile is consistent with Weber's result for a constant intensity background (see figure 3.14). At high intensities the curve deviates significantly from Weber's result, increasing relatively slowly and reaching a maximum value only two times as large as the minimum profile value.

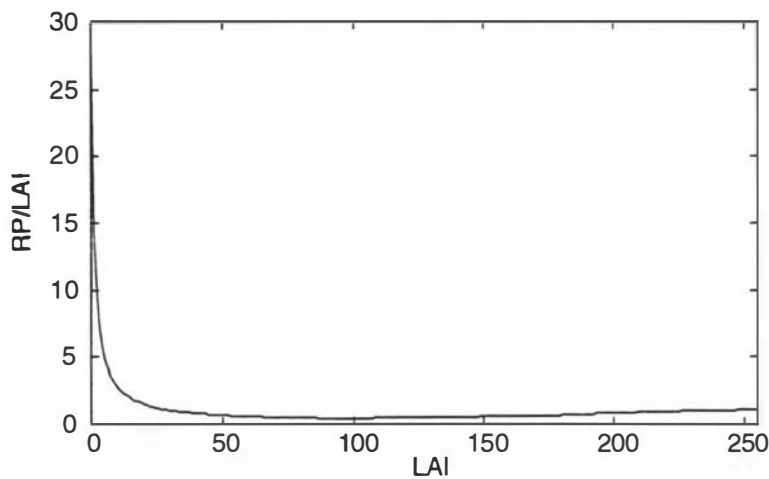


Figure 3.16: Weber threshold profile

The original work of McIlroy *et al* was primarily concerned with the actual hardware implementation of their Weber's law based edge detection algorithm. Their experimental results for a range of test images, indicated the potential effectiveness of their simple algorithm, but only with respect to *hand-tuned* threshold profiles. It was unclear how the profile generation could be automated and hence adaptively updated in a feedback loop for real-time edge processing of dynamic scenes. The next two sections in this chapter outline original work to investigate the effectiveness of the adaptive threshold algorithm of McIlroy *et al* and to extend their results with the potential for automatic threshold profile generation and hence the reliable detection of edges in cineradiographic sequences.

3.4.2. Target Based Computation of Threshold Profiles

The Weber's law based edge-detection algorithm assumes that the edge information in an image is uniquely defined by a range of Weber ratios ($\Delta I / I$), represented by the threshold profile function. To investigate the degree to which this relationship holds, a method was devised where a human operator manually traced the desired edges in an image. The coordinates of these traced edges form an *edge-template*. The edge-template is then used as a *target* from which analysis can begin. A secondary advantage of this approach is that in using a human to specify the edges of interest, some of the mechanisms of human vision (namely Weber's law) will be incorporated implicitly into the selection process.

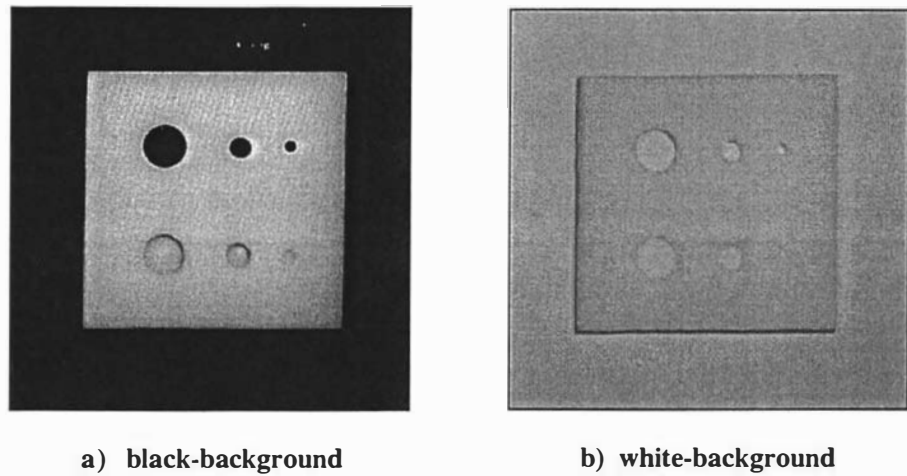


Figure 3.17: Component test images

To ensure that there was little subjectivity in the defining the edges of interest, two test images with specific properties were created. Both the images are of a simple manufactured component containing both blind and through holes. The test images are shown in figure 3.17. The first image, figure 3.17a, is of the component on a near black background, and the second image, figure 3.17b, is of the component on a near white background. The black-background image contains sharply defined edges and is relatively noise free. Most of the information in the white-background image is contained in a narrow range of intensities. The image appears relatively noisy and of low contrast.

The main edges associated with the test component (holes and outside perimeter) were traced on each test image to produce the corresponding edge templates. These edge templates are shown overlaid on the original images in figure 3.18.

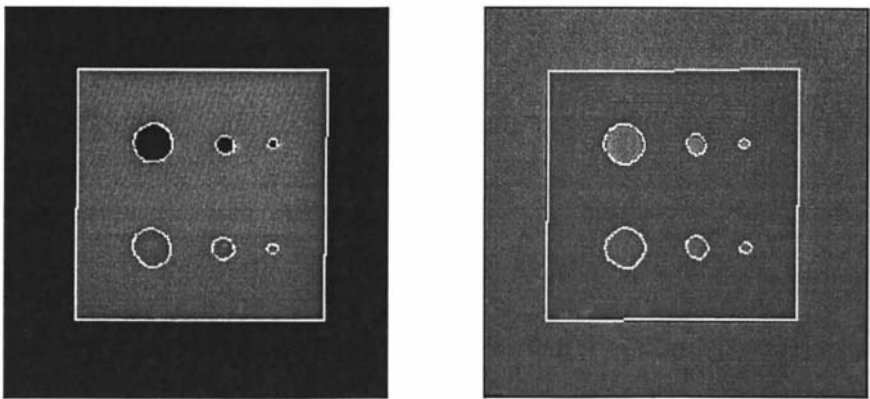


Figure 3.18: Manually generated edge templates

The edge-strength local-average-intensity statistics of the original image were computed at the pixels defined by the edge-templates, using the 2x2 operators of McIlroy *et al.* Figure 3.19 shows the minimum, maximum and mean edge-strength profiles generated from the computed statistics. The mean profiles are calculated from the average of the range of edge-strengths recorded at each LAI in the image. If a value of LAI was not present in the image at an edge template position, then the profile value was set to 100 for clarity.

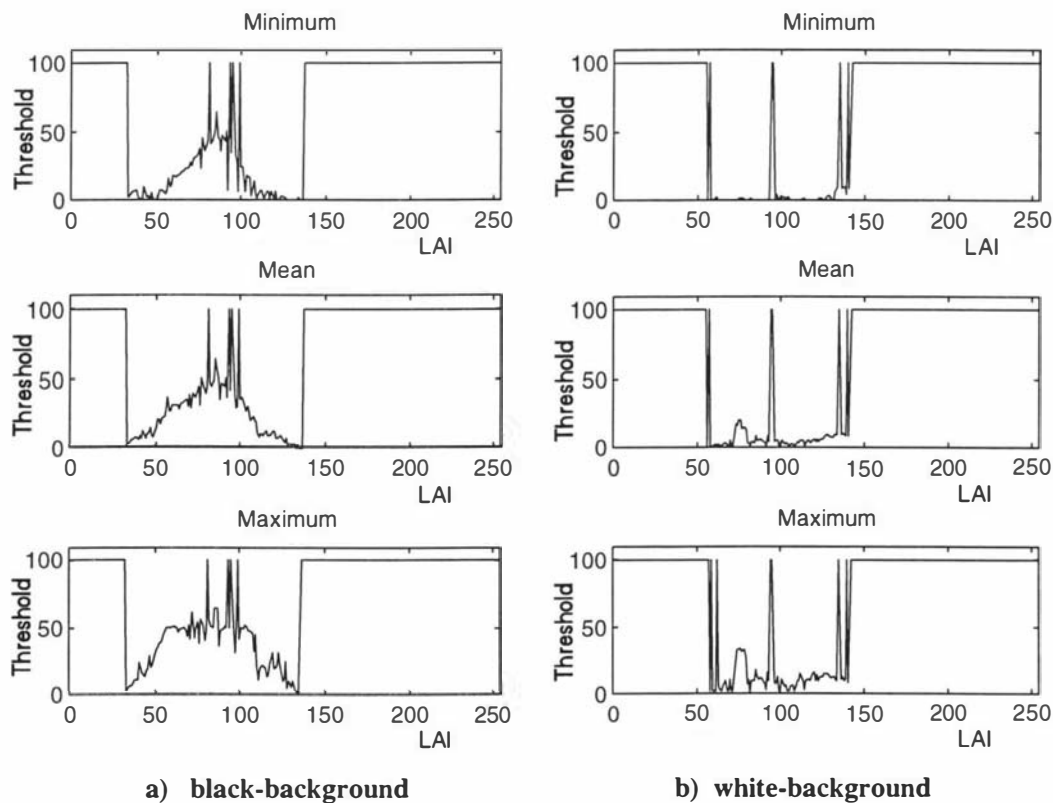


Figure 3.19: Edge-strength profiles

The black-background image profiles (figure 3.19a) cover a LAI range from approximately 35 to 135. There is little difference in shape between the minimum and mean profiles except at the ends of the active LAI range. The maximum profiles show elevated edge-strengths in the LAI

region from 30 to 70, with a number of high spiky values in the range 100 to 135. The profiles for the white-background image (figure 3.19b) cover a LAI range from approximately 55 to 140. This dynamic range is slightly smaller than the black-background image profiles. There are some large edge-strengths recorded at about 75 LAI, indicating the potential presence of high contrast edges. These values are most likely due to local shadowing at the edges of the test part. The minimum profile edge-strengths are generally very low relative to the mean and maximum profiles and reflect the noisy nature of the image. The maximum profile is similar in shape to the mean profile except that the edge-strengths and variance are increased

The minimum, mean and maximum edge-strength profiles of the two test images computed at the pixels defined by the edge-templates were used in the Weber's law based edge detection algorithm of McIlroy *et al.* The resulting edge images are shown in table 3.7.

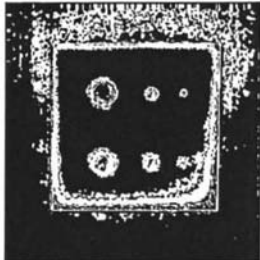
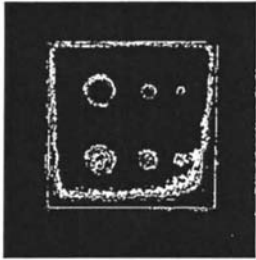

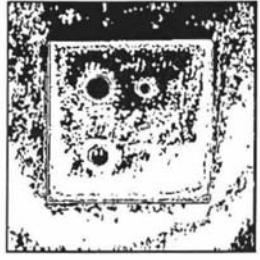
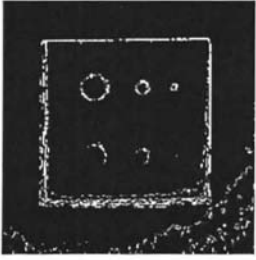

Table 3.7	Minimum Profiled	Mean Profiled	Maximum Profiled
Black background			
White background			

Table 3.7: Direct profile thresholding of test images

Minimum edge-strength profiling tends to give greater emphasis on the LAI values rather than the edge-strengths. For both test images, the minimum profile edge image contains all the pixels of the edge template. However many other pixels are recovered because they also have an ES greater than or equal to the value of the template edge pixels at each LAI. This is particularly obvious in the white-background case where about 50% of all pixels are detected. The mean profile edge images show greater rejection of non-edge template pixels, though this has been at the expense of the non-detection of some of the genuine edge template pixels. Maximum edge-strength profiling recovers only the pixels defined by the edge-template with the largest ES recorded at each LAI. For the black-background image many of the edge-template pixels are recovered, though the edges are highly fragmented and the lower surface contour, due to uneven

illumination, is still present. Few surface pixels are recovered from the white-background image, but there are also few edge-template pixels detected and those detected are highly fragmented. This is particularly obvious for the bottom row of blind holes, where only a single pixel is detected on the large hole.

The direct profiling results of table 3.7 are poor, even for the black-background image with its high contrast. This may be due to slight pixel placement inaccuracies in the target edge templates. To test this hypothesis, threshold profiles were recomputed for the growth of a one and two pixel layer about the original edge-template. The resulting mean profiles are shown in figure 3.20.

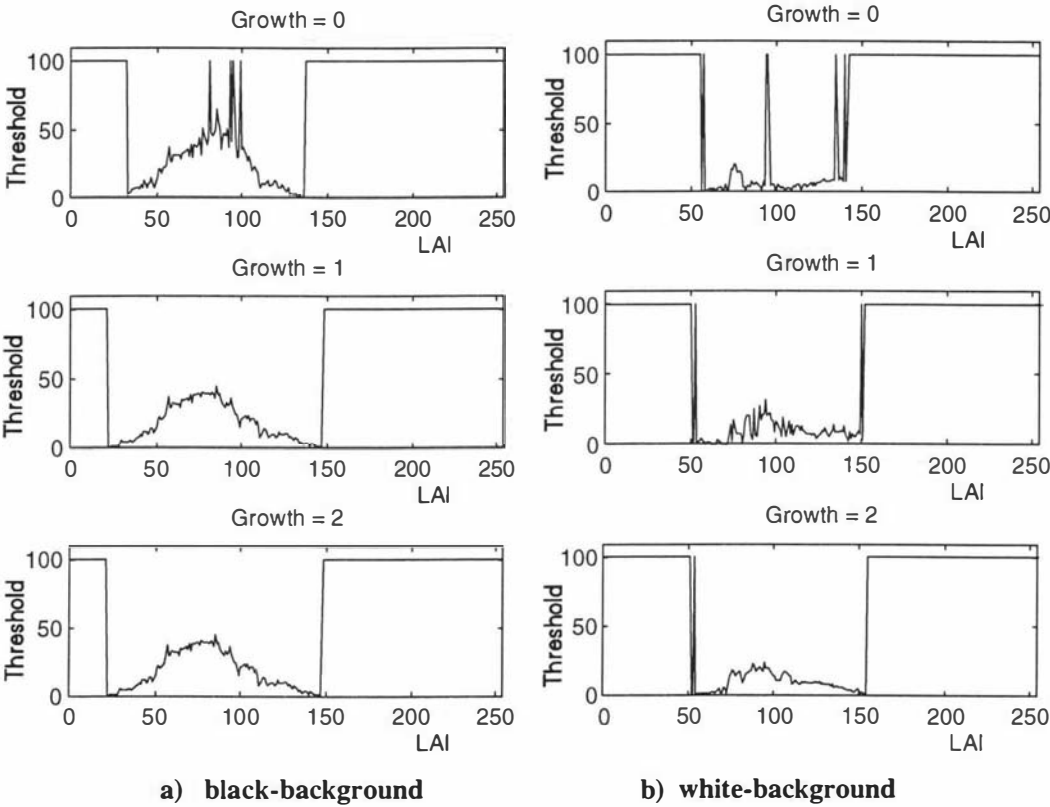


Figure 3.20: Mean edge-strength profiles with growth

The mean profiles for the black-background test image (figure 3.20 a) show little change after the first edge expansion. This is because the image is relatively binary in nature, with only two main region types being distributed about the target edges. On the other hand, the profiles for the white-background test image change rapidly for the first pixel layer, indicating the noisy low contrast nature of this image. Very little change occurs with the growth of the second layer, indicating that most pixels types (ES-LAI combinations) are now included in the profile. Using the minimum, mean and maximum edge-strength profiles with pixel growth, in the Weber edge detection algorithm, the results of table 3.8 are produced. It is immediately noticeable that the minimum profile edge images now contain a very high percentage of the total image pixels for

both test images. The extreme case of this is the white-background edge image for a growth of two layers, where all but two pixels are not detected.

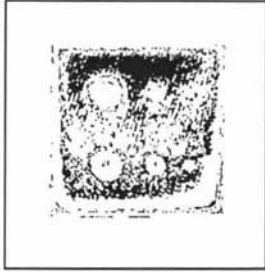
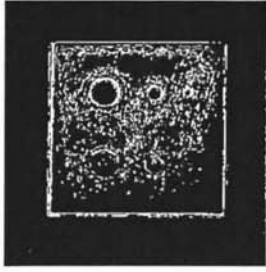


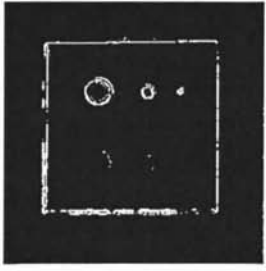
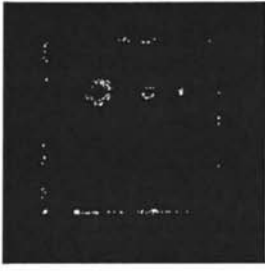
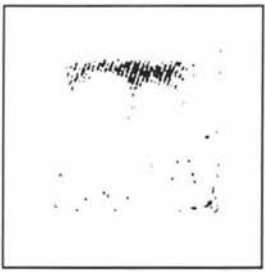
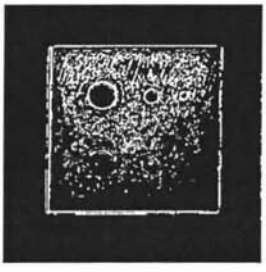

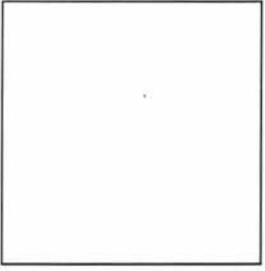
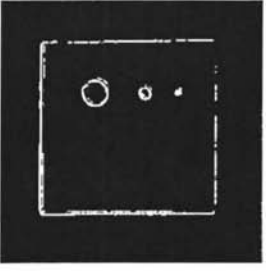

Table 3.8	Minimum Profiled	Mean Profiled	Maximum Profiled
Black background (growth=1)			
White background (growth=1)			
Black background (growth=2)			
White background (growth=2)			

Table 3.8: Direct profile thresholding of test images with growth

The mean profile edge images for the black-background test part now show a complete recovery of the outer edge without the appearance of the lower contour shown in row one of table 3.7. The top through holes are now also complete, however this is accompanied by the detection of surface noise and the partial loss of the blind holes at the bottom. There is little difference between the mean profile results for a growth of one layer and two layers. This is to be expected as the mean edge-strength profiles of figure 3.20a change very little. The most significant

change for the white-background test image is with the mean profiles. The edge image for a growth of one layer no longer shows the contouring at the bottom right-hand side due to uneven illumination. There is improved edge continuity on the left and bottom outside edges, accompanied by some loss of the blind hole edges. Growing another layer on the edge-template and profiling the image further enhances the edge continuity at the bottom and left-hand edges, but results in partial fragmentation of the top and right-hand edges, with the complete loss of the blind holes.

The results of table 3.8 are still relatively poor given the simple composition of the test images. and do not support the hypothesis that a profile computed from pixels on or local to specified edges, uniquely define the edge pixels when used in the McIlroy *et al* algorithm. To determine how well the McIlroy *et al* algorithm can perform on these test images, code was written to tune the mean profiles to maximise Pratt's figure of merit. At each occupied LAI position in the mean profile, the tuning procedure adjusted the edge-strength over a range of plus or minus three standard deviations. The current LAI position in the profile was then set to the edge-strength that maximised the figure of merit. The procedure then moved on to the next position, tuned that location and then continued until it had covered the occupied LAI range. Figure 3.21 contains the tuned profiles and the resulting edge images.

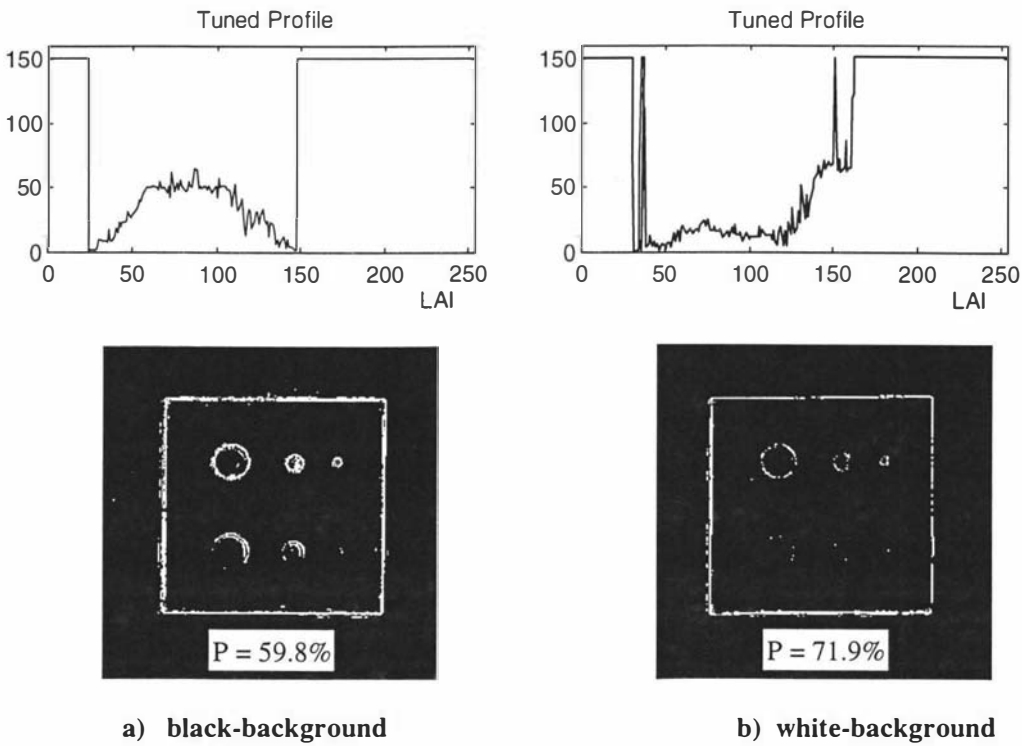


Figure 3.21: Tuned profiles and resulting images

In both cases the tuned profiles are significantly different from the original mean profiles that they started from. The black-background tuned profile is similar in shape to the original zero

growth mean profile of figure 3.19, except that the threshold values are all lower. The white-background tuned profile also shows elevated values compared to the original mean profile. There is a rapid increase at a LAI of approximately 40, accompanied by an almost ramp increase in the 120 to 150 LAI range. The output edge images are greatly improved compared to all the previous results. On tuning, Pratt's figure of merit for black-background image increased from a mere 11.7% to 59.8%. An even more dramatic improvement occurred for the white-background image where the merit figure increased from 22.3% to 71.9%. The edges for the black-background image are contiguous except around edges of the bottom holes. The edges for the white-background image are not quite as complete, particularly down the right-hand side and around the bottom holes, where there is significant edge loss. This image scores a very high figure of merit because the recovered edges are generally only of one pixel width.

It is worth while at this stage to compare best McIlroy *et al* algorithm results with the best that can be achieved using the Canny algorithm at a single filter resolution. Figure 3.22 contains the best experimental results (with respect to Pratt's figure of merit) that could be achieved using the Canny algorithm of section 3.3.2. In both cases the optimal standard deviation of the Gaussian filter was found to be 0.6. The percentile for determining the lower threshold was initially set to 80%, but the resulting images showed complete loss of the bottom holes and the loss of the right-hand and top edge of the white-background image. Setting the value to 30% vastly improved the results for the white-background image, but a value this low would not normally be used because it is not consistent with the basic Canny edge/noise model. In reality this simply reflects the difficulty the Canny algorithm has in dealing with edges with such dramatically different edge strengths. The black-background edge image looks extremely good, yet only achieves a figure of merit of 59.8%. This relatively low score is partly due to the loss of edges around the bottom holes, but mostly due to a one pixel difference in the diameter between the detected holes and the edge-template holes. The white-background edge image also looks very good, except for a double edge on the left hand and bottom edges, and the near total loss of the bottom row of holes. This image scored a figure of merit of only 29.3% because of these last two characteristics. It may be argued that there is in fact a double edge at the left-hand and bottom sides of the test part due to shadowing. However this edge was not considered to be significant by the human observers due to its very close proximity to the physical edge of the test part.

The results of figure 3.18 indicated that the McIlroy *et al* algorithm can be very effective, even when compared to the superior, though highly computationally expensive Canny algorithm. The difficulty with the McIlroy *et al* algorithm is that there is still no defined way to derive the optimal profile as it is not uniquely defined by the ES-LAI statistics of the edges.

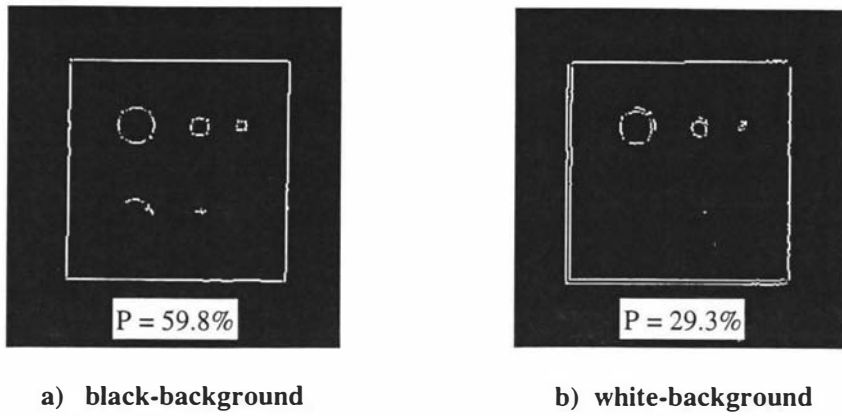


Figure 3.22: Single resolution Canny results

3.4.3. Direct Thresholding using ES-LAI Edge-template Statistics

Back at the start of this section, Weber's law was introduced and defined by equation 3.21 as the ratio of change in local intensity (edge-strength) to the local average intensity and was equal to a constant. The McIlroy *et al* algorithm does not actually implement this equation, but instead declares an edge pixel if the edge-strength is greater than the profile value at the computed LAI. In effect, the algorithm only allows a very limited range of unique Weber ratio's (ES to LAI) to be defined by the 1-D profile function. It may be because of this that the mean profiles were not found to uniquely define the target edges of the test images. To test this hypothesis, the two dimensional ES-LAI statistics, computed about the edge-template, should be used to directly threshold the image. Suitable code was written in which a pixel is declared an edge pixel, only if at the ES-LAI location in the 2D statistics array at least one pixel is recorded. The results of applying this algorithm to the two test images is shown in table 3.9.

The first column of this table are the results of applying the ES-LAI statistics from the manually generated edge-templates to the two test images. All the specified edges are recovered but they are accompanied by many non-edge noise pixels, particularly for the white-background image. The recovered edges do have the redeeming property that they are generally only one pixel wide. An exception to this are the edges of the blind holes at the bottom which are lost in noise. This effect is further illustrated by the results in the third column of table 3.9. These results were produced by applying the ES-LAI statistics from the edited Canny edge-templates of column two. These edge-templates were generated from the Canny algorithm output images of figure 3.22, by manually filling in any missing edge pixels and removing any noise pixels. The black-background edge image of column three shows significant improvement over the result of column one. There are virtually no background noise pixels and the top row of holes is recovered fully with few noise pixels. Little improvement is shown for the bottom holes and edge, in fact there is a small increase in noise pixels in this area. The white-background edge image of column three

is significantly worse than the result of column one. Many more non-edge pixels are recovered over most areas in the image.

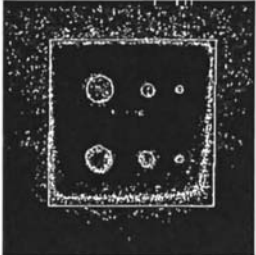
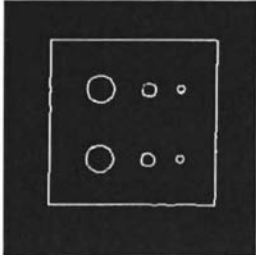
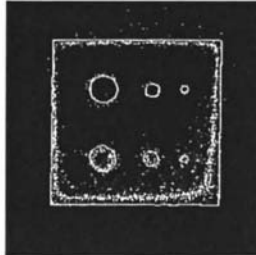
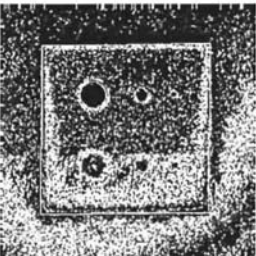
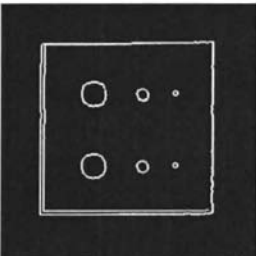
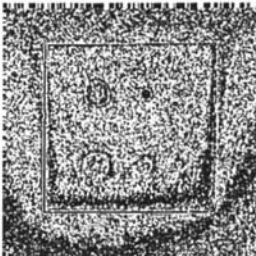
Table 3.9	2-D profiled using manual edge-template	Edited Canny edge-template	2-D profiled using edited edge-template
Black background	 <p>$P = 25.1\%$</p>		 <p>$P = 35.9\%$</p>
White background	 <p>$P = 6.9\%$</p>		 <p>$P = 5.8\%$</p>

Table 3.9: Direct thresholding with the two dimensional ES-LAI statistics

The overall conclusion that can be drawn from these results is that the ES-LAI statistics computed about the target edges using 2×2 neighbourhoods, define relatively uniquely the edge pixels only when there is a high signal-to-noise ratio. A potential reason for this result is the poor performance of the Roberts product in noisy images and the asymmetric response of 2×2 kernels. More robust results should be produced by using a 3×3 neighbourhood and computing the edge-strength using a Sobel mask.

Table 3.10 contains the results of using the ES-LAI statistics computed about the edge-templates, on a 3×3 neighbourhood, to directly threshold the test images. The first column used the manually generated edge-template and the second column the edited Canny templates of table 3.9. There is little difference between the black-background edge images for either template and this is reflected in their merit scores. Overall the images are slightly better than the best results obtained using a 2×2 neighbourhood. The white-background edge images show a significant improvement with respect to noise rejection, particularly for the Canny based template. This confirms that the pixel placement in edge-template is more important when the image is of low signal-to-noise ratio.

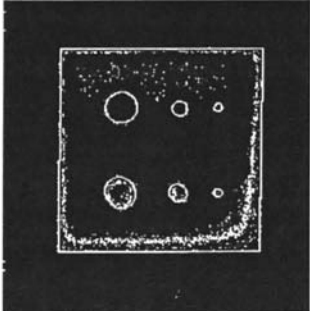
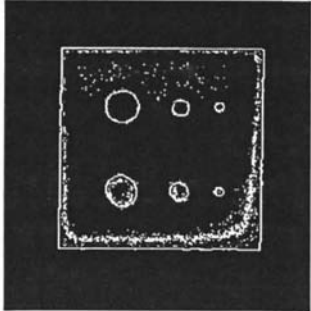
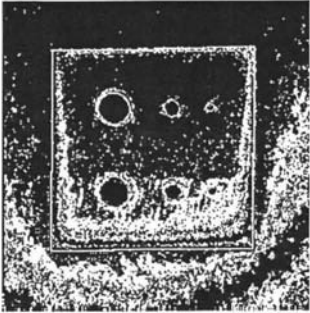
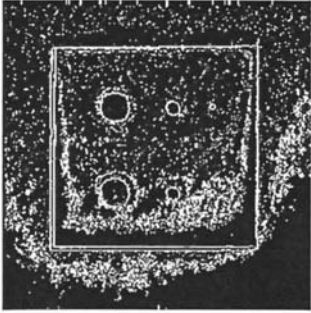
Table 3.10	2-D profiled using manual edge-template	2-D profiled using edited edge-template
Black background	<div><p>P = 41.9%</p></div>	<div><p>P = 42.3%</p></div>
White background	<div><p>P = 7.7%</p></div>	<div><p>P = 16.9%</p></div>

Table 3.10: Direct thresholding based on 3x3 neighbourhood ES-LAI statistics

Although the use of a 3x3 neighbourhood to compute the ES-LAI edge statistics improved the quality of the recovered edges and hence implies that the edge pixels are better defined by these statistics, overall the results are still unsatisfactory and do not support the hypothesis that edge pixels are uniquely defined by their ES and LAI statistics.




Manually generated edge-template	2-D profiled 2x2 neighbourhood	2-D profiled 3x3 neighbourhood
		

Table 3.11: Direct thresholding of cervical spine test image

The main purpose of this section was to develop a high-performance edge-extraction algorithm suitable for cineradiographic images. Although the performance on the test-part images was not that good, it is worth while to see how well direct thresholding using ES-LAI statistics performs on cineradiographic images. Table 3.11 contains the results of applying the ES-LAI statistics computed about the edge-template, to directly threshold the cineradiographic cervical spine test image. Both edge images clearly show the recovery of the original edge-template pixels with full connectivity. The 3x3 neighbourhood edge image displays far greater suppression of non-edge pixels than the 2x2 neighbourhood image, but overall the results are still unsatisfactory.

3.5. Summary of Edge Detection in Cineradiographic Images

At the start of this chapter the concept of image segmentation was introduced. Image segmentation attempts to decompose a scene into its components. With respect to determining vertebral motion in spinal cineradiographic sequences, the components of interest are the vertebral boundaries. If in each frame of the sequence the vertebral boundaries can be reliably extracted, then the problem of determining the vertebral motion is relatively straight forward. Measuring the difference in position and orientation of each vertebral boundary through the sequence will describe the vertebral motion and hence the motion of the spine as a whole. None of the edge detection methods investigated in this chapter have been able to reliably extract the vertebral boundaries of the cervical spine test image. Recovered edges were generally fragmented and noisy.

The test image used to evaluate the various edge detection schemes was from the middle of a typical cervical spine sequence. An image from a cervical sequence was chosen because these images tend to be of lower quality than images of the lower (lumbar) or middle (thoracic) spine. There are two reasons for this loss of quality. Firstly, the vertebrae occupy only a small fraction of the area of the frame due to the requirement to include the entire skull in the frame. This results in a relatively low spatial resolution for the vertebrae once digitised. The second image quality factor concerns the dynamic range of intensity. Although there is little soft tissue to scatter the X-ray beam, part of the scene will contain the X-ray beam unattenuated. The portion of the total area containing the unattenuated beam will vary depending on where the neck is in the scene. This would not be a problem if the intensity dynamic range of the imaging system was very large. A further complication is that the X-ray strength when imaging this area of the spine has to be kept low in order to prevent *burn-in* of the image intensifier tube. Even with this low level, the range of intensities that the camera experiences are very large. Cameras designed to work with X-ray image intensifiers are very sensitive compared to daylight cameras, but this high sensitivity is gained at the expense of increased noise and reduced intensity range handling. In order to prevent the camera sensor overloading, cameras are fitted with auto-gain circuitry. This circuitry dynamically adjusts the effective gain of the sensor based on the average intensity

in the scene. The effect of this gain adjustment over the passage of a sequence is to modulate the image contrast. For sequences of the cervical spine, with direct X-ray feed-through, this gain modulation is significant and can dramatically reduce the contrast of the images.







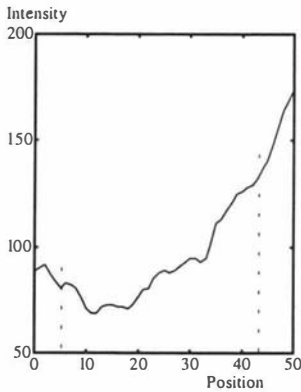
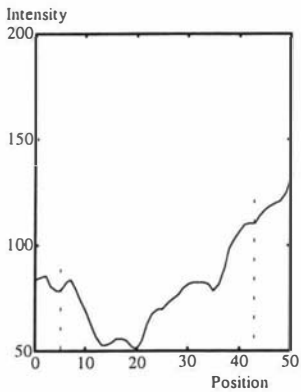
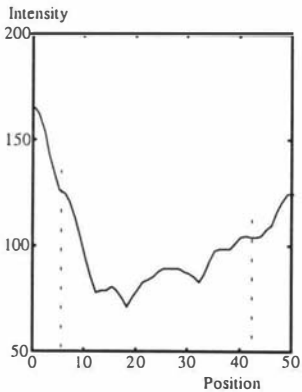
Table 3.12	First Image	Middle Image	Last Image
Full			
Section			
Line Profile C2			

Table 3.3: Contrast Modulation

The first row of table 3.12 contains frames from the start, middle and end of a typical cervical cineradiographic sequence in which the subject moved from full *flexion* (neck tucked down) to full *extension* (neck extended back). The second row of this table contains enlarged views of the centre of the neck covering the first to fifth vertebrae. It is clear from these views that there is a considerable difference in contrast through the sequence. Maximum contrast is reached in the middle of the sequence where the amount of direct X-ray feed-through is balanced either side of the neck. At the extremes of the movement the feed-through area is predominantly at the back or front of the neck resulting in an increase in the average intensity. The auto-gain circuitry

compensates for this by reducing the camera gain and hence the contrast of the image. This effect is further illustrated by an intensity line profile taken through the second vertebrae. In these profiles the approximate position of the soft tissue to bone interface is marked with a vertical dotted line. At these transitions it is clear that the difference in intensity level is maximum for the middle image but is significantly reduced for the two end images.

How the change in image contrast affects edge detection performance has not been explored so far in this chapter as none of the methods investigated were able to reliably extract the vertebral edges from an image in the centre of a sequence. To briefly explore the effect contrast modulation has on the Canny edge-detection algorithm (this algorithm scored the highest figure of merit on a single frame), the algorithm was applied to the three cervical sections contained in table 3.12. Only a single resolution was used with the internal algorithm parameters adjusted individually for each image to produce the best visual results. The resulting edge images are shown in table 3.13. It is clear from these images that the change in image contrast due to the auto-gain circuitry, dramatically effects edge-detection performance. The edges detected in the middle image are reasonably good, while the edges recovered for the two end images are extremely fragmented and very noisy, with almost complete loss of the vertebral outlines.




First Image	Middle Image	Last Image
		

Table 3.4: Contrast effects on Canny Algorithm performance

Section 3.3.4 of this chapter covered several methods for combining fragmentary or noisy edge data. Tracing methods are generally only effective for filling in small gaps in boundaries and thus are unlikely to be useful on the edge images of table 3.13. Generalised fitting methods, such as the Hough transform, would require a geometric model for each vertebral boundary. Determining a suitable model to cover all the natural vertebral shape variations would be possible, but it is likely to be highly complex. The reason for this complexity can be easily illustrated by looking at the variation in vertebral shape that occurs in the middle image of table 3.13. The shape of the front of a vertebral body is similar for most of the vertebrae, only the relative size changes. This is not the case for the back of the vertebrae where the

spinous-process protrudes. Here the complex shape varies dramatically between individual vertebra. Assuming that a suitable geometric model has been defined, then the computational cost of the fitting process will be extremely high, unless some of the model parameters of the individual vertebra can be estimated. The estimation of the main model parameters may be able to be achieved by first applying an edge-linking procedure to the edge-fragments computed by a Canny algorithm on the middle image in a sequence. However towards the ends of a cervical sequence the recovered edge information becomes so sparse and noisy (see table 3.13) that the fitting procedure is unlikely to produce stable results.

More effort could be expended on exploring the vast range of edge detection algorithms described in the literature. Current thinking would indicate that a multi-resolutional, multi-model approach involving adaptive techniques would be the most appropriate place to start. This approach, if successful, is likely to be extremely computationally expensive and require special hardware if the operation is to be completed in a realistic time. Assuming that all this is can be successfully achieved, then the original task of measuring the vertebral motion could be tackled.

The reason for attempting to use edge-detection methods to extract the vertebral boundaries was that this would greatly simplify motion measurement since tracking binary boundaries is relatively straight forward. As reliable segmentation has not been achieved, then an alternative approach would be to attempt to track the actual gray-scale characteristics of the vertebra in the original sequence. The next chapter introduces the fundamental difficulties associated with motion perception. Various strategies for measuring and tracking motion are overviewed and a suitable approach to motion measurement in cineradiographic images is proposed.

3.6 References

- [3.1] **Rosenfeld A. and Kak A.C.** *Digital picture processing*, 2nd edition, Volume II, Academic Press, Orlando, USA, 1982.
- [3.2] **Burt P.J.** *The pyramid as a structure for efficient computation*. In Multiresolution image processing and analysis, edited by Rosenfeld, A., Springer Series in Information Sciences, Vol. 12, Springer, New York, 1984.
- [3.3] **Brice C.R. and Fennema C.L.** *Scene Analysis using Regions*. In Computer methods in Image Analysis, edited by Aggarwal J.K., Duda, R.O., and Rosenfeld, A., Los Angeles: IEEE Computer Society, 1977.
- [3.4] **Horowitz N.J. and Pavlidis T.** *Picture segmentation by split-and-merge procedure*. Proceedings of the 2nd IJCPR, pp 424..433, August 1974.
- [3.5] **Attneave F.** *Some informational aspects of visual perception.*, Psychological Review No. 61, 1954.

- [3.6] **Roberts L.G.** *Machine perception of three-dimensional solids.* In Optical and Electro-optical Information Processing, Tippet J.P. et al editors, Cambridge, MA: MIT Press, 1965.
- [3.7] **Prewitt J.M.S.** *Object enhancement and extraction.* In Picture Processing and Psychopictorics, Lipkin, B.S., and Rosenfeld, A. editors, New York: Academic Press, 1970.
- [3.8] **Pratt W.K.** *Digital Image Processing.* New York: Wiley Interscience, 1978.
- [3.9] **Kirsch R.A.** *Computer determination of the constituent structure of biological images.* Computers and Biomedical Research Vol. 4 (3), pp 315..328, June 1971.
- [3.10] **Kreyszig E.** *Advanced Engineering Mathematics*, 6th Edition, pp 457..504, Wiley, 1988.
- [3.11] **Jain A.K. and Ranganath S.** *Image restoration and edge extraction based on 2-D stochastic models.* Proceedings of ICASSP-82, Paris, May 1982.
- [3.12] **Jain A.K.** *Fundamental of Digital Image Processing*, Prentice-Hall Information and System Sciences Series, Prentice-Hall, 1989, pp. 285-286, ISBN -013-332578-4.
- [3.13] **Pratt W.K.** *Digital Image Processing.* New York: Wiley Interscience, 1978.
- [3.14] **Abdou I.E.** *Quantitative methods of edge-detection*, USCIP Report 830, Image processing Institute, University of Southern California, July 1978.
- [3.15] **Kitchen L. and Rosenfeld A.** *Edge evaluation using local edge coherence.* IEEE Transactions of Systems, Man, and Cybernetics, Vol. SMC-11, No. 9, pp 597..605, Sept. 1981.
- [3.16] **Delp H.J and Chu C.H.** *Detecting Edge Segments.* IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-15, No. 1, pp 144..152, Jan./Feb. 1981,
- [3.17] **Abdou I.F. and Wong K.Y.** *Applied Numerical Analysis.* 2nd Edition, Addison-Wesley, 1978.
- [3.18] **Marr D. and Poggio T.** *A computational theory of human vision.* Proceeding of the Royal Society of London, B 204, pp 301..328, 1979.
- [3.19] **Logan B.F. Jnr.** *Information in the zero-crossings of band pass signals.* Bell Systems Technical Journal, No. 56, pp. 487-510, 1977.
- [3.20] **Marr D. and Hildreth E.** *Theory of Edge Detection.* Proceedings of the Royal Society of London, B 207, pp 187..217, 1980.
- [3.21] **Canny J.** *A computational approach to edge detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, p. 679, November 1986.

- [3.22] **Martell A.** *Edge-detection using Heuristic Search Methods.* Proceedings of Computer Graphics and Imaging 1, pp 169..182, August 1972.
- [3.23] **Leipnik R.** *The extended entropy uncertainty principal.* Information Control 3, pp 18..25, 1960.
- [3.24] **Marr D., Poggio T. and Ullman S.** *Bandpass channels, zero-crossings, and early visual information processing.* Journal Of The Optical Society of America 69, pp 914..916, 1979.
- [3.25] **Wilson H.R. and Bergen J.R.** *A four mechanism model for spatial vision.* Vision Research 19, pp. 19-32, 1979.
- [3.26] **Page W.H. and Pugmire R.H.** *Internal communiqué,* Department of Production Technology, Massey University, New Zealand, 16 January 1994.
- [3.27] **Page W.H. and Pugmire R.H.** *Internal communiqué,* Department of Production Technology, Massey University, New Zealand, 8 February 1994.
- [3.28] **Spacek L.A.** *Edge detection and motion detection.* Image Vision Computing, vol. 4, pp 43..51, 1986.
- [3.29] **Deriche R.** *Using Canny's criteria to recursively implemented optimal edge detector.* International Journal of Computer Vision, Vol. 1, No. 2, 1987.
- [3.30] **Frei W. and Chen C.** *Fast boundary detection: A generalisation and a new algorithm.* IEEE Transactions on Computing, pp 988..998, Oct. 1977.
- [3.31] **Bracewell R.N.** *The Fourier Transform and its application.* McGraw-Hill, pp 56, 1986.
- [3.32] **Davis L.** *A survey of edge detection techniques.* Computer Graphics and Image Processing, Vol. 4, pp 248..270, 1975.
- [3.33] **Robinson G.** *Edge detection by compass gradients.* Computer Graphics and Image Processing, Vol. 6, pp 492..501, 1977.
- [3.34] **Nevatia R.** *Locating Object boundaries in textured environments.* IEEE Transactions on Computing, Vol. 25, pp 1170..1180, Nov. 1976.
- [3.35] **Martelli A.** *Edge detection using heuristic search methods.* Computer Graphics Image Processing 1, pp 169..182, Aug. 1972.
- [3.36] **Bellman R.E. and Dreyfus S.** *Applied Dynamic Programming,* Princeton, N.J., Princeton University Press, 1962.
- [3.37] **Hough P.V.C.** *Method and means of recognising complex patterns.* US Patent no. 3,069,654, 1962.

- [3.38] **Anderberg M.R.** *Cluster Analysis for Application.* New York: Academic Press, 1971.
- [3.39] **Ballard D.H. and Brown C.M.** *Computer Vision.* Prentice-Hall, 1982.
- [3.40] **Princen J.** *Hough Transform methods for curve detection and parameter estimation.* PhD thesis, University of Surrey, June 1990.
- [3.41] **Stockman G.C. and Agrawala A.K.** *Equivalence of Hough transform curve detection to template matching.* Communications of the ACM, Vol. 20, pp 820..822, 1977.
- [3.42] **Hecht S.** *The visual discrimination of intensity and the Weber-Frechner law.* Journal of General Physiology, pp 241..250, 1924.
- [3.43] **McIlroy C.D., Linggard R. and Monteith W.** *Hardware for real-time image processing.* IEE Proceedings, Vol. 131, part E, no. 6, November 1984.

Motion-tracking: An Overview

4.1 Visual Motion Perception and Description

Our visual perception of motion is fundamental to our interaction and understanding of the world. It is the principle method by which we determine relative motion between different objects in the world. It also provides a rich source of information about depth and surface structure of objects. Pictures or two-dimensional projections of a scene represent a single shot in time and space where all information is represented by the intensity variation across the projection. Changing the viewpoint of the projection over time enables three-dimensional structural information to be collected and visual motion inferred.

When an object moves relative to a camera, points on the surface of the object generate trajectories in space and time. The projection of these 3-d trajectories onto the camera sensor, as shown in figure 4.1, produces 2-d paths, the derivatives of which are *2-d velocities*. The entire collection of these 2-d velocities forms what is commonly referred to as a 2-d motion field. The measurement of this motion field is called the *optical flow* [4.1].

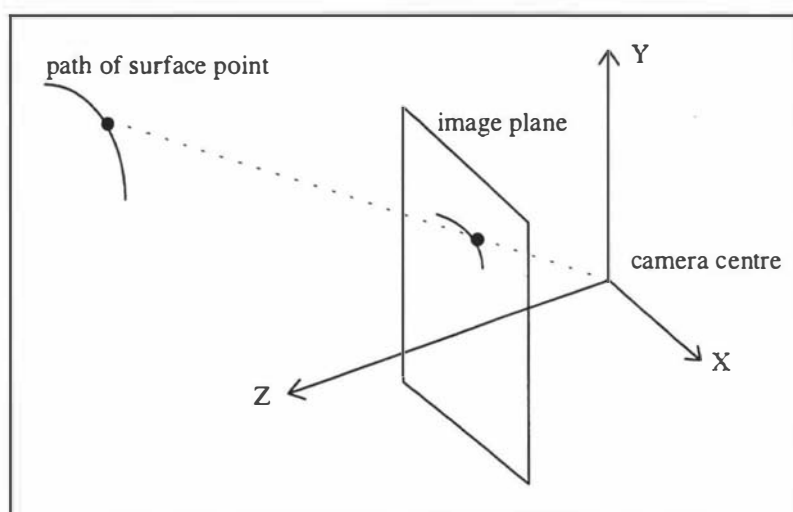


Figure 4.1: Motion Field

In order to measure the image velocity, local descriptions of the motion field must be extracted from the spatiotemporal patterns of image intensity. The difficulty in measuring the image

velocity is that the intensity depends on several independent aspects of the image formation process. For the simple case where the object is a plane surface, temporal image intensity variations may be caused by:

Perspective Projection: As the surface moves in 3-d space the relative image locations of the projection of nearby points on the surface, undergo geometric deformations from image to image. Translation occurs as the surface moves across the *line-of-sight* of the camera. Rotation and dilation occur as the surface rotates or moves toward or away from the camera;

Photometric Effects: The reflectance properties of the surface such as shading or mirror-like reflection can significantly affect the image intensity over time. Similarly variations in scene illumination can also be significant;

Camera Distortions: The sensor may cause variations in relative image intensity due spatiotemporal smoothing (motion blur) or automatic gain control.

Because of the different sources of intensity variation, the image intensity corresponding to a single surface point is unlikely to remain constant at all times. Hence tracking points of constant intensity will not, in general, produce satisfactory estimates of the motion field.

The fundamental problem in defining image velocity is that of devising a complementary measurement technique that yields a good approximation to the 2-d motion field. An alternative approach to the measurement of image velocity is to track an image property or feature from frame to frame. With sufficient features or image properties specified, it would be possible to build up an entire velocity field or optical flow of a motion sequence.

Once the complete optical flow of a sequence has been constructed it must then be interpreted to extract information about the scene. This interpretation generally begins by going back to the original intensity images and spatially segmenting each frame into areas or features. This segmentation process is most commonly based on the detection of sudden changes in intensity, potentially indicating edge boundaries. The areas defined in the intensity spatial segmentation stage are then used to spatially segment the velocity field of each frame in the sequence. The velocity field for a simple sequence where a planar square object is moving at a constant velocity to the right of the visual field would be depicted as in figure 4.2a. The length of the arrows indicate the magnitude of the local velocity and the orientation the direction. The interpretation of this simple velocity field is straight forward, all vectors point in the same direction and are of the same length. In figure 4.2a there are only two features indicated based on edge information, a square object (dotted lines) and a constant intensity background. If this motion continued through the sequence then the interpretation of the entire scene would be straight forward. However if at some point in the sequence the translation stopped and object began to rotate

anti-clockwise at a constant speed then the velocity field would look like figure 4.2b. Edge segmentation still reveals a single object, however describing the motion based on the magnitude and orientation of the individual velocity vectors is now more difficult. Making the assumption that the square is rigid and moving in a single plane enables the vectors to be grouped together into a system of equations that can be solved to yield an estimate of the rotational velocity. Similarly if both translation and rotation occur simultaneously in the scene and the two previous assumptions hold, a system of equations can be formed and solved for the translational and rotational velocity (speed and orientation). For this simple approach to work it is vital that each frame in the sequence can be reliably segmented spatially to define each rigid object. This implies that photometric effects are minimal and edge information is near complete. If spatial segmentation cannot be performed reliably then grouping individual vectors is no longer trivial and requires additional assumptions concerning their association.

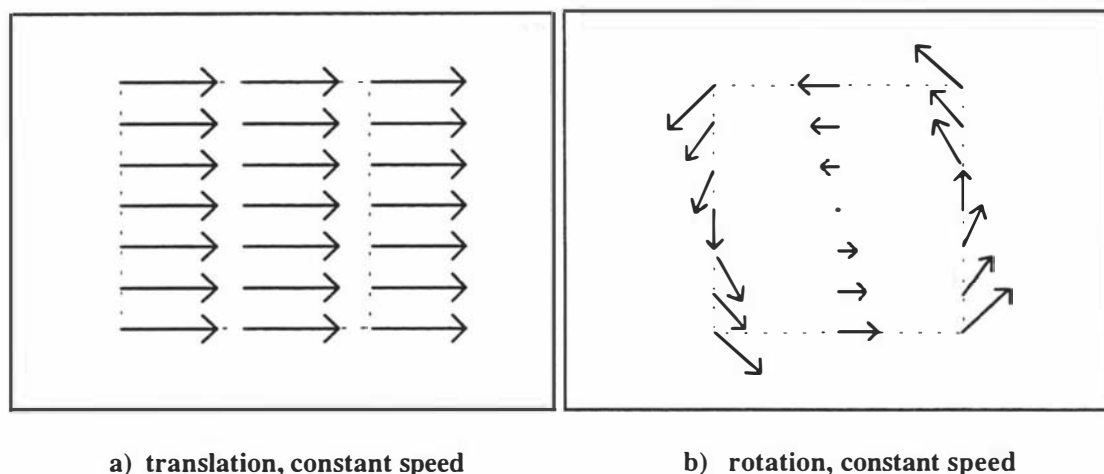


Figure 4.2: Simple velocity field

In real scenes the measured velocity fields will be noisy, perspective and photometric effects will be significant, camera distortion due to motion blur may be significant, and object occlusion will occur. Furthermore, objects will not all be rigid, they are likely to deform increasing photometric effects and invalidating the previous motion assumptions. The real difficulty is in interpreting the local velocity information contained in the optical flow data, to determine the nature of the motion in a scene.

A simpler approach is possible, that is free from the difficulty of interpreting the local velocity information when the objects of interest in the scene can be specified in advance. If the objects of interest can be spatially segmented in the first frame, then the visual motion problem is reduced to tracking the objects or features through the sequence. This method is not free from the difficulties caused by temporal image intensity variations. However if the object tracking is reliable, then motion interpretation is straight forward. The most significant problem with

tracking an entire object concerns how to deal with object occlusion and deformation. The apparent deformation may be due to perspective effects or be real as in the case of an object changing its shape. The tracking algorithm must adapt to the changing shape of each object. Partial occlusion will also result in an apparent change in shape, while total occlusion will cause this method to fail all together.

4.1.1 The Aperture Problem

Two-dimensional translation is the simplest form of intensity variation that can be used to model the local temporal variations in an image sequence. In digital image sequences the information is discrete both in time and space. Estimating the local 2-d translation from the intensity variation usually involves defining a window in both space and time over which the translation is calculated. As long as this spatiotemporal region of support or aperture is narrow, the 2-d translation model provides an adequate description of the local motion without having to resort to a more complex description (account for rotation, dilation, etc. effects) over a larger neighbourhood. This localisation also helps to ensure good resolution of the velocity field as it is not necessary to determine where within the window of measurement support that the velocity estimate applies. Localisation also helps to minimise the adverse effects of occluding boundaries and the measurement of spatially disjoint objects, by treating them independently.

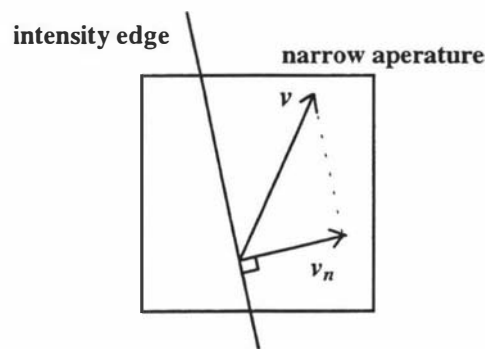


Figure 4.3: Component velocity and the aperture problem

If measurements are restricted to a narrow aperture, then the intensity structure upon which the measurements are based will often be one dimensional. This effect is illustrated in figure 4.3 where the intensity structure appears as a single edge. In this situation it is only possible to reliably measure the normal component v_n , of the 2-d velocity field v . This is referred to as the *aperture problem* [4.2]. The reason for this restriction can be readily explained by the fact that there are no clues to indicate motion in-line with the 1-d intensity structure as all positions along the edge appear the same when viewed from the small aperture. Hence the component of

velocity tangential to the edge will not be distinguishable. Only movement perpendicular to the orientation of the edge will be readily perceivable, and hence be able to be reliably measured.

This problem is inherent to any local scheme for calculating motion. The flow-on-effect of this problem is that it is not possible to fully recover motion using local information alone.

4.2 Matching Approaches

Region-based matching techniques aim to find the *best match* between image regions in one frame with neighbouring regions in the subsequent frames. Commonly this has been formulated as finding the *degree-of-fit* based on some form of correlation measure.

Let $I_k(m, n)$ be image k in a sequence and $W(i, j)$ denote a 2-d window function centred at (x, y) with $|W(x+i, y+j)| \rightarrow 0$ as $\|(i, j)\| \rightarrow \infty$. Then $W(x+i, y+j)I_k(x, y)$ denotes a windowed patch in image k . The estimate of the local motion can be calculated by finding the displacement $(\Delta x, \Delta y)$ about (x, y) in image $k+1$ that yields the best fit between the image patches:

$$A_k(x, y) \equiv W(x+i, y+j)I_k(x, y) \quad \text{eq. 4.1}$$

$$A_{k+1}(x+\Delta x, y+\Delta y) \equiv W(x+\Delta x+i, y+\Delta y+j)I_{k+1}(x, y) \quad \text{eq. 4.2}$$

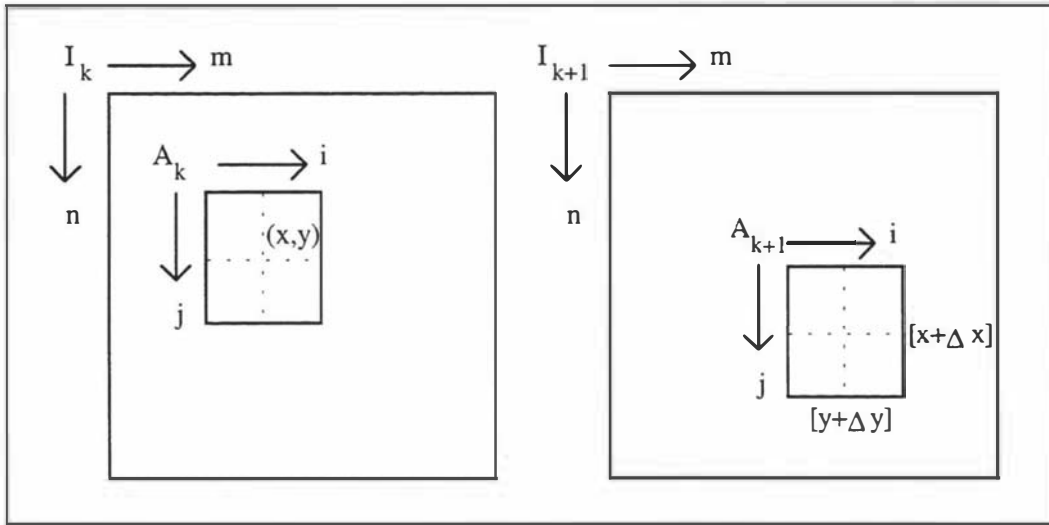


Figure 4.4: Region-based matching

Finding the best match is essentially an optimisation problem in which a similarity measure is maximised or a difference measure minimised, over the local displacement $(\Delta x, \Delta y)$. The most commonly used similarity measure involves the maximisation of the normalised discrete cross-correlation:

$$C(A_k(x, y), A_{k+1}(x + \Delta x, y + \Delta y)) = \frac{\langle A_k(x, y), A_{k+1}(x + \Delta x, y + \Delta y) \rangle}{\|A_k(x, y)\| \|A_{k+1}(x + \Delta x, y + \Delta y)\|} \quad \text{eq. 4.3}$$

A commonly used difference measure involves the minimisation of the sum-of-squares difference (SSD):

$$D(A_k(x, y), A_{k+1}(x + \Delta x, y + \Delta y)) = \|A_k(x, y) - A_{k+1}(x + \Delta x, y + \Delta y)\|^2 \quad \text{eq. 4.4}$$

4.2.1 Matching in the Spatial Domain

Spatial domain methods work directly in the image space with the intensity value of each pixel. The window function $W(i, j)$, is simply an aperture into the image that limits the match area to the object or neighbourhood of interest. Effectively $A_k(x, y)$ is an object template for which a local search is made in the next frame to find the best match.

The sum-of-squares difference, defined in equation 4.4, can be applied directly in the spatial domain by simply taking the difference between each pixel intensity in the object template A_k and the corresponding pixels in the match test area A_{k+1} , of the next frame. This measure can be written as:

$$D_2 = \sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) - A_{k+1}(i, j)]^2 \quad \text{eq. 4.5}$$

An alternative measure to SSD is the absolute sum difference (ASD). The squared difference is replaced by the computationally less expensive absolute function. Equation 4.6 defines the ASD measure.

$$D_1 = \sum_{i=0}^m \sum_{j=0}^n |A_k(i, j) - A_{k+1}(i, j)| \quad \text{eq. 4.6}$$

Similarly the normalised discrete cross-correlation measure of equation 4.3 can be applied directly in the spatial domain as:

$$C_1 = \frac{\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) * A_{k+1}(i, j)]}{\sqrt{\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j)]^2 * \sum_{i=0}^m \sum_{j=0}^n [A_{k+1}(i, j)]^2}} \quad \text{eq. 4.7}$$

If the correlation measure is used as the match statistic then the object is located in the next frame by searching for the largest peak. Alternatively if one of the difference measures is used then the object will be located by searching for the largest trough. In both cases this will occur when the observed area and the template are spatially registered. In real sequences the object will not only be spatially translated but will also be rotated and possibly scaled. This is

illustrated by equation 4.8 where s_x and s_y are scale factors, p and q are the displacement values, and θ is the angle of rotation of the observed area with respect to the template:

$$A_k(x, y) = A_{k+1}\left(\frac{x+p}{s_x}, \frac{y+q}{s_y}, \theta\right) \quad \text{eq. 4.8}$$

Finding the best match now involves searching the parameter space (p, q, s_x, s_y, θ) . This becomes impractical unless reasonable estimates of s_x, s_y and θ are known.

Moment Invariants

One way to work in the spatial domain but reduce the complexity of searching the parameter space is to use a match function that is invariant to geometric transformations such as translation, scaling, rotation and reflection.

Let $f(x, y) \geq 0$ be a real bounded function with support on a finite region R . The discrete $(u+v)$ th order moment is defined as:

$$m_{u,v} = \sum_R \sum f(x, y) x^u y^v \quad u, v = 0, 1, 2, \dots \quad \text{eq. 4.9}$$

The moment representation theorem [4.3] states that the infinite set of moments $\{m_{u,v}, u, v = 0, 1, 2, \dots\}$ uniquely determines $f(x, y)$, and vice-versa. A reconstruction formula exists by which the original signal can be recovered from the infinite set of moments. However the series representation of the reconstruction formula cannot be simply truncated to find an approximation to the original signal.

In spite of this difficulty, if moments of $f(x, y)$ are calculated up to order N , then a match statistic can be formed by comparing each moment to the corresponding moment for the test area. A minimisation function may be formed by taking the sum-of-squares difference over all N moments.

In situations where geometric scaling is not significant, the limiting factor in searching the parameter space is dealing with rotation. Performing a direct template match would require the rotation of the match area with respect to the template. The geometric rotation of discrete data is in itself not a trivial task. Great care has to be taken to minimise the effects of spatial aliasing [4.4] if a correct match is to be found. A suitable procedure would consist of interpolating the match area to form a continuous description, rotating the continuous description, then finally resampling the rotated description back onto the original discrete pixel grid. This series of operations would have to be performed for each iteration of the search with great computational cost.

Using the theory of algebraic invariants [4.5] it is possible to find certain polynomial moments that remain unchanged under the general linear coordinate rotation transform defined in equation 4.10, where θ is the angle of rotation.

$$\begin{bmatrix} x \\ y \end{bmatrix}_{New} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{Old} \quad eq. 4.10$$

These moment invariants are constructed from central moments defined by:

$$\mu_{p,q} = \sum_R \sum [(x - \bar{x})^p (y - \bar{y})^q f(x, y)] \quad eq. 4.11$$

where $\bar{x} = m_{1,0} / m_{0,0}$ and $\bar{y} = m_{0,1} / m_{0,0}$.

Table 4.1 contains the first, second and third-order polynomial central moments that are invariant to both rotation and reflection.

First-order moments (always invariant)	$\mu_{0,1} = \mu_{1,0} = 0$
Second-order moments ($u+v=2$)	$\phi_1 = \mu_{2,0} + \mu_{0,2}$ $\phi_2 = (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2$
Third-order moments ($u+v=3$)	$\phi_3 = (\mu_{3,0} - 3\mu_{1,2})^2 + (\mu_{0,3} - 3\mu_{2,1})^2$ $\phi_4 = (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{0,3} + \mu_{2,1})^2$ $\phi_5 = (\mu_{3,0} - 3\mu_{1,2})(\mu_{3,0} + \mu_{1,2})^*$ $\quad [(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2]$ $\quad + (\mu_{0,3} - 3\mu_{2,1})(\mu_{0,3} + \mu_{2,1})^*$ $\quad [(\mu_{0,3} + \mu_{2,1})^2 - 3(\mu_{1,2} + \mu_{3,0})^2]$ $\phi_6 = (\mu_{2,0} - \mu_{0,2})[(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2]$ $\quad + 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{0,3} + \mu_{2,1})$

Table 4.1: Central moment invariants

Higher-order moment invariants can be formed, but their computational complexity increases significantly. Moment invariants up to third-order have been used successfully in shape recognition and scene matching applications [4.6, 4.7].

Surface fitting

An alternative way to work in the spatial domain and reduce the computational cost of the match statistic is to approximate the image data with a simple geometric function. A low-order

polynomial surfaces is fitted to the match template and the test region in the next frame. A similarity measure can then be computed by some sort of weighted difference of the polynomial coefficients. In approximating the image regions with a polynomial surface, the information or structure has become filtered in some way. If the form of the polynomial is changed then the nature of the filtering effect also changes. An effective way to deal with this problem is to adaptively fit a polynomial to the match template [4.8]. This can be accomplished by starting with the simplest form of the surface polynomial and computing a *goodness-of-fit* measure such as the total sum-of-squares difference. The form of the polynomial is slowly made more comprehensive (more terms are added) until the *goodness-of-fit* measure reaches a maximum. The form of the polynomial at this stage is then used in the *fit-and-match* process outlined above. This adaptive process ensures that the simplest polynomial that best approximates the template data is used.

Preprocessing

The sharpness of the best match is dependent on both the match statistic and the structure of the intensity information contained in the template. Sharpness relates to how rapidly the match value changes for small perturbations away from the optimal position in the parameter space. One way to improve the match sharpness is to enhance any structure present in the match template and the search area before performing the search. The enhancement may take the form of local histogram stretching or equalisation, or spatial filtering to reduce noise, or enhance intensity gradients around the boundaries of objects. In practice all these operations may be applied jointly. Chapter 5 introduces the concept of sharpness and selectivity with appropriate measures and shows how proper preprocessing can improve discrimination.

4.2.2 Matching in the Transform Domain

A classical method of signal representation is by orthogonal series expansions [4.9]. For a one dimensional discrete signal $\{u(n), 0 \leq n \leq N-1\}$, represented as a vector \mathbf{u} of size N , a unitary transformation is written as:

$$\mathbf{v} = \mathbf{A}\mathbf{u} \Rightarrow v(k) = \sum_{n=0}^{N-1} a(k,n)u(n), \quad 0 \leq k \leq N-1 \quad \text{eq. 4.12}$$

where \mathbf{A} is the transform matrix and \mathbf{v} the transform vector.

Unitary implies that the inverse of the transform matrix is equal to its conjugate transpose, $\mathbf{A}^{-1} = \mathbf{A}^{*T}$. The corresponding inverse transformation is given by:

$$u = \mathbf{A}^{*T}\mathbf{v} \Rightarrow u(n) = \sum_{k=0}^{N-1} v(k)a^*(k,n), \quad 0 \leq n \leq N-1 \quad \text{eq. 4.13}$$

Equation 4.13 can be viewed as a series representation of the signal $u(n)$. The columns of \mathbf{A}^{*T} are called the *basis vectors* of \mathbf{A} . The series coefficients $v(k)$ provide a representation of the original signal that can be used in filtering, feature extraction and other types of analysis.

The general orthogonal series expansion for an $N \times N$ image $u(m,n)$ is a pair of transformations of the form:

$$v(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} u(m,n) a_{k,l}(m,n), \quad 0 \leq k,l \leq N-1 \quad \text{eq. 4.14}$$

$$u(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} v(k,l) a_{k,l}^*(m,n), \quad 0 \leq m,n \leq N-1 \quad \text{eq. 4.15}$$

where $\{a_{k,l}(m,n)\}$, called an *image transform*, is a set of complete orthonormal discrete basis functions.

If the two-dimensional image transform can be written as the product of two one-dimensional transforms, then the transform is termed separable, that is,

$$a_{k,l}(m,n) = a_k(m) b_l(n) \equiv a(k,m) b(l,n) \quad \text{eq. 4.16}$$

where $\{a_k(m), k = 0, \dots, N-1\}$, $\{b_l(n), l = 0, \dots, N-1\}$ are one-dimensional complete orthonormal sets of *basis vectors*.

By choosing $\mathbf{A} \equiv \{a(k,m)\}$ to be the same as $\mathbf{B} \equiv \{b(l,n)\}$ equations 4.14 and 4.15 reduce to:

$$\mathbf{V} = \mathbf{A} \mathbf{U} \mathbf{A}^T \Leftrightarrow v(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} a(k,m) u(m,n) a(l,n) \quad \text{eq. 4.17}$$

$$\mathbf{U} = \mathbf{A}^* \mathbf{V} \mathbf{A}^{*T} \Leftrightarrow u(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a^*(k,m) v(k,l) a^*(l,n) \quad \text{eq. 4.18}$$

In the general sense matrix \mathbf{A} is referred to as the transform kernel. By choosing different kernel generating functions all the standard linear transformations including the discrete Fourier, Sine, Cosine and Hadamard transforms, and the inverses, can be computed using equations 4.17 and 4.18 respectively.

Energy Compaction and Decorrelation

An important property of most unitary transforms is the tendency to pack a large fraction of the average energy of the image into relatively few of the transform coefficients. Energy in this context is measured by the square of the Euclidean distance, represented by the symbol $\| \cdot \|$. Since the unitary transform preserves the total energy,

$$\| \mathbf{U} \|^2 = \| \mathbf{V} \|^2 \quad \text{eq. 4.19}$$

it follows that many of the coefficients will be small and contain very little energy. Even if the energy of original image values is uniformly distributed, the energy of the transform coefficients will be non-uniformly distributed. Also if the image data is highly correlated the transform coefficients will tend to be uncorrelated.

Matching

Matching in the transform domain consists of first transforming the object template, and the test match area in the next frame. These two signals are then compared using one of the difference or similarity measures, and a value computed. As with the spatial domain methods, the best match in the parameter space is found by searching for peaks or troughs in the local area. Many of the transform coefficients of the object template will be small due to energy compaction, thus only a small percentage of the transform coefficients are required in the computation of the match statistic. The actual coefficients of significance can be determined in advance by inspection of the transformed object template. A suitable criterion may be to sort the coefficients into order, based on their absolute size and choose the largest 10%. A large fraction of the template structure will be contained in this small percentage of coefficients due to the energy compaction and decorrelation properties of unitary transforms. Once the coefficients have been chosen, the test match area is transformed and the corresponding 10% of the coefficients are used to compute the match statistic.

The Fourier Transform

One of the most fundamental transforms in signal processing is the Fourier transform. If $f(x, y)$ is a continuous intensity image defined on the spatial coordinates x, y , then ξ_x, ξ_y are the spatial frequencies that represent the changes in intensity with respect to the spatial distances. The units of ξ_x and ξ_y are reciprocals of x and y , respectively. Equations 4.20 and 4.21 define the Fourier transform and the inverse Fourier transform of $f(x, y)$.

$$F(\xi_x, \xi_y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) \exp[-j2\pi(x\xi_x + y\xi_y)] dx dy \quad \text{eq. 4.20}$$

$$f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(\xi_x, \xi_y) \exp[j2\pi(x\xi_x + y\xi_y)] d\xi_x d\xi_y \quad \text{eq. 4.21}$$

From the form of these equations it is clear that the Fourier transform is a complex function. It takes the real valued function $f(x, y)$ and produces the complex spatial frequency description $F(\xi_x, \xi_y)$.

The Fourier transform possesses a number of very useful properties. The most commonly exploited property is that of convolution [4.10]. The results of the convolution theorem can be

readily extended to the *spatial correlation* of two functions. Let \star represent a correlation operator, then the spatial correlation of two functions $f(x, y)$ and $g(x, y)$ is defined as:

$$c(x, y) = h(x, y) \star f(x, y) \quad \text{eq. 4.22}$$

In the Fourier domain the correlation operator is replaced by a sign reversal of the spatial frequency terms in the first function and a simple multiplication.:

$$C(\xi_x, \xi_y) = H(-\xi_x, -\xi_y) F(\xi_x, \xi_y) \quad \text{eq. 4.23}$$

Another property of the Fourier transform that can be exploited in scene matching is the shift property. A shift or translation $(\Delta x, \Delta y)$ in the spatial domain results in the multiplication by a complex exponential in the Fourier domain. Equation 4.24 defines the Fourier shift property.

$$f(x \pm \Delta x, y \pm \Delta y) \Leftrightarrow \exp[\pm j 2 \pi (\Delta x \xi_x + \Delta y \xi_y)] F(\xi_x, \xi_y) \quad \text{eq. 4.24}$$

Real Transforms

Equations 4.20 and 4.21 show that the Fourier transform involves complex arithmetic. If the original signal is also complex then there is no computational overhead. However most practical applications involve real valued signals. Thus the use of the Fourier transform results in increased storage and computational requirements. Furthermore, once a signal has been processed in the Fourier domain and inverse transformed back, care must to be taken to ensure that the returned signal contains only real values.

There exists a number of kernels that can be used in equation 4.17 and 4.18 to unitary transform a real signal, requiring only real arithmetic and producing only real values. They generally possess excellent energy compaction for highly-correlated data and result in highly decorrelated transform coefficients. The most well known of these kernels produce the Cosine, Sine and Hadamard transforms. All of these transforms have a fast discrete implementation due to their Kronecker product separability.

The $N \times N$ cosine transform kernel (matrix) $\mathbf{C} = c(k, n)$, also called the *discrete cosine transform* (DCT), is defined as:

$$c(k, n) = \begin{cases} \frac{1}{\sqrt{N}} & k = 0, 0 \leq n \leq N-1 \\ \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) & 1 \leq k \leq N-1, 0 \leq n \leq N-1 \end{cases} \quad \text{eq. 4.25}$$

The cosine transform is related to the symmetric extension of the *discrete Fourier transform* (DFT).

Similarly the $N \times N$ sine transform kernel (matrix) $\mathbf{S} = s(k, n)$, also called the *discrete sine transform* (DST), defined as:

$$s(k, n) = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(n+1)}{N+1}\right), \quad 0 \leq k, n \leq N-1 \quad \text{eq. 4.26}$$

is related to the antisymmetric extension of the DFT.

The DST is a self inverse due to it being symmetric and orthogonal, resulting in an implementation that is somewhat faster than both the fast Fourier and fast Cosine transforms.

Unlike the Fourier, cosine and sine transforms, the Hadamard (or Walsh-Hadamard) [4.11] transform matrix contains only the binary values ± 1 . This makes it well suited for digital signal processing resulting in an exceptionally fast transform requiring only additions and subtraction. The Hadamard transform matrices, \mathbf{H}_n , are $N \times N$ matrices, where $N \equiv 2^n, n = 1, 2, 3, \dots$. These are easily generated from a core matrix \mathbf{H}_1 and Kronecker product recursion. Equation 4.27 defines the core matrix, while equation 4.28 defines the recursion relationship (where the symbol \otimes represents the Kronecker product operator) for generating larger matrices.

$$\mathbf{H}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{eq. 4.27}$$

$$\mathbf{H}_n = \mathbf{H}_{n-1} \otimes \mathbf{H}_1 = \mathbf{H}_1 \otimes \mathbf{H}_{n-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_{n-1} & \mathbf{H}_{n-1} \\ \mathbf{H}_{n-1} & -\mathbf{H}_{n-1} \end{bmatrix} \quad \text{eq. 4.28}$$

When dealing with the Hadamard transform the concept of transform frequency is replaced with the term *sequency* since the function only takes on the values ± 1 . For sinusoidal signals frequency can be defined in terms of the number of zero crossings, *sequency* is simply the number of sign changes.

4.2.3 Search Strategies

Whether working in the spatial domain or transform domain, motion tracking using template matching requires the search of a parameter space to find the best fit. If the maximum interframe motion with respect to the parameter space can be estimated reliably, then the extent of the local search can be constrained. Even for relatively small areas, an exhaustive search can become computationally prohibitive.

Direct search methods tend to be applied in the spatial domain although in many cases the approach is perfectly general to other forms of optimisation. A number of efficient direct search techniques exist.

Two-dimensional logarithmic search

Assuming that the motion can be approximated by pure translation, consider a local neighbourhood size $\pm p$ about the estimated position in the image space, for which a match with an object template of size $M \times N$ is sought. An exhaustive search would require $(2p+1)^2$ iterations, the logarithmic search reduces this to $\log(2p+1)$ iterations. A mean distortion function can be defined as:

$$D(i, j) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N f(I(m, n) - I_k(m+i, n+j)), \quad -p \leq i, j \leq p \quad \text{eq. 4.29}$$

where a suitable choice of the function $f(x)$ may be x^2 or $|x|$, corresponding to the sum-of-squares difference and absolute-sum-difference of equations 4.5 and 4.6. Define the *direction of minimum distortion* (DMD) as the direction of the vector (i, j) that minimises $D(i, j)$. The template match occurs when the DMD has been found within the search area.

If $D(i, j)$ increases monotonically moving away from the DMD in any direction, then the search can be speeded up by successively reducing the area of search. Figure 4.5 illustrates the logarithmic search procedure for $p=4$, the \square symbol marks the search positions. The algorithm begins by searching five locations between the midpoints of the centre and the four boundaries of the search area. These locations have been marked with 1. The optimum direction (the circled numbers) give the location of the centre for the next step. The procedure continues until the search plane is of size 3×3 in which all nine locations are searched to find the DMD. The final optimum direction is indicated by * in figure 4.5. In this example the number of searched locations has been reduced to 13 out of the possible 81 positions.

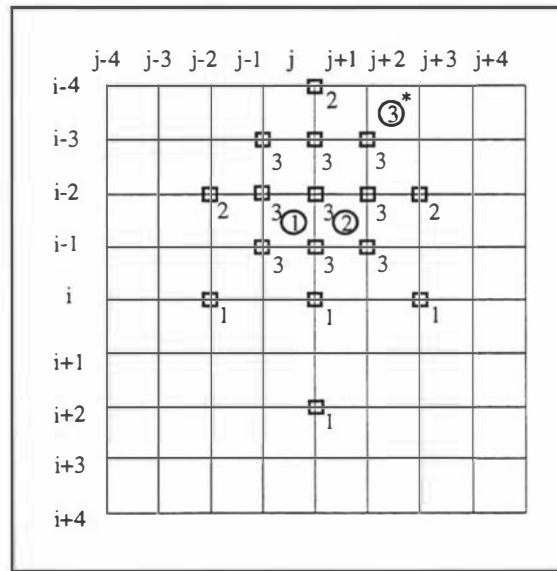


Figure 4.3: 2-d logarithmic search in the direction of minimum distortion

The logarithmic search is simply a variation on a discrete 2-d implementation of the well known *steepest decent* algorithm.

Sequential search

There are several alternative ways in which a sequential search can be speeded up. The first involves computing a cumulative error and terminating the direction of search when some predefined threshold is exceeded. The search may only be continued in those directions that are below the threshold.

Another common algorithm that is used as an alternative to steepest decent is the *conjugate gradient* algorithm [4.12]. With respect to searching a 2-d space, the search algorithm proceeds in the i direction until a minimum is found and then switches to the j direction until the next minimum is found. The search continues in alternating conjugate directions until the location of the minimum remains unchanged.

Hierarchical search

A hierarchical search is effective when the search parameter space is large. The method involves searching a low-resolution reduced copy using a likewise reduced copy of the template. If multiple matches occur, then these locations are searched at higher resolution to further refine and reduce the search area. The algorithm continues until the best match is found at the original resolution.

Image processing structures that contain multiple copies of an image at successively lower resolution are called pyramids. The rule by which the resolution is reduced is important as it governs how information is preserved proceeding up the pyramid. A complete pyramidal representation of signal requires approximately 33% more storage than the original signal. This increased storage is offset by the greatly reduced computational cost in searching the pyramid from the top level down. For a simple half-scale rule (half size at each level), the *coarse-fine* search is logarithmically efficient.

The resolution reduction rule for the pyramid may also be different at various levels. It can be adaptively chosen to preserve specific characteristics, at different levels of the pyramid.

4.2.4 Motion Prediction

If a reliable estimate of the most likely position and the maximum deviation from this position in the parameter space can be computed, then the size of the match region to be searched can be greatly reduced. Predicting the motion of the object to be tracked usually involves some *a priori* knowledge about the nature of this motion. The simplest assumption that can be made is the object moves smoothly and hence the interframe motion for the next frame will be similar to the

current frame. This single interframe step approach amounts to the linear interpolation of the motion parameters. If the smoothness assumption holds, then the only difficulty is in finding the first interframe match to form the prediction. This first step will usually involve an extensive search with practical estimates on the maximum possible extent of the search region. Once the method is started, the only difficulty that can occur is that the smoothness assumption may become invalid at some point, resulting in the predicted region not including the true match region. One way to deal with this problem is to compare the value of the best-match in the current frame with the best-match in the previous frame. If the difference exceeds some threshold then the tracking process resets itself and restarts from this frame as if it was the first frame.

The effectiveness of motion prediction can be enhanced by computing the prediction over more than one frame. A quadratic predictor can be computed by initially starting with the linear predictor until two interframes have been tracked. At this stage the motion from these two interframes is fitted with a quadratic function which in turn is used to predict the next interframe motion. This fitting and prediction process then continues as the sequence is tracked. An added advantage of using multiple frames to form the prediction is that the threshold used to detect whether or not the prediction has failed (*ie.* the true match not in the prediction region) can in itself be made adaptive. The standard deviation of the best-match over several frames can be computed and used as the \pm deviation for the allowable difference of match.

Higher order interpolators such as splines can also be used in the prediction process, but computational cost begin to increase sharply. Also care has to be taken to ensure the stability of the interpolant. Cubic splines tend to '*ring*' and overshoot if the spacing between points becomes too non-uniform. This situation will occur if the sequence motion is rough, and can result in large prediction errors.

4.3 Optical Flow Methods

Motion tracking using optical flow methods involves computing the interframe optical flow or instantaneous velocity field of a scene and then locating the objects being tracked and computing their motion from the structure of the velocity field. One of the important features of optical flow is that it can be calculated using only local information. Five basic methods for the calculation of optical flow have been studied in the literature.

1. Point-based differential methods
2. Region-based local match methods
3. Contour-based approaches
4. Energy-based approaches
5. Phase-based approaches

All of these techniques extract local estimates of the image translation, and in some case the gradient of the velocity field. These methods will be reviewed in brief, in the following sections.

4.3.1 Differential Approaches

Differential techniques were the first methods to be studied in the computation of optical flow [4.13, 4.14]. They modelled the image motion by a continuous variation of image intensity as a function of position and time. The intensity function $I(x, y, t)$ is then expanded as a Taylor series.

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\partial I}{\partial x} dx + \frac{\partial I}{\partial y} dy + \frac{\partial I}{\partial t} dt + \text{higher terms} \quad \text{eq. 4.30}$$

Terms higher than the first term are discarded, making the partial differential description linear.

If the image at some time $t + dt$ is the same image at time t , translated by dx and dy then:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) \quad \text{eq. 4.31}$$

Combining equations 4.30 and 4.31 leads to the result:

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \quad \text{eq. 4.32}$$

known as the *gradient constraint equation*.

In this equation $\frac{\partial I}{\partial t}$, $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ are all measurable quantities, and $\frac{dx}{dt}$ and $\frac{dy}{dt}$ estimate what we are looking for, the local velocity in the x and y direction.

$$\text{Let,} \quad \frac{dx}{dt} = u \quad \frac{dy}{dt} = v \quad \text{eq. 4.33}$$

$$\text{thus equation 4.32 gives,} \quad -\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \quad \text{eq. 4.34}$$

$$\text{or equivalently,} \quad -\frac{\partial I}{\partial t} = \nabla I \cdot \mathbf{v} \quad \text{eq. 4.35}$$

where ∇I is the spatial gradient of the image, ' \cdot ' is the dot product operator, and $\mathbf{v} = (u, v)$ the velocity.

Explicit measurement using equation 4.35 yields only the normal velocity with respect to the direction of the spatial gradient operator. This is because only one linear equation constrains both components of the velocity.

Many additional constraints have been proposed to over determine the system and hence enable a least-squares or similar solution. The most popular constraint involves global *smoothness* of \mathbf{v} . This smoothness constraint can be defined by the following energy function over a local region of interest R , that is minimised with respect to the velocity field.

$$\min_{\mathbf{v}}(E) = \min \int_R \left((I_x u + I_y v + I_t)^2 + \lambda (u_x^2 + u_y^2 + v_x^2 + v_y^2) \right) dx dy \quad \text{eq. 4.36}$$

where u_x, u_y, v_x, v_y are partial derivatives of the velocity field with respect to the x and y , and λ reflects the relative importance of smoothness in the solution.

Implicit in the differential or gradient techniques is the calculation of the spatial and temporal derivatives. When only two frames are used in this computation the method is limited to only one pixel per frame displacement, and a linear intensity change in a region as large as the expected displacement. Practically this has led to the use of multiple frames in the calculation of the optical flow using this approach. This method also assumes that the derivatives of the velocity components do exist. However for the discrete image case both spatial and temporal aliasing may occur, corrupting the signal and leading to severe instability in the numerical differentiation.

4.3.2 Matching Approaches

Most of the essential elements in matching approaches to the computation of optical flow have already been covered in section 4.21 on spatial domain matching approaches to motion tracking. Region-based optical flow approaches are aimed at overcoming the difficulties of the point-based differential methods when accurate differentiation is not possible due to noise or where only a small number of frames exist. The scene is broken up into small regular areas and a match is sought for each area in the following frame. A similarity or difference measure is then used to find the best match. The velocity vector for each area is then computed from the differences in coordinates between the original location and the best fit location in the next frame.

In practice two issues have proven to be important in applying matching methods to the computation of optical flow. First, the aperture problem (see section 4.1.1) arises when the local intensity structure in the match template is one-dimensional. In this case the match statistic gives rise to surfaces that are *ridge-like*, without clearly defined maxima (or minima) as a function of shift. A conditioning measure such as the curvature of the match statistic may prove useful in dealing with this problem, but cannot eliminate it. The second difficult issue involves textured and periodic structures that may exist in a scene. For large interframe displacements and in the absence of good motion prediction, this can lead to multiple, equally-likely ridge matches in the search region. To handle this problem additional constraints or some form of control structure

can be incorporated. Band-pass filtering may prove useful in the removal of low frequency structure and can be applied in addition to a coarse-to-fine match strategy [4.15].

4.3.3 Contour-based Approaches

Contour-based or *feature-based* approaches appear in two basic forms. In the first form the image contours are explicitly used as features that are tracked from frame-to-frame. The second form uses the contours as locations in the image where some form of differential method can be applied.

Most *feature-based* approaches use edges. The motivation behind this choice is that it has been conjectured that edges are stable over time, correspond to salient image properties and are relatively high contrast features for images with a good signal-to-noise ratio. However practically, the contours extracted with the current edge detection processes are rarely confined to a single smooth surface.

The most commonly used edge detection methods such as Canny and Marr-Hildreth (see chapter 3) provide a reasonably sparse collection of features from which image velocity can be measured. It is assumed that the edges move small distances relative to their density in the image, enabling relatively straight forward correspondence. The local velocity is then computed from the perpendicular distance between one contour and the corresponding contour in the next frame. However the apparently straight forward contour correspondence problem has proved to be the major difficulty in applying this method to motion analysis. Many methods have been devised to tackle this problem including the iteration of initial estimates [4.16] and dynamic optimisation by the gradient descent procedures [4.17].

In the second form of contour-based approach, edges are usually defined in terms of the output of a band-pass edge-enhancement filter. The velocity measurement is then based on a differential method restricted to edge locations and their local neighbourhoods.

Several specific problems exist that have been argued to be a significant obstacle to feature-based approaches to optical flow. Correspondence is the most intractable, however the assumption that features (edges in particular) are well localised and stable over time has also proved to be weak in practical situations. Implicit to these method is that contours provide a rich description of the image so that no velocities will go undetected and that sparse isolated edges will facilitate easy correspondence. This has not proved to be the case [4.18].

4.3.4 Energy-based Approaches

Energy-based approaches use frequency analysis and an initial velocity-specific representation of the image sequence. A range of tuned filters are used to sample the local power spectrum of the

signal. The relative amplitudes of the output of these filters is then used to compute the image velocity.

Let the vector $\mathbf{x} \equiv (x, y)$ denote the x and y spatial coordinates and $\mathbf{v} \equiv (v_x, v_y)$ denote the x and y spatial velocities. Consider a 2-d intensity function $i_o(\mathbf{x})$, translating at velocity \mathbf{v} .

$$i(x, t) = i_o(\mathbf{x} - \mathbf{v}t) \quad \text{eq. 4.37}$$

Taking the Fourier transform of this equation and using the Fourier shift property (eq. 4.24), it follows that:

$$\begin{aligned} I(\mathbf{k}, \omega) &= \iint i_o(\mathbf{x} - \mathbf{v}t) e^{-j(\mathbf{x}^T \mathbf{k} + t\omega)} dx dt \\ &= I_o(\mathbf{k}) \int e^{-jt\mathbf{v}^T \mathbf{k}} e^{-jt\omega} dt \\ &= I_o(\mathbf{k}) \delta(\omega + \mathbf{v}^T \mathbf{k}) \end{aligned} \quad \text{eq. 4.38}$$

where $I_o(\mathbf{k})$ is the Fourier transform of $i_o(\mathbf{x})$ and δ is the Dirac or impulse function.

The result of equation 4.38 indicates that when the translating intensity function is expressed in the Fourier domain, all of its non-zero power lies on a plane containing the origin. This is because $\delta(\omega + \mathbf{v}^T \mathbf{k})$ is non-zero only when $\omega = -\mathbf{v}^T \mathbf{k}$. The magnitude of the velocity (speed), $\|\mathbf{v}\|$, determines the angle between the planes $\omega = -\mathbf{v}^T \mathbf{k}$ and $\omega = 0$. While the direction of \mathbf{v} determines the orientation of the velocity plane about the ω -axis.

The above analysis implies that if the temporal variation in an image is due to translation, then all the power will be concentrated on the appropriate plane in the frequency space. This result can be extended by considering the 3-d power spectrum \mathfrak{R}^3 , of a region R , in a spatiotemporal image, as a distribution [4.19]. The principle directions of this distribution are given by the eigenvectors of its covariance matrix and the eigenvalues specify the variance of the power spectrum in each of the principle directions. The main results of this analysis in terms of velocity measurement are:

- A translating 1-d profile has all its power concentrated about a line through the origin resulting in only one of the three eigenvalues being significantly non-zero. From the corresponding eigenvector the velocity of the line can be estimated
- A translated 2-d textured image patch has its power concentrated about a plane in frequency space resulting in two significantly non-zero eigenvalues. As the eigenvectors are orthogonal, they will span the plane enabling the estimation of the 2-d velocity.
- For constant intensity patches all three eigenvalues will be zero, while for more complex spatiotemporal variations all three eigenvalues will be non-zero.

The above results would seem to indicate that the computation of local image velocity by this method is straight forward. However in order to get an adequate estimate of the velocity several filters, tuned to different spatial and temporal frequencies, are needed. The output of these filters are then combined in some way to produce the final velocity estimate. Because the final estimate is based on the results of multiple filters, velocity resolution is significantly sacrificed. This poor resolution can lead to confusion, two different component velocities in the same neighbourhood may have the same distribution of output filter amplitudes as a single component velocity. In addition, there is little information with respect to robustness of these methods for common deviations from pure translation, such as incremental rotation and scale change.

4.3.5 Phase-based Approaches

Phase-based approaches to optical flow use a collection of velocity-tuned filters as in the energy-based approaches, however the phase behaviour of the filter outputs rather than the amplitude is used to compute the component velocity. The motivation behind the use of phase information is that spatial contours of constant phase provide a better approximation to the temporal evolution of a motion field than do contours of constant amplitude [4.20].

The importance of phase information can be readily demonstrated by the following. Fourier transform an image, convert the complex result to polar form (magnitude and phase); set all the magnitudes to unity, convert back to component form and inverse transform. The resulting image will be very low in contrast but will clearly be recognisable as the original image. This is because the phase information contains much of the images spatial structure. This result can be readily extended to 3-d signals such as image sequences.

In principle if the change in image intensity in a motion sequence was solely due to translation, then all four methods covered in sections 4.3.1 to 4.3.4 would produce accurate estimates of the local velocity, so long as the aperture problem and aliasing of the signal were not severe. However, the typical time-varying behaviour of intensity image sequences is only crudely approximated by local translation. Tracking phase contours is less sensitive to contrast changes and local variations in scale, speed, and orientation, compared to tracking amplitude contours. This is because the calculation of the component velocity from phase information requires the computation of the phase gradient and it can be shown that the phase gradient uniquely defines the instantaneous spatial and temporal frequencies of a signal [4.21]. Unlike the energy-based approaches, the collection of filters used to compute the final velocity estimate are only tuned slightly differently from each other. This ensures that velocity resolution is not sacrificed.

The accurate calculation of the phase gradient is not a trivial matter as the effects of phase wrapping/unwrapping and discontinuities have to be taken into account. Many of these difficulties can be avoided by computing the phase gradient directly from the response and

gradient of the response of the filters. This computation explicitly involves numerical integration and differentiation of the sampled signals. To avoid many of the difficulties associated with numerical integration and differentiation, the complex signals have to be low-pass filtered first. They may then be convolved with appropriate interpolation and differentiation kernels to yield fairly robust results.

Overall the computational cost of phase-based techniques for the measurement of optical flow is very high, but results can be very good.

4.3.6 Adapting Optical Flow for Radiographic Images

In modelling the spatiotemporal intensity variation in an image sequence, optical flow techniques exploit a relatively simple scene model incorporating smooth surfaces, with both diffuse and specular components of reflection. This reflectance model does not include the effects of shadowing, but in many cases proves to be quite adequate. Radiographic images do not conform to the reflectance model. Chapter 2 clearly showed that radiographic information is the result of the integrated absorption of the X-ray beam as it passes through material. The relative absorption is dependent on the thickness of the material and its mass density. Thus the intensity of an object is dependent on its density and not its surface reflectance. This implies that measured velocities no longer represent a single point on the object. Making the assumption that density does not change with time, intensity changes will represent depth changes.

In angiograms [4.22] a radio-opaque dye is injected into the bloodstream. The primary absorbers of the transmission in this situation are the dye and calcified bone. The assumption that the intensity changes represent depth changes in the heart is true in this case since the dye-filled heart is the primary source of motion. However for cineradiographic sequences of the human spine there is little bone deformation only small density changes due to stretching and compression of soft tissue.

Continuity Equation

The continuity equation arises from the study of fluid flow. It relates the spatiotemporal behaviour of a conserved fluid quantity to the fluid velocity.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad \text{eq. 4.39}$$

where $\rho = \rho(\mathbf{x}, t)$ is the fluid density, $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ the fluid velocity, and \mathbf{x} is the spatial coordinates (x, y) in two dimensions and (x, y, z) in three dimensions.

This fluid flow model, based on the continuity equation, is particularly appropriate in imaging situations where the intensity is proportional to the density of some conserved fluid quantity,

such as mass, charge or the projection of such a density. This is clearly the case for radiographic imaging [4.23].

In section 4.3.1, differential approaches to the computation of optical flow were primarily based on the *gradient constraint equation*, equation 4.35. When this is applied to radiographic images, the intensity term represents density, resulting in the equivalent equation:

$$\frac{\partial \rho}{\partial t} + \nabla \rho \cdot \mathbf{v} = 0 \quad \text{eq. 4.40}$$

where $\mathbf{v} = v(\mathbf{x}, t)$ is now the velocity of 'constant-density'.

Expanding the gradient and dot-product operations in the continuity equation (4.39) enables a comparison between this equation and the gradient constraint equation.

$$\frac{\partial \rho}{\partial t} + \nabla \rho \cdot \mathbf{u} + \rho \nabla \cdot \mathbf{u} = 0 \quad \text{eq. 4.41}$$

Comparing equations 4.39 and 4.41 it is readily observed that if $\rho \nabla \cdot \mathbf{u} = 0$, then \mathbf{u} and \mathbf{v} obey the same equation. This situation corresponds to the case where the fluid is both conserved and incompressible. Combining equations 4.40 and 4.41 produces the following result:

$$\mathbf{v}_g = \mathbf{u}_g + \frac{\nabla \cdot \mathbf{u}}{|\nabla(\log \rho)|} \quad \text{eq. 4.42}$$

where the subscript 'g' denotes the component measured in the direction of the density gradient $\nabla \rho$.

This equation shows that the additional term resulting from the fluid flow model involves two components. The first component $\nabla \cdot \mathbf{u}$ is the local divergence of the fluid flow (the gradient in the direction normal to the fluid velocity) and the second component is the gradient of the log of the density. The overall effect of this term is that the optical flow velocity in the direction of the gradient will be more positive than the fluid flow component at locations where the fluid flow is diverging, and more negative where the flow is converging.

Although the continuity equation appears to be more appropriate than the standard optical flow equation for radiographic images since it is based on actual physical flow, it is insufficient in itself to uniquely determine the flow. Additional constraints are required, similar to those applied to the optical flow equation in section 4.3.1 to ensure a solution. In practice it is the application of these constraints and not the base model that is the most problematic.

4.4 A Suitable Tracking Method for Cineradiographic Images

All of the methods covered in this chapter are constrained by the effects of both the aperture problem and the correspondence problem. The aperture problem (covered in section 4.1.1) restricts the estimation of the local velocity to its normal component. The correspondence problem is solved explicitly by object-based matching approaches when they find the best match to the object template in the next frame. Optical flow methods deal with this problem when they calculate the interframe temporal gradients on which local velocity estimates are made. Also when optical flow methods are used to track complete objects, local velocity estimates have to be associated (correspondence found) with the objects. These corresponding velocity vectors are then used in the estimate of the object motion.

Another significant problem encountered by all the motion-tracking methods is occlusion. In the general case one object may pass behind another object and thus become partially or totally obscured only to reappear later in the sequence. The effect of even partial occlusion in reflectance images will dramatically alter the recorded intensity information. Dealing with occlusion in radiographic images is in principle a far less difficult task as total occlusion can never occur since the image is formed by a density projection.

In cineradiographic images of the human spine the motion of interest is the movement of the vertebrae. A description of the entire velocity field is not required. Occlusion is not generally a problem in the important cervical and lumbar regions. Thus template-based matching approaches would appear to be the most appropriate method for cineradiographic images of the human spine.

Some means of segmenting the image and identifying the objects of interest (the vertebrae) is necessary. Chapter 3 covered in detail the application of edge-based segmentation methods to cineradiographic images of the human spine. The low contrast of these images combined with their noisy characteristics made this task very difficult. In a practical situation manual segmentation of a single frame in a sequence and labelling the objects of interest would be acceptable to a health care practitioner so long as the remainder of the tracking process is automated.

A practical system for the measurement of spinal motion must perform the necessary calculations in an acceptable time. Acceptable time in this situation implies that it should take no longer than 30 minutes and preferably less. Given that a cineradiographic sequence may contain a hundred or more frames (even at half-video rates), then the overall computation required to track the vertebrae will be extremely high. Of the various methods reviewed in this chapter, template matching approaches are the least computationally expensive, especially when combined with good motion prediction and an effective match statistic.

In the following chapter the performance of the various match statistics proposed in sections 4.2 and 4.3, are evaluated on real data from cineradiographic images of the human spine.

4.5 References

- [4.1] **Horn B.K.P.** *Robot Vision*, MIT Press, Cambridge, 1986
- [4.2] **Marr D. and Ullman S.** *Directional selectivity and its use in early visual processing*, Proceedings of the Royal Society of London, B 211, pp 151..180, 1981.
- [4.3] **Aggarwal J. K., Duda R. O. and Rosenfeld A.** (eds.). *Computer Methods in Image Analysis*. Chapter 4: Hu K., Visual Pattern Recognition by Moment Invariants. Los Angeles: IEEE Computer Society, 1977.
- [4.4] **Pavlidis T.** *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.
- [4.5] **Gurevich G. B.** *Foundations of Theory of Algebraic Invariants*. Groningen, The Netherlands: P Noordhoff, 1964.
- [4.6] **Dudani S., Breeding K. and McGhee R.** *Aircraft Identification by Moment Invariants*. IEEE Transactions on Computers C-26, No. 1, pp 39..45, Jan. 1977.
- [4.7] **Wong R. and Hall E.** *Scene Matching With Moment Invariants*, Computer Graphics and Image Processing 8, pp 16..24, 1978.
- [4.8] **Besl P.J. and Jain R.C.** *Segmentation Through Variable-Order Surface Fitting*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 2, pp 167..92, March 1988.
- [4.9] **Ahmed, N. and Rao, K.R.,** *Orthogonal Transforms for Digital Signal Processing*. Springer Verlag, New York, 1975.
- [4.10] **Bracewell R.N.** *The Fourier Transform and its application*. McGraw-Hill, pp 24..48, 1986.
- [4.11] **Walsh, J.L.** *A closed Set of Orthogonal Functions.*, Proceedings of the London Mathematical Society., vol. 34, pp 241..279, 1932.
- [4.12] **Abdou I.F. and Wong K.Y.** *Applied Numerical Analysis*. 2nd Edition, Addison-Wesley, 1978.
- [4.13] **Fennema C. and Thomson W.** *Velocity determination in scenes containing several moving objects*. Computer Graphics and Image processing 9, pp 301..315, 1979.
- [4.14] **Horn B.K.P. and Schunck B.G.** *Determining optical flow*. Artificial Intelligence 17, pp 185..204, 1981.

-
- [4.15] **Ballard D.H. and Brown C.M.** *Computer Vision*. Prentice-Hall, 1982.
 - [4.16] **Glazer F., Reynolds G. and Anandan P.** *Scene matching through hierarchical correlation*. Proceedings IEEE CVPR, Washington, pp 432..442, 1983.
 - [4.17] **Wu J. Brockett R. and Wohn K.** *A contour-based recovery of image flow: Iteration method*. Proceedings IEEE CVPR, San Diego, pp 124..129, 1989.
 - [4.18] **Kass M., Witkin A. and Terzopoulos D.** *Snakes: Active contour models*. International Journal of Computer Vision 1, pp 321..331, 1988.
 - [4.19] **Waxman A.M., Wu J. and Bergholm F.** *Convected activation profiles: Receptive fields for real-time measurement of short range visual motion*. Proceedings of CVPR, Ann Arbor, pp 717..723, 1988.
 - [4.20] **Fleet D.J. and Jepsen A.D.** *Computation of image velocity from local phase information*. International Journal of Computer Vision 5, pp 77..104, 1990.
 - [4.21] **Adelson E.H. and Bergen J.R.** *Spatiotemporal energy models for the perception of motion*. Journal of the Optical Society of America A2, pp 284..299, 1985.
 - [4.22] **Stevens A.S.** *Medical Imaging*, Gold-Publishing, 1991.
 - [4.23] **Fitzpatrick J.M.** *A method for calculating velocity in time dependent images based on the continuity equation*. IEEE Motion: Representation and Perception, pp 78..81, 1985.

Match Statistic Performance

5.1 Template Matching

It was concluded in chapter 4 that a template-based matching approach would be the most suitable method for tracking motion of the human spine in cineradiographic images. This was principally based on the fact that a general description of the motion field was not required since only the vertebra were of interest. The outlines of the vertebral boundaries in a single frame would form object templates for which a best match position would be sought in subsequent frames. Such an approach has the added advantage of directly solving the correspondence problem (see section 4.3.3). The potential success of this approach is further enhanced by the fact that the vertebral motion is reasonably well constrained to a single plane, resulting in little geometric distortion of the projected vertebral image. Also, due to constraints in the degrees of freedom associated with the motion of the spine, only partial occlusion of vertebrae can occur and its effect is significantly reduced since the image is a density projection.

Template based tracking approaches are of low computational cost only when the size of the template is small and good estimates of both translation and rotation are available. While dealing with translation to within one pixel is straight forward, matching with template rotation is far from simple and can consume most of the computation in the motion tracking scheme. The issue of rotation is dealt with in detail in the following chapter where a method is developed to create cineradiographic sequences with prescribed motion. Matching with rotation requires that for each increment of rotation, the discrete template is interpolated to form a continuous description. This continuous description is then rotated and sampled back onto the original pixel grid, ready for matching. Ensuring that the rotation operation does not introduce any spatial aliasing artefacts or significant smoothing requires careful design and significant computational effort. If the object represented by the template changes very little, both geometrically and photometrically, then the rotated templates need only be computed once at the start of the tracking procedure for each increment of rotation. However in cineradiographic sequences, there are significant photometric changes principally due to the camera auto-gain control changes necessary to prevent system overload. Dealing with this and geometric effects would require that the template be incrementally updated after each interframe match and new rotated versions of template computed. This would introduce significant computational cost.

The effect of rotation on the template match can be made insignificant if the template is made small relative to the feature it represents. For cineradiographic images digitised to 288x288 pixels (half PAL composite video frame size), the bodies of the vertebrae are typically 30x18 pixels for the cervical region and between two and three times that size in the lumbar region. Maximum speed of rotation and translation occurs in a lateral cervical view where the neck moves from full flexion to full extension. In this situation the vertebrae undergo rotation from approximately -45° to $+45^{\circ}$. Assuming that the capture rate is sufficiently fast to restrict the inter-frame vertebral rotation to only a few degrees, there will still be at least half a pixel movement due to rotation at the edges of the vertebra.

The disadvantage of making the template small enough so that rotation can be ignored is that it may not contain enough information to give a reliable match. It was shown in section 4.1.1 of chapter 4 that the aperture problem restricts the accurate estimation of local velocity to its normal component when the intensity structure upon which the measurement is based is nearly one-dimensional. Table 5.1 illustrates some simple (noise-free) grey-scale intensity feature images and their relative dimensionality when viewed from a window centred at the middle of each image.

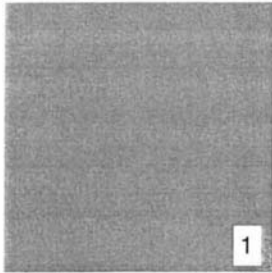
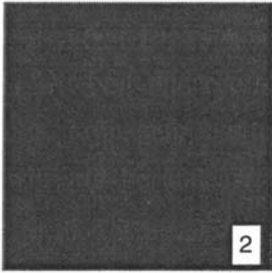
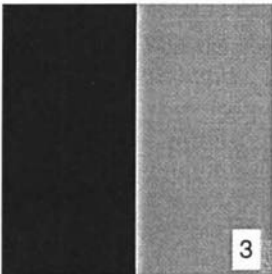
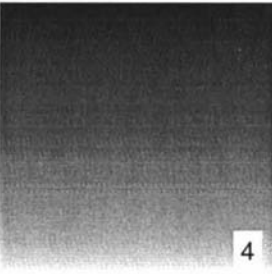
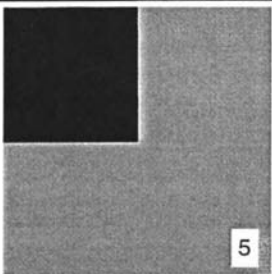
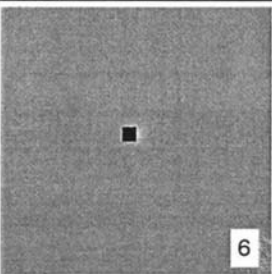
Relative dimensionality	Simple grey-scale feature images	
close to zero	 1	 2
close to 1	 3	 4
between 1 and 2	 5	 6

Table 5.1: Simple grey-scale features and their relative dimensionality

Matching methods will suffer aperture problem effects when the feature forming the template is of low dimensionality. This will tend to occur when a template is made so small that it does not contain any corner and/or boundary information. Without this two-dimensional information, the match will be poor in the direction of the one-dimensional feature, and thus be prone to mismatching along the direction of the feature in the presence of noise. A method proposed to solve this problem is covered in detail in chapter 8, where the complete motion-tracking algorithm is assembled and tested. The solution involves defining a global match for a given vertebra by combining a number of small sub-templates positioned about the vertebra. Although any one of the sub-templates may contain a relatively one-dimensional feature, so long as the angle of orientation for each sub-template feature is significantly different, then the global match will not be prone to mismatch due to aperture effects.

5.1.1 Match Statistics

From the range of match statistics covered in chapter 4, four were chosen for evaluation on cineradiographic images. These statistics are all computed in the spatial domain where they offer low computational cost when small match templates are used [5.1]. The first two match statistics correspond to normalised versions of the measures described by equations 4.5 and 4.6.

Match Statistic	Computational Cost
$MAD = \frac{1}{mn} \sum_{i=0}^m \sum_{j=0}^n A_k(i, j) - A_{k+1}(i, j) \quad eq. 5.1$	additions: $2.5 mn$ multiplications: 11
$RMS = \frac{1}{mn} \sqrt{\left(\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) - A_{k+1}(i, j)]^2 \right)} \quad eq. 5.2$	additions: $2 mn$ multiplications: $mn + 25$
$Corr = \frac{\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) * A_{k+1}(i, j)]}{\sqrt{\left(\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j)]^2 * \sum_{i=0}^m \sum_{j=0}^n [A_{k+1}(i, j)]^2 \right)}} \quad eq. 5.3$	additions: $3 mn$ multiplications: $3 mn + 26$
$MCor = \frac{\sum_{i=1}^6 [M_k(i) * M_{k+1}(i)]}{\sqrt{\left(\sum_{i=1}^6 [M_k(i)]^2 * \sum_{i=1}^6 [M_{k+1}(i)]^2 \right)}} \quad eq. 5.4$	additions: $33 mn + 48$ multiplications: $48 mn + 57$
NOTE: See appendix B for full details of the computational cost calculations.	

Table 5.2: Match statistics and their computational cost

The Mean of Absolute Differences (MAD) statistic is a normalised version of the absolute sum of differences (ASD) of equation 4.6. The Root Mean of Squared differences (RMS) is a normalised version of the sum-of-squares difference (SSD) of equation 4.5. These first two statistics are based on intensity differences, for which a best match between the reference template A_k and target area A_{k+1} produces a minimum. The third match statistic corresponds directly to the (discrete) normalised product Correlation (CORR) of equation 4.7, while the fourth match statistic is a normalised product correlation based on Moment invariants, (MCor). Table 5.1 lists these four measures along with an estimate of their computational cost.

The two correlation based statistics produce a maximum at the best match position. The moment based correlation uses the first six normalised central moment invariants listed in table 4.1 of chapter 4. The moments for the template, $M_k(i)$, are matched to the target area moments, $M_{k+1}(i)$, using a normalised product correlation. In principal this statistic should be invariant to rotation, scale and translation changes. Only the first six moment invariants have been used in this calculation as the computational cost of high order moment invariants increases rapidly.

5.1.2 A Match Statistic Sharpness and Selectivity Measure

An important performance measure in template matching is the uniqueness of a best match with respect to the local neighbourhood. One measure of uniqueness is that rate at which the match statistic decreases (or increases) as the template is moved away from the best match position. If the statistic changes monotonically in all directions about the best match, then the feature represented by the template is unique in a local sense. However, if the feature represented by the template is relatively one-dimensional (usually the case for small templates), then moving the match window normal to the orientation of the feature, the match statistic will decay rapidly, whereas moving parallel to the feature the decay will be significantly less rapid.

Williams and Glazer [5.2] defined two sharpness measures for assessing the uniqueness of a template match. Their sharpness measures are based on calculating the sum of the differences between the value of the statistic at the best match position and values about this position moving out in a square annulus. This sum is then normalised by the difference between the best match value and the maximum value in a neighbourhood of radius r . The minimum sharpness rule, defined in equation 5.5, is sensitive to one-dimensional features. It will give a low value for a one-dimensional feature since it is based on the minimum difference in each annulus.

$$S_{Min} = \frac{\sum_{i=1}^r (M_i - M_o)}{\sum_{i=1}^r (M_{Max} - M_o)} \quad eq. 5.5$$

$$S_{Ave} = \frac{\sum_{i=1}^r (M_{Ave(i)} - M_o)}{\sum_{i=1}^r (M_{Max} - M_o)} \quad eq. 5.6$$

where: M_o is the match central minimum, M_i is the minimum match in the square annulus of radius i , $M_{Ave(i)}$ is the corresponding annulus average and M_{Max} is the maximum in the entire neighbourhood of radius r .

The average rule on the other hand (equation 5.6) will be less affected by one-dimensional features since it involves a weighted sum of values in each annulus.

Williams and Glazer compared their sharpness measures to the interest operators of Moravec [5.3] and Hannah [5.4] on a number of test images using several simple match statistics. Their match statistics were all computed as Euclidean distances for both point, area and gradient (Kirsch) differences. They acknowledged that their match statistics were not normalised and thus sensitive to changes in the average or *DC* intensity, but then proceeded to ignore this effect for the remainder of their analysis. The results of the last part of their work are of importance to general template matching in that points of high interest as defined by the Hannah operator were classified manually into nine categories according to their feature characteristics. The categories included whether or not the position was near a border, with three sub-categories for the border type (straight, curved, or sharp corner). If the position was not near a border but within an object, then three sub-categories were used to describe the local area as homogeneous, somewhat textured or strongly textured. Although all these categories were subjectively assessed, the results provided considerable insight into the behaviour of the various match performance measures. Their minimum and average sharpness measures were used graphically to show how each responded to a different category feature. Based on these results, a match selectivity measure is proposed, computed from the ratio of the difference between the minimum and average sharpness values, and the average sharpness. The difference component is a measure of dimensionality. It will be large if the feature is of low dimensionality, such as in the case of a homogeneous surface or a one-dimensional boundary. The average component provides a measure of the local mean match sharpness. For a highly selective feature, the difference component should be small and the average component large, thus producing a small ratio. One minus this ratio gives a selectivity measure (equation 5.7) that will be close to unity for a highly selective feature match and close to zero for a poor feature match.

$$Selectivity = 1 - \left(\frac{S_{Ave} - S_{Min}}{S_{Ave}} \right) = \frac{S_{Min}}{S_{Ave}} \quad eq. 5.7$$

Applying this selectivity measure to the simple images of table 5.2 using the MAD match statistic and a window and neighbourhood radius of five pixels, produces the results of figure 5.1. With no noise added to the images the results are consistent with the relative dimensionality of each image. Images 1 through to 4 show zero selectivity since their dimensionality is less than or equal to one and hence their minimum sharpness value is zero. The other two images show high selectivity with image 6 recording a perfect value due to its single point structure.

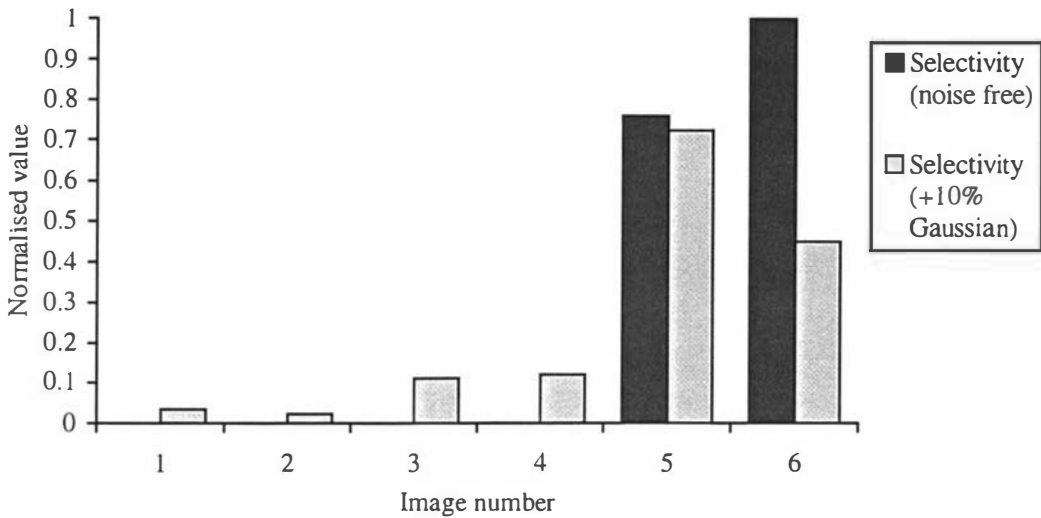


Figure 5.1: Selectivity of the simple grey-scale images of table 5.2

The addition of 10% gaussian noise to the images significantly changes their selectivity as is shown by the second set of bars in figure 5.1. The low dimensional images 1 through to 4 now show a small selectivity value. The one dimensional images 3 and 4 display a larger selectivity value than do the zero dimensional images 1 and 2. Image 5 (the corner image) displays a small drop in selectivity whereas the single point image, image 6, displays a large drop in selectivity. This large drop in selectivity is due to the point nature of this image that is easily corrupted by the noise. The corner image, image 5, is significantly less effected by the noise as its underlying structure covers a larger area.

5.2 Salient Features and Optimising Match Selectivity

Biological vision is inherently active. Humans and animals actively acquire visual information about their environment to guide their behaviour. The paradigm of active computer vision has in recent years led to the study and development of "*seeing systems*", as outlined by Aloimonos *et al* [5.5] and Pahlavan *et al* [5.6]. The most important concept underlying this approach is the concept of fixation. When a human views a scene the visual system is often directed to certain types of simple geometric features. The scene is scanned by jumping from one fixation feature to another in order to build up global and local geometric information. Bajcsy and Campos [5.7],

and Rimey and Brown [5.8] have proposed a general framework for what has become known as the "where to look next" problem. Other researchers [5.9] have most recently acknowledged that the active choice of fixation points is task dependent. In the case of manufactured objects, important cues to shape are junctions, and their types, and boundary edges with classification as being curved or straight. When a scene is dynamic and contains moving objects, then fixation points become important to the task of object tracking and leads to the concept of feature saliency. Salient features should change only slowly during the motion of the objects thus greatly aiding the ability to track their motion.

For cineradiographic images of the human spine the salient objects of interest are the vertebrae. The corners and junctions between the vertebral body and spinous-process could be considered as salient features of the individual vertebra. In order to evaluate the four match statistics defined in table 5.1, salient points were defined on the skull, and first and second vertebra of the middle frame of a typical cervical cineradiographic sequence. The position of these points is indicated in figure 5.1 by the white crosses. The ease at which the individual feature points can be located visually varies significantly between points. The point on the back of the skull is of high contrast but is fairly one-dimensional, whereas the front bottom point of the second vertebra is of low contrast but highly two-dimensional as it occurs on a sharp corner. This variation in feature characteristics should be reflected in the match minimum and average sharpness measures and the selectivity measure. A single parameter determines the behaviour of the sharpness measures for a given feature and match statistic, and this is the shell radius. The bigger the radius, the larger the area over which sharpness is computed and hence the less localised the measurement becomes. A suitable choice of radius should reflect the differences between each of the feature points.

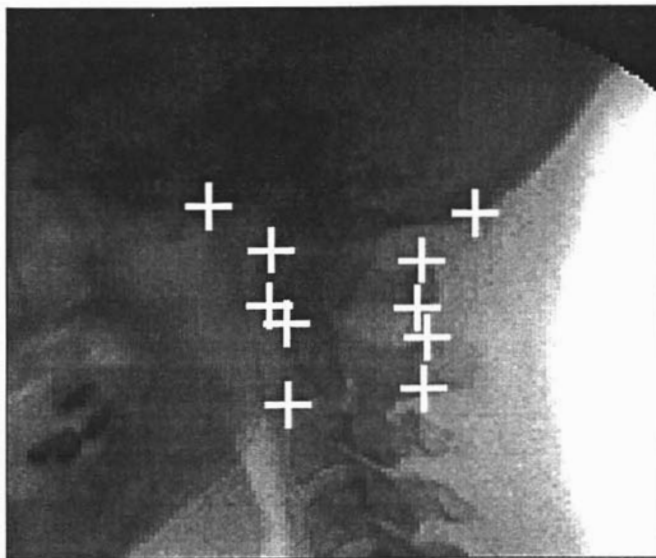


Figure 5.2: Salient feature points defined on a typical cervical cineradiographic image

All ten of the feature points defined in figure 5.2 were processed by first computing an auto-match surface about each position and then calculating selectivity for a range of shell radii. The match was computed with a window radius from two to four pixels. At four pixels radius the feature window is approximately half the height of a typical cervical vertebra.

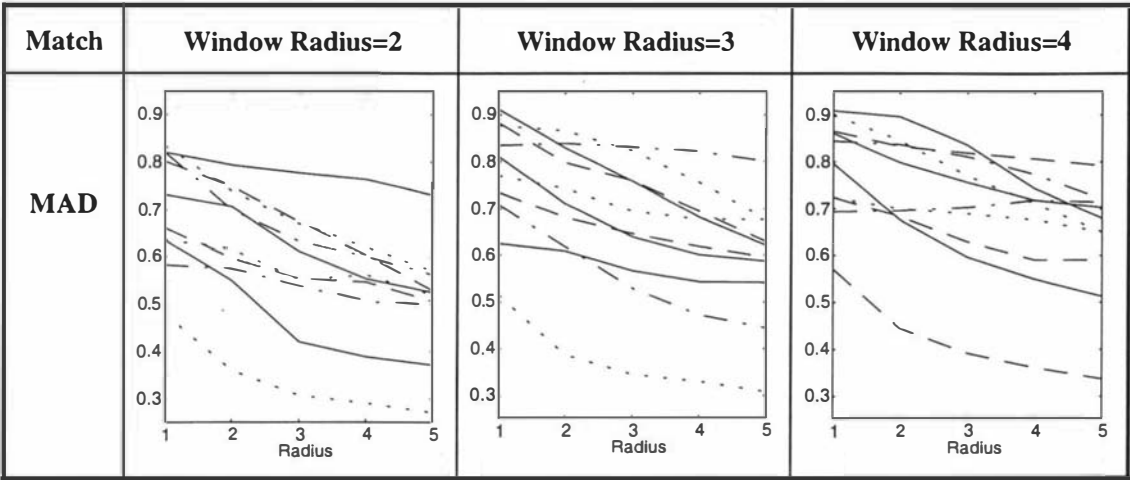


Table 5.3: Variation in selectivity with increasing shell radius for MAD statistic

Table 5.3 contains the selectivity results for the MAD match statistic for the three window sizes. Each of the traces on the graphs correspond to one of the ten feature points. The graphs show a general downward trend in selectivity for almost all match feature points. This general behaviour was also observed for the other three match statistics. The decline begins to level off for a shell radius of about three pixels. As the shell radius increases further past this value, the selectivity traces tend to converge for several of the feature points. This convergence implies that feature discrimination based on selectivity is lost at large radii. At a radius of three pixels the selectivity traces tend to be maximally separated from each other. This size radius will be used for the remainder of the match statistic performance evaluation.

5.3 Match Statistic Evaluation

One of the first steps in evaluating the performance of the match statistics on the feature points is to determine how sharpness and selectivity change as the size of the feature template increases. A single parameter, the window radius, controls the size of the template if it is made square.

5.3.1 Auto-frame Feature Point Matching

An initial sense of the behaviour of a match statistic can be obtained by matching the template onto itself in a local neighbourhood. This results in a match surface from which sharpness and selectivity can be computed. Table 5.4 contains examples of match surfaces for the position at the back of the skull and the front of the second vertebra. The data have been normalised and are

displayed as both an intensity image and as a 3-D plot. The dynamic range of the MAD and RMS statistics is inverted and displayed with the same polarity as the correlation data.

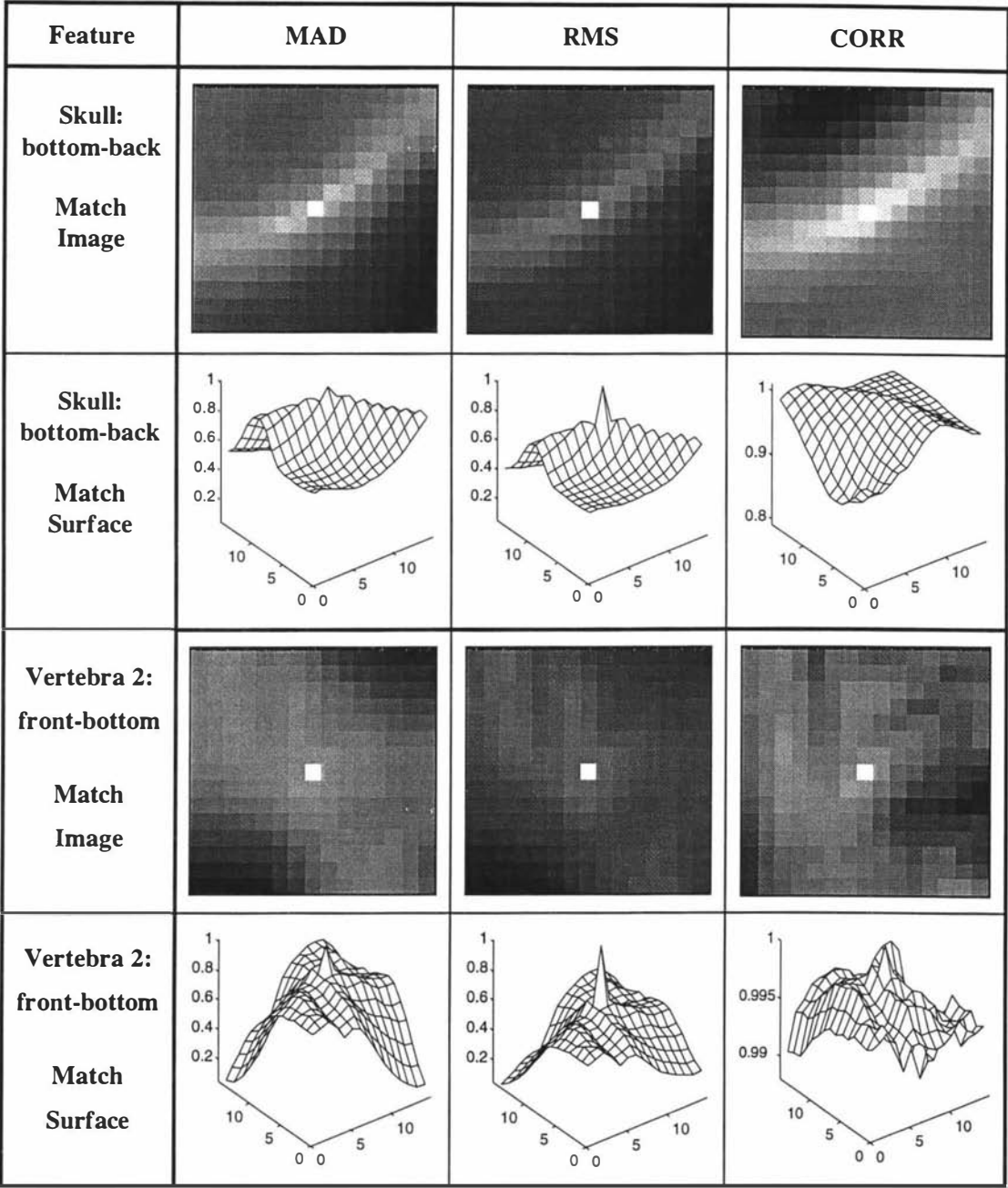


Table 5.4: Auto-match data for a window radius of three pixels

For the position marked on the bottom of the skull at the back (bottom-back), the local feature is of high contrast but is relatively one-dimensional. The one-dimensionality of this region is reflected in the match data for all three match statistics. Parallel to the direction of the skull-line the match decreases very slowly whereas normal to this direction the match decays rapidly. Both difference statistics produce a sharp peak at the original template position whereas the

correlation match is much smoother in all directions without a sharply defined peak. The peak for the RMS match is significantly greater than for the MAD match.

The second set of match data shown in table 5.4 corresponds to the bottom of the front of the vertebral body of the second vertebra (front-bottom). This local feature is of much lower contrast than the skull feature due to absorption and scattering in the local soft tissue. It is also more two-dimensional than the skull feature as it is located on a corner of the body of the vertebra. This increased dimensionality is reflected in the match data for all three match statistics. The match decreases fairly rapidly in most directions with only a slight bias in one direction as indicated by a broad diagonal band. All of the statistics show a peak at the original template position with the RMS statistic showing the largest peak. The correlation match is significantly less smooth than the other two match statistics to the extent that the broad diagonal band displayed by the two difference statistics is not visible.

Table 5.5 contains graphs of match sharpness as a function of match window radius for all feature points for each match statistic. The feature points have been grouped in terms of the feature with which they are associated. The solid traces represent the average sharpness while the broken traces represent the minimum sharpness. Both difference statistics (MAD and RMS) show minimum and average sharpness increasing with increasing window radius for almost all the feature points. The average sharpness for a few of the points peaks and then decreases past a certain window size. This is particularly obvious for the top trace on the skull graphs where the trace represents the position at the back of the skull. This effect is due to the match window encompassing a greater area containing only skin, which is homogeneous and noisy. However the minimum sharpness continues to increase as more of the curvature of the skull-line is contained in the match window. This effect levels out at around six pixels by which time the slight corner of the skull-line is now contained in the match window and any further increase only introduces more of the essentially one-dimensional skull-line either side of the corner.

Overall, sharpness for the RMS statistic is significantly greater than for the MAD statistic. This is principally due to the peaking effect illustrated in the match surfaces of table 5.3. Both the area correlation and moment-based correlation display significantly lower sharpness than either of the two difference statistics. The minimum sharpness for MCor is particularly low. There is also significant variation in sharpness between feature points. The correlation statistic (CORR) displays dramatic changes in both minimum and average sharpness for the feature points of the two vertebra. This variation is consistent with the notable visual difference between these feature points. An optimal window radius is reached for many of the vertebral feature points where sharpness is maximised.

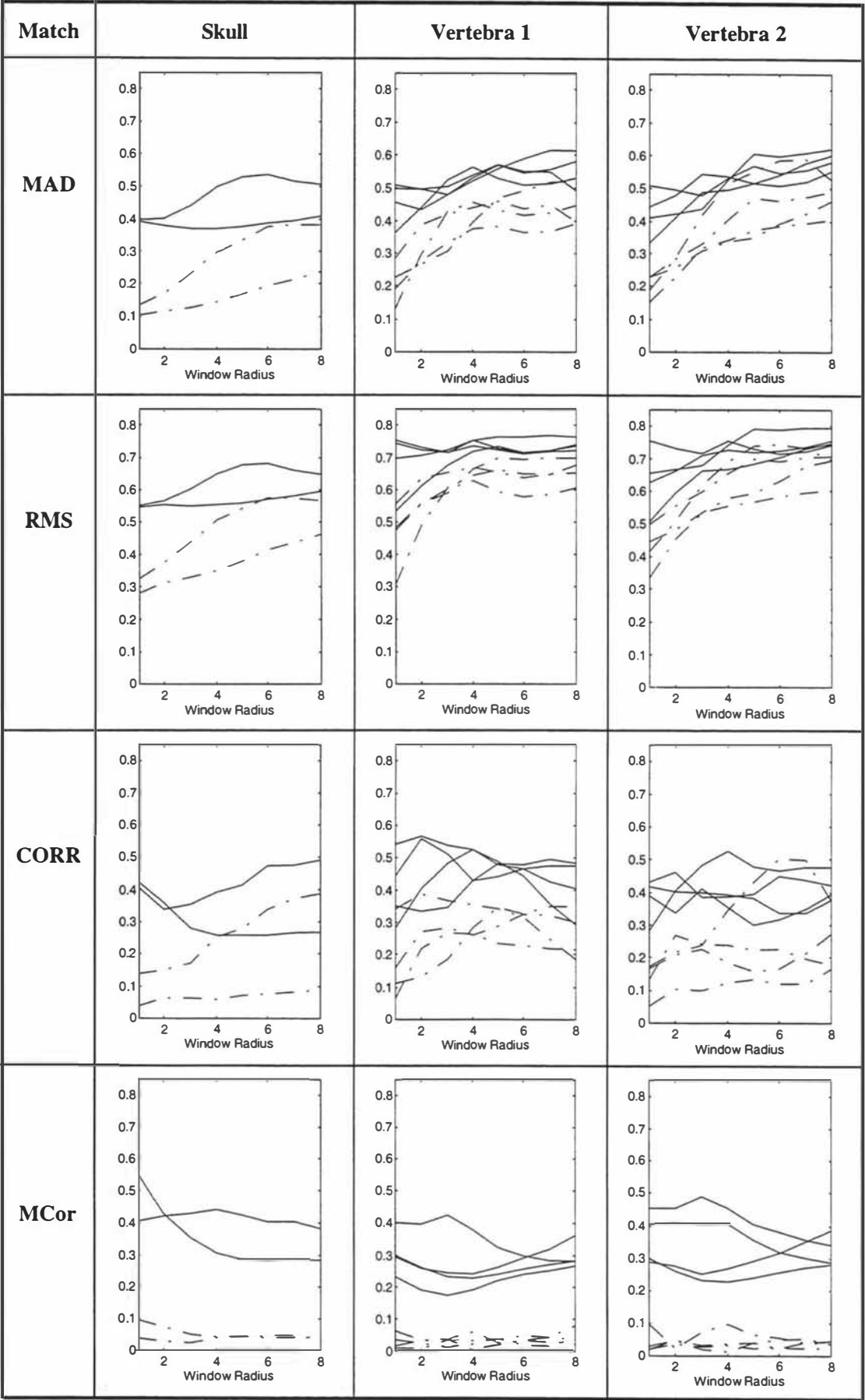


Table 5.5: Match sharpness for increasing window radius

The average sharpness for the moment based correlation statistic changes relatively smoothly, reaching a maximum at about a four-pixel radius for half of the feature points. For the other feature points the average sharpness goes through a minimum at this radius. Minimum sharpness varies erratically as the window radius is increased going through several local minima and maxima.

The selectivity measure described by equation 5.7 combines the minimum and average sharpness measures into a single measure describing the uniqueness of a given feature point and match statistic combination. Table 5.6 contains selectivity graphs for the three features for each of the match statistics. Each of the graph traces can be keyed back to the actual feature position marked in figure 5.1, using table 5.6.





Line-type and feature				
	(1)	(2)	(3)	(4)
Skull	bottom-back	-	bottom-front	-
Vertebra 1 and 2	back-bottom	back-top	front-bottom	front-top

Table 5.6: Key to match selectivity graph traces of tables 5.7 to 5.10

The selectivity of the skull feature points increases with increasing window radius for all but the MCor statistic. Selectivity for the MCor statistic is low and relatively flat with small local minima. For the first vertebra the selectivity of both the MAD and RMS matches peak at about three pixels and then level out to similar values. Selectivity of the correlation match on the other hand peaks at different radii for each of the feature points, thus illustrating the greater discrimination of this statistic. The moment-based correlation selectivity is highly erratic and goes through several local maxima and minima, all comparatively low in value. This behaviour also occurs for the second vertebra. Greater selectivity discrimination can be seen for the two difference statistics for the second vertebra and this response carries over to the correlation match. This improvement in selectivity discrimination is primarily due to the second vertebra being significantly larger than the first vertebra and that it contains more sharply-defined corners. The selectivity of the moment-based correlation still shows highly erratic behaviour, as observed for the first vertebra.

The reason for the poor performance of the moment based match can be explained in terms of the properties of this statistic. Because it is based on moment invariants the match should not vary for changes in translation, rotation and scale. While rotation and scale invariance are highly desirable, translation invariance is not desirable when an accurate match in position is required.

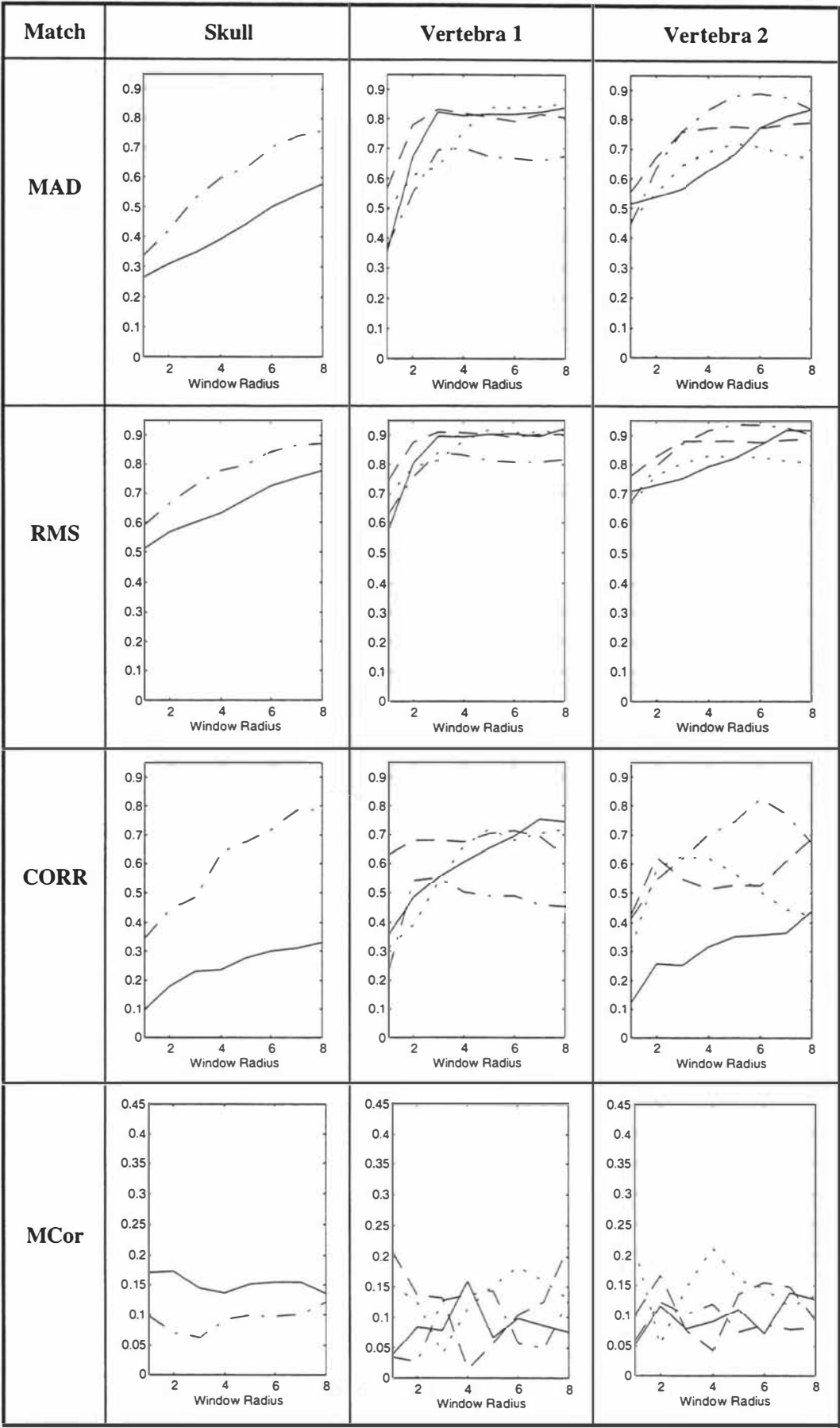


Table 5.7: Match selectivity for varying window radius

Other authors [5.10] have noted that moment-based correlation will match silhouettes or binary outlines with high positional accuracy but matching grey-scale data generally does not produce a sharp match. General scene matching schemes based on other translation invariant methods [5.11] for grey-scale data have used the invariant match as the top stage of a hierarchical multi-scale procedure. In such procedures potential template match candidate positions are first found using large translation steps up to half the size of the match template. These potential sites are then investigated locally using a finer step size and the remaining candidates then tested in detail using an area correlation to give the desired positional accuracy. For cineradiographic images of the human spine, locality is already known to be within a few pixels of the previous position due to the relatively small interframe motion. Thus moment-based matching with simple translation will not produce a sufficiently accurate positional match.

5.3.2 Interframe Feature Point Matching

In the previous section an initial idea of the performance of the four match statistics was obtained by computing a match on the original frame in a local neighbourhood about the feature position. An evaluation of this type is equivalent to the case where there is only interframe translation of pixel size increments. No information is gained about the behaviour of the statistics in the presence of noise, scale changes or rotation in such a situation. To assess the performance of the match statistics under real-world conditions (all these factors present), an interframe local match was performed about the middle frame of the cervical test sequence. The middle of the sequence was chosen because this is where maximum velocity usually occurs and hence maximum rotation and translation. The match process starts at the original frame position in the next frame, and searches in square annuli until the best match is found, or the limit of the search radius is reached. If a best match is found, then the selectivity is calculated at this location and the x and y coordinates recorded.

Selectivity and Positional Stability

The two factors important to interframe matching, and hence motion-tracking, they are the positional accuracy and positional stability of the best match. In order to assess positional accuracy for a given type of image data, image sequences with known motion are required. This issue is dealt with in the next chapter where methods are developed for the creation of cineradiographic sequences with prescribed motion. Interframe match positional stability can be assessed by plotting the position of the best match as the match window radius is increased. Any sudden change in either the x or y coordinate will be indicative of positional instability. This should only occur when match selectivity is low and/or there is a significant change in selectivity between window radii.

Tables 5.8 and 5.9 contain selectivity and best match x and y position graphs for the MAD statistic for each of the feature points, for previous and next interframe matches. The feature point represented by each trace is keyed according table 5.6. Positional data are plotted with respect to the original reference frame such that $(x,y)=(0,0)$ is the original position. A selectivity value or positional value that is at the limit of a graph scale implies that a best match was not found within the local search radius of five pixels.

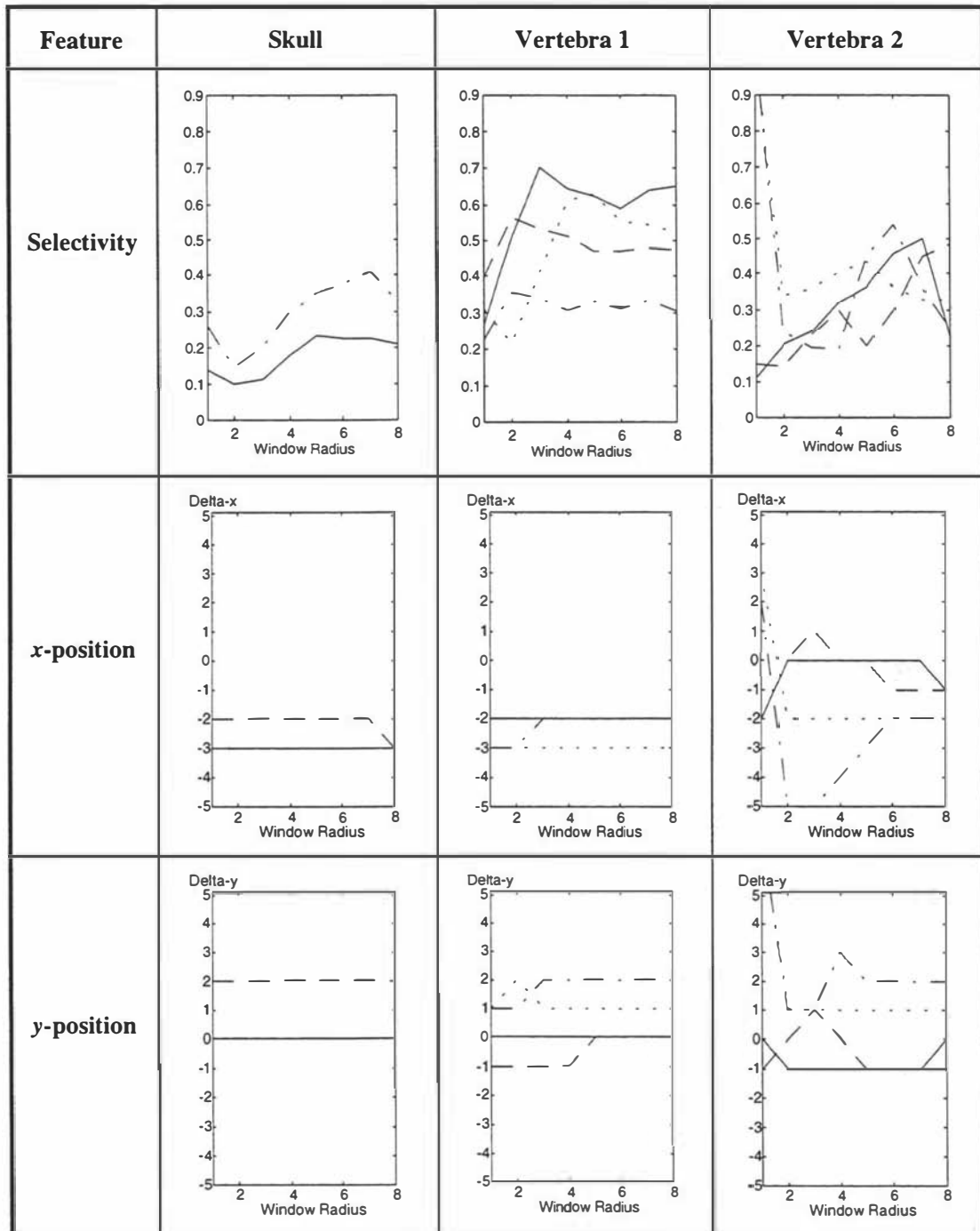


Table 5.8: MAD previous frame match: Selectivity and x - y position

Both skull feature points for the MAD statistic for the previous interframe show low selectivity that decreases and then increases with increasing window radius. Selectivity for the bottom-back feature peaks at five pixels and levels out, while selectivity for the bottom-front feature peaks at seven pixels and then rapidly decreases. This sudden drop in selectivity is accompanied by a jump in the x -position. For all other values of the window radius the x and y position do not change, indicating that the best match is stable. The selectivity of the feature points of the first vertebra rapidly increases, then peak and stabilise at different window radii. The front-top feature peaks at the largest radius, five pixels, and then decreases. A transition occurs in the x position in only one of the feature points, all others are stable. However, the y position of most of the feature points undergo a transition at the selectivity peak and then stabilise. Quite a different behaviour can be seen for the second vertebra. Two of its feature points do not find a best match in the search area until the window radius is two pixels. The selectivity traces are also much more peaky and do not stabilise with increasing window radius. Both the x and y position undergo many transitions. The position for most of the feature points is stable for a window radius between five and seven pixels. This significant difference in behaviour compared to the first vertebra can be explained in terms of the difference in size of the vertebra, the difference in feature contrast, and also the proximity of other features.

The second vertebra is significantly larger than the first and thus it is reasonable that match selectivity will peak at a larger window radius. The rapid drop in selectivity after the peak, as the window radius is increased, is due to nearby features, namely the vertebrae above and below the second vertebra, beginning to appear in the match template. Since the motion of these vertebrae is different to the second vertebra, the local match degrades rapidly as more of the interfering feature appears in the template area. This effect can also be seen for the front-bottom feature point of the first vertebra, with its close proximity to the top of the second vertebra.

The selectivity and positional graphs for the next interframe match using the MAD statistic are shown in table 5.9. As with the previous frame match, the skull feature points show low selectivity that decreases and then increases with increasing window radius. However, unlike the previous frame match, it is less obvious what the optimal window radius is as there is no well-defined peak in selectivity. Furthermore, the bottom-front feature point does not find a best match until a window radius of two pixels. The x and y position become stable between four and seven pixels window radius and then at eight pixels the x position moves a pixel. This shift in position occurs where there is a sudden increase in selectivity due to a nearby feature point on the skull becoming included in the match template area. The selectivity behaviour of the first vertebral feature points is similar to the previous interframe match except that selectivity is generally lower and the peaks occur at different window radii.

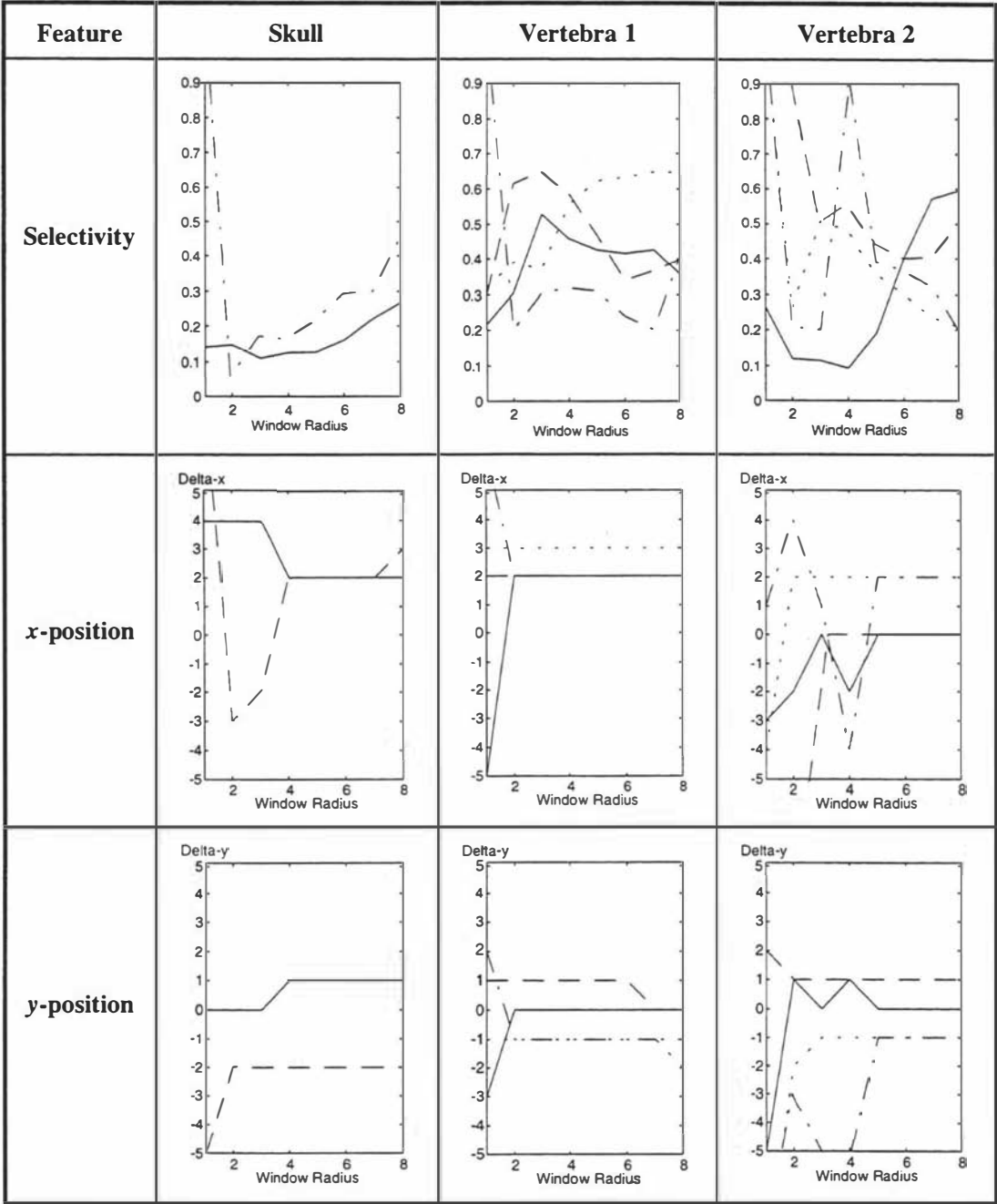


Table 5.9: MAD next frame match: Selectivity and x-y position

Also, the front-bottom feature does not find a best match until a window radius of two pixels. Position is stable between two and six pixels radius after which the y position of two of the features moves due to a sudden increase in selectivity. The next interframe match behaviour of the second vertebra is significantly different to the previous interframe match. A best match is not found for three of the four feature points until a window radius of two pixels and is then lost at four pixels radius for the front-bottom feature. As with the skull feature points, it is difficult to define an optimal window radius for several of the second vertebra feature points due to the lack of a well defined selectivity peak. Selectivity of the back-top feature peaks at four pixels, begins to decrease and then increases past seven pixels as the bottom of the first vertebra begins

to appear in the template area. The position data undergo many transitions but then stabilise after five pixels radius.

The notable difference in selectivity and positional stability for the next interframe match compared to the previous interframe match can be explained in terms of the actual interframe motion. Visualisation of a sequence consisting of only the middle frame and the two frames either side clearly revealed that the forward direction (next frame match) movement is significantly greater than in the backward direction (previous frame match). This increased movement results in greater vertebral rotation and hence a general reduction in feature point selectivity.

Interframe Match Summary

The selectivity and positional behaviour of the MAD, RMS and CORR match statistics, for previous and next interframe matches, has been summarised in table 5.10. The moment based match, MCor, has not been included in the table because for almost all feature points a best match was not found within the search radius. The reason for this is that the value of the Moment correlation is extremely close to unity throughout the search area, for all window radii. Hence the match surface is essentially flat and the minimum sharpness is nearly zero. The reason for this lack of discrimination in the interframe match is as explained at the end of section 5.3.1 for the auto-frame match. Rotation, scale and translation invariant schemes generally do not produce accurate positional matches but do indicate that the general area contains the feature of interest.

The selectivity graphs have been summarised in terms of average selectivity and average degree of selectivity peakiness, for the three vertebral features. These two summary categories are scored using a low, medium or high (L,M or H) rating. Also contained in the summary table is the best window radius for each of the feature points. The best window radius is defined as the window radius at which selectivity reaches a local maximum and the x and y position are stable. The window radii are in order of the feature points defined left to right in table 5.6. The pattern of selectivity and selectivity peakiness is almost the same for the two difference statistics (MAD and RMS) for the previous interframe match, and covers the full range in both categories. In contrast, the correlation match ratings are more stable and do not score high in either category. There is also a strong degree of consistency in the best window radius for the two difference statistics, covering a range from two to seven pixels across all feature points. Less variation in the best window radius is observed for the correlation match, with the range now from three to six pixels.

The pattern of selectivity and selectivity peakiness for the next interframe match for both the MAD and Correlation statistic is the same as for the previous interframe match. The next

interframe RMS match on the other hand shows a slightly different pattern to the previous interframe match. High-score cases have been reduced to a medium score. For all the statistics match, the best window radius tends to be smaller for the next interframe match than the previous interframe match. This reduction in size is principally due to the greater forward interframe vertebral rotation that degrades the match for larger window radii.

Frame	Statistic	MAD			RMS			CORR		
	Feature	S	V1	V2	S	V1	V2	S	V1	V2
Previous	Selectivity	L	H	M	L	H	M	L	M	M
	Selectivity peakiness	L	M	H	L	M	H	L	L	M
	Best window radius	5,7	3,2 2,5	7,4 6,5	5,5	3,2 2,4	7,4 6,5	4,6	5,3 3,4	6,5 6,4
Next	Selectivity	L	H	M	L	M	M	L	M	M
	Selectivity peakiness	L	M	H	L	M	L	L	L	M
	Best window radius	5,6	3,3 4,6	7,4 5,3	4,4	3,3 4,3	6,3 5,4	4,4	5,4 5,6	2,3 3,3
KEY: S = skull, V1 = first vertebra, V2 = second vertebra, L, M, H = low, medium, high										

Table 5.10: Summary - Interframe matching

Interframe Selectivity Constancy

Although the best match window radius for the forward interframe was generally smaller than for the backward interframe match, it is important that the selectivity of each feature point at the best match window radius should remain relatively constant between interframes. Visually it is reasonable that this should be the case as there is very little observable change in local feature characteristics between the two interframes. Thus plotting interframe match selectivity at the best window radius for each feature point should show little difference between the previous and next interframe data. Table 5.11 contains the best interframe match selectivity graphs for the MAD, RMS and Correlation statistics. The continuous trace on each graph represents the previous interframe match, while the broken trace is the next interframe match. Each position on the x-axis corresponds to the feature points listed in order, in table 5.6.

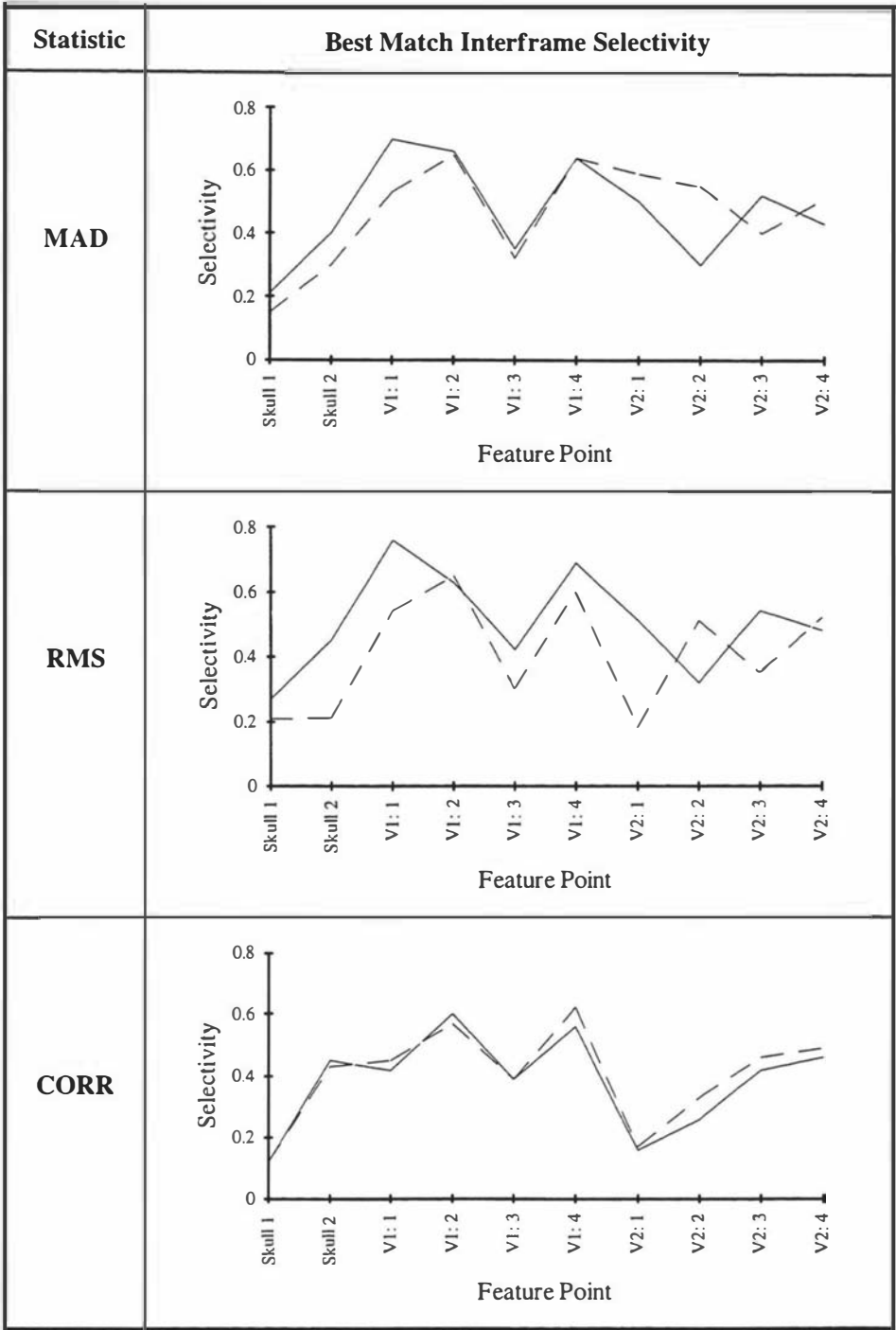


Table 5.11: Best interframe match selectivity

For the MAD statistic there is little difference in selectivity between the previous and next interframe match for some of the feature points, while for others points there is a fairly large difference. High-selectivity constancy occurs for three of the four feature points on the first vertebra. The RMS match consistently shows a large variation in selectivity between the two interframes for all the feature points. Only the correlation based match displays best match selectivity that changes very little between the forward and backward interframes. This is in

spite of the fact that for some of the feature points the best match window radius has been reduced significantly due to the greater vertebral rotation.

5.3.3 Global Image Match Selectivity

Determining the locations in an image that are locally unique in the sense that they have a high match selectivity for a particular match statistic and template window size, will indicate features that are potentially good for template based motion-tracking. The same procedure that was used in the auto-frame and interframe feature point matching described in the two previous sections, can be applied globally to any given frame or interframes. About each location in the reference image a square template of a specified window radius is matched in a local neighbourhood (auto- or interframe) of a specified search radius. This will produce a match surface for each location from which a single selectivity value is calculated about the best match position. The selectivity values can then be formed into a matrix and visualised as an image. The advantage of this approach for interframe matching is that the only assumption made is that the maximum interframe translation anywhere in the image is smaller than the search radius.

Table 5.12 contains selectivity images for the previous and next interframe matches, and also the current frame (auto-frame) match, for the middle of the cineradiographic cervical test sequence. The first column in the table contains MAD statistic images, while the second column contains the product correlation selectivity images. A window radius of four pixels has been used for all the images, this being the average of the best window radii for the interframe vertebral feature point match, summarised in table 5.10. A large search radius of seven pixels was used to ensure that the maximum observed interframe translation would be matched. If a best match was not found within the search radius then the recorded selectivity was set to zero. For interframe matching, this will only occur when the template contains no significant structure, such as in the relatively homogenous soft tissue areas.

Looking at the MAD auto-frame selectivity image obvious features such as the skull-line show low selectivity due to their local one-dimensional characteristic. However, many locations in the image show high selectivity in areas that are visually relatively homogenous. This effect is particularly obvious in the area of the skin at the back of the neck. Local to the circular aperture produced by the image intensifier, a wide selectivity band occurs. This indicates that the MAD statistic is not particularly discriminating, for high contrast features. In comparison, the correlation statistic shows much greater discrimination throughout the image. Very few points in soft tissue areas score high in selectivity, while high contrast features such as the circular aperture are localised to only a few pixels. The centre of both auto-frame selectivity images clearly show that the vertebral bodies have much higher selectivity than do the vertebral tails (the spinous-processes).

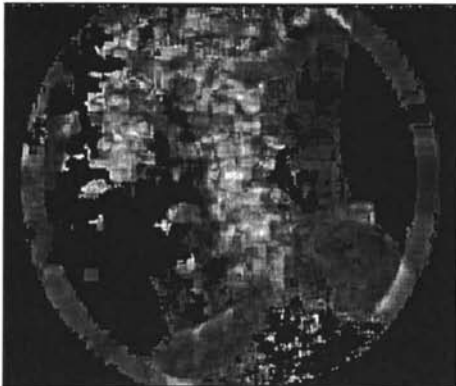
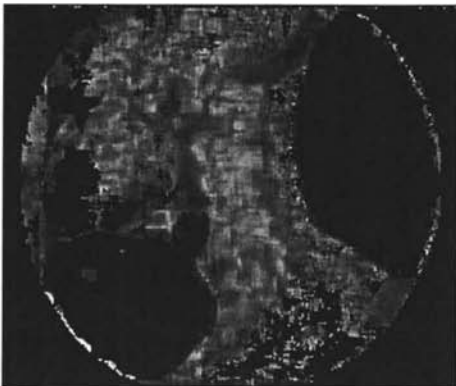

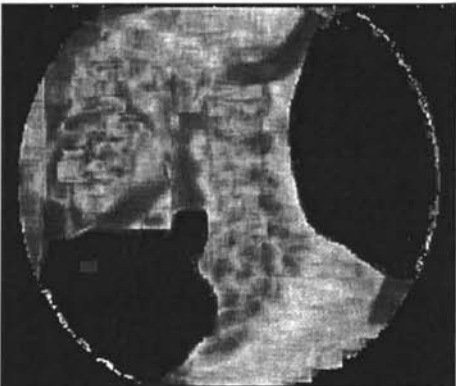
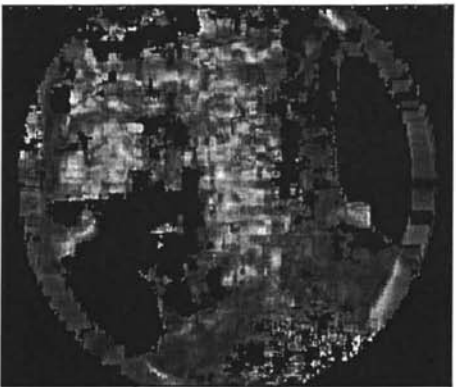
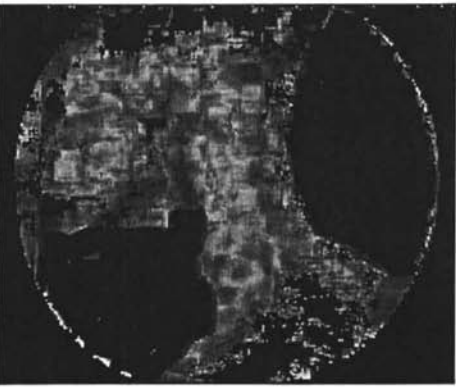
Frame	MAD-4	Coor-4
Previous		
Current (Auto)		
Next		

Table 5.12: Global Match Selectivity

Looking at the interframe match selectivity images, far fewer points are displayed with high selectivity and hence potentially good candidates for tracking. This is to be expected since there is significant motion (rotation and translation) between these images. Selectivity of the skull in particular is greatly reduced due to its comparatively large rotation. Selectivity of many of the vertebrae is reduced to a larger extent in the forward or next interframe match than the previous interframe. This is once again due to the greater motion (mainly rotation) present in this interframe. The correlation statistic consistently displays better selectivity discrimination and feature localisation than does the MAD statistic for both interframe matches.

5.4 Match Statistic Conclusions

This chapter has primarily been concerned with the experimental assessment of the performance of four template match statistics. The purpose of the assessment was to choose a suitable match statistic to be used in the evolution of a complete motion-tracking algorithm for cineradiographic images of the human spine. A feature selectivity measure was developed to aid in this choice. This measure incorporates the effects of both feature dimensionality and local sharpness. While feature selectivity is an important property in its own right, it is the positional stability and positional accuracy of the match that are of prime importance to a motion-tracking algorithm.

The MAD statistic showed good selectivity characteristics for the ten chosen feature points for both interframe matches, but positional stability for some of these feature points was not particularly good. The banding seen in the selectivity images of table 5.12 further illustrated this property. This general behaviour was also observed for the other difference statistic, where even greater positional instability was detected.

The moment invariant based correlation statistic proved to be totally unsuitable for template matching of these type of images due to its low selectivity and poor positional stability. Only the simple product correlation produced reliable and consistent behaviour for all feature points for both interframe matches. Match positional localisation was also shown to be very good, as clearly illustrated in the global interframe match selectivity images. Furthermore, there was a high degree of selectivity constancy between the interframes for all ten prescribed feature points. The computational cost of a product correlation is relatively low, particularly when small sized templates are used, as will be the case for the proposed motion-tracking algorithm. One further advantage of the correlation measure described in this chapter is that it is inherently normalised. This is important because the complete motion-tracking algorithm described in chapter 7 will take advantage of this property by forming a global match for an object based on a number of small sub-templates positioned about the object.

The development of methods in this chapter to assess the performance of the match statistics does in itself provide a means of making a motion-tracking algorithm adaptive to the changes in the characteristics of a moving object. The initial choice of the feature points at the start of the tracking procedure is critical to the overall tracking success. By computing the forward and backward optimal (best template window radius) interframe selectivity of a feature point, a figure of merit could be calculated to score the potential success of tracking this location. Alternatively, having defined a feature point, a local optimisation could be performed to find the local optimal feature position based on a forward and backward interframe selectivity assessment. These ideas will be expanded upon later in the thesis.

5.5 References

- [5.1] **Gonzalez R. and Wintz P.** *Digital Image Processing*. Addison-Wesley, pp 69..72, 1977.
- [5.2] **Williams T.D. and Glazer F.** *Comparison of Feature Operators For Use In Matching Image Pairs*. Image Sequence Processing and Dynamic Scene Analysis. Springer-Verlag, Berlin-Heidelberg, pp 395..423, 1983.
- [5.3] **Morevec H.P.** *Rover Visual Obstacle Avoidance*. Proceedings of IJCAI-7, Vancouver B.C., August 1981.
- [5.4] **Hannah M.J.** *Bootstrap Stereo*. Proceedings of Image Understanding Workshop, DARPA Conference, pp 201..208, April 1980.
- [5.5] **Aloimonos Y., Weiss I. and Bandyopadhyay A.** *Active Vision*. Proceedings of the First International Conference on Computer Vision, pp 35..54, 1987.
- [5.6] **Pahlavan K., Uhlin T. and Eklundh J.O.** *Integrating Primary Ocular Processes*. Proceeding of the Second European Conference on Computer Vision, Vol. 588, pp 526..541, Lecture Notes in Computer Science, Springer-Verlag, May 1992.
- [5.7] **Bajcsy R. and Campos M.** *Active and Exploratory Perception*. CVIGIP: Image Understanding, Vol 56 (1), pp 31..40, July 1992.
- [5.8] **Rimey R.D. and Brown C.M.** *Where to Look Next Using a Bayes Net: Incorporating Geometric Relations*. Proceeding of the Second European Conference on Computer Vision, pp 542..550, May 1992.
- [5.9] **Brunnstrom K. and Eklundh J.** *Active Fixation for Junction Classification*. Proceedings of the Second European Conference Computer Vision, pp 452..459, 1992.
- [5.10] **Wong,R. and Hall E.** *Scene Matching With Moment Invariants*, Computer Graphics and Image Processing 8, pp 16..24, 1978.
- [5.11] **Schutte H., Frydrychowicz S. and Schroder J.** *Scene Matching With Translation Invariant Transforms*. Proceedings of the 5th International Conference on Pattern Recognition. pp 195..198, IEEE, New York, 1980.

Prescribed Motion Cineradiographic Sequences

6.1 Introduction

Many of the elements necessary to develop a motion-tracking algorithm for cineradiographic sequences of the human spine have already been described. The previous chapter was about the selection of a suitable template match statistic. Of the four statistics evaluated, the normalised product correlation was shown to have the best selectivity and positional stability under interframe matching conditions. Although good positional stability does not necessarily imply high positional accuracy, a severe mismatch will generally only occur when the location has poor selectivity, as in the case of low dimensional or low contrast features. Assuming at this stage that a system for the measurement of vertebral spinal motion has been completed, the evaluation of such a system would be extremely difficult as there is no independent method by which the internal vertebral motion can be measured.

One approach to solving this problem might be to record a range of real sequences and then get a number of experts to mark each frame manually in a sequence to locate defined feature points. Experience has shown that manually tracking individual feature points in cineradiographic images is extremely difficult, even over just a few interframes. This is because the feature points of interest are generally poorly defined. In the previous chapter (see section 5.5.3) it was shown that the corners of the vertebral bodies were the best feature points to track. However, even these so called corner points are not sharply defined. Thus attempting to manually track them with good reliability over 50-100 interframes would be extremely difficult.

More reliable tracking results could be obtained *manually* by using outline templates. In this approach the outline of the entire vertebra is traced onto a transparency (or stored digitally). In each subsequent frame a best match is found for the entire outline by positioning it over the corresponding vertebra. Good results can be obtained using this method since the entire structure of the vertebra is used in the match. Like the manual point-based case, the only way to ensure an accurate result over an entire sequence is to combine the results of a number different experts.

An alternative to taking a real sequence and trying to measure the motion by some other means would be to create sequences with known motion. An artificial sequence must possess similar properties to a real sequence to ensure that it provides an effective test for any proposed motion tracking and measurement scheme and it must not introduce any additional artefacts.

6.2 The *Stop-and-shoot* Method

A common approach used to create sequences with known motion is the *stop-and-shoot* method [6.1]. In this method the objects that are to appear to be in motion in the scene are manually positioned at the desired starting location for the sequence and a single frame is captured. The objects are then moved to the next location and another frame is captured. In this way sequence will be built up in which the objects appear to be in motion. Superficially, this approach appears to be a totally satisfactory method for the creation of a prescribed motion sequence. This is true if the motion to be prescribed is sufficiently small that at a normal rate of capture the object can be considered to be near stationary. Consider the case in which the area of a scene is 1 x 1 metre and is digitised linearly to one pixel per millimetre. Assuming that the capture rate is 25 frames per second (equivalent of 50 fields per second in conventional interlaced video) and that stationary simply implies that movement should be substantially less than one pixel per frame, say a tenth of a pixel, then the maximum velocity of an object must not exceed $0.1 * 25 = 2.5$ millimetres per second. At this speed an object would take two hundred seconds to move from one side of the scene to the other. Even if the stationary criterion is relaxed to half a pixel per frame, at maximum speed an object would still take some forty seconds to traverse the image.

If the actual motion in a scene exceeds half-a-pixel per frame, as would be the case in most cineradiographic motion sequences, then during the frame integration time a moving object would traverse across many sensor elements. The effect of this motion is that edges of the object in the direction of the motion will not appear sharp, but will be spread across several pixels. This phenomenon is called *motion blur* [6.2], and is readily observed in photographs of fast-moving objects such as racing cars where the detail of the car is blurred in the direction of motion. Although motion blur is sometimes seen as a problem, it is the natural result of ensuring the integrity of the data representation and hence the temporal derivatives. This guarantees that the data can be interpolated and differentiated (see chapter 4, section 4.2.1), two essential requirements for the success of most motion-tracking algorithms. This happens because the signal becomes temporally band limited, thus reducing or preventing temporal aliasing [6.3]. Temporal aliasing manifests itself as sudden jumps in the position of an object known to be in a smooth continuous motion. In a sequence created using the *stop-and-shoot* method the real object is always stationary when the frame is recorded, thus no motion blur occurs and hence temporal aliasing will take place. If however the positional jumps are relatively small, then when the sequence is viewed by a human observer at an appropriate frame rate the motion will appear

continuous due to *persistence of vision*. Persistence of vision results from the temporal integration that is a fundamental property of the human visual system. Several different temporal integration time-constants have been experimentally measured [6.4], the longest being approximately 100ms and the shortest 15ms. There is good practical and theoretical evidence [6.5] to suggest that temporal integration over several resolutions is responsible for many of the remarkable motion-related properties of the human visual system.

6.3 Articulated Equivalent Phantom

The stop-and-shoot method for the creation of a motion sequence described in the previous section, will suffer temporal aliasing unless the interframe motion is at a sub-pixel level or some form of velocity-dependent smoothing is applied. When considering the application of such an approach to the creation of cineradiographic sequences of the human spine the first thing that must be considered is the spine itself. Long-term X-ray radiation exposure is a health risk. Furthermore, only the external position of the spine can be accurately measured. A partial solution to this problem, commonly used by medical biomechanics researchers, is cadaver use. This clearly overcomes the exposure problem, but the only way to get an accurate measurement of the location of the vertebrae is to remove much of the soft tissue surrounding the spine. This approach is unsatisfactory for a number of reasons. Firstly, there are the ethical concerns of using a cadaver for research and the practical issues of availability and hygiene. There is also an important biomechanical issue often ignored by researchers in this field, that is a preserved body cannot be positioned in the same way as a live one since the preservation process significantly changes the mechanical properties of the major connecting tissue such as tendons and ligaments.

A better solution to the problem would be to build an *equivalent phantom* human spine that can be accurately positioned and has similar X-ray absorption characteristics to a real living spine. A basic human skeleton can be used as the starting point for building such a phantom. Firstly the important connecting soft tissue needs to be simulated with suitable materials. The entire structure can then be sealed and immersed in a saline solution or embedded in a suitable flexible resin to simulate the bulk soft tissue X-ray absorption properties. The only major difficulty remaining with such a *near* equivalent phantom is animating the structure through a range of motion in a known measurable way. For generating equivalent sequences, only a section of the spine needs to be animated since the largest image intensifiers available are only 350 mm in diameter (see chapter 2, section 2.2.2). This could be done using some form of mechanical jig made out of non X-ray absorbing material that is attached to a number of vertebra. The jig could then be moved under machine control to produce a range of motion. However this approach still leaves unresolved the problem of accurately measuring the produced motion in each vertebra. Various sensor technologies could be used, but getting many of them to work reliably under X-ray bombardment is an entirely different matter.

6.4 Artificially Animated Cineradiographic Sequences

Rather than taking the conventional approach of producing a physical phantom spine which is then mechanically animated and cineradiographed, a more flexible alternative would be to synthesise the X-ray sequence directly. Synthesis in the computer graphics sense usually begins with a model for the components that will form the scene. For the biomechanical radiographic case both the mechanical and X-ray absorption related physical properties of the components must be quantified and modelled. The main components of a cineradiographic sequence of the human spine are the vertebra themselves (including the skull) and to a lesser extent the soft tissue. Figures for the relative X-ray absorption coefficient of bone and soft tissue can be determined from conventional radiographs, while the three-dimensional structure of typical vertebrae can be obtained from standard reference CT (*computerised tomography*) [6.6] images. Next the mechanical articulation of the structure has to be quantified. With all this data the task of building an accurate three-dimensional animation model could be embarked upon but would be a substantial undertaking. An alternative and more tractable task would be to begin with a single frame from a real cineradiographic sequence and to produce a two-dimensional animated sequence from it.

6.4.1 Digital Image Warping

Digital image warping is an active research area in computer graphics. Image warping involves a geometric transformation, for which there are three integral components: spatial transformation, resampling, and anti-aliasing. A *spatial transformation* defines a geometric relationship or mapping that establishes a spatial correspondence between all points in the input and output images. The input image is treated as consisting of a set of reference points that are known precisely, while the output image comprises the observed or *warped* data.

A general spatial mapping can be specified in two forms:

$$I_{out}[x, y] = I_{in}[X(u, v), Y(u, v)] \quad eq. 6.1$$

$$I_{in}[u, v] = I_{out}[U(x, y), V(x, y)] \quad eq. 6.2$$

where I is the image intensity, $[u, v]$ refers to the input image coordinates and $[x, y]$ to the output image coordinates, and X, Y, U and V are arbitrary mapping functions that uniquely specify the spatial transformation.

Functions X and Y map the input image to the output image and are referred to as the forward mapping, while U and V map the output image back to the input image and are known as the inverse-mapping functions. Both forward and inverse mapping can be used to produce a warp. However the forward or target mapping has significant advantages when the input image is to be

read sequentially as in the case of raster scanned images. It is particularly useful for separable algorithms that operate in scanline order (see Two-Pass Mesh Warping later in this chapter).

In a forward mapping each input pixel is copied onto the output image at positions determined by X and Y . Figure 6.1 illustrates a forward mapping for a simple one-dimensional case. The input and output spatial distributions are shown for a line of grayscale pixels lying on a discrete integer grid. Each input pixel is passed through the spatial transformation where it is assigned a new output coordinate. The spacing of the horizontal arrows represent the input and output spatial distribution of the forward mapping. In this example the distance between input pixels A and B has been compressed resulting in a closer projected spacing between output pixels A' and B'. Conversely the distance between input pixels C and D has been expanded, resulting in a wider projected spacing between output pixels C' and D'. Whereas the input pixels lie on an integer grid, the projected output position of the mapping may take on continuous or real values. Thus an interpolation stage is required to fit a continuous function to the discrete input data. The continuous function may then be sampled at arbitrary positions. This interpolation operation is known as *image reconstruction*. Jointly, image reconstruction followed by sampling is known as image resampling [6.7].

The real valued output positions assigned by the forward-mapping functions X and Y cause complications since the output pixels must also lie on a discrete integer grid. Holes may appear where the input-output mapping bypasses an output position as in the case of pixel E' in figure 6.1. Alternatively an overlap may occur where two or more consecutive input samples collapse into a single output pixel, as depicted by output pixel F'.

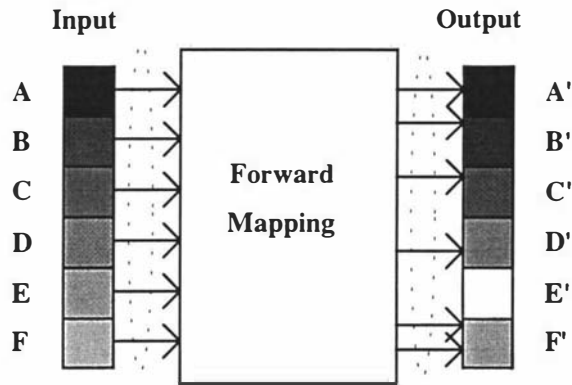


Figure 6.1: One-dimensional forward mapping

These two resampling problems arise because *point sampling* (zero-spread) has been used on the reconstructed input to assign the discrete output. Point sampling is where each input sample point is taken independently of its neighbours and thus can affect only one output point. With point sampling, entire intervals between samples may be discarded and their information content lost. If the input data changes smoothly the lost data may be recovered through appropriate

interpolation (reconstruction). However, when the discarded interval contains complex changes, interpolation may be inadequate and the lost data unrecoverable. In this situation the input signal is said to be *undersampled* giving rise to *aliasing*. The filtering used to counter aliasing is known as *anti-aliasing* and typically requires that the input be blurred before resampling, ensuring that the sampled points are influenced by their discarded neighbours.

In theory, aliasing can be prevented entirely by sampling at a sufficiently high frequency, as dictated by sampling theory or by designing an ideal anti-aliasing filter to suit the data. But due to practical limitations in the implementation of an ideal filter, many approximate anti-aliasing methods have been proposed. The *four-corner mapping* paradigm is one such solution to the problems produced by the simple point-to-point mapping. Four-corner mapping is an area sampling method in which input pixels are considered as square patches that may be transformed into arbitrary quadrilaterals in the output image. This ensures that the input remains contiguous after the mapping. However, this approach introduces two other complications that require the use of adaptive sampling of the input, based on the size of the projected quadrilateral, in order to resolve them. If the projected input patch covers more than one output pixel, as would be the case for local magnification, intermediate values must be interpolated to correctly fill the output positions. Alternatively if more than one projected input patch resides in an output position, as is the case for local minification (reduction in scale), then the patches must be averaged, based on their area, to fill the output position.

Two-Pass Transforms

Consider the case of a spatial transform specified as a forward mapping by functions X and Y such that:

$$[x, y] = T(u, v) = [X(u, v), Y(u, v)] \quad \text{eq. 6.3}$$

The transformation T is said to be *separable* if $T(u, v) = F(u)G(v)$. The order of the functions implies that G is applied after F , thus $T(u, v)$ is said to be *two-pass transformable*. The functions F and G are known as the *two-pass functions*, each operating along a different axes. This approach enables the forward mapping of equation 6.1 to be rewritten as two one-dimensional mappings, F along the horizontal axis and G along the vertical axis.

Two pass algorithms have been shown to apply to a wide class of transformations [6.8] of general interest including perspective projection of rectangles and bivariate patches.

Two-Pass Mesh Warping

An important class of the two-pass warp algorithm is defined in terms of piecewise continuous mapping functions. The input and output images can be partitioned into a mesh of patches.

Each patch delimits the area over which the continuous mapping function applies. The mapping operation is reduced to transforming each patch onto its counterpart in the second image, thus giving rise to the term *mesh warping*. The term *morphing* (short for geometric metamorphosis) has been coined to describe a particular application of this method used by Industrial Light and Magic [6.9] to create the extra special effects seen in popular films such as *The Abyss* and *Terminator-2*. Figure 6.2 shows an example of a non-uniform mesh of patches with the control point placed at the vertices. The control points are indexed by integer (u,v) coordinates and serve as pointers to their real position. In effect this is a parametric grid that partitions the image into a contiguous set of patches. Each patch can be fitted with a bivariate function to produce a piecewise continuous mapping function.

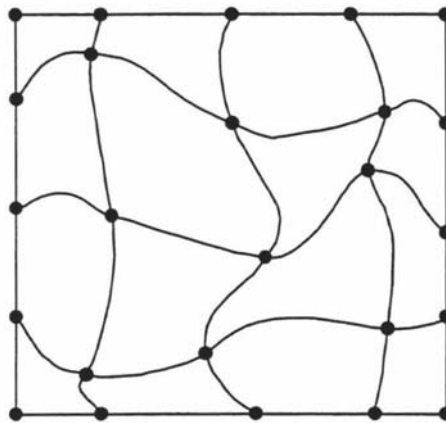


Figure 6.2: A mesh of patches

The algorithm [6.10] accepts a source image and two two-dimensional arrays of mesh coordinates. The first array contains the control points for the source image while the second array specifies the corresponding coordinates in the destination image. These two arrays must be the same size to establish a one-to-one correspondence but the points are real-valued numbers and can lie anywhere in the image plane. The only requirement of these arrays is that they be topologically equivalent. That is to say, there must be no folding or discontinuities and hence no self-intersection. Figure 6.3 shows an example where a regularly-spaced grid has been used for the source mesh array and the non-uniform patches of figure 6.2 used as the destination mesh array. Each (u,v) point in the source array is mapped to the corresponding (x,y) point in the destination array and the direction of the warp indicated by the arrows. A complete description of the algorithm can be found in Wolberg [6.11]. Below is a brief outline of the steps of this two-pass, forward mesh warp algorithm.

Horizontal Warp, Phase 1:

Take each horizontal line of the input and output meshes and interpolate them onto the input image base coordinates. This produces the input and intermediate image tables for the y -intercepts of horizontal scanlines.

Horizontal Warp, Phase 2:

Resample each horizontal scanline of the input image based on the intermediate and input tables to produce an intermediate image.

Vertical Warp, Phase 1:

Take each vertical line of the input and output meshes and interpolate them onto the input image base coordinates. This produces the intermediate and output image tables for the x -intercepts of vertical scanlines.

Vertical Warp, Phase 2:

Resample each vertical scanline of the intermediate image based on the intermediate and output tables to produce the output image.

This algorithm is fast and produces anti-aliased images over a wide range of input-output scale changes. Linear interpolation is used for magnification and box filtering for minification. A cubic interpolator is used in the first phase of each pass to regularise the input and output mesh grids onto the input image base coordinates.

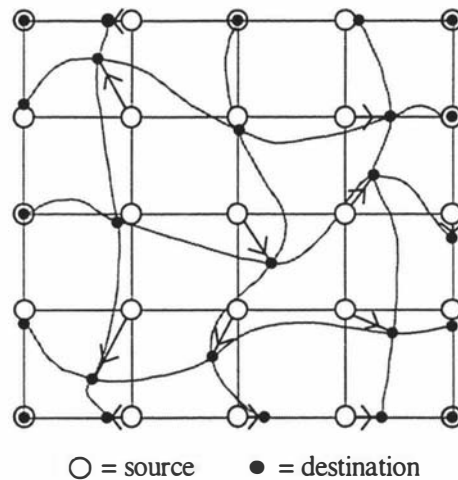


Figure 6.3: Example of source and destination arrays

At the time of this research (November 1993) there were no readily available commercial image-warping software packages. Based on C code-fragments found in the book *Digital Image Warping* [6.11], a Pascal language unit called *WarpFtns* was written to implement the complete warp algorithm. This code can be found in its entirety in appendix C starting at page C-52.

6.4.2 Image Rotation

Translating an object and changing its scale digitally is a much easier task than rotating the object. In order to produce a prescribed motion sequence the creation algorithm must be able to perform translation, scaling and rotation simultaneously. Before looking in detail at the task of

creating a motion sequence using two-pass mesh warping, the issue of how accurately this type of algorithm can perform object rotation must be investigated.

Many simple spatial transformations can be expressed in terms of a general 3x3 transformation matrix T shown in equation 6.4. For two-dimensional image projections (eg. mapping between the uv - and xy -coordinate system) the third dimension can be ignored without loss of generality.

$$[x', y', z'] = [u, v, z] T \quad \text{eq. 6.4}$$

where

$$T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

When this general matrix is used to specify a two-dimensional coordinate transformation it operates in what is known as the *homogeneous coordinate system* [6.12]. The use of homogenous coordinates was introduced to provide a consistent representation for *affine* and perspective transformations. The general representation of an *affine transformation* is:

$$[x, y, 1] = [u, v, 1] \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \quad \text{eq. 6.5}$$

This is a special case of the 3x3 transformation in which the homogeneous coordinate z' has been set such that $z' = z = 1$. An affine mapping corresponds to an *orthographic* or *parallel plane projection* from the source uv -plane to the destination xy -plane. It will accommodate only simple planar mappings such as translation, rotation, scaling and shearing. Each of these operation can be seen by partitioning the affine matrix. Translation by offsets T_u and T_v to u and v coordinates respectively takes the form

$$[x, y, 1] = [u, v, 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_u & T_v & 1 \end{bmatrix} \quad \text{eq. 6.6}$$

Similarly scaling u and v by arbitrary factors S_u and S_v respectively is specified by

$$[x, y, 1] = [u, v, 1] \begin{bmatrix} S_u & 0 & 0 \\ 0 & S_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{eq. 6.7}$$

Rotation about the origin anti-clockwise through an angle θ is given by

$$[x, y, 1] = [u, v, 1] \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{eq. 6.8}$$

Finally, a shear along the u -axis by a factor H_u and the v -axis by a factor H_v is defined by

$$[x, y, 1] = [u, v, 1] \begin{bmatrix} 1 & H_u & 0 \\ H_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{eq. 6.9}$$

A composite transformation consisting of any combination of these four basic transforms can be computed by forming the matrix product of the individual matrices. Shown below in equation 6.10 is an example of a composite transform consisting of rotation followed by a scale change and finally translation. Because matrix multiplication is generally non-commutative, changing the order of the transforms will result in a different composite transformation matrix.

$$[x, y, 1] = [u, v, 1] \mathbf{M}_{Comp} \quad \text{eq. 6.10}$$

$$\mathbf{M}_{Comp} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_u & 0 & 0 \\ 0 & S_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_u & T_v & 1 \end{bmatrix}$$

where

$$= \begin{bmatrix} S_u \cos \theta & S_v \sin \theta & 0 \\ -S_v \sin \theta & S_u \cos \theta & 0 \\ T_u & T_v & 1 \end{bmatrix}$$

As stated in section 6.4.1, the only restriction on the form of the warp that can be created using the two-pass mesh algorithm is that there can be no self-intersection of the mesh coordinates. Hence all transformations realisable by the general *affine mapping* should be possible. The only obvious limit on this statement concerns the maximum angle of rotation achievable. For angles greater than $\pm 45^\circ$, mesh self-intersection will tend to occur thus limiting the useful rotation range to this value. This does not present any problems with respect to the creation of cineradiographic sequences of the human spine because in general the vertebrae do not exceed this range of rotation.

Rotation Evaluation

To assess the quality of image rotation produced by a two-pass mesh warp, an algorithm designed specifically to perform image rotation is required. One of the most significant algorithm developments in image rotation was proposed recently by two independent researchers

[6.13, 6.14]. They demonstrated that the rotation transform could be decomposed into three one-dimensional shear transforms. This decomposition is shown in equation 6.4.

$$\begin{aligned}
 R &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 0 \\ -\tan(\theta/2) & 1 \end{bmatrix} \begin{bmatrix} 1 & \sin \theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\tan(\theta/2) & 1 \end{bmatrix}
 \end{aligned}
 \tag{eq. 6.11}$$

The first shear skews the image in a horizontal direction by displacing each row. The resultant image is then skewed in the vertical direction and the last part finally performs another skew in the horizontal direction to complete the rotation.

The main advantage of this algorithm is that there are no scaling operations. By not requiring any scaling, sampling and filtering complications, and their associated degradations, are avoided. Furthermore, a shear transformation can be performed efficiently requiring little more than linear interpolation. It can also be shown that the sum of the pixel intensities along any scanline remains unchanged after a shear operation, thus no visible spatial-variant artefacts are produced by this algorithm. Code for this algorithm can be found in appendix C page C-49 in a Pascal language unit called *Rotate*.

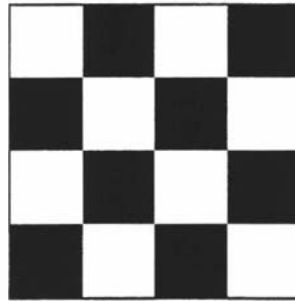


Figure 6.4: Checker-board test image

One of the most popular test images for assessing the performance of an implementation of a two dimensional transform is the checker board. This is because the image contains very high spatial frequency terms due to the binary intensity transitions. These clearly-bounded transitions provide an effective test for an algorithm since excessive smoothing will appear as blurring of the boundaries and aliasing (due to under-sampling or poor reconstruction) will appear as jaggedness in the boundaries. Figure 6.4 contains the checker-board test image used to evaluate the rotation algorithms. The squares are size 16 pixels, this being approximately the height of the smallest vertebra in the cervical spine.

Table 6.1 contains the results of rotating the checker-board test image through a range of angles up to 40°. The images in the third column were produced using a popular algorithm based on

bicubic interpolation [6.15], for additional comparison. Mesh warp rotation was performed using a grid resolution equal to the pixel size in the original checker-board image.












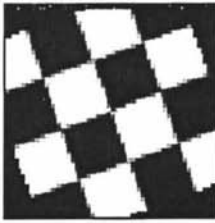





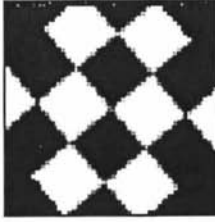
Rotation Angle	Three-pass shear	Two-pass Mesh Warp	Bicubic Interpolation
5 degrees			
10 degrees			
15 degrees			
20 degrees			
30 degrees			
40 degrees			

Table 6.1: Rotation of checker-board test image

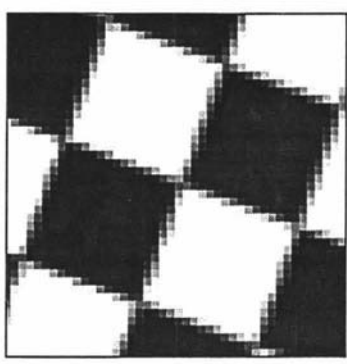
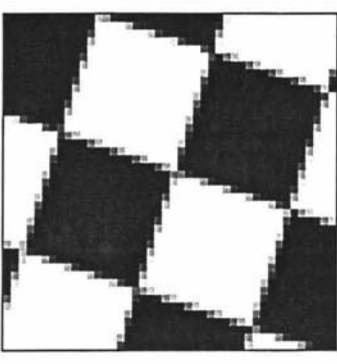
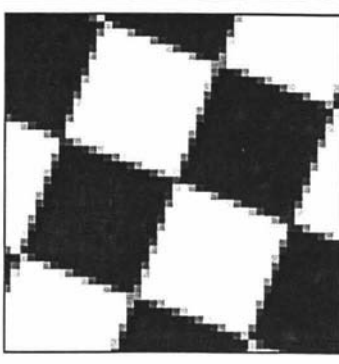
Three-pass shear	Two-pass Mesh Warp	Bicubic Interpolation
		

Table 6.2: 4x magnified 20° rotation of checker-board test image section

Visually there is little difference between the quality of the images generated by all three methods. The images show good maintenance of the checker-board boundaries, free from excessive jaggedness. Table 6.2 contains the centre section of the three images for the 20° rotation case magnified by 4. Even at this higher magnification there is very little visual differences between the three methods, all show good reconstruction of the boundaries.

The results of table 6.1 and 6.2 clearly show that the general two-pass mesh warp algorithm is capable of image rotation with an accuracy comparable to purpose build rotation algorithms and thus is suitable for the creation of prescribed motion cineradiographic sequences.

6.5 Motion Sequence Generation

In principle, motion sequence generation using the digital image warping offers the ultimate in flexibility in the specification of the motion characteristics and hence special effects. Using the basic mesh warping algorithm described in section 6.3.1 a wide range of planar motion transformations can be generated. The illusion of full three-dimensionally is also possible if range information is available. The start of the creation of a cinematic illusion using image warping normally begins with the separate recording of the two (or more) sequences. The first sequence is the source or start sequence and the last sequence is the target or destination sequence. The warp will form a new sequence in which the first frame is the same as the first frame of the start sequence and the last frame is the same as the last frame of the last sequence. The intermediate frames will be a spatial and colour combination of the intermediate frame of each sequence. For complex scenes each sequence is often recorded on a plain background (usually chroma-tone blue) to prevent scene elements other than the objects of interest from interfering with the each other during the warp. The final background can then be merged in last of all. The remaining steps required to achieve the warp can be illustrated with the simple case where there is just one source frame and one destination frame. The popular case of a picture of

one human face morphing to another face is a typical example. In each frame points of correspondence are marked, that is locations in the source frame that will become the new location in the destination frame. Once this has been done geometric mesh grids for the source and destination frames are formed from the correspondence points. A sequence of intermediate meshes is then formed that starts from the source mesh and ends with the destination mesh. To generate each frame of the sequence both the source frame and the destination frames are warped based on the intermediate mesh data. For the source frame the warp is a forward warp in which its original mesh is the input mesh and the intermediate mesh is the output mesh. In the destination frame case the intermediate mesh becomes the input mesh and the original mesh is the output mesh producing a reverse warp. These two intermediate frames are then combined using a method called *cross-dissolve* in which each frame is weighted by a value dependent on how far through the sequence the frame is and the two frames added together. For colour sequences the process is applied three times, once to each of the red, green and blue image planes. The main steps of sequence generation are thus summarised as:

1. Define correspondence points in the source and destination frames and form a geometric mesh grid from the points for each frames.
2. Generate an intermediate mesh grid interpolated between the source and destination grids.
3. Warp the source frame towards the destination image and the destination frame towards the source frame based on the intermediate mesh grid.
4. Cross-dissolve the warp source and destination frames to produce the a new intermediate frame.
5. Repeat steps 2 to 4 to produce all subsequent frames of the sequence.

Simple animation warping programs commonly automate the mesh generation by using linear interpolation between the input image mesh and the final output image mesh, over the specified number of frames. Sophisticated animation warping programs such as those used by Industrial Light and Magic [6.9], enable the partitioning of the input mesh and the definition of functions that specify how the partitioned areas move towards the final output mesh over the desired number of frames. This increased complexity in mesh specification enables more visually realistic warp sequences to be created since the rate of metamorphosis of small and extensive detail can be directly controlled.

6.5.1 Extended Image Warping

Image warping is normally used to create a visual illusion in which a source image and destination image geometrically distorts towards each other as the sequence progresses. The only requirement of the warp is that it is aesthetically pleasing. In this sense the sequence must appear to be visually smooth and certain key components such as eyes, mouth and nose of a face

must be clearly recognisable through the warp and not dissolve into a blur of the source and destination frames. The goal of the application of image warping to cineradiographic images of the human spine is to produce an animation in which the components of the spine move through a prescribed path of motion in accordance with physical constraints

Application to Cineradiographic Sequences

In order to produce a realistic animated cineradiographic sequence of the human spine, general constraints on the spatial and temporal motion characteristics need to be known. At a simplistic level the most obvious constraint on the allowable motion is that the vertebra must behave as rigid bodies conserving their geometric shape. Soft tissue surrounding the vertebrae must therefore mould or distort to accommodate the movement of the vertebrae. Digital image warping can readily achieve this since sections of the mesh grids can be made to keep their relative shape while the surrounding areas deform (stretch or compress) to account for the new locations of the fixed sections.

When a vertebra (the fixed sections of the mesh grids) is moved, the local geometric distortion will be propagated away from the vertebrae into the surroundings as if the surrounding was made of an elastic material that obeyed natural physical laws. The soft tissue between the vertebrae (the discs) has very different mechanical properties to the general soft tissue (skin and muscle, etc.). Thus constraints must be applied to ensure that compression and shear on the disc tissue is within known mechanical limits. An additional constraint can be used to simplify the problem when applied to *lateral* and *anteriposterior* (AP) views of the human spine. Vertebra in these views are generally non-occluding, allowing only close contact interaction.

As stated previously in section 6.5, image warping requires a source and a target frame and the definition of points of correspondence defining object of interest. To create complex warps containing many objects of interest each object is usually recorded separately or is separated from a frame and then processed individually. The difficulty in applying this technique directly to cineradiographic images is that the separation of each component, the skull, vertebrae and soft tissue, is not readily possible. Radiographing a skeleton could be used to specify the hard tissue, but the layering effect of the density projection of both soft and hard tissue would be difficult to achieve. Cineradiographic images of the spine are in effect only available as two-dimensional projections of the complete structure. A useful animated cineradiographic sequence could be produced from a single frame in a real sequence if a detailed spatial description of the main components (skull, vertebrae, soft tissue boundaries) is available. From these spatial descriptors, all objects would then have to be processed collectively and all spatial interactions adequately accounted for.

The manual placement of landmark points on static radiographs is routine when making simple biomechanical measurements. This same manual approach can be applied digitally to a single frame from a cineradiographic sequence to define the corner points of the vertebrae and other important features. Interpolating between these corner points yields an outline or boundary description of the object enclosed within. These outline coordinates can then be used to form a suitable spatial description of each object. Alternatively the Canny edge detection algorithm described in chapter 3 could be used to generate an initial edge map. Although this map will be noisy and the vertebral boundaries discontinuous, it can be readily tidied up by overlaying it on the original image and manually editing each pixel. The resulting edge image can then be chain coded [6.16] to produce a spatial representation of the vertebra and other boundaries.

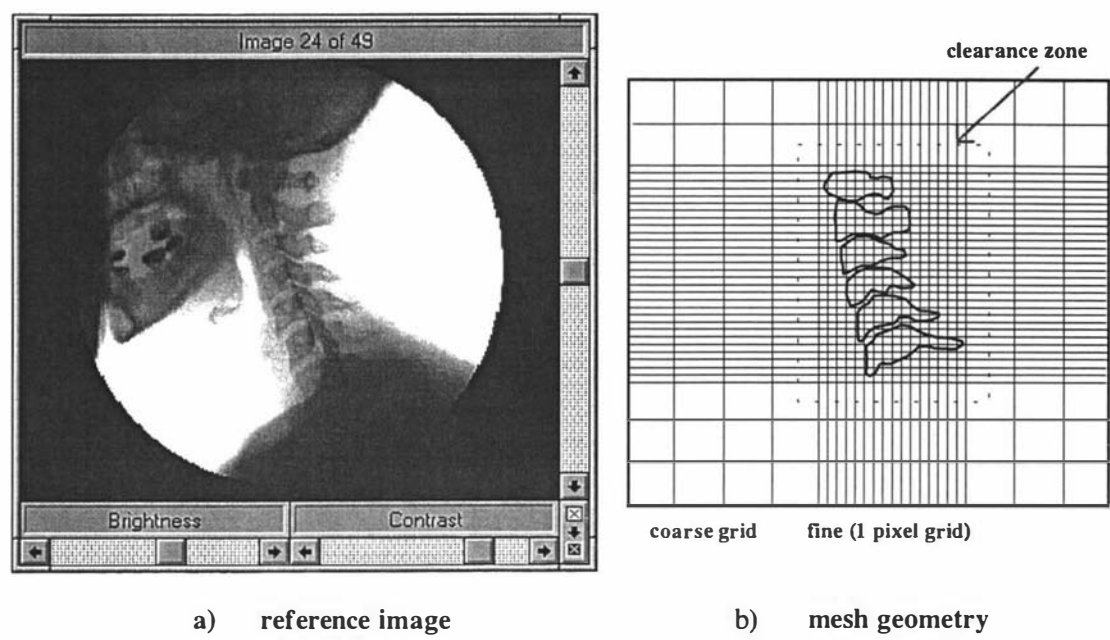
6.5.2 Initial Implementation

A very simplistic approach was used for the initial implementation of the warp-based animation algorithm. A trajectory file containing x - y position and orientation of each vertebra was predefined to prescribe the motion. The steps of this implementation are summarised as follows:

- Outline each vertebra (object) of interest on the reference image
- From the outline compute the minimum enclosing rectangle of each object
- Use a 1-pixel resolution grid in the object area and a coarse grid outside. Include a clearance zone around each object (see figure 6.5 b)
- From a prescribed trajectory file (x, y, θ) compute the new output mesh coordinates of each object. All internal coordinates of the enclosing rectangle are transformed, point by point using a general coordinate transformation
- Warp the image using the computed mesh grids
- Get the next trajectory data, compute new mesh coordinates, and repeat the process

This initial implementation has a number of significant limitations. Firstly the use of a minimum enclosing rectangle for the vertebral outlines results in motion boundary artefacts. Also due to the irregular shape of the vertebrae, a minimum enclosing rectangle outline description severely limited the range of motion that can be generated without having to explicitly deal with object interference. It was also found that for some trajectory values the output images appeared as if they have been stretched to all four corners and folded back on themselves. This was initially considered to be a bug in the mesh generation routine. However a thorough investigation revealed that the sudden transition from the fine single-pixel grid in the object area to the coarse grid in the surroundings caused the cubic spline interpolator used in the first phase of each pass of the warp algorithm, to become unstable and *overshoot*. This overshoot resulted in the unexpected special effects. This problem can be easily illustrated for the 1-D case in which

irregularly sampled data is regularised to a constant sampling interval by cubic spline interpolation.



middle section significant undershoot occurs. This behaviour is the classic result of severely undersampling the original data for which the only valid solution is to increase the number of samples. In the case of the initial implementation of the cine-warp program reducing the coarse grid spacing below 10 pixels stabilised the interpolation in the region outside the clearance zone.

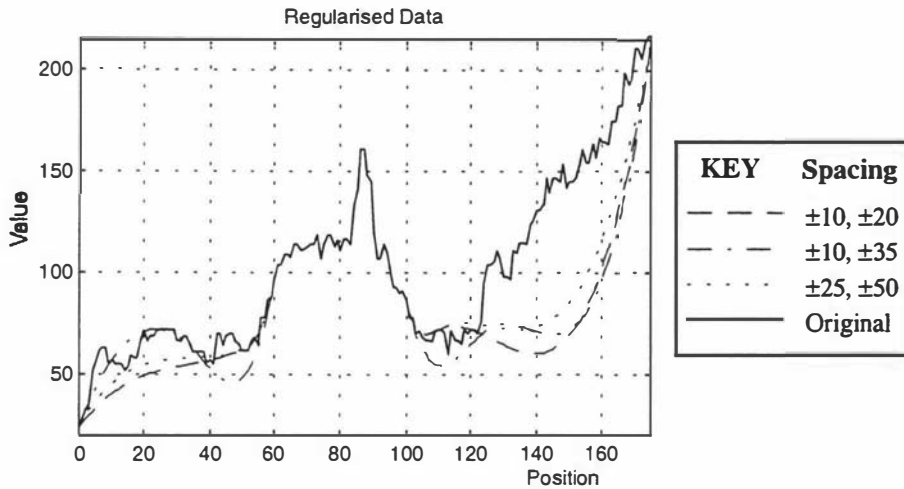


Figure 6.7: Regularised data by cubic spline interpolation

The use of an overall clearance zone, computed from the maximum pixel shift of any object in the scene, was a simple step designed to deal with propagating local distortion due to object motion, into the surroundings. The assumption was that the interpolation inherent in the warping routine would adequately propagate the distortion due to the fine (1 pixel grid) to coarse mesh grid transition. This assumption holds adequately for a small range of motion but with increasing translation the distance over which the distortion is propagated varies markedly between vertebra. This results in the spacing between the last position on a vertebrae and the first position outside the clearance zone varying markedly. This change in an interval spacing on a line-by-line basis results in local regions of the mesh still severely overshooting after regularisation.

The other severe limitation of this implementation is due to the use of a rectangular outline to describe each object. This limits the useful rotation to only a few degrees with a fine mesh spacing of 1-pixel, before mesh self-intersection occurs. Making the fine grid spacing larger does enable a wider range of motion to be prescribed but the accuracy of image rotation in particular is reduced due to the more coarsely defined mesh co-ordinates. The use of a rectangular outline in layered warping does not cause any problems since each layer contains only the object on a plain background. Boundary artefacts are not visible in each layer since the background is homogeneous in value. When the object layers are combined, cross interactions are dealt with by either merging or overlaying. This emphasises the fact that warping is usually used to generate cinematic illusions, not create accurately prescribed motion sequences.

6.5.3 Enhanced Implementation

In principle most of the limitations of this first implementation are eliminated by the use the object outline as the natural boundary and automating the generation of the entire output mesh co-ordinates. The following programme routines perform the required functions.

FormObjects:

Take each defined object and linearly interpolate an outline from its edge points. From these outlines an object description is created in terms of the start and end co-ordinates of both horizontal and vertical scanline segments.

CollectScanlineSegments:

Scan all object descriptions and collect all horizontal scanline segments existing on a common y-coordinate and vertical scanline segments on a common x-coordinate. Sort into order to produce the horizontal and vertical mesh interference arrays.

InitialiseOutputGrid:

Initialise output mesh grid to a regular one-pixel spacing.

PlaceObjects:

Using the trajectory data (x, y, θ) adjust the mesh coordinates of each object (defined by the scanline segment coordinates) to reflect the new position and orientation.

PropagateObjectDistortion:

Propagate the distortion (using interpolation) due to object movement into the surrounding area. Try to propagate back over the distance of the local mesh coordinate displacement plus ObjectClearance, otherwise propagate back to the nearest object.

CineWarp (Main procedure)

Read the reference image and object edge-point data

Read the trajectory data for each object

FormObjects

CollectScanlineSegments

WarpSequence (Number of images)

InitialiseOutputMesh

PlaceObjects

PropagateObjectDistortion

MeshWarpImage

The last part of the algorithm uses the original *MeshWarpImage* function to perform the actual warp. However, because the input mesh grid is already on a regular one-pixel spacing, the regularisation of input mesh performed in the first phase of both the vertical and horizontal passes, is not actually necessary. Modifying the warping function to detect this significantly

speeds up the execution time per image warp. Also since the *PropagateObjectDistortion* function is already having to perform much of the mesh regularisation, the mesh warp routine only has to perform image scanline resampling. This results in a further speed-up in warp execution.

The most complex part of the entire *CineWarp* algorithm is the generation of the output mesh with the coordinates adjusted to propagate the distortion produced by the motion of each object. Two functions are integral to this operation. The first function is *CollectScanlineSegments*, this produces a global spatial representation for all potential interfering horizontal and vertical scanline segments (*ie.* segments on same x or y coordinate) across all objects. It is from this representation that the second function *PropagateObjectDistortion*, is able to deal with the local spatial distortion produced by each moving object.

\rightarrow (x,y) \downarrow	29	30	31	32	33	34	35	36	37	38
26	(29.0,26.0)	(30.0,26.0)	(31.0,26.0)	(32.0,26.0)	(33.0,26.0)	(34.0,26.0)	(35.0,26.0)	(36.0,26.0)	(37.0,26.0)	(38.0,26.0)
27	(29.0,27.0)	(30.0,27.0)	(31.0,27.0)	(32.0,27.0)	(33.0,27.0)	(34.0,27.0)	(35.0,27.0)	(36.0,27.0)	(37.0,27.0)	(38.0,27.0)
28	(29.0,28.0)	(30.0,28.0)	(31.0,28.0)	(32.0,28.0)	(33.0,28.0)	(34.0,28.0)	(35.0,28.0)	(36.0,28.0)	(37.0,28.0)	(38.0,28.0)
29	(29.0,29.0)	(30.0,29.0)	(31.0,29.0)	(32.0,29.0)	(34.5,30.5)	(35.5,30.5)	(35.0,29.0)	(36.0,29.0)	(37.0,29.0)	(38.0,29.0)
30	(29.0,30.0)	(30.0,30.0)	(31.0,30.0)	(32.0,30.0)	(34.5,31.5)	(35.5,31.5)	(36.5,31.5)	(36.0,30.0)	(37.0,30.0)	(38.0,30.0)
31	(29.0,31.0)	(30.0,31.0)	(31.0,31.0)	(33.5,33.5)	(34.5,31.5)	(35.5,32.5)	(36.5,32.5)	(36.0,31.0)	(37.0,31.0)	(38.0,31.0)
32	(29.0,32.0)	(30.0,32.0)	(31.0,32.0)	(33.5,34.5)	(34.5,32.5)	(35.5,33.5)	(36.5,33.5)	(36.0,32.0)	(37.0,32.0)	(38.0,32.0)
33	(29.0,33.0)	(30.0,33.0)	(31.0,33.0)	(32.0,33.0)	(33.0,33.0)	(35.5,34.5)	(36.5,34.5)	(36.0,33.0)	(37.0,33.0)	(38.0,33.0)
34	(29.0,34.0)	(30.0,34.0)	(31.0,34.0)	(32.0,34.0)	(33.0,34.0)	(34.0,34.0)	(35.0,34.0)	(36.0,34.0)	(37.0,34.0)	(38.0,34.0)
35	(29.0,35.0)	(30.0,35.0)	(31.0,35.0)	(32.0,35.0)	(33.0,35.0)	(34.0,35.0)	(35.0,35.0)	(36.0,35.0)	(37.0,35.0)	(38.0,35.0)
36	(29.0,36.0)	(30.0,36.0)	(31.0,36.0)	(32.0,36.0)	(33.0,36.0)	(34.0,36.0)	(35.0,36.0)	(36.0,36.0)	(37.0,36.0)	(38.0,36.0)

Table 6.3: Mesh coordinates after object placement (x, y, θ) = (1.5, 1.5, 0)

Table 6.3 shows part of the output mesh for a small irregular object (indicted by the black outline) just after the *PlaceObjects* function has adjusted the mesh coordinates in accordance with the trajectory data for the current frame. In this case the motion is a translation of 1.5 pixels in both the x and y direction with no rotation. This is easily recognised by scanning across in the x -direction or down in the y -direction from the top left-hand corner. If the coordinates do not increase incrementally in unit steps in the direction of the scan then motion has occurred. Scanning across grid line $y = 29$ a sudden change occurs at $x = 33$ where the x -coordinate jumps from 32.0 to 34.5 and the y -coordinate jumps from 28.0 to 30.5. The normal change should only be one unit, thus this object start coordinate has translated 1.5 pixels in both directions. Continuing down or across this the mesh from this position, the grid resumes the normal unit

pixel change until the end of the object scanline. At this location there is a negative 0.5 unit change in both directions after which the unit pixel spacing is resumed once more.

The *PropagateObjectDistortion* function begins at the top of the list of (potentially) interfering horizontal scanline segments. If there is only one segment in this top entry then no interference can occur and hence the function is free to propagate the distortion without interference. If there is more than one segment in the top entry, then the horizontal distance between the end of each segment and the start of the next segment is computed along with the horizontal displacement of these end points from their reference (original) mesh position. The signed horizontal distance of the end points from their reference position is a direct measure of the end point local horizontal displacement. If the sum of the signed displacement between two contiguous segments is less than the original segment end point separation then the space between the pair of segments has become compressed. If the sum is greater, then there has been an overall local expansion between the segments. This local expansion could be the result of compression in one segment direction and a greater expansion in the other segment direction, or expansion in both segment directions. Each segment is now considered in turn to determine the nature of the end-point translation. These results are then combined together and the mesh x -coordinates adjusted to propagate the distortion due to the inter-segment motion. Simple linear interpolation is used to propagate the motion distortion. This is equivalent to assuming that the inter-segment material is homogenous and perfectly elastic over the propagation distance.

The above procedure is repeated for all remaining segments in the top entry of the horizontal scanline interference array. Once all segments are processed the function moves to the next entry (on the next valid y -coordinate) and repeats the operation. With all the horizontal scanline interference array entries processed, the same procedure is repeated for the vertical scanline interference array moving in the y -direction.

At the end of this stage the x -coordinates along each horizontal line and the y -coordinates along each vertical line have been adjusted to account for the motion distortion. Test sequences generated with an implementation of *PropagateObjectDistortion* to this stage worked correctly for either horizontal or vertical translation and for small rotation angles, but with combined motion the object outlines began to lose their integrity in the vertical direction whenever expansion occurred. Because this effect appeared in one direction only it was initially assumed to be due to a coding error. However, a thorough check of the code showed that the mesh generation in both the horizontal and vertical direction were identical. Close inspection of the way in which the object outlines began to lose their integrity revealed that it was due to a shearing action. The vertical shear increased with horizontal movement, beginning from the corner where expansion was present. It was concluded that the reason for this effect was that the propagation of distortion had ignored the interaction between the horizontal and vertical

peripheral scanlines. Careful inspection of a printout of a section of the mesh around the object periphery eventually revealed this *edge-effect*. The reason it had not been readily apparent previously, was the confusion between mesh index coordinates and physical image coordinates. The mesh index coordinates do not represent the object coordinates, but the entries at the location do. For simple motion they are very similar because the initial mesh spacing is one also pixel. This shearing problem did not show up in the horizontal direction because of the scanline nature of the *MeshWarpImage* routine. Reversing the order of the horizontal and vertical passes of this routine shifted the shearing effect to the horizontal direction.

With this new insight revealed, the *PropagateObjectDistortion* function was extended to account for this effect. The extra code begins from the outer ends of each peripheral horizontal scanline segment. Based on the vertical displacement at this position the *x*-coordinate is maintained vertically spaced away from the object over the number of mesh elements equivalent to the local vertical displacement. The same procedure is performed on the vertical peripheral segments where the *y*-coordinate is maintained horizontally away from the object over the number of mesh elements equivalent to the local horizontal displacement. The effect of these two one-dimensional operations is to move the peripheral shear outside the physical region of the object.

\rightarrow (<i>x</i> , <i>y</i>) \downarrow	29	30	31	32	33	34	35	36	37	38
26	(29.0,26.0)	(30.0,26.0)	(31.0,26.0)	(32.0,26.0)	(33.0,26.0)	(34.0,26.0)	(35.0,26.0)	(36.0,26.0)	(37.0,26.0)	(38.0,26.0)
27	(29.0,27.0)	(30.0,27.0)	(31.0,27.0)	(32.0,27.0)	(34.5,27.5)	(35.5,27.5)	(36.0,27.0)	(36.0,27.0)	(37.0,27.0)	(38.0,27.0)
28	(29.0,28.0)	(30.0,28.0)	(31.0,28.0)	(33.5,28.0)	(34.5,29.0)	(35.5,29.0)	(36.0,28.5)	(36.0,28.0)	(37.0,28.0)	(38.0,28.0)
29	(29.0,29.0)	(30.0,29.0)	(31.5,30.5)	(33.5,30.5)	(34.5,30.5)	(35.5,30.5)	(36.0,30.5)	(36.5,30.5)	(37.0,29.0)	(38.0,29.0)
30	(29.0,30.0)	(30.0,31.0)	(31.5,31.5)	(33.5,31.5)	(34.5,31.5)	(35.5,31.5)	(36.5,31.5)	(37.0,31.5)	(37.5,31.5)	(38.0,30.0)
31	(29.0,31.0)	(30.5,32.5)	(32.0,32.5)	(33.5,32.5)	(34.5,32.5)	(35.5,32.5)	(36.5,32.5)	(37.0,32.5)	(37.5,32.5)	(38.0,31.0)
32	(29.0,32.0)	(30.5,33.5)	(32.0,33.5)	(33.5,33.5)	(34.5,33.5)	(35.5,33.5)	(36.5,33.5)	(37.0,33.5)	(37.5,33.5)	(38.0,32.0)
33	(29.0,33.0)	(30.0,33.0)	(31.0,34.5)	(33.5,34.5)	(34.5,34.5)	(35.5,34.5)	(36.5,34.5)	(37.0,34.5)	(37.5,34.5)	(38.0,33.0)
34	(29.0,34.0)	(30.0,34.0)	(31.0,34.0)	(33.5,34.5)	(34.5,34.5)	(35.5,35.0)	(36.5,35.0)	(37.0,34.0)	(37.0,34.0)	(38.0,34.0)
35	(29.0,35.0)	(30.0,35.0)	(31.0,35.0)	(32.0,35.0)	(33.0,35.0)	(35.5,35.5)	(36.5,35.5)	(36.0,35.0)	(37.0,35.0)	(38.0,35.0)
36	(29.0,36.0)	(30.0,36.0)	(31.0,36.0)	(32.0,36.0)	(33.0,36.0)	(34.0,36.0)	(35.0,36.0)	(36.0,36.0)	(37.0,36.0)	(38.0,36.0)

Table 6.4: Mesh coordinates after the final phase of distortion propagation
 $(x,y,\theta) = (1.5,1.5,0)$, *ObjectClearance* = 1

Table 6.4 shows the mesh coordinates of table 6.3 after distortion propagation has been completed. The extent of the distortion propagation is shown geometrically by the grey border. Note that this is in mesh index coordinates not the actual spatial coordinates of the image. To determine the physical propagation distance the mesh grid entries need to be inspected.

An additional user-defined parameter *ObjectClearance*, enables control of the distance over which motion distortion is propagated. The sum of this value and the actual local displacement is the physical propagation distance. This sum is in effect the distance over which the surrounding material behaves as an elastic medium.

Although this enhanced implementation removed the corner shearing effect local to the object it has done so by shifting it away from the object by a distance that is dependent on the object motion and the value of *ObjectClearance*. This causes problems when the source image contains many objects in close proximity since there is no space to shift the distortion into. Also when two objects in close proximity undergo a vastly different range of motion the regions of overlap and underlap are not adequately described by 1-dimensional interpolation. This is easily illustrated in figure 6.8 where due to horizontal translation to the right of the top vertebra relative to the bottom one, what was an overlap in one frame becomes an underlap in the next and vice-versa.

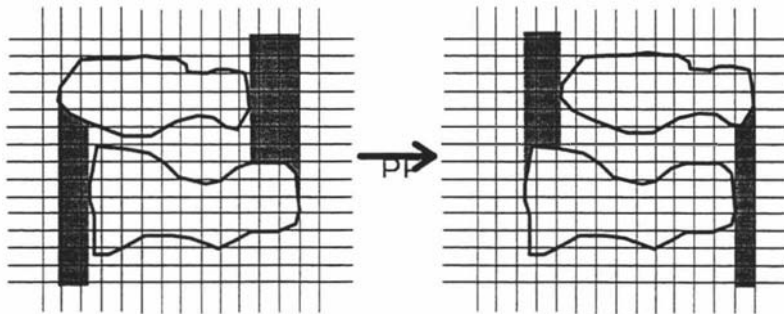


Figure 6.8: Vertebral overlap becomes underlap

6.5.4 Image Warping Reassessed

Although an affine mapping is separable and the prescribed motion of the vertebrae can be represented by an affine transform of the vertebral coordinates, the need for boundary conditions at the edge of the image means that distortion of the inter-vertebral regions is not defined by an affine transform but by a perspective transform. A perspective transform is not separable hence the limited effectiveness the adjustment of the mesh coordinates to account for object movement using 1-dimensional interpolation in the x and y directions. The effectiveness is further limited by the fact that the control points are sparsely defined and hence severely undersample the coordinates (see figures 6.6 and 6.7).

The image warping problem can be restated as;

given the initial coordinates of a set of control points on a reference image (defined by the vertebral outlines) and their new coordinates on a new image (defined by the prescribed motion) geometrically distort the base image so that it is aligned at the new coordinates.

The problem is, in effect, the same as the standard geometric correction or rectification problem where the control points are tessellated into a number geometric patches. Each patch can then be mapped to a new patch defined by the new coordinates of the control points. The mapping function can then be used to interpolate internal coordinates of each patch and hence enable the regularisation of the entire image mesh.

A Surface Fitting Paradigm for Geometric Correction

The problem of determining the forward mapping functions U and V can be conveniently posed as a surface fitting problem. Consider M control points labelled (x_k, y_k) in an observed image and (u_k, v_k) in the reference image, where $1 \leq k \leq M$. The mapping process is equivalent to determining two smooth surfaces: one that passes through points (x_k, y_k, u_k) and the other that passes through (x_k, y_k, v_k) , for $1 \leq k \leq M$. The simplest mapping is achieved when only three control points are used. In this situation the fitted surfaces will be a delineated by triangular regions. It can be shown [6.17] that fitting a linear interpolant, *ie.* a plane, enables the mapping of an arbitrary triangular region to another arbitrary triangle region in which the two regions are related by an affine mapping. Since the general *MeshWarpImage* algorithm is limited to defining affine warps, triangular regions will thus be adequate.

The equation of a plane through three points (x_1, y_1, u_1) , (x_2, y_2, u_2) and (x_3, y_3, u_3) is given by

$$A_u x + B_u y + C_u u + D_u = 0 \quad \text{eq. 6.12}$$

$$\text{where } A_u = \begin{vmatrix} y_1 & u_1 & 1 \\ y_2 & u_2 & 1 \\ y_3 & u_3 & 1 \end{vmatrix} \quad B_u = - \begin{vmatrix} x_1 & u_1 & 1 \\ x_2 & u_2 & 1 \\ x_3 & u_3 & 1 \end{vmatrix} \quad C_u = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D_u = - \begin{vmatrix} x_1 & y_1 & u_1 \\ x_2 & y_2 & u_2 \\ x_3 & y_3 & u_3 \end{vmatrix}$$

Similarly the second plane through the three points (x_1, y_1, v_1) , (x_2, y_2, v_2) and (x_3, y_3, v_3) is given by

$$A_v x + B_v y + C_v u + D_v = 0 \quad \text{eq. 6.13}$$

$$\text{where } A_v = \begin{vmatrix} y_1 & v_1 & 1 \\ y_2 & v_2 & 1 \\ y_3 & v_3 & 1 \end{vmatrix} \quad B_v = - \begin{vmatrix} x_1 & v_1 & 1 \\ x_2 & v_2 & 1 \\ x_3 & v_3 & 1 \end{vmatrix} \quad C_v = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D_v = - \begin{vmatrix} x_1 & y_1 & v_1 \\ x_2 & y_2 & v_2 \\ x_3 & y_3 & v_3 \end{vmatrix}$$

and the parallel lines '||' represent the determinant of the 3x3 matrices.

Once the surface coefficients A, B, C and D have been determined, equations 6.12 and 6.13 can be rearranged into equations 6.14 and 6.15 and used to interpolate intermediate values of u and v for any arbitrary x - y location in the triangular surface.

$$u = -(A_u x + B_u y + D_u) / C_u \quad \text{eq. 6.14}$$

$$v = -(A_v x + B_v y + D_v) / C_v \quad \text{eq. 6.15}$$

By choosing these x - y coordinates for all patches to lie on a regular grid, piecewise mesh grids representing the complete forward mapping functions U and V can be realised in a form suitable for the *MeshWarpImage* routine.

Triangulation is the processes of tessellating the convex hull of a set of N distinct points into triangular regions. Although many configurations are possible the most useful is one in which points inside the triangles are closer to their vertices than to the vertices of other triangles. This form of partitioning is called optimal triangulation and avoids the production of long triangles with sharp corners. Lawson [6.18] defined three criteria for optimality.

1. **Max-min criterion:** For each quadrilateral in a set of triangles, choose the triangulation that maximises the minimum interior angle of the two obtained triangles. This biases the partitioning against undesirable long thin triangles.
2. **The circle criterion:** For each quadrilateral in a set of triangles fit a circle through three of its vertices. If the fourth vertex does not lie within the circle, split the quadrilateral into two triangles by drawing a diagonal that does not pass through the fourth vertex.
3. **Thessian region criterion:** For each quadrilateral in a set of triangles, construct the Thessian region [6.19]. In computational geometry the Thessian region is the result of intersecting the perpendicular bisectors of the quadrilateral edges. This serves to ensure that regions around a control point are closer to the point than to other control points.

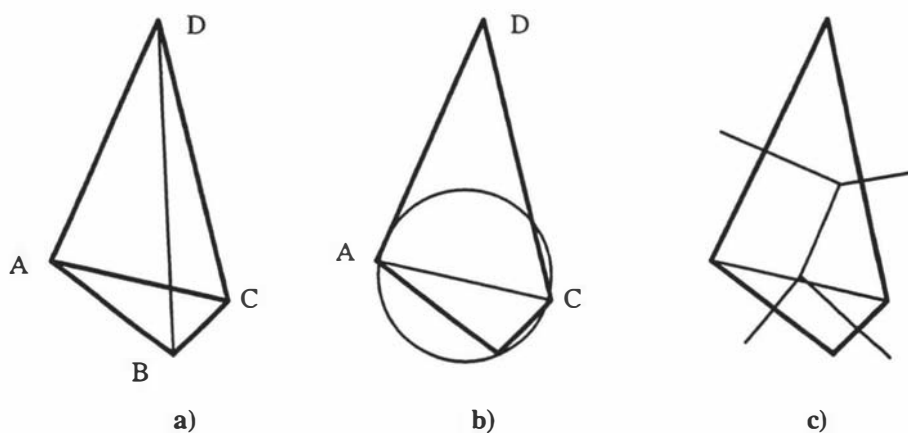


Figure 6.9: Application of the optimal triangulation criteria

Figure 6.9 contains examples of the application of these three criteria. In figure 6.9a, triangle ABC is chosen in preference of triangle BCD on application of the first criterion. Point D in figure 6.9b lies outside the fitted circle (application of the second criterion) and thus the diagonal

is not drawn through this point. While in figure 6.9c, the intersection of the edge bisectors shows that the interior region of each triangle is closer to its own vertices than to the vertices of the other triangle.

The only component remaining is the initial triangulation of the control points prior to optimisation using Lawson's criteria. This can be conveniently handled by the recursive algorithm of Lee [6.20] in which the control points are split into halves based alternately on their x and y coordinates. The splitting continues until each subset contains only three or four points which can then be easily triangulated using Lawson's three criteria. Each subset is then merged into larger subsets until all the triangular subsets are consumed. This results in a convex hull of triangles that must be extended in some way to the boundaries of the image. This can be performed by considering the triangles on the border of the hull and extending their size to the limit dictated by their intersection with other neighbouring triangles and the image borders.

6.5.5 Final Implementation

The final implementation of the warp-based animation routine uses a very efficient recursive Delaunay triangulation algorithm described by Watson [6.21]. This algorithm is very fast and can be applied equally to two and three dimensional data. Although the algorithm does not apply Lawson's optimality criteria directly, it does in effect use the second or circle criterion of Lawson. A revised list of the functions used in the final implementation is shown below.

CineWarp (Main procedure)

Read the reference image and object edge point data
Read the trajectory data for each object
FormControlPoints
Triangulate

WarpSequence (Number of images)

InitialiseOutputMesh
TransformTriPoints
FitTriangularPatch
InterpolateOutputMesh
SmoothOutputMesh
MeshWarpImage()

The function *FormControlPoints* extracts the edge points of each object and then adds boundary control points to the list. Because all the boundary points lie along the image border, after triangulation the resulting convex hull is the same as the image border. Thus a separate operation to extend the hull to the image border is not required. *Triangulate* performs the actual triangular tessellation of the control points and is a 2-D implementation of the Watson algorithm. With the triangulation completed the tessellation information is passed to the main *WarpSequence*

procedure to begin the animation. The output mesh grids are initialised and the trajectory data is then used by *TransformTriPoints* to adjust the output coordinates of each triangular patch. Based on these new coordinates of each patch, the linear coefficients for both the *x* and *y* triangular planes are computed by *FitTriangularPatch*. Next the function *InterpolateOutputMesh* uses the coefficients of each patch to regularise the output mesh grids ready for the *MeshWarpImage* routine.

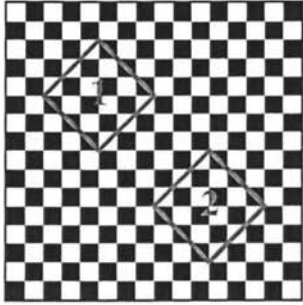
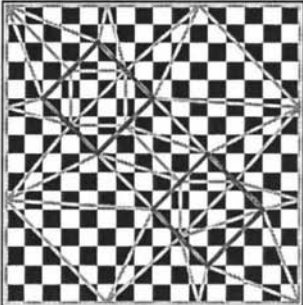
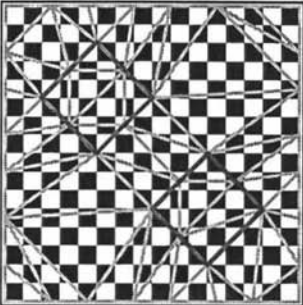
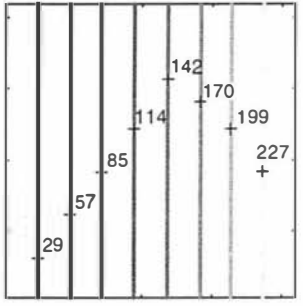
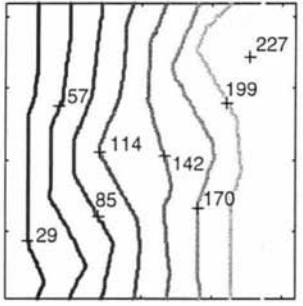
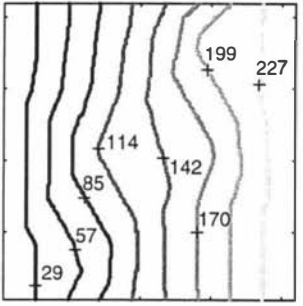
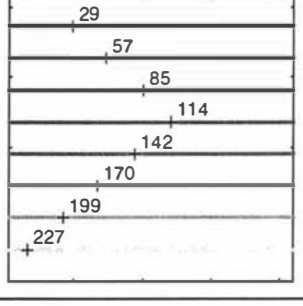
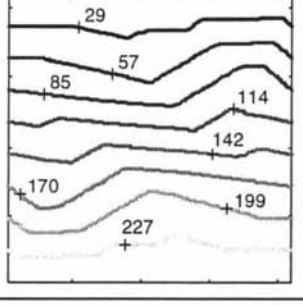
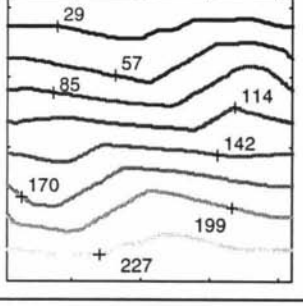
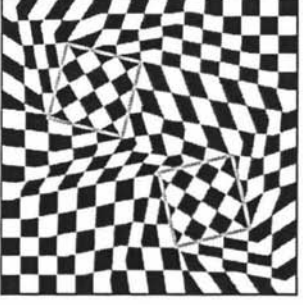
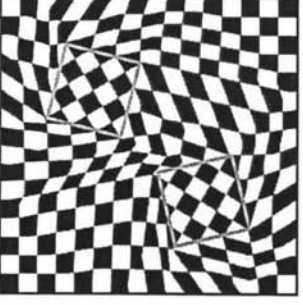
6.5	Original (two objects)	Four Boundary Points	Eight Boundary Points
Tri-Patches			
xMesh			
yMesh			
Forward Warp Images	Object trajectories $(x,y,\theta)_{\text{Object 1}} = (-8,8,30^{\circ})$ $(x,y,\theta)_{\text{Object 2}} = (8,-8,-30^{\circ})$		

Table 6.5: MeshWarp Stages

Mesh-smoothness problems [6.22] can occur with the above approach since each patch involves only a linear mapping and the patches can vary significantly in size. A simple solution adopted by the author to alleviate this problem was to smooth the mesh coordinates (using a 7x7 box filter) in areas outside the vertebrae. This ensures the vertebrae maintain their geometric integrity while boundaries between linear patches are concealed. The function *SmoothOutputMesh* performs this operation.

Table 6.5 illustrates many of the steps involved in the warp-based animation algorithm. The first entry in the first row is the reference image (a 16 pixel size checker-board) with two diamond-shaped objects defined by four vertex points and four mid-points. The next two entries in the row show the reference image with the tessellation resulting from triangulation superimposed. Increasing the number of boundary points per axis from four to eight increases the number of triangular patches overall, but not all of these new patches improve the spatial distribution. This is due to the formation of boundary patches where all three vertices are fixed at the boundary. The effect can be seen in the bottom-left and top-right corners of the tessellation and will produce poor spatial interpolation in these areas. In practice the objects contain many more outline points than is illustrated in this simple example and consequently very few boundary patches occur.

Rows two and three of the table 6.5 contain the x and y output mesh grids displayed as contour plots. There are eight equi-spaced contour lines and each line is labelled with its value and coded with a grey-scale intensity (black to white, smallest to largest). The first entry in each row is the grid with no object displacement, resulting in a linear plane with evenly spaced contours. The other two entries correspond to the top diamond moving with a trajectory $(x, y, \theta) = (-8, +8, 30^\circ)$ and the bottom diamond moving $(x, y, \theta) = (+8, -8, -30^\circ)$, where the displacements are in pixels. The contours for these entries clearly show the non-linear spacing of the x and y grid coordinates required for this simple mesh warp. Increasing the number of boundary points from four to eight improves inter-object coordinate sampling as is indicated by the gradual smoothing of the contour lines.

Figure 6.10 shows the tracking points dialogue from CineMetriX [6.23] software package incorporating some of the elements developed by the author in this chapter. The point group list to the right of the figure shows that the second vertebra of the cervical spine is selected. The point groups indicated by the '>' symbol represent the tri-patches formed automatically by triangulation. In this case 210 tri-patches were generated for the six defined vertebrae. The sixteen-corner boundary patches shown in the figure do not reduce spatial acuity since they lie outside the circle of the image intensifier.

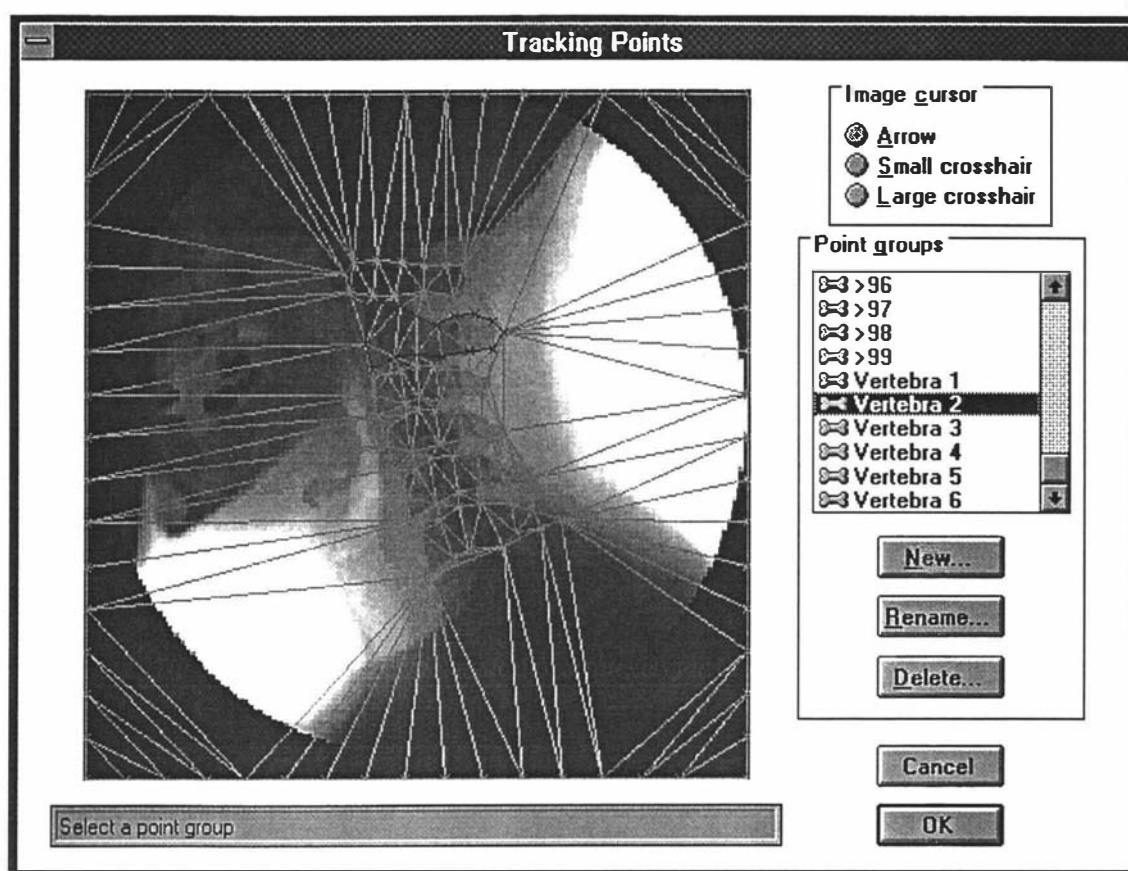


Figure 6.10: Track Points dialogue - cervical spine fully tessellated

Image Noise

One remaining component needs to be added to the warp-based animation routine in order to produce realistic sequences. Noise in a cineradiographic system was shown in chapter 2, section 2.3 to be limited by the smallest number of quanta (photon or electrons) utilised at any stage of the process. In practice there are two main sources of image noise. The first is due to X-ray scattering (see chapter 2, section 2.1.2) while the second results from the amplification process in the X-ray image intensifier tube. The amount of X-ray scattering for a given material is proportional to its thickness and absorption coefficient. The effective noise of an image intensifier tube is dependent on its design and overall amplification factor. However these noise contributions are effectively *frozen* in the warp based animation since it begins from just a single frame of a real sequence. Furthermore, the image resampling used in the animation process involves spatial filtering (box filtering for minification and linear interpolation for maxification) that tends to smooth this frozen noise in a velocity dependent manner. This smoothing effect is readily apparent when viewing one of the prescribed motion sequences. The reference frame which is usually the centre frame in a sequence is noticeably sharper but more noisy than the created frames. Thus in order to produce a realistic motion sequence using warping, appropriate noise needs to be *added* on a frame-by-frame basis.

In order to determine the characteristics of the noise present in a typical digitised cineradiographic sequence, two factors first need to be considered:

1. Video camera auto-gain effects
2. Area of interest, soft tissue and hard tissue

Both factors can be addressed by using a frames from cineradiographic sequence of the cervical spine. In imaging this part of the spine a very wide range of intensities are present and camera auto-gain effects are at a maximum to prevent overload. It is possible to find both soft tissue and hard tissue areas of reasonable homogeneity. Using ten frames from the middle of a sequence three regions were defined to sample the image noise. One of the frames with the three sample regions indicated is shown in figure 6.11.

The first region is near the circular edge of the intensifier and is part of one of the lead shutters of the *collimator*. This was used as a reference to ensure that camera auto-gain changes between frames did not significantly alter the recorded noise. Region two is at the back of the neck and encompasses a large area of relatively homogenous soft tissue. The third region is on the lower centre of the skull and was chosen to be representative of an area with the greatest thickness and highest absorption coefficient.

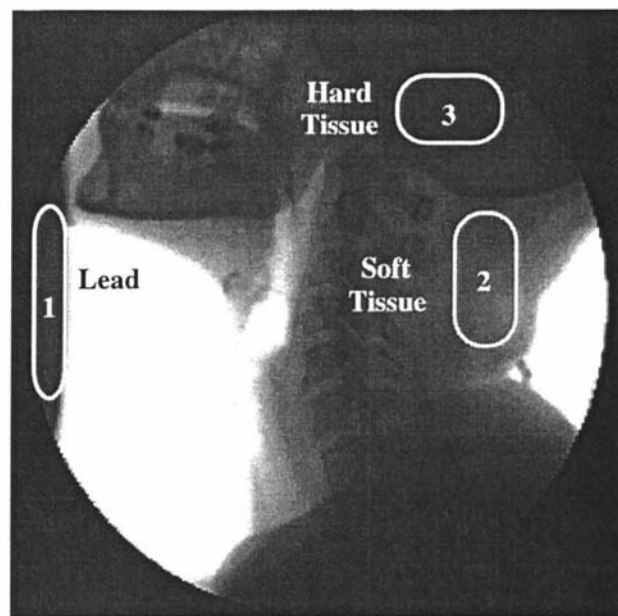


Figure 6.11: Image noise estimate regions

In order to remove the effects of any underlying structure in these three regions from the estimated noise, each frame was subtracted from a filtered version. Filtering involved the recursive application of a 7x7 box filter. By visually inspecting a magnified copy of these regions it was found that five passes of the filter adequately removed the image noise while

preserving any general underlying intensity trends. The differences in data for each region over the ten frames were then combined together and normalised histograms of their distributions generated.

Figure 6.12a through to 6.12c show the normalised distribution of the noise in the three regions with a least-squares best fit gaussian distribution superimposed (see equation 3.12 of chapter 3).

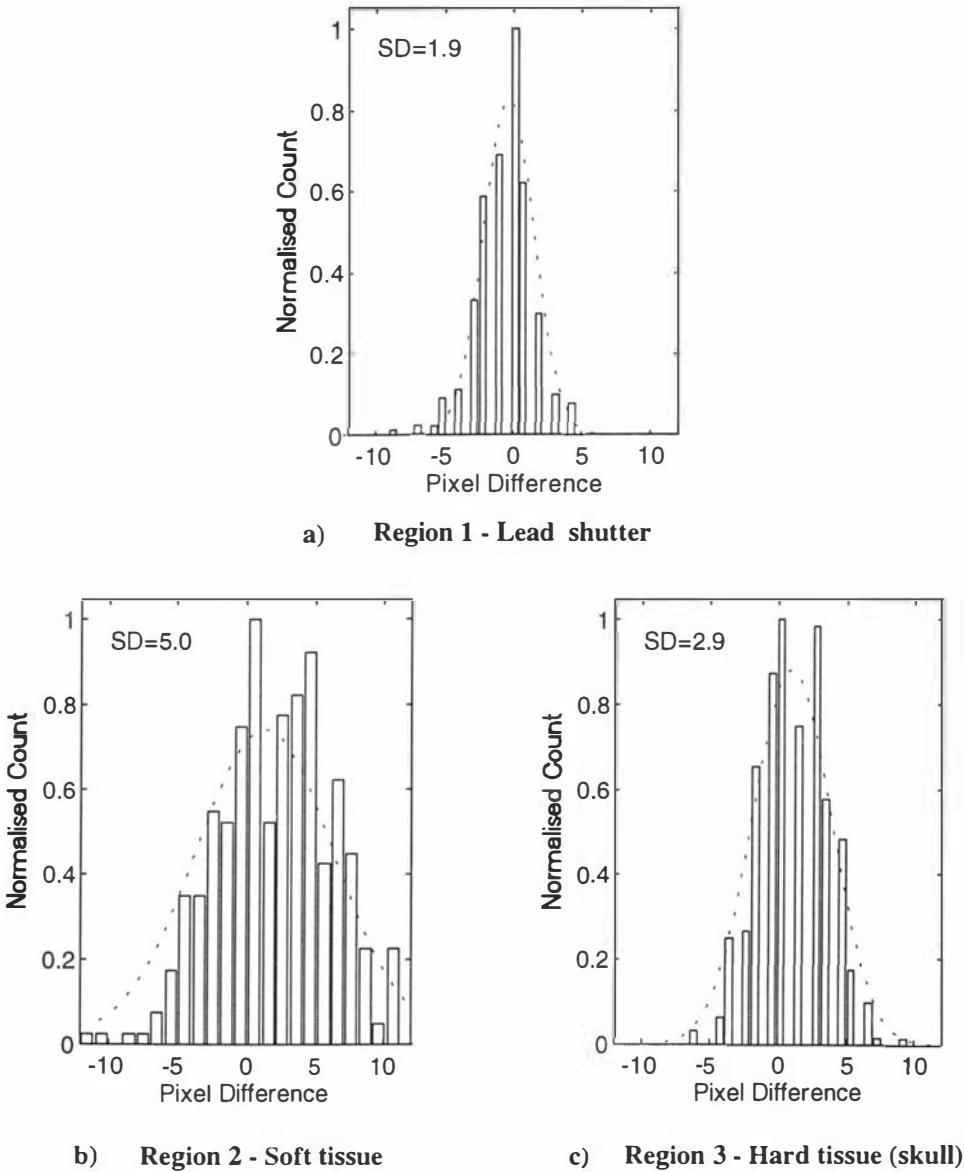


Figure 6.12: Normalised noise distribution

For the reference region (region 1) of figure 6.11a the fit of the gaussian is reasonably good with a standard deviation of 1.9. For the other two regions the gaussian fit is not particularly good, nevertheless the standard deviations do provide a comparative measure of noise between the regions. Region 2 (figure 6.11 b) containing the soft tissue at the back of the neck was the noisiest region with a standard deviation over twice that of region 1. This high noise in a low

absorption region is principally due to scattering in the layers of the soft tissue. The third region shows a noise level between those of regions 1 and 2 (see figure 6.11 c). Total absorption in this region is comparable to that of region 1 containing the lead collimator shutter, but due to the greater thickness of the skull, scattering is increased and hence noise is also increased.

Based on the noise estimate results above, the magnitude of the noise added to each frame of the warp sequence should vary on a pixel-by-pixel basis dependent on the recorded intensity and hence the total absorption. In the low-absorption area of the skin the noise should be greater than in a higher absorption area such as the skull. However in practice the region of interest is the vertebrae where absorption is moderate but significant scattering occurs due to the layers of soft tissue. Thus rather than make the standard deviation of the noise added to each pixel dependent in the intensity of the pixel, a simpler approach was taken in which noise was added to each frame drawn from a zero mean gaussian distribution with standard deviation of 4.0.

Complete Pascal source code for the final implementation of the warp-based animation routine can be found in appendix C page C-67. This code includes an additional routine for object collision detection and limiting. This routine is necessary since the warp based animation can not allow object cross-over and hence mesh intersection. Object collision detection is achieved by computing the intersection of each object outline for the given trajectory data. If any of the objects are found to collide in any frame of the prescribed sequence then the trajectory of the bottom colliding object is limited. The routine is then run recursively until no objects are found to collide at which stage the actual animation begins with the new trajectory.

6.6 Motion Test Sequences

With the method for prescribed motion sequence generation by digital image warping now described, attention is shifted to considering what constitutes a suitable test sequence for a motion-tracking algorithm for cineradiographic images of the human spine. In testing any motion-tracking algorithm it is desirable to start off with sequences that contain simple motion and then increases the complexity and speed of the motion. This approach enables incremental assessment of tracking performance and the setting of limits on the accuracy and reliability of a given method. The final prescribed motion should realise a level of complexity that is at least equivalent to that found in real-world sequences of the type for which the algorithm is designed.

In diagnostic cineradiographic images of the human spine translation is mainly constrained to the horizontal axis. The only exception to this is when considering pelvic alignment in the lumbar-sacral region. It was noted in section 5.1 that the maximum speed of rotation and translation usually occurs in the lateral cervical area where the neck moves from full flexion to full extension. This cycle of movement is normally completed in a little less than four seconds, in which the top vertebra (*atlas*, *axis* and skull) may under go a range of rotation from

approximately -45° to $+45^\circ$. At half video frame rate (12.5 frames per second) the mean interframe rotation will be constrained to less than two degrees. Interframe rotation of this magnitude was shown in section 5.3.2 to not significantly degrade the interframe match. However, this calculation assumes normal biomechanical operation. Under conditions of dysfunctional operation, where a vertebra may lock and an adjacent vertebra then hyper-extend (*-mobilise*) to compensate, interframe rotation may be much larger than two degrees.

Sequence length is important for incremental motion-tracking algorithms working to single pixel accuracy (\pm half a pixel). This is because the overall accuracy will tend to degrade as the sequence length becomes longer and the errors in the best match positions tend to accumulate. At half-video capture rate, four seconds of motion (equivalent to a single cervical cycle) will require approximately fifty frames. With pre-captured sequences the number of frames to track can be halved by starting in the middle of the sequence and tracking to both ends. Since only half the number of frames is traversed, error propagation should be substantially reduced.

Based on the preceding discussion, a suitable strategy for motion sequence generation would be to create sequences of the cervical spine in the following order:

1. Horizontal linear translation of individual vertebra (no adjacent vertebral interference).
2. Simple linear rotation of individual vertebra (no adjacent interference).
3. Extended length, multi-cycle sequences to access mis-match propagation effects.
4. A comprehensive sequence with full adjacent vertebral interference with typical range and type of motion.

Animation of the skull presents a slight problem with respect to the third point in this list. This is because the skull is large, has few *corner* points, and its outer edges often disappear from the field of view. One way of dealing with some of these difficulties is to segment the skull into a number of objects. These objects are then collectively animated such that their geometrical integrity is maintained.

The last point in the list of prescribed motion generation offers the ability to place limits on the fastest moving (complex motion) object that can be reliably tracked. Algorithms working to fixed spatial accuracy (eg. \pm half a pixel) will often introduce *jitter* [6.24] style noise into the data. Knowing the frequency limits (translation and rotation) of a given motion-tracking algorithm many enable the prefiltering of the data before analysis to reduce the effects of this noise.

Prescribed motion sequences created in accordance with the above philosophy will be utilised in the next chapter in the assessment of the motion-tracking algorithm described in early sections of that chapter.

6.7 References

- [6.1] **Duntta R., Manmatha R., Williams L. and Risman E.M.** *A Data Set For Quantitative Motion Analysis.* Proceedings IEEE CVPR, San Diego, pp 159..164, 1989.
- [6.2-3] **Fleet D.J.** *Measurement Of Image Velocity.* Kluwer Academic Publishers, pp 26..28, 1992.
- [6.4] **Overington I.** *Vision and Acquisition.* Pentech Press, London, 1976.
- [6.5] **Overington I.** *Computer Vision - A unified biologically-inspired approach.* pp 231..239, Elsevier, 1992.
- [6.6] *IEEE Proceedings, Special Issue on Computerised Tomography.* Vol. 71, No. 3, March 1983.
- [6.7] **Wolberg G.** *Digital Image Warping.* IEEE Computer Society Press, California, USA, pp 138..145, 1991.
- [6.8] **Smith A.R.** *Planar 2-Pass Texture Mapping and Warping.* Computer Graphics, vol. 21, no. 4, pp 263..272, July 1987.
- [6.9] Industrial Light and Magic, a division of Lucasfilm Ltd, Twentieth Century Fox, USA
- [6.10] **Smythe D.B.** *A Two-Pass Mesh Warping Algorithm for Object Transformation and Interpolation.* Industrial Light and Magic (IML) Technical Memo #1030, Computer Graphics Department, Lucasfilm Ltd., 1990.
- [6.11] **Wolberg G.** *Digital Image Warping.* IEEE Computer Society Press, California, USA, pp 222..241, 1991.
- [6.12] **Maxwell E.A.** *General Homogeneous Coordinates in Place of Three Dimensions.* University Press, Cambridge , 1951.
- [6.13] **Tanaka A.M., Kameyama M., Kazama S. and Watanabe O.** *Processing Method for the Rotation of an Image.* U.S. Patent 4,618,991, Hatachi Ltd., October 21, 1986.
- [6.14] **Paeth A.W.** *A Fast Algorithm for General Raster Rotation.* Graphics Interface '86, pp 77..81, May 1986.
- [6.15] **Hou H.S. and Andrews H.C.** *Cubic Splines for Image Interpolation and Digital Filtering.* IEEE Transactions on Acoustics, Speech, Signal Processing (ASSP), Vol. 26, pp 508..517, 1987.
- [6.16] **Freeman H.** *Computer Processing of Line Drawing Images.* Computer Surveys 6, pp 57..98, March 1974.
- [6.17] **Goshtasby A.** *Piecewise Linear Mapping Functions for Image Registration.* Pattern Recognition, Vol. 19, No. 6, 1986.

- [6.18] **Lawson C.L.** *Software for C^1 Surface Interpolation*. Mathematical Software III, Ed. by Rice J.R., Academic Press, London, pp 161..194, 1977.
- [6.19] **Read F.J.** *Applied Computational Geometry*, 2nd Edition, Waite-Group, 1988.
- [6.20] **Lee D.T.** *Two Algorithms for Constructing Delaunay Triangulation*. International Journal in Computer Information Science, Vol. 9, pp 219..242, 1980.
- [6.21] **Watson D.F.** Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. The Computer Journal, Vol. 24(2), pp 167 ..172, 1981.
- [6.22] Doctor Dobbs Journal, pp 67..75, May 1993.
- [6.23] CineMetriX - see the note at the end of Chapter 1, Palmerston North, New Zealand.
- [6.24] **Smith Y.K.** Noise and Stochastic Processes. Cambridge-Press, 1986.

Motion-tracking Cineradiographic Images

7.1 Introduction

Information on the detailed kinematics of the human spine is potentially of great interest to spinal health professionals in the diagnosis of spinal dysfunction. This information is in principle contained in a cineradiographic sequence of the spine in normal motion. The application of image processing techniques in the form of a motion-tracking algorithm offers the potential to automatically extract this information, so making it more readily available for analysis.

In chapter 4, it was concluded that template based matching approaches were the most appropriate method for motion-tracking in cineradiographic images of the human spine. This conclusion was reached by careful consideration of the many alternative approaches described in the literature. One of the important criteria with respect to the development of a practical motion-tracking system is computational cost. Template based matching approaches are the least computationally expensive method when they are combined with good motion prediction and an effective match statistic. This assertion will now be discussed.

When the object templates are small, matching is most efficiently achieved directly in the spatial domain. However to find a match for an entire object such as a cervical vertebra, that is undergoing both rotation and translation (assuming no scale change), a three dimensional parameter space (x, y, θ) needs to be searched. Searching the first two dimensions to within single pixel accuracy is straight forward since it only involves sliding the template in the x and y direction in the local neighbourhood and finding the best match. The real difficulty arises in dealing with object-template rotation. The template(s) need to be rotated through a range of angles at each x and y location where a match is to be computed. Image rotation was shown in chapter 6 section 6.4.2 to be a resampling process and although fast scanline rotation algorithms are available, their computational cost is high relative to a typical template match based algorithm using a sum-of-squared differences (see SSD in section 4.2.1 of chapter 4).

Speeding up the template interframe match search is only possible if the match function is well behaved as was illustrated in section 4.2.3 of chapter 4. In general this will not be the case so

motion prediction is one of the few practical ways to improve the search speed. Since the vertebra in a spine normally move in a relatively smooth manner, simple prediction based on one-dimensional cubic interpolation [7.1] of all three search parameters (x, y, θ) should prove adequate.

It was illustrated in chapter 4 that all motion-tracking methods are constrained by the effects of the aperture and correspondence problems. The correspondence problem is solved explicitly by object-based matching approaches when they find the best match to the object template in the next frame. However, if the feature points representing the object template are of low dimensionality (one dimension or less), then aperture effects will prevent a good match except in a direction normal to the local motion of the feature. The selectivity measure developed in chapter 5 incorporates the effects of both feature dimensionality and local sharpness. When this measure was applied to ten feature points of interest in a cineradiographic cervical spine sequence, the low dimensionality and sharpness of these local features was evident. In selecting the best match statistic for a motion-tracking algorithm, positional stability and positional accuracy of the match were also of prime importance. The normalised product correlation statistic was found to exhibit the best positional stability and accuracy, displaying reliable and consistent behaviour for all the feature points in both forward and backward interframe matching. One further advantage of the correlation measure is that it is inherently normalised. This important property will be exploited by the complete motion-tracking algorithm that will be shortly described.

7.2 Practical Implementation Issues

There are a number of practical implementation issues concerning the actual algorithm and motion-tracking system that will be briefly explored before the algorithm is described and the results of comprehensive tests reported. These issues include:

- Defining the objects of interest (*'the vertebrae'*)
- Vertebral displacement and orientation
- Deriving suitable motion measures
- Displaying motion data effectively

7.2.1 Defining Objects Of Interest

For a motion-tracking algorithm to begin the objects of interest must be specified. Chapter 3 looked at various methods of segmenting a cineradiographic image and identifying the objects of interest (*'the vertebra'*). However the low contrast of these images combined with their noisy characteristics made this task very difficult. A pragmatic solution to this problem is to get a

human operator to manually segment the objects of interest in a single frame in the sequence to be tracked. This can be done by placing a number of landmark points to outline each vertebra of interest. Since this procedure is very similar to what is performed by spinal health specialist on standard static radiographs, the same well established landmark points can be used. Such points include the front and back of the vertebral bodies and the centre of the tip of the spinus-processors.

Figure 7.1 contains the reference frame for a cervical cineradiographic sequence part way through the point placement procedure. This figure represents a graphical dialog from the software package CineMetriX [7.2] and currently shows that the skull and first three vertebra have been completed and the fifth point on the fourth vertebra is being placed. The top corner indicates that a range of marking cursors are available. Currently a large cross-hair is selected and the active point is at the intersection of the lines. The cross-hair lines are set to the inverse gray-level of the image pixels they cover. This makes the cross-hair lines appear near transparent as they are moved, thus preventing masking of the underlying structure and assisting greatly in point placement.

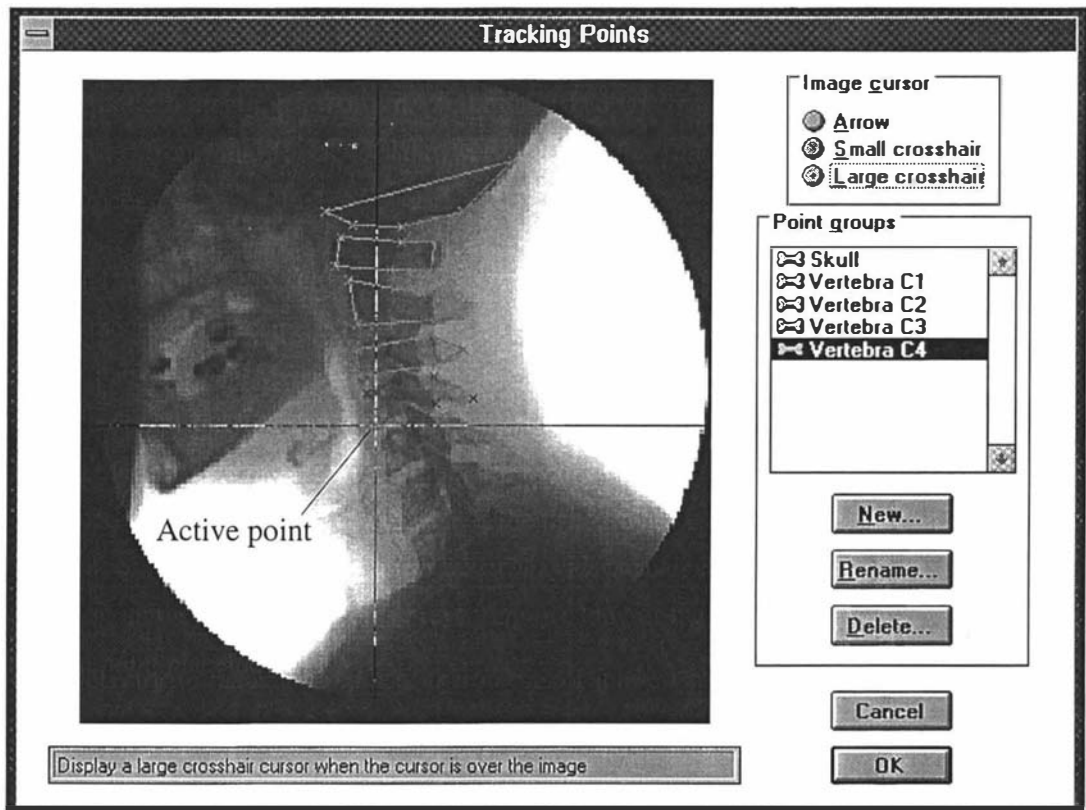


Figure 7.1: CineMetriX track points dialog - placement of the fifth landmark for vertebra C4

The coordinates of each landmark point are stored as a point-group structure to be used by the motion-tracking algorithm. The dialog also shows buttons for adding new point-groups and renaming and deleting previously defined groups.

7.2.2 Vertebral Displacement and Orientation

The orientation of an object is simply a measure of its rotation from a given view. In the context of a human observer the idea of *natural* or *visual* orientation arises [7.3]. For a naturally occurring item such as a kiwi fruit, measuring its length requires knowledge of its orientation. This is usually performed by the observer physically manipulating the fruit until the position of natural orientation is found. The actual orientation determined by the observer is dependent on the context, the objective (in this case measuring the length), the frame of reference, any geometric symmetry of the object and an individual's perception of orientation. Thus determining the natural orientation of an object requires contextual information and is often highly task dependent.

The two most commonly used terms when dealing with geometric objects are *axis* and *centroid*. *Axis* refers to a line indicating the rotation of a shape and is analogous to the standard x , y and z cartesian axes used to indicate the orientation of the Cartesian plane. A circle for instance would have an infinite number of axes all through its centre, whereas a square would have four axes through its centre at most. Symmetry plays an important roll in the determination of orientation. A symmetric object may have many axes whereas most asymmetrical shapes will have only two axes, commonly termed the *major* and *minor* axes. These two axes are usually orthogonal (perpendicular) and rotation of the object through the *major* axis is very closely related to visual orientation. The axes are usually drawn through the centroid of the object. For a simple two-dimensional object the centroid corresponds to the centre of area, or the point on the surface of the object about which the total area is evenly distributed. In the case of three-dimensional objects (which may have be non-homogeneous) the centroid corresponds to the centre of mass. An idealised point inside the object that has the mass of the object evenly distributed about it.

When considering groups of connected objects such as the vertebrae in the human spine, knowledge cues such as the *continuity* of orientation and its smoothness are also important. Deciding on the orientation of a particular vertebra in isolation is complicated by the irregular shape and density of a vertebra when viewed as a density projection in a radiograph. Thus for practical reasons it is common to use the base-line of the body of each vertebra as a datum for measuring its orientation. The base-line of a vertebra is also important from a biomechanical point of view since it represents a projection of the main loading surface. When there are several landmark points available that may or may not include points along the base-line, it essential to have an automatic method that will determine orientation in a consistent and meaningful way. If the orientation of each vertebra is not determined in a meaningful way with respect to the spine as a connected structure, then only relative measurements may be displayed in an intelligent fashion.

Orientation by Moments

Orientation of an arbitrary two-dimensional shape is commonly computed using a moment based approach. In chapter 4 section 4.2.1 central moments were introduced with respect to matching approaches. Using the three second-order central moments, what is commonly called object *orientation* can be computed using equation 7.1. The definition of orientation in this case is the angle of the axis of the least moment of inertia.

$$\theta = \frac{1}{2} \tan^{-1} \left[\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right] \quad \text{eq. 7.1}$$

where μ_{11} , μ_{02} and μ_{20} are the three second-order central moments.

This axis is drawn through the centroid of the shape at an angle θ to the horizontal. If this approach is applied to a simple shape similar to one formed by the land-mark points placed to define the corners of a vertebra, then the figures of table 7.1 result. The middle of the circle in each figure is the centroid of the shape while the crosses mark the position of the outline points.

The results in the first column of this table are based on central moments computed using just the perimeter or outline of the shape interpolated with ten discrete evaluations per unit length on the grid. The second column results are computed using the entire area of the shape with ten evaluations per unit area. Beginning at the first row of the table, the shape is defined by four points to form a simple rectangle of 2:1 aspect ratio. In both cases the moment-based approach results in an axis drawn horizontally (an orientation angle of zero) as expected. If a fifth point is added to the outline set three-quarters the way along the top edge then the result is the two figures in the second row of table 7.1. The first thing that is immediately noticeable is that the orientation of the axes is no longer horizontal for the perimeter case even though the object shape has not changed. The centroid position is unaltered for the area method while there is considerable shift upwards towards the new point for the perimeter method. The reason the first method produces a slight shift in the orientation from the horizontal even though the shape is unchanged is that in practice it is necessary to compute the moments at a number of discrete positions along each edge segment. Increasing the number of evaluations by a factor of ten to one hundred per unit length still leaves an orientation of -0.11° .

If the top right-hand point is now moved down and to the left by one grid unit and the centroid and orientation recomputed, the results are those of the third and fourth row of the table 7.1. The centroid for both implementations have barely moved from their previous location but the axis of orientation has suddenly shifted downward. This shift in angle increases rapidly as the point is moved another grid unit downward and to the left as can be seen in the third row.

The simple set of results of table 7.1 clearly show that computing orientation using moment based methods is very sensitive to small changes in the position and number of outline-points.

The calculated angle for a shape similar to a vertebra can change greatly with small variations in the placement of the landmark points and thus is unsuitable for use in a spinal motion measurement system.

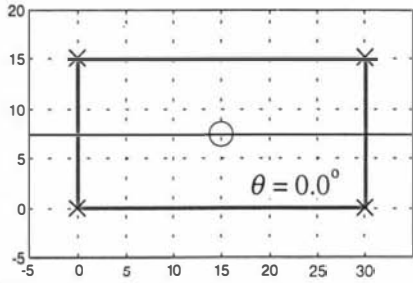
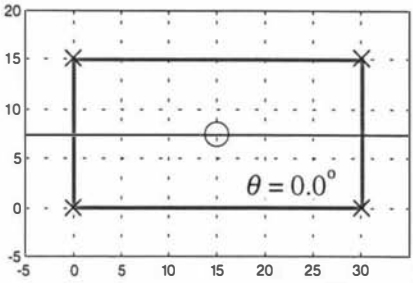
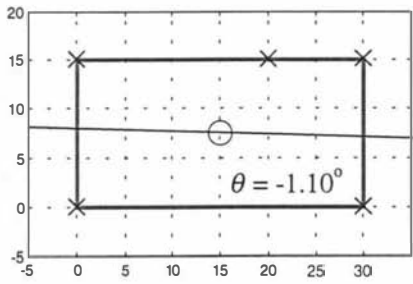
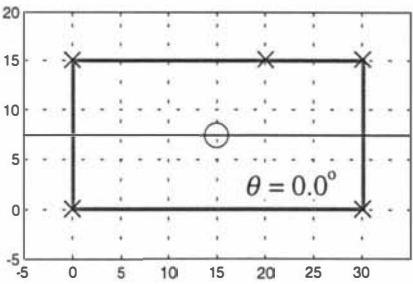
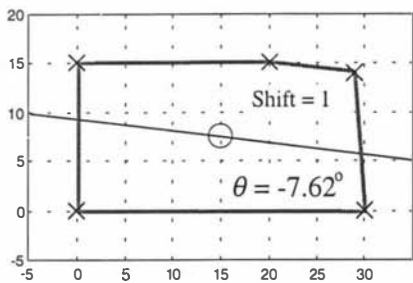
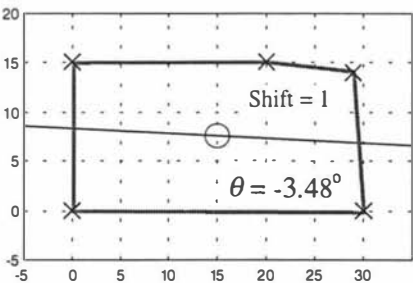
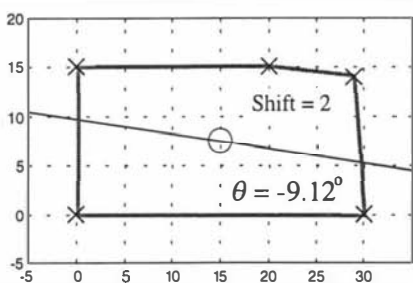
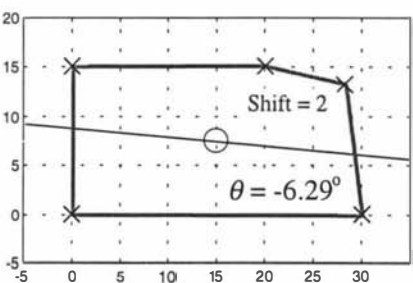
Table 7.1	Perimeter based (10 x interpolation)	Area based (10 x interpolation)
Four outline points		
Five outline points		
Third outline point moved 1 pixel		
Third outline point moved 2 pixels		

Table 7.1: Moment based orientation of a simple shape

Orientation by *Fold-and-Match*

Moment-based methods, when used to compute object orientation, do not produce results consistent with natural orientation because they are based on the axis of minimum inertia, a concept only relevant with respect to forces and mass distribution. Experimental work carried

out by the author [7.4] in conjunction with an undergraduate project came up with a new definition for natural orientation.

Natural orientation is (almost) purely related to the axis of (projected) area balance, it is the angle (usually to the horizontal) of the longest axis of maximum symmetry, drawn through the centroid.

With this new and more precise definition it becomes possible to formulate an algorithm to compute natural orientation for an arbitrary shape. The basis of the algorithm is quite simple. If a piece of card is cut to the same shape as the that formed by the given set of outline points and the centroid marked (a task humans have little difficulty in performing for almost any shape). Then the natural orientation axis is found by folding the card in half through the centroid and finding the fold angle that results in the best match between the two halves which is along the longest fold line. This description of the algorithm thus resulted in an appropriate name, the *fold-and-match* method.

The algorithm can in principle be implemented by simple numerical integration using the trapezoidal rule [7.5]. If a line is drawn through the centroid at an angle θ and then lines are drawn from the vertices (outline points) to this line at right angles, then this partitions the shape into a number of trapezoidal areas (see figure 7.2). These areas are computed using the trapezoidal rule and the total area above the fold-line is subtracted from the total area below the fold to produce an area mismatch for the given angle. If θ is swept from -90° to $+90^\circ$ in $\Delta\theta$ increments and the area mismatch recorded at each position, then the natural orientation axis is at the angle with the minimum mismatch along the longest axis.

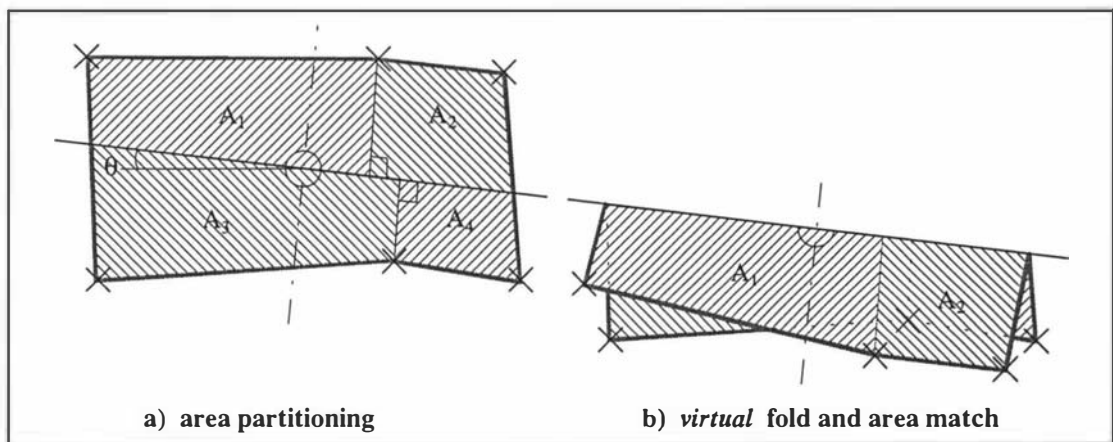


Figure 7.2: The *fold-and-match* method steps

In practice θ needs to be swept through a range slightly greater than $\pm 90^\circ$ to ensure that the true position is found. A *coarse-fine* strategy is used to minimise computation by initially using an angular increment of 10° to find the approximate position and then 1° increments to find the

orientation to within half a degree. Pascal source code for this algorithm can be found in appendix C (page C-85) as a part of the motion-tracking code.

Table 7.2 contains the results of applying the fold-and-match method to the same five outline point shapes processed by the moment based methods. These figures show the change in orientation angle as the third point is progressively moved downward and to the left in one unit increments. As can be seen the method is robust with respect to the slight shift in the position of the point since the computed angle changes very slowly. These results are also consistent with the visual or natural orientation shift of this simple object as it slowly changes shape.

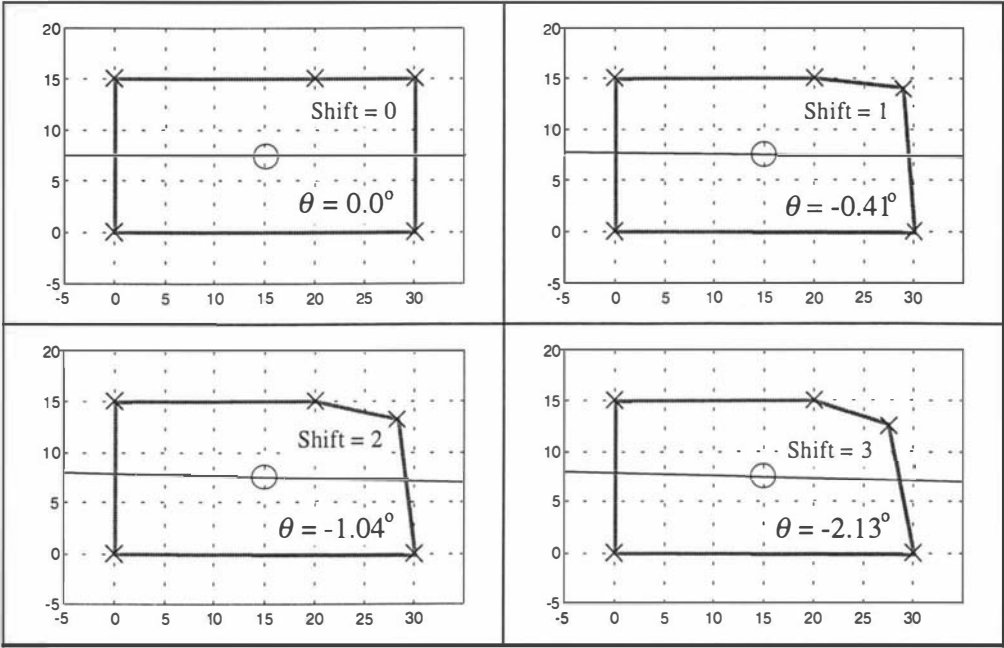


Table 7.2: Orientation of a simply changing shape by the *fold-and-match* method

If the shape of the object was much more complex than the simple examples shown above, in particular if it contained concavities or highly asymmetric protrusions, then the task of determining the natural orientation becomes very dependent on interpretation. Depending on what a human observer thinks the shape represents will greatly affect the selected orientation. Thus the simple fold-and-match algorithm would not necessarily produce results consistent with the *average* human observer. However for the simple shapes formed by placing landmark points to define vertebrae in the human spine the results will be highly consistent with perceived orientation.

7.2.3 Deriving Suitable Motion Measures

Early work by Jones [7.6] in 1959 on qualitatively assessing motion in the normal cervical spine using cineradiography determined that the motion of vertebrae during a normal cycle from full flexion to full extension is not simple. The kinematics for this region of the spine differ greatly

between the neutral to full flexion and neutral to full extension parts of the cycle. This is reflected in the shape of the curve drawn along the posterior aspects of the vertebral bodies in these two positions. Figure 7.3 shows a typical A/P cervical view of the spine in full extension and full flexion. The smoothness of the curve in extension indicates relatively equal participation of the vertebra in the overall motion. In contrast the *bow-shaped* curve for the flexion case indicates varying participation. This is due to both a gliding and rocking motion of the cervical segments.

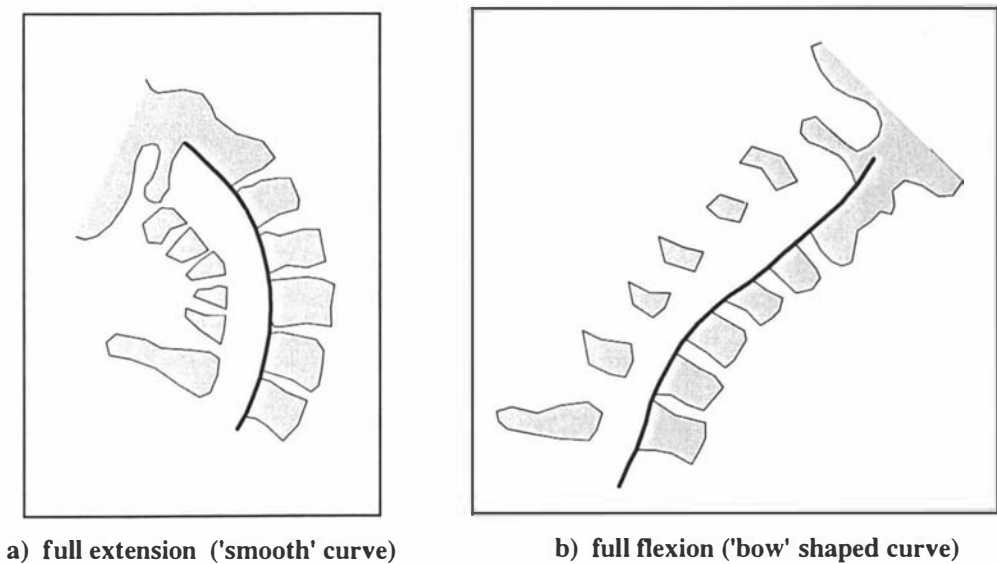


Figure 7.3: A/P cervical spine vertebral curve

A gliding motion implies translation of a vertebra near the plane of its baseline. This sort of motion should be readily apparent by comparing the relative translation of the centroid of adjacent vertebrae. A rocking motion implies a cyclic change in the orientation of vertebrae and should be readily apparent by viewing the rotational displacement of each vertebra.

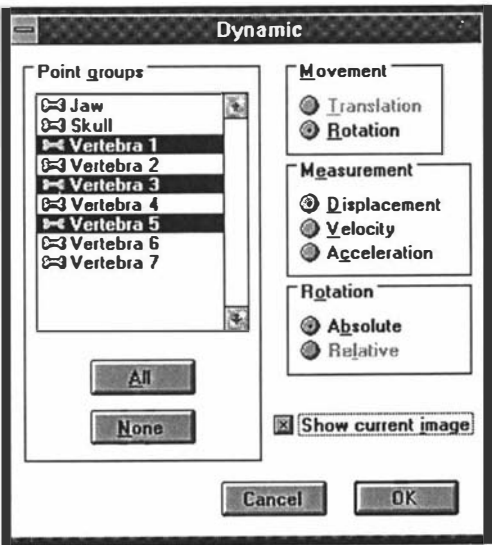


Figure 7.4: Graph control dialog

Figure 7.4 shows the dialog for controlling the data displayed in the graph window (see figure 7.7b) of CineMetriX. Consistent with the work of Jones, two basic types of movement translation (of the outline centroid) and rotation (orientation of the outline about the centroid) have been defined. All or a selected set of the tracked features (in this case the first, third and fifth vertebra) can be chosen. For each of the movements three measurements can be displayed. The first measurement is the raw data from the feature-tracking algorithm, the displacement (either translational or rotational). The other two measurements are velocity and acceleration. Discontinuities in the vertebral motion should be highlighted by looking at the velocity. While vertebral acceleration may give an indication of forces exerted on each vertebra. These measurements can be derived from the raw displacement values using suitable first and second order derivative filters [7.7]. Great care has to be taken in the design of the filter to ensure that noise does not invalidate the derived measurements.

One other indicator of biomechanical motion of the spine that has recently [7.8] been investigated by manually tracking vertebra in a short cervical spine cineradiographic sequence is instantaneous centres of rotation or ICR's. These were originally proposed by Penning [7.9] and Dvorak *et al* [7.10] as an indicator of cervical segment instability. The basic idea is that any object that is undergoing motion containing rotation and/or translation has an apparent point where the rotation could be said to be centred. For smooth motion this *virtual* centre should change slowly. However for unstable motion the centre or ICR may move erratically from frame-to-frame. The ICR may be calculated by finding the intersection of the major axis between adjacent frames in a sequence. This is shown diagrammatically in figure 7.5 below.

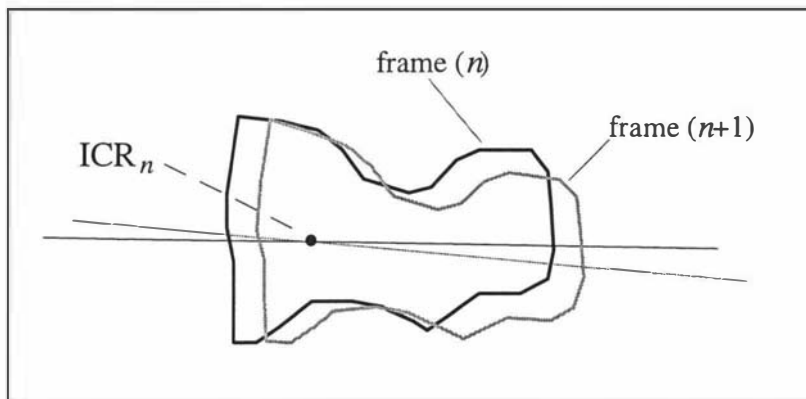


Figure 7.5: Determining ICR's from main axes intersection

For the ICR to be reasonably accurate the difference in angle must be significant. Van Mameren [7.8] suggested that seven degrees was the minimum angular difference necessary for repeatable results. Within an accurate automated measurement system this should be able to be significantly improved using data filtering and interpolation techniques.

7.2.4 Displaying Motion Data Effectively

Spinal health practitioners routinely have to deal with and interpret static radiographs. Most are not familiar with the interpretation of time-varying data such as the measurements from a cineradiographic measurement system. Technologists, engineers and scientists on the other hand routinely deal with time-varying, multi-dimensional data. Most commonly this data is presented in the form of a number of two-dimensional graphs. Interpretation of these graphs becomes difficult when the data comes from a number of different and disparate sources. Somehow the data need to be unified or connected together so they can be viewed from a common framework and interacted with by the observer to gain understanding. An active research area central to dealing with this problem is *Data Visualisation* [7.11] and most recently *Virtual Reality* (VR) [7.12] has become synonymous with this research area. Although work in this area was not central to this research, a number of VR inspired approaches have been implemented to assist in the interpretation of cineradiographic data. The general philosophy of these implementations was to maximise operator interaction.

Motion Visualisation

Visualisation is an important technique applied in the preliminary assessment of any spinal condition. A static radiograph of the spine is placed on a light box and magnifying glass used to assist in the drawing lines to extract simple geometric information. To assist visualising the information present in a cineradiographic sequences three viewing modes were provided. Figures 7.6a to 7.6c illustrate images in each of the modes. In Figure 7.6a, the standard mode, the image appears just as it would directly from the output of the video camera (coupled to the image X-ray intensifier), as a photographic positive. The brightness and contrast of the image can be adjusted interactively as the sequence is played using the image controls at the bottom of the image window. The settings of these sliders are stored as a part of the cineradiographic document and are automatically loaded when the document is opened. These controls can be hidden once set to provide greater display area. An image inversion button to the right of the slider controls provides the ability to view the sequence in the usual radiographic mode (photographic negative). Figure 7.6b shows the same image as in figure 7.6a but viewed in this mode. The image now appears like a standard static radiograph, a negative image. The invert button can be toggled interactively as the sequence is played, greatly assisting in perceiving low contrast features.

The third image mode has its origins in the early work by the author on gradient-based edge detection. The later half of this work is reported in chapter 3 section 3.4 on an alternative HVS (Human Visual System) inspired approach to edge detection (see the side panel). Figure 7.6c shows the result of this process. The edge content of the image has been enhanced while suppressing much of the intensity information.

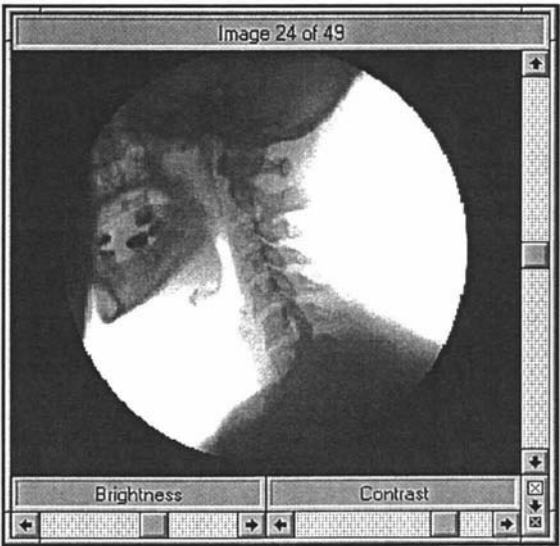


Figure 7.6 a) standard mode

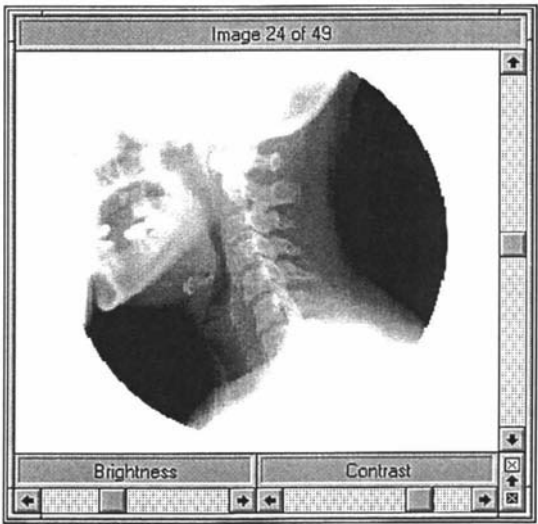


Figure 7.6 b) radiographic mode

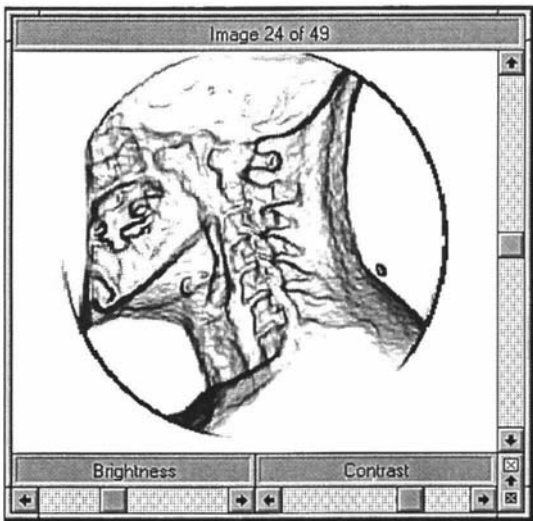


Figure 7.6 c) Gradient enhancement mode

By taking a surface-fitting approach it can be shown (see appendix A) that the heuristically derived Sobel operator (see chapter 3 section 3.2.1) can be arrived at by fitting a parabolic surface to a 3x3 neighbourhood. In practice it has been shown by the author that better gradient information is often extracted if the orthogonal axes of the surface are not made collinear with the image axes but are rotated by 45°. Setting the axes up this way and refitting the parabolic surface results in the pair of *modified* Sobel gradient masks shown in table 7.3 below.

+45	-45
$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$

Table 7.3: Modified Sobel gradient masks

Applying these masks to an image produces an estimate of the intensity gradient at each location. For cineradiographic images the resulting gradient data occupies a large dynamic range with most of the information about the vertebrae residing at the low end of the range. Thus it is necessary to scale the gradient values upwards to make the vertebral information visible when presented as a grey-scale image. After scaling large gradient values are truncated to the maximum grey-scale value.

The image looks very similar to that produced by Magnetic Resonance Imaging (MRI), although the information presented is different. This mode has been found by the author to be very effective in the subjective assessment of spinal motion.

Figure 7.7a shows the dialog box 'Cine Options' that controls the playback characteristics of the cineradiographic sequence. The first control of importance is the playback speed slider. Very often visual information can be seen more readily if the frames are played faster or slower than the original rate. In the current implementation of CineMetriX the speed can be adjusted interactively from less than one frame per second to up to twice the original capture rate. Low-speed playback enables image-by-image assessment while high-speed playback permits more global assessment of motion trends.

The next feature controllable from this dialog box is the playback mode. There are three modes currently defined, *Continuous*, *Reversing* and *Swinging*. The continuous mode runs the entire image sequence in a loop, from start to finish, repeating endlessly. The reversing mode in contrast replays the sequence from the beginning reversing at the end and replays backwards to the beginning, in a continuous loop. This produces a sequence with a more natural appearance than the continuous mode since there is no abrupt transition at the ends of the loop. In the swinging mode the loop goes from start to end and back to the start as in the reversing mode, but with a special additional feature. As the sequence advances forward say by five images, it then reverses back say four images. The overall motion in this case is forward but with a backward swing. This swinging technique has been utilised previously in meteorology [7.13], proving effective in the visualisation of complex dynamic weather patterns.

Operator Interaction

It is important that measurements derived from a cineradiographic sequence can be readily related back to what can be seen visually since this provides the opportunity for greater understanding of the motion. Figure 7.7b shows a full-screen snap-shot from CineMetriX with both the graph and image windows displayed. The graph window currently shows the rotational displacement or orientation of the three cervical vertebra C1, C3 and C5 versus the frame number.

The top of the image window indicates that the current frame displayed is number 22 of 35. This is reflected in the position of the vertical scroll bar on the right-hand side of the image window which is just over half-way down. It is also reflected in the position of scanning bar on the graph window. Outlines of each tracked feature (skull, jaw and C1 to C7) have been overlaid on the image window. Their angle indicates their orientation in the current frame while their centroid position is marked by a small dot at the centre of each outline.

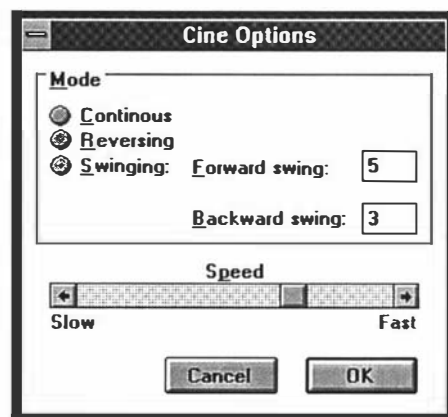


Figure 7.7a: Cine Options dialog

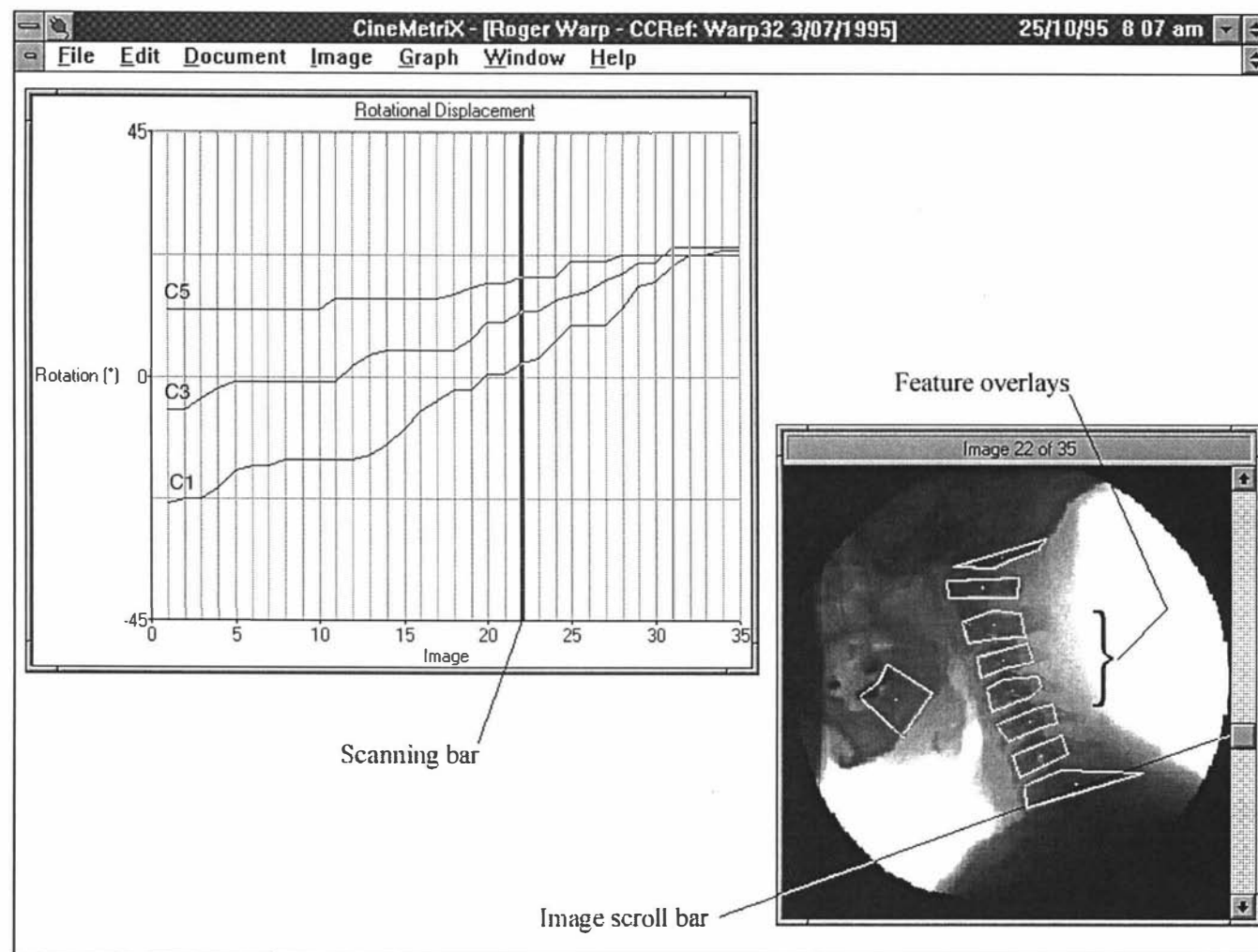


Figure 7.7b: Graphical Interaction in CineMetriX

As the sequence is played the image window slider and graph window scanning bar move in synchronism. The effect of this is that each position on the graph is visually locked to the image it occurs in and vice-versa. An operator can manually drag either the image slider or graph scanning bar while in any of the replay modes and the two windows will track together. This is particularly useful when placed in the swinging playback mode with the number of forward and backward frames set to the same value, say five.

The graph scanning bar can then be positioned at the centre of a discontinuity in one of the traces and the image window will repeat a short reversing sequence of five frames about the discontinuity. This approach enables the quick isolation of just the frames relevant to the discontinuity.

Both the image window and graph window are interactively scaleable by simply clicking on the window and dragging a corner to the desired size. This is particularly useful for the image window when displayed with feature outlines overlaid as scaling the window up allows simple visual confirmation of the tracking accuracy.

7.3 The Motion-tracking Algorithm

In the introduction of this chapter the difficulties in searching for the match to a template representing an object undergoing both translation and rotation, were summarised. One of the most obvious problems with this approach is the computational cost associated with rotating the object templates for each interframe match. One solution to this might be to pre-compute an ensemble of templates based on a rotated version of each base object template. One degree increments over an range of $\pm 45^\circ$ should be adequate for the worst case of the cervical spine vertebrae, resulting in a total of 96 templates for each object. This approach reduces the search match problem to simply seeking the best match at a given x - y location from the ensemble of rotated object templates. Since the template rotation is performed only once at the beginning of the tracking procedure, computational cost is low and storage requirements will be modest for typical vertebral dimensions. However, there are a number of problems associated with this approach. Firstly, although a template match based on the normalised product correlation will compensate for overall intensity changes between template and match area, it can not compensate for local changes within the area of the template. Local changes in intensity and contrast through a cineradiographic sequence were clearly illustrated in section 3.5 of chapter 3. This contrast modulation, due to the auto-gain control necessary to prevent camera overload, was shown graphically in table 3.12 where a line intensity profile through the second cervical vertebra at the beginning middle and end of a sequence was extracted. The last line of this table (reproduced below as table 7.4) clearly shows the significant change in intensity and contrast across the vertebra through the sequence. Thus there is no simple way to adequately compensate

for this effect if the rotated template ensemble is precomputed once at the beginning of the sequence.

The second problem with this approach is that all rotation algorithms involve image resampling and thus will introduce a degree of blur or smoothing in the rotated template that is angle and algorithm dependent. To compensate for this effect, the match area would need to be equivalently blurred before the match, adding further to the computational burden. The third problem with precomputing the rotated templates concerns scale change effects. Although the motion in a typical cineradiographic sequence of the human spine is constrained predominantly to be within the image plane, there is some out of plane motion that results in small scale changes. On an interframe basis scale change is not significant but when taken over a large number of frames it may result in significant mismatch at the true (x, y, θ) position.




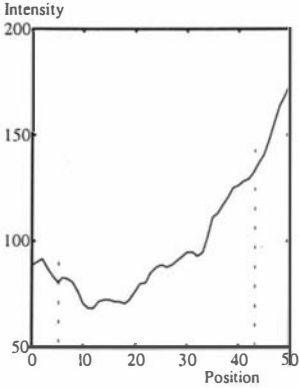
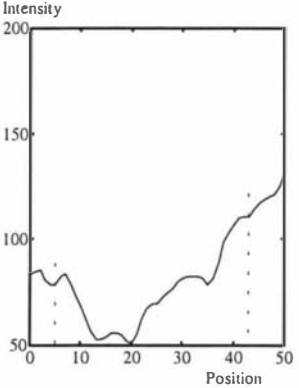
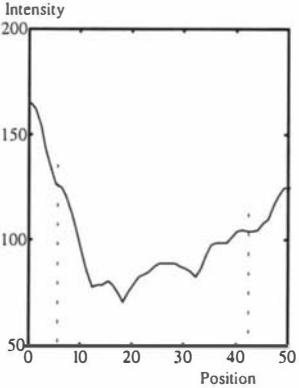
Table 7.4	First Image	Centre Image	Last Image
Cervical Section			
Intensity Profile through C2			

Table 7.4: Contrast Modulation

When these three problems are taken into account, the original direct search approach involving rotating each template over an appropriate range for each interframe match would appear the only effective solution. However there is an alternative approach that exploits one of the properties of small templates some of which were discussed in section 5.1 of chapter 5.

Small template related properties:

1. If the interframe motion is small (specifically rotation and scale change), then there is no significant error in the template match when approximated by pure translation (to within pixel resolution).
2. Small templates will contain only a small amount of information about the underlying object and this information is likely to be of low dimensionality.
3. A normalised match statistic can adequately compensate for intensity and contrast changes on an interframe basis for small templates.

Although the first property looks attractive with respect to reducing computational cost for general matching, the second property indicates that not enough information is likely to be contained in a small template to make the match reliable. However the third property indicates that it should be possible to combine the match results of a number of small templates distributed about an object of interest to produce an overall match for the object. In order for the resultant match to be useful, the sub-templates or sub-windows need to be *locked* together in the original geometry of the object. Thus in order to search for the best match to the object in the next frame, an object outline defined by the centre coordinates of the sub-windows, is moved through the search space (x, y, θ) and at each discrete location the centre coordinate of each sub-window is assigned to the nearest pixel grid. Each sub-window is then matched and the results combined together to produce an estimate of the total object match at that location in the search space. Since the assignment of the sub-window centre coordinates to a pixel grid is not until the local match is computed, the object outline can be translated and rotated in *real* coordinates through the search space.

With the best match of the entire template located in the next frame, the sub-templates are updated to use the new pixel values in this frame. The process is then repeated for each subsequent frame with the sub-template pixel values being updated after each match. Thus the algorithm works incrementally by matching between pairs of frames.

Figure 7.8 shows an enlarged section of the centre of a cervical spine from a frame of a cineradiographic sequence where the outline of the middle vertebra has been manually traced in black. Four points represented by diagonal white crosses have been placed to indicate the important landmark points of the front and back of the vertebral body. These four locations are the centres of four square sub-templates (drawn in grey) which are locked together geometrically to form an object template whose outline is represented by the white trapezoid. The centroid of this structure is indicated by the white cross in the middle of the figure.

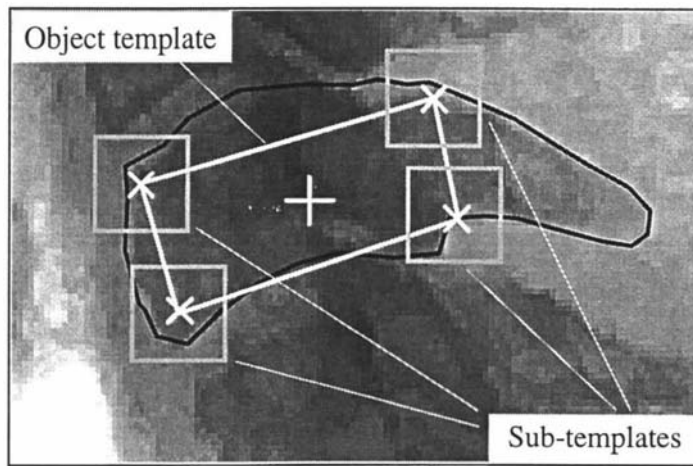


Figure 7.8: Sub-templates geometrically locked together to form an object template

As the object template is translated and rotated about the centroid (in specified increments of x , y and θ) and the sub-templates are in effect 'shuffled' like blocks to best approximate the new *real* position of the object outline.

There are a number of potential advantages to this new, geometrically-locked small-template matching approach:

1. Each sub-template match is local and thus can readily compensate for intensity contrast changes such as present cervical cineradiographic images.
2. The method is in principle able to achieve sub-pixel resolution since it involves moving the object-template in real coordinates while finding the best match of each sub-template to the nearest pixel.
3. If the original locked geometry of the sub-templates is relaxed over time by performing some additional independent local match of each sub-template, it will be possible to track objects that slowly undergo scale and perspective changes.

7.3.1 Theoretical Tracking Constraints

For perfect (noise free) data it is possible to estimate the theoretical minimum interframe movement that is detectable by this new algorithm. Three basic motion cases are summarised below:

Pure translation: With the object-template initially defined only on an integer pixel grid, an object undergoing pure translation (in x and/or y direction), can only at best be located to within one pixel.

Pure Rotation: With rotation both the x or y coordinates of the object-template outline points will change (in real coordinates) thus offering the potential of significant sub-pixel

resolution. The improvement is dependent on the size of the template and distance from the object-template centroid to the centre of rotation.

Translation and Rotation: If the motion results in greater than half a pixel change in either the x or y coordinate of an outline point then improved tracking resolution can be achieved.

As stated above for the pure rotation case, the amount of sub-pixel resolution improvement is dependent on the size of the object-template and its centre of rotation. The worst case occurs when the centre of rotation corresponds to the centroid of the object-template since at this position the average distance to the outline-points is at its minimum. This situation is illustrated below in figure 7.9 for the top right-hand corner of a rectangular object-template with its centroid used as the origin.

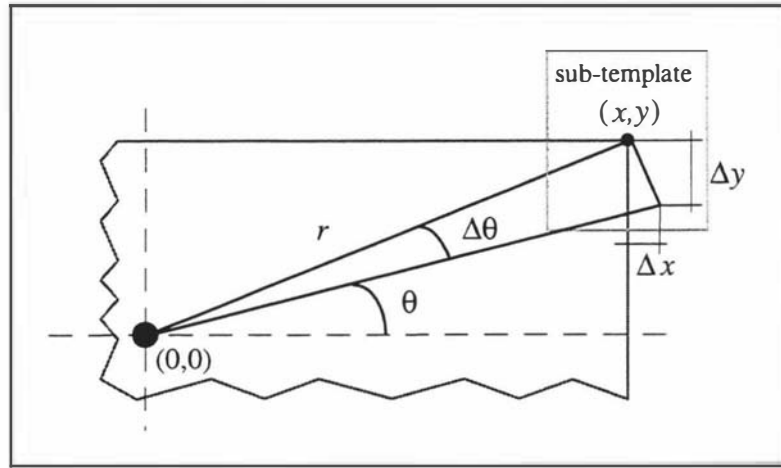


Figure 7.9: Sub-template geometry to determine the minimum detectable rotation

By applying basic geometry and considering the change in coordinate of the centre of the corner sub-template as the object-template is rotated, equations 7.1 and 7.2 result. If for a change in angle ($\Delta\theta$) there is a change in the x or y coordinate of greater than half a pixel, the position of the match sub-template will change.

$$x \pm \Delta x = r \cos(\theta \mp \Delta\theta) \quad \text{eq. 7.2}$$

$$y \pm \Delta y = r \sin(\theta \pm \Delta\theta) \quad \text{eq. 7.3}$$

where: $\theta = \tan^{-1}(y/x)$ and $r = \sqrt{x^2 + y^2}$

For a cineradiographic image of the cervical spine, digitised to 256x256 pixels, a typical vertebral body has the approximate dimensions height=15, width=30 (pixels). Substituting these vertebral dimensions into the equations above and rearranging to solve for the positive and negative deviation in angle for a 0.5 pixel shift in either coordinate, gives:

$$\begin{aligned}
 (\Delta\theta_x^+, \Delta\theta_x^-) &= (-4.12^\circ, 3.60^\circ) \\
 (\Delta\theta_y^+, \Delta\theta_y^-) &= (1.92^\circ, -1.89^\circ)
 \end{aligned}
 \tag{eq. 7.4}$$

Thus for noise free data an angular deviation of about 4° is required for a detectable change in the x -coordinate while for the y -coordinate it is only 2° .

Applying this same approach to the case where the motion contains both rotation and translation there should be an improvement in the translational resolution from 1.0 to within 0.5 of a pixel. This improvement is due to the object-template position being computed in real coordinates and only the position of the sub-templates being constrained to the nearest pixel grid when computing each local match. Once a best match is found, the position recorded is the location in the *real* parameter space (x, y, θ) .

7.3.2 Practical Algorithmic Implementation

In coding a practical implementation of the small-template based motion-tracking algorithm, a number of variations are possible that can significantly alter the computational time. One simple approach is to pre-compute a translation path or trail. A suitable path would be an integer spiral made from a set of square annular shells. All that is then required to traverse the (x, y) part of the search space is to increment along this predefined path. It was shown in the previous chapter (section 6.4.2) that a transform involving both translation and rotation is separable and can be performed in two passes. This property can be exploited by placing the computationally more expensive rotation part of the search in the outer loop. Thus for each value of theta in the search space the full translational path is traversed.

Shown over the page is pseudo code for the central section of the tracking algorithm. This code segment is called during each interframe comparison in which the new location of each object template is searched for using the normalised correlation match statistic.

Since the object tracking is based on interframe matching and the best match for a particular interframe may be slightly in error due to noise or finite resolution limitations, it is possible for the mismatch to accumulate and result in a significant tracking error. The greater the number of frames to match the more likely this will occur. However with pre-recorded sequences the number of frames to traverse from the reference frame can be halved by beginning the tracking procedure in the middle of the sequence and tracking to both ends. With only half the number of frames to traverse from the middle reference frame error propagation should be significantly reduced.

In order to track an object while keeping the computation to a minimum some limits on the maximum range of translation and rotation and a suitable sub-template size are required. It was stated in chapter 5 section 5.1 that the fastest movement (rotation and translation) of the human

spine under normal diagnostic conditions occurs in a lateral cervical view where the neck moves from full flexion to full extension. In this situation the vertebrae undergo rotation from approximately -45° to $+45^\circ$. If the motion is assumed to be nearly cyclic then an estimate of the maximum change in angle per interframe is given by dividing this range by the half the number of frames in a given sequence. At a third speed video capture a sequence will contain about 50 frames resulting in a maximum interframe angle of about 3.6° . Most of the translational movement in a typical lateral cervical sequence is constrained to be horizontal due to vertical support produced by the spinal column. For an image digitised to 256x256 pixels the maximum movement is about ± 40 pixels horizontally and about a third of this value in the vertical direction. If the motion is assumed once more to be cyclic then the maximum interframe translational values are about $\Delta x = 3.2$ and $\Delta y = 1.06$ pixels.

```
{Main object-template matching}
loop for each objectTemplate
  loop Theta from minTheta to maxTheta in steps of deltaTheta,
  rotate the objectTemplateCoords by angle Theta
  loop though the path from startPath to endPath,
  translate the objectTemplateCoords by current path
  assign objectTemplateCoords to nearest integer grid
  initialise objectMatch
  loop through integer objectTemplateCoords,
    if position has changed
      then compute the new subTemplate match and store it
    else use the previously computed match
    add the current subTemplate match to objectMatch
  continue with integer objectTemplateCoords loop
  if objectMatch is greater than bestMatch then
    replace bestMatch by objectMatch and store path as
    bestPath and Theta as bestTheta
  continue with path loop
continue with theta loop
store the best match location of objectTemplate given by
the coordinates at the bestPath position and bestTheta
continue with next objectTemplate
```

The only parameter that remains to be specified is the size of the sub-templates or sub-windows used in the match. Based on the experimental results of chapter 5, a square template of radius 5 pixels should prove adequate for most cervical spinal feature points.

Figure 7.10 shows the track options dialog used to select the objects or point-groups to track and the track parameters. Currently the second and fifth vertebrae have been selected and the 'defaults' button has been pressed to set the tracking parameters. The default translation step and rotation step size are set at the theoretical limits of the sub-pixel resolution achievable by the algorithm. The 'radio-button' has been activated to display the outline of the point-groups

representing each selected vertebra, as it is tracked. All that remains is to press the 'OK' button to start the tracking procedure.

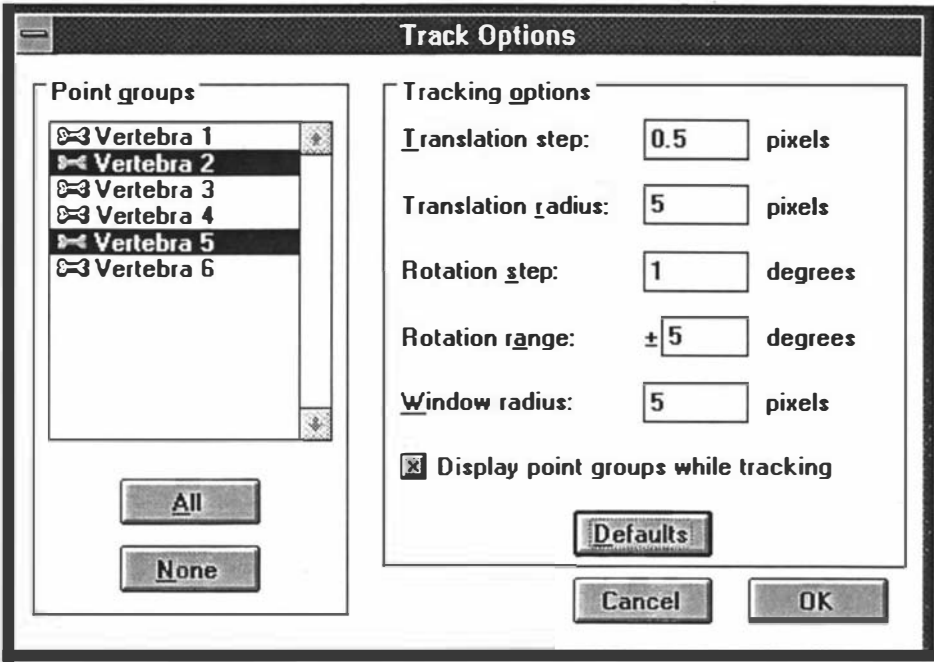


Figure 7.10: CineMetriX track options dialog

7.3.3 Image Noise

It was initially found that without performing some form of image noise filtering prior to running the tracking algorithm, the best-match interframe location would begin to wander away from the true match after only four or five interframes. The reason for this becomes apparent when looking at the image noise estimates presented in figure 6.2 of section 6.5.5 of chapter 6 and reproduced in part over the page in figure 7.11. In both the hard and soft-tissue noise estimates a best-fit (least-squares) gaussian has been superimposed on the histogram. It is clear from the middle of the histograms that the fit of the gaussians is not particularly good, with many values extending above and below the curves. The effect of this non-gaussian noise is to produce a biased estimate of the best-match and hence match position. For the normalised product correlation (like many match measures), it can be shown that the estimate of the true match will be unbiased only for data contaminated by normal gaussian noise [7.14].

The nature of noise in cineradiographic images was discussed previously in detail in chapter 2 section 2.3 where it was shown that the primary limitation on noise is the initial number of *quanta* recorded in the first conversion stage. However this result deals only with the average noise level and not its characteristics. For homogeneous materials (both in density and X-ray absorption) the noise is well approximated by a normal gaussian as was illustrated in the noise estimates through the lead collimator shutters in figure 6.2 of chapter 6. For non-homogeneous

materials such as a human spine, the noise deviates significantly from a gaussian due to secondary X-ray emission and scattering through the soft and hard tissue layers.

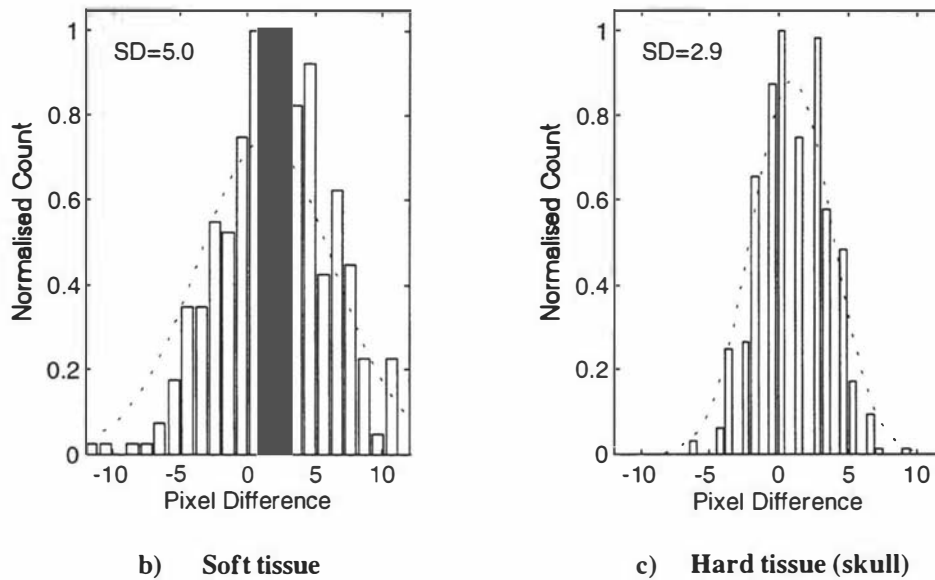


Figure 7.11: Cervical spine cineradiographic image noise

The median filter is a commonly used non-linear filter from the set of rank-based ordered statistical filters [7.15]. An extended class of the median filter is the weighted median filter (WMF) [7.16] for which the more commonly used nearest neighbour filter (NNMF) [7.17] is a sub-class. The median filter and its near relatives have general edge preserving and noise reduction properties [7.18]. They are particularly effective in reducing noise for distributions that show outliers or impulse variations similar to those shown in figure 7.10. Thus before each interframe match is performed by the motion-tracking algorithm the images are median filtered using a 3x3 window.

7.4 Tracking Performance

At the end of chapter 6 the composition of a range of motion sequences suitable for testing the tracking performance of a given algorithm on cineradiographic sequences of the human spine, was described. This range can be broadly split into those that involve adjacent vertebral motion and those that do not. This distinction was made so that testing could begin with relatively simple isolated motion and progress to more complex and realistic motion.

Simple cyclic motion has a number of advantages in terms of analysis. Firstly, in a single cycle there are three zero-points which can be used as reference points to test long-term tracking accuracy and hence mis-match error propagation. Further, if the cyclic motion is made linear, hence forming a triangular wave, then for a fixed length sequence a single control parameter, the wave amplitude uniquely defines the motion. The other advantage of linear-cyclic motion is that

the magnitude of the velocity is constant (except at the waveform peaks) enabling velocity to be used as a single analysis parameter. The other cyclic motion that is useful in testing a tracking algorithm is sinusoidal motion. In sinusoidal motion the velocity changes continuously throughout the sequence reaching a maximum value at the zero-crossings and a minimum value at the peaks. Thus the search for the best match will be exercised more fully than for the linear case and any discretisation effects of an algorithm should be exposed.

The results of tracking tests for cervical spine vertebrae in cineradiographic sequences using the sub-template based matching algorithm are presented in the following sections for a wide range of prescribed motion. In all cases the default settings defined by the track options dialog (see figure 7.9) have been used [TranslationStep=0.5 (pixels), TranslationRadius=5 (pixels), RotationStep=1°, RotationRange=±5°, WindowRadius=5 (pixels)].

An odd number of frames in a sequence is preferable to ensure that there is a genuine middle frame. Thus all the graphs in tables 7.5 to 7.10 all contain 35 frames since this is an odd number and enabled a single sequence to be fitted on a single 1.44 MByte floppy disk without loss, using simple run-length coding.

7.4.1 Single-Cycle Linear Motion: Tracking Results

The first and simplest prescribed motion test sequences used to evaluate tracking performance consists of single-cycle linear motion independently in x , then y and finally θ , for a range of amplitudes. Table 7.5 contains typical prescribed trajectories in (x, y, θ) and their values determined by the motion-tracking algorithm. These graphs are presented here to orientate the direction of the reader. Error graphs (prescribed minus tracked) will be used for the majority of the remainder of this section.

The prescribed amplitude for all the graphs in the table 7.5 is 20. This value corresponds to 20 pixels for translation in x and y and 20° for rotation in θ . The *key* at the bottom of the table defines what each of the three traces on the graphs represent. Working down the first column of the table, the traces show that the prescribed motion changes from pure translation in x across the first row, to pure translation in y across the second row and finally pure rotation in third row. All of these prescribed trajectories display the expected three zero-crossings (the middle and at either end) but also show a flattening of the waveform peaks. This truncation of the peaks is due purely to sampling the waveform over a fixed number of frames and none of the frames exactly occurring at the peaks in the wave. Since the waveform already had a discontinuity in the velocity at these points, the effect of the truncation is to limit the velocity at these points to less than would ideally occur.

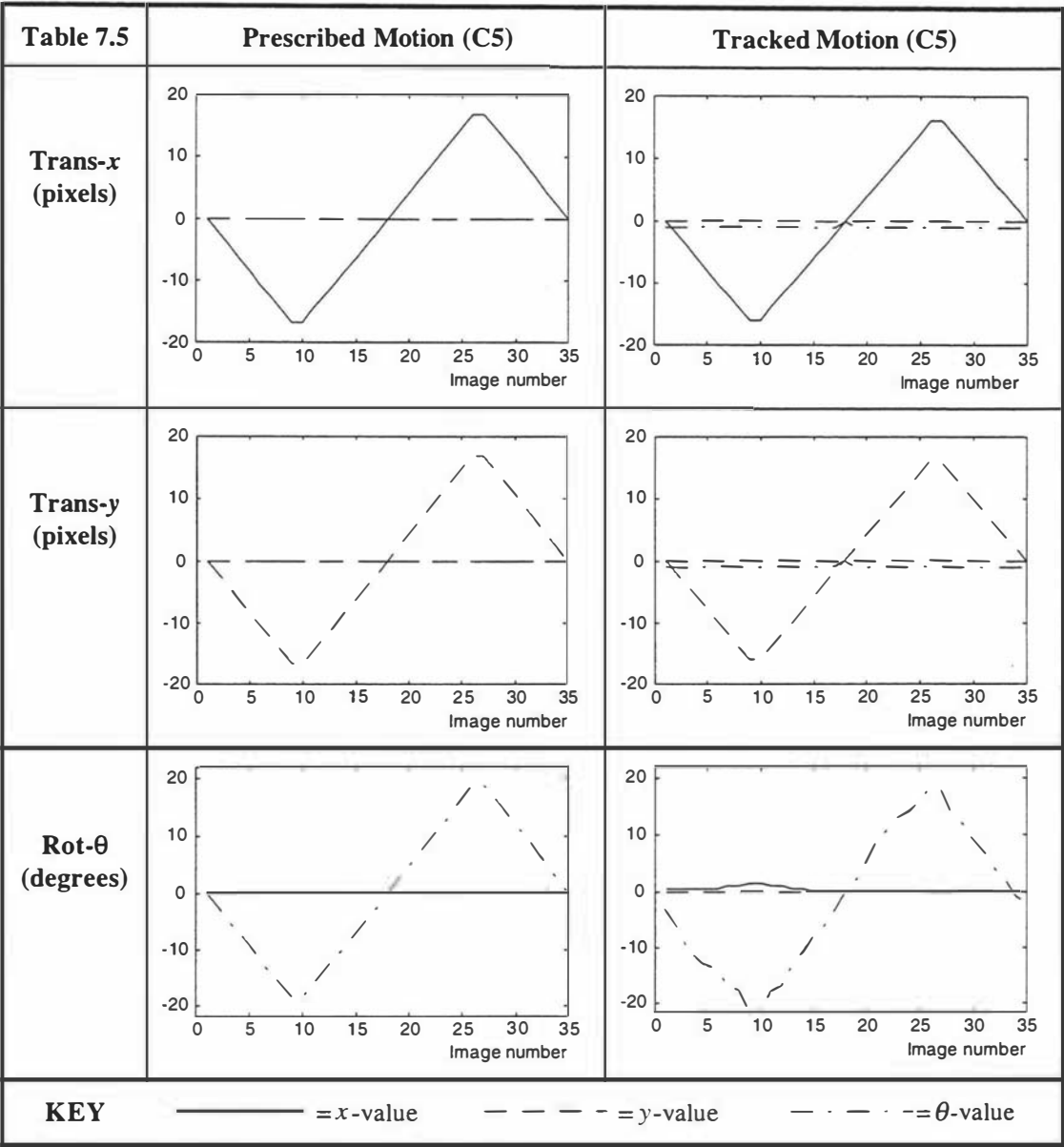


Table 7.5: Typical prescribed trajectories and tracked values for single-cycle linear motion

Visually there is little difference between the prescribed trajectory and the tracked results for pure translation in either x or y . The only slight difference occurs immediately after the first interframe match in both directions about the middle frame (frame 17), the best-match was found at an angle 1° less than the starting value. Once this slight rotation was adopted, the remainder of the sequences shows vertebral rotation, as prescribed. In simple terms this is due to the algorithm finding a more stable match to one or more of the sub-templates forming the object-template, at a location a pixel away from the original position. Once adopted, this new sub-template geometry becomes the geometry that is tracked through the remainder of the sequence. The reason for this slight change will be discussed in detail in section 7.4.3.

The results of tracking the sequence containing only rotation is somewhat different to the translation cases. Firstly, the rotation trace (dotted line) is not particularly smooth but varies above and below a straight-line rotational velocity. There is slight under-shoot when the rotation is at its minimum, with the x -coordinate moving slightly positive to compensate. The reason for this behaviour will be explained in detail shortly, but it is principally due to the algorithm attempting to work to sub-pixel performance. With a peak angle of 20° , the angular velocity is only about 1.75° per frame which is slightly below the minimum theoretical interframe angle that can be detected by the algorithm.

Single-cycle Linear Translation

Six single-cycle linear sequences were generated for a range of peak amplitudes in x , y , and θ . The maximum amplitude for the pure translation sequences was set at 40 pixels, this being twice the value measured experimentally in real cineradiographic of the cervical spine. The maximum amplitude of the pure rotation sequences was set at 40° , this being just below the limit of the ability of the animation algorithm to produce correct motion and about the same as the value measured in real sequences of the cervical spine. The mean and peak tracking error for two vertebra in the cervical spine have been plotted against interframe velocity for the three groups of one-dimensional motion. The results are contained in tables 7.6 through to 7.8. The two vertebra, the second and fifth cervical (C2 and C5) were chosen because they are representative of the difference in geometric structure between the vertebra at the ends of this section of the spine.

From table 7.6 it can be seen that the mean tracking error in the x -coordinate for translational motion prescribed only in the x -direction, decreases linearly from one pixel at an interframe velocity of 4 pixels per frame to less than a third of a pixel at one pixel per frame for both vertebrae. It then quickly rises and then falls once more, as the velocity is decreased further. The sudden rise and peaking in error at 0.5 pixels per frame is because below this value the algorithm loses lock of the vertebra due to the interframe motion being below the resolution of the algorithm under the condition of simple translational motion. This effect is seen more clearly in the peak tracking error graphs where, at the theoretical limit of one pixel per frame, the error is equal to the theoretical value of half a pixel. In all cases the maximum peak error occurred at the peak or trough of the triangular trajectory. This is because at these locations the prescribed velocity is in fact zero due to the truncation of the waveform as shown in table 7.5.

The results for the second set of prescribed sequences with motion only in the y -direction (table 7.7), are in many respects similar to the previous set for motion only in the x -direction, however there are some important differences.

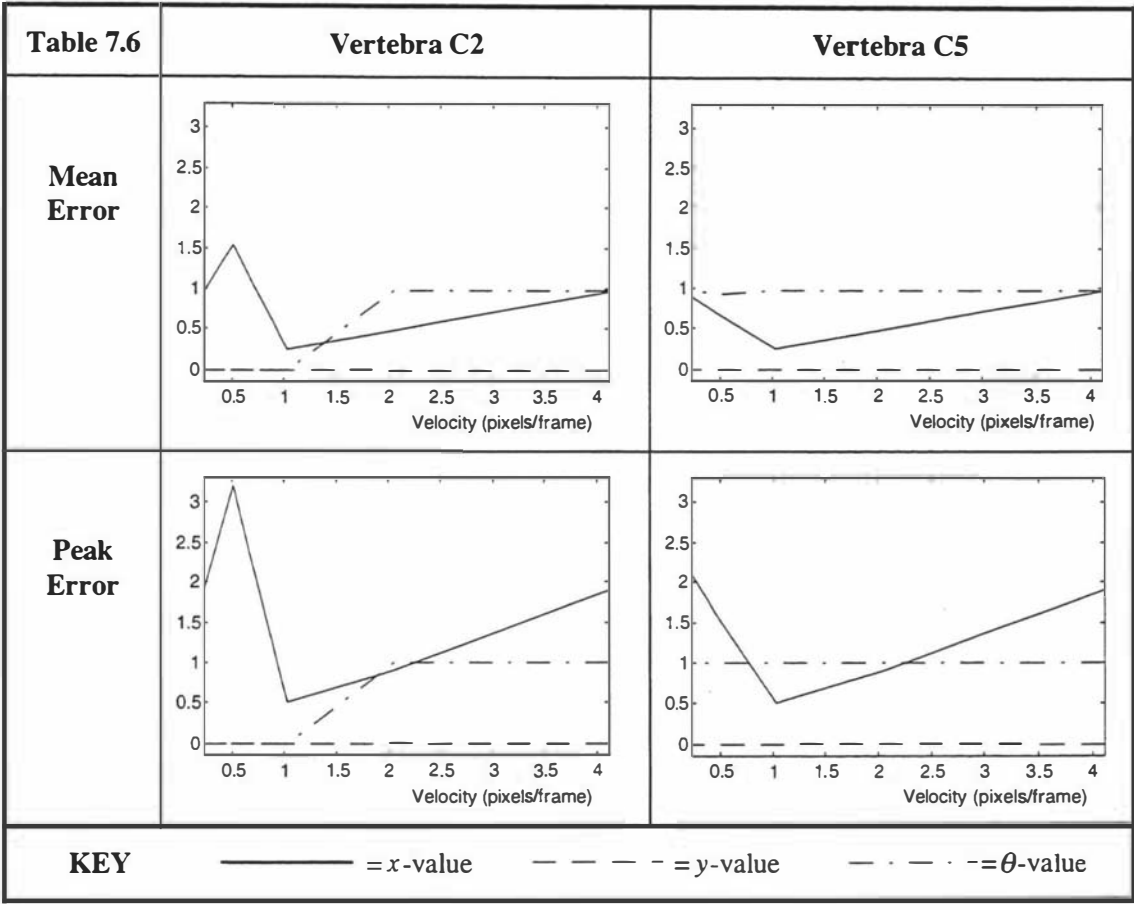


Table 7.6: Tracking error versus velocity for translational motion in the x-direction

Looking down the first column of the table 7.7 it can be seen that in tracking vertebra C2 there is speed dependent change in angle. This is quite different to the small fixed value that occurred in the previous set and which is displayed in this set by the second test vertebra C5. The reason for this velocity-dependent rotation is that the algorithm is having difficulty maintaining a reliable lock on the vertebra and is incrementally shifting the angle in an attempt to find the best match. The actual angular shift is below the theoretical minimum detectable angle of about 2° and so is in fact the result of some of the sub-templates translating slightly. The reason for this effect can be found by going back to chapter 5 section 5.3.2 on interframe feature-point matching. The selectivity of the various match statistics was measured for all the defined feature-points for both the forward and backward interframe about the middle of a sequences and the results presented in table 5.10.

The last row of this table containing the normalised product correlation match results has been extracted and is presented below as figure 7.12. If the small difference in selectivity between the forward (solid line) and the backward (dashed line) interframe match is ignored, it can be seen that there is a significant difference in selectivity between the four feature-points defined on the second vertebra C2 (last four points on the graph).

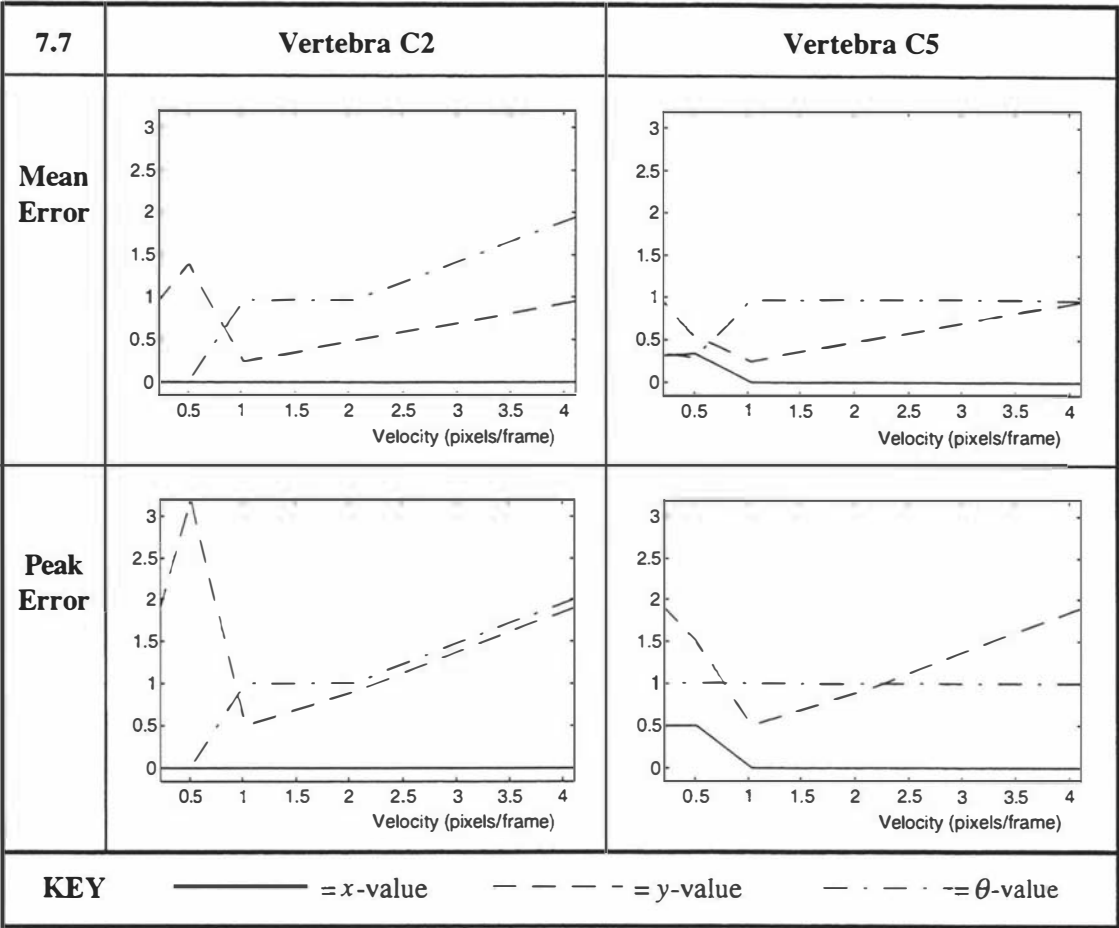


Table 7.7: Tracking error versus velocity for translational motion in y-direction

The two points defined at the front of the vertebral body (C2: 1 and C2: 2) show much lower selectivity than the two points at the back (C2: 3 and C2: 4). Visually the two front points are of moderate contrast but they are also very one-dimensional since they lie along the almost straight front edge of the vertebra. This low dimensionality in the vertical direction means that if the motion of the vertebra is predominantly in the y-direction, then the best-match will tend to slide along the front edge in the direction of the motion. Also since the motion-tracking algorithm in its current form gives equal weighting to all the sub-templates forming the total object-template match, the effect of these front points sliding vertically is to produce a small incremental rotation of the object-template match. This does not occur for the second test vertebra C5 since the front and back of the vertebral body is far more irregular and thus of high dimensionality and selectivity.

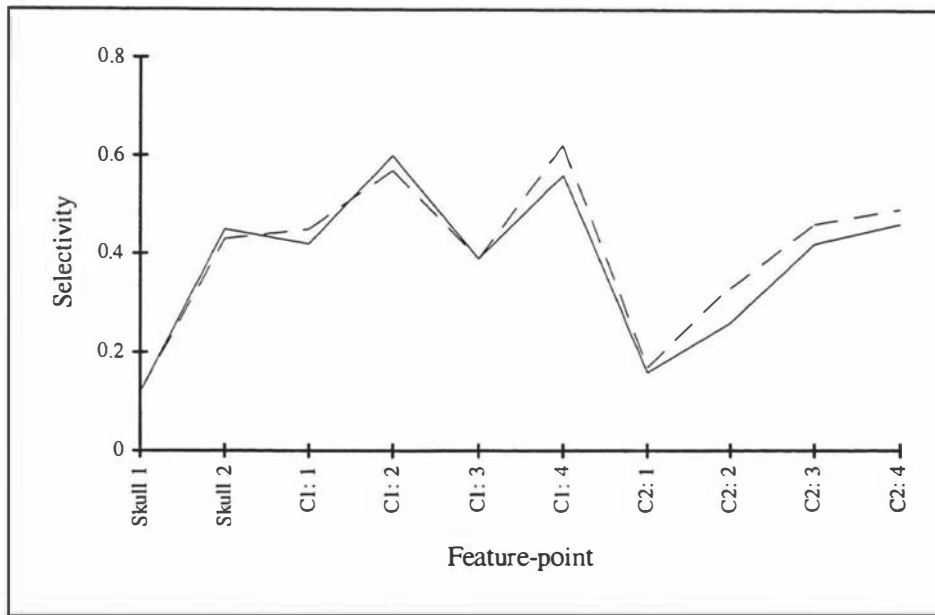


Figure 7.11: Interframe best-match selectivity for the normalised correlation measure

Single-cycle Linear Rotation

The third set of single-cycle prescribed motion test sequences are for a constant linear rotational velocity. The mean and peak tracking errors for two of the vertebrae in these sequences are presented in table 7.8 for a range of velocities.

The first thing that is obviously different from the previous series of results is that all three search space parameters now show a velocity dependent variation. This variation is particularly evident in the peak tracking error graph for vertebra C2. An important observation is that right across the range of prescribed angular velocities the mean tracking error for both vertebrae is closely bounded by the theoretical minimum measurable angle of about 2° . However the peak tracking error shows values over two times this range with the error generally increasing with decreasing velocity until it suddenly drops at just below the limit of trackability. Translational errors in x and y decrease with decreasing velocity but the error in the y -value is about twice the x -value which is in itself within the theoretical trackable range of ± 1 pixel. In order to understand this complex interaction it is necessary to look at actual tracked parameter values and their respective errors for a typical sequence.

Table 7.9 contains the results for a sequence with a prescribed peak angular amplitude of 20° . This corresponds to a velocity of about 2.5° per frame on the graphs in table 7.8. The results for vertebra C5 contained in the second column of table 7.9 are in close agreement with the theoretical trackability constraints of the algorithm. The maximum angular error occurs at the peak of the waveform where the velocity is zero for one frame and then changes sign as it

continues. Also at the waveform peak positions the x and y values are non-zero as they try to compensate for the slight overshoot and undershoot of the measured angle. Thus it can be concluded that the traces for C5 are within the range of what is to be expected from the algorithm.

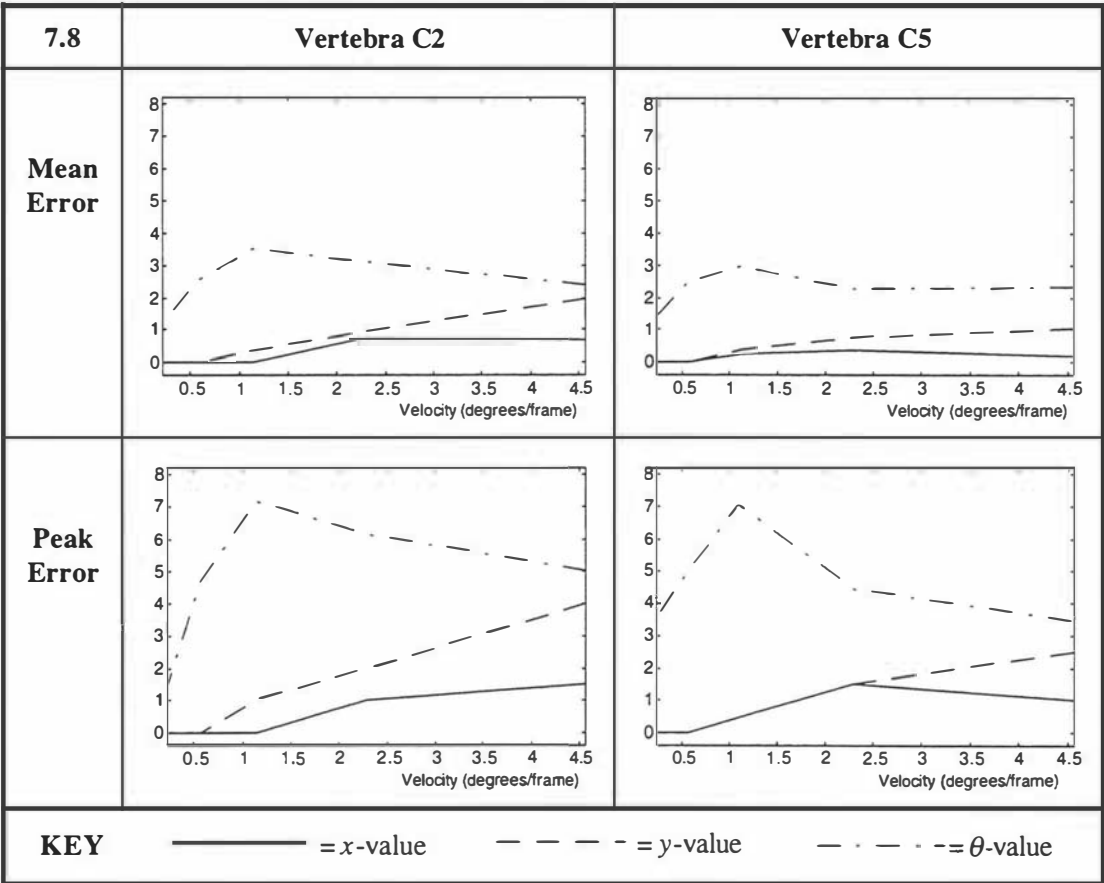


Table 7.8: Tracking error versus angular velocity for pure rotational motion

However, the traces for vertebra C2 are very different and display a highly irregular shape. The reason for this irregularity stems from the low selectivity of this feature as a whole and the degree to which a match *lock* can be maintained. The start of this problem was first seen in the tracking results for C2 for the case of pure translation in y where the error in the y coordinate changed in a velocity dependent way. Since rotation involves both a change in the x and y coordinates, then if the match lock is not particularly good in the y direction due to poor selectivity (low dimensionality and low contrast) then the best match will not be found at the true position. The general effect of this is that the best match will tend to lag behind the true position to such an extent that small changes in the x and y coordinates can not compensate sufficiently to maintain a reliable lock on the feature.

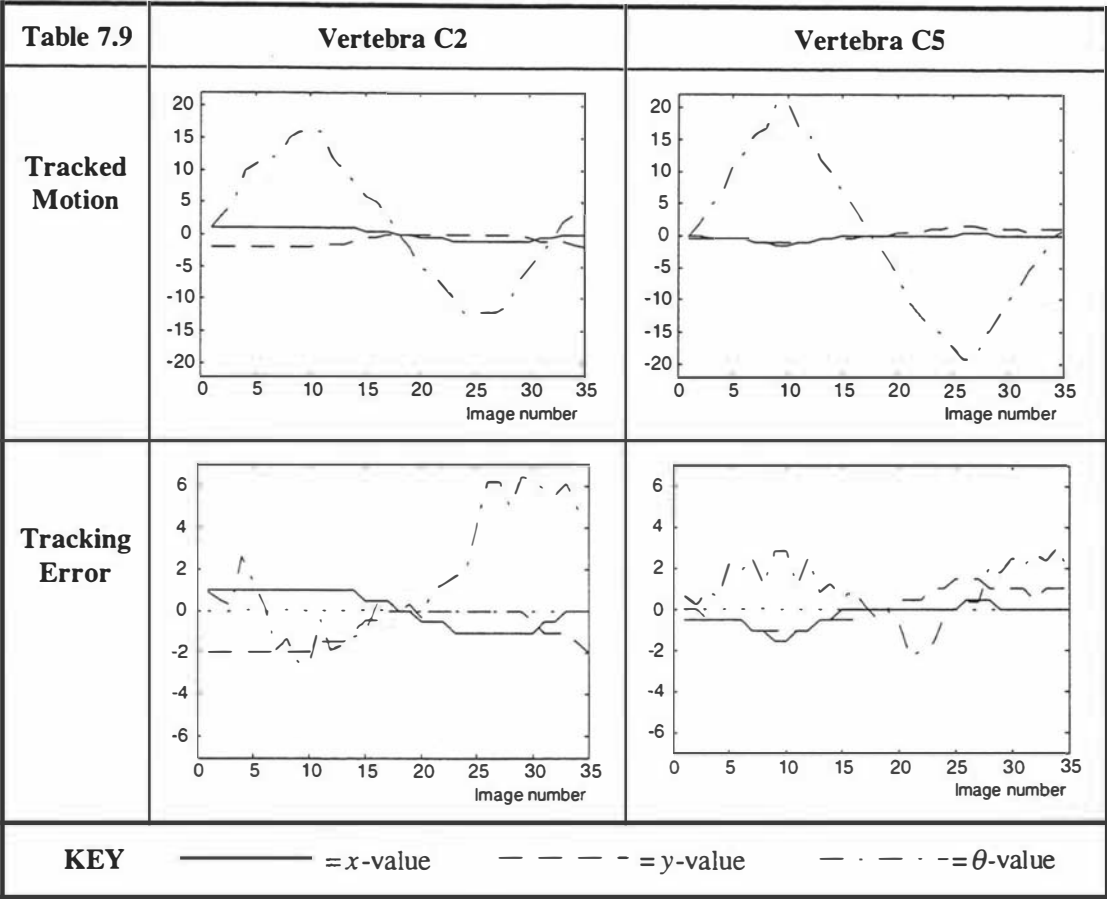


Table 7.9: Single-cycle linear rotation - tracked parameter values and resulting errors

Single-cycle Composite Motion

The last set of results in this section are for the case where the prescribed motion is linear and cyclic in all three dimensions of the search space. The relative proportions of peak amplitude of the motion have been set to $(x,y,\theta)=(18,9,20^\circ)$ which are comparable to those observed experimentally in the middle of the cervical spine in real cineradiographic sequences. The results are presented in table 7.10 for vertebra C5 which is typical for this part of the spine.

Although the tracking error varies throughout the sequence this variation is generally irregular and below the theoretical minimum trackable values of the algorithm. The reason the algorithm is able to produce these results which are significantly better than the three previous cases where the prescribed motion was only in one of the dimensions of the search space, is to do with the way the *sliding* of the sub-templates attempts to find a best match for the object-template as a whole. This effect is evident in the size of the incremental variations in each of the search space dimensions. Translational changes as small as a quarter of a pixel and angular changes of a fraction of a degree, can be seen. The maximum error in all three dimensions occurs at the peak of the wave where the angular error is just approaching the theoretical minimum detectable value and the x and y coordinates are moving slightly positive to compensate. A similar but smaller

effect is seen towards the beginning of the sequence (Note - this is the end of the first phase of the algorithm since it tracks from the centre frame to the beginning of the sequence and then from the centre frame to the end of the sequence).

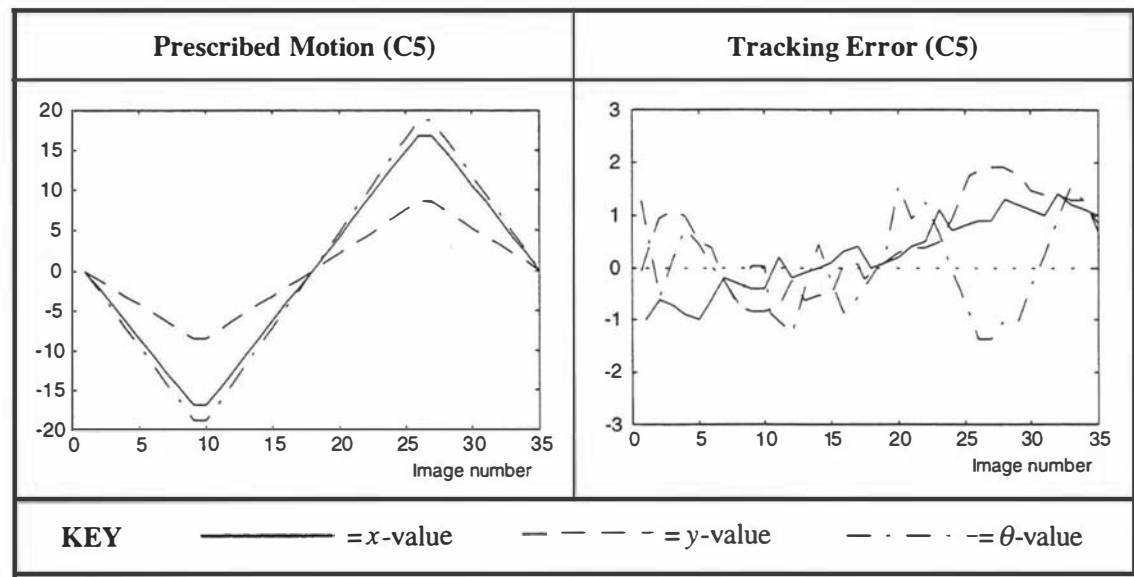


Table 7.10: Tracking a sequence with composite motion prescribed in both x , y and θ

7.4.2 Double-Cycle Composite Motion: Tracking Results

In order to assess the long-term tracking ability of the algorithm, two styles of sequences were constructed. The first set used a total of 35 frames as previously, but the prescribed motion now covered two linear cycles. The peak amplitude of the motion was reduced to half the value of the single-cycle sequences in order to keep the interframe velocity of all three search parameters constant between sequences. The second style of test sequences was designed to specifically assess interframe mismatch propagation. These sequences covered two linear cycles and maintained the same interframe velocity of the three search parameters as previously, but were twice the length. In this case the algorithm has to perform twice as many interframe match comparisons in order to track the entire sequence, thus highlighting mismatch propagation effects.

Double-cycle Composite Motion

Table 7.11 contains the tracking results for a prescribed sequence in which the peak amplitude of the motion of the two vertebra is $(x,y,\theta) = (9,5,10^\circ)$, where x and y are in pixels. Looking at the tracking error traces for C2 it can be seen that the error in x and y varies approximately cyclically with the prescribed motion. The maximum error in both cases occurs roughly at the peaks of the prescribed motion, but are generally within a ± 1 pixel band. By contrast the angular error is proportionally greater and at most positions is positive in value. This result is consistent with the loss of tracking lock previously displayed in table 7.9 for this same vertebra.

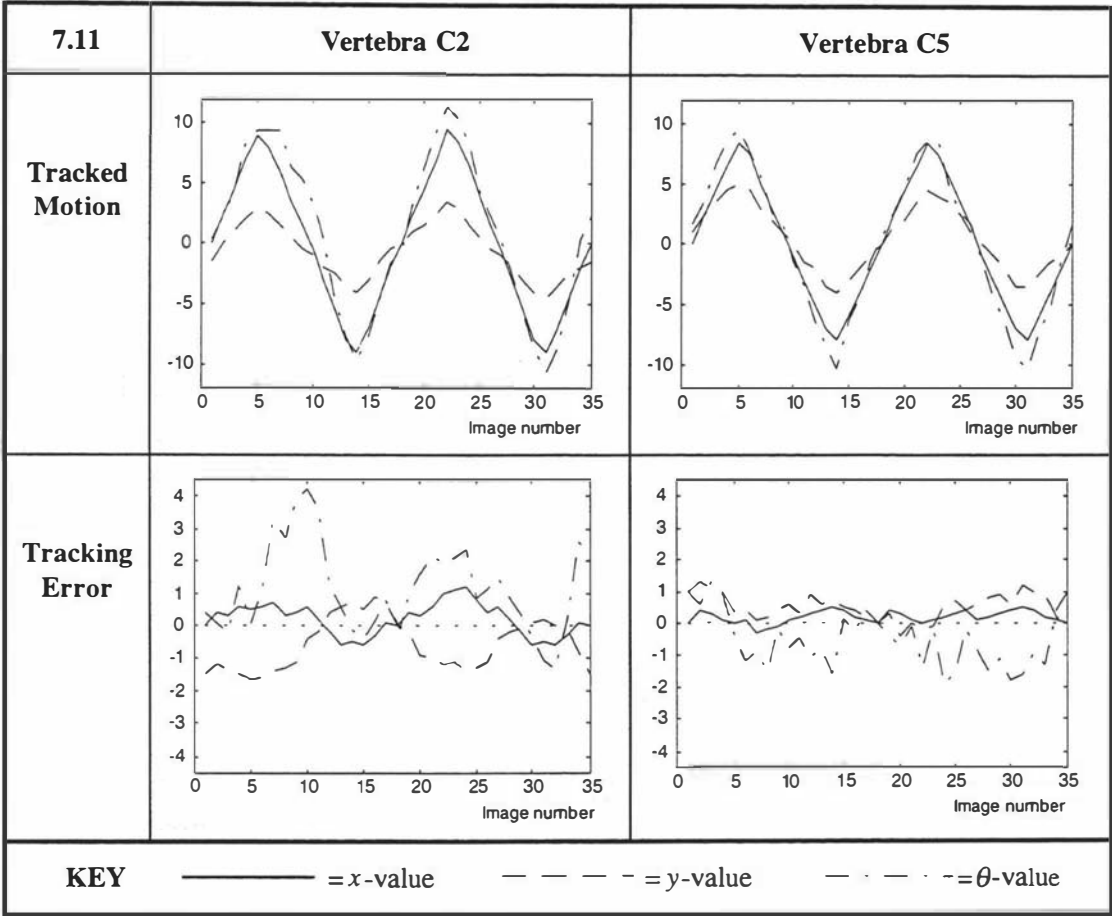


Table 7.11: Double-cycle composition motion - tracked parameter values and resulting errors

Although the centroid remains close to the true position as indicated by the displacement errors, a good match is not maintained for the angle due to the low dimensionality of the overall match-template, for this vertebra. A much better picture is seen for the other vertebra C5. The displacement errors are cyclic but are in most cases close to the theoretical tracking limit of half a pixel. Angular tracking accuracy is extremely good, displaying an error less than one degree throughout the sequence. The error is about half the value predicted from the simple theoretical analysis of section 7.3.1.

Double-cycle, Extended length, Composite Motion

The second set of results in this section are for an extended length, double-cycle, composite motion sequence. What is immediately noticeable about these results (presented above in table 7.12) is that the relative difference in tracking angle error between vertebra C2 and C5 is almost the opposite of what was shown previously. The displacement errors are generally within one pixel (particularly for C5) but a match lock on the vertebral angle is not be maintained for C5. The effect of this is that the match angle slides increasingly positive with respect to the prescribed motion for about two third of the forward and backward track paths. In the forward path it then stabilises as a better lock is achieved around frame 50, whereas the backward path

suddenly changes direction at about frame 10 and then rapidly converges back to the original location.

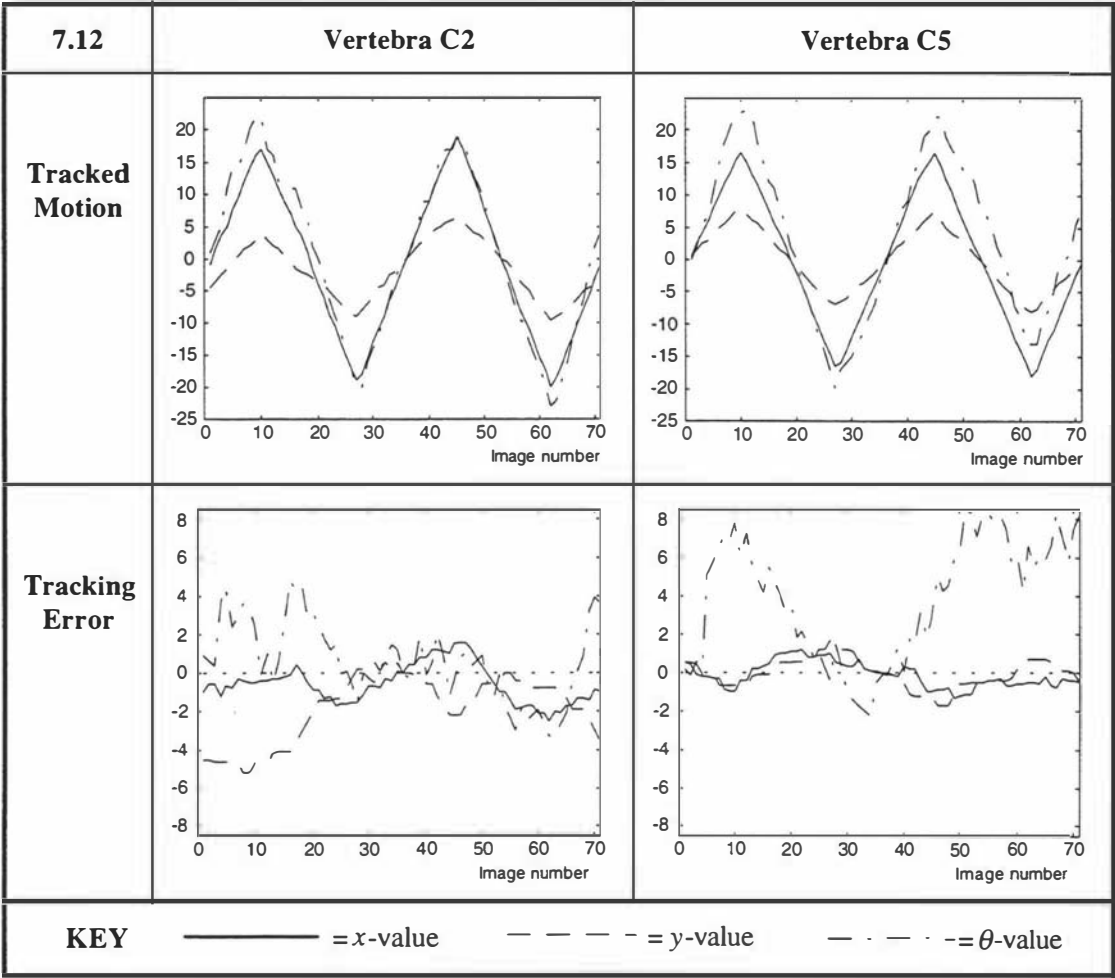


Table 7.12: Double-cycle composite motion, extended sequence - tracked parameter values and resulting errors

Experimenting with the manual placement the points that define the sub-template centres for C5 could not achieve significantly better results than those displayed in table 7.12. Consistently the angle would slide and then stabilise in the forward direction, while in the backward direction the angle would slide and then converge rapidly back to the true location. Although the interframe motion with respect to the parameter search space is constant through the test sequences, increasing the number of frames and hence the number of interframe comparisons greatly effect the overall tracking performance.

7.4.3 Full Motion with Vertebral Interaction: Tracking Results

All of the previous test sequences have been produced with no prescribed adjacent vertebral motion. This was achieved by defining a clearance zone around each vertebra and then

prescribing motion only to alternate vertebrae in the cervical region. Some local interaction does however occur using this approach since the space between alternate vertebra is distorted in order to produce the animation. This approach was used so that testing the motion-tracking algorithm could begin with relatively simple isolated motion and progress to more complex and realistic motion.

Realistic Motion

Producing realistic motion in which there is full vertebral interaction with typical range and type of motion requires knowledge of the biomechanics of the spine. If the cervical region is considered to be a simply connected system in which the bottom vertebra (C7) is fixed with respect to displacement, but is free to rotate. Then for a cycle of motion from full extension to full inflection, rotation and displacement will decrease progressively from top to bottom and the motion will be relatively linear but decreasing towards the extremes of the range. The approach used to determine typical trajectories for vertebrae in the cervical spine was to take a real sequence, place land-mark points to define all the vertebrae (including the skull and jaw) and then motion track the sequence with the algorithm. The resulting track data (x, y, θ) can then be passed to the animation routines along with the object outline information and a sequence will be produced with motion very similar to the original sequence. For this approach to be successful the motion-tracking algorithm has to perform fairly well on the real sequence. Slight differences between the real (as yet unknown) motion and the measured motion however should not cause any problems since actual test sequence(s) will be the prescribed (known) motion from the warp-based animator.

Table 7.13 illustrates typical motion in the cervical spine of a real sequence measured using the algorithm. Only the motion for alternate vertebrae starting at C1 is shown for clarity. The quality of this data was assessed visually by playing the sequence in a continuous *swinging* loop (see section 7.2.4) with the vertebral outlines superimposed. Visually there was close correspondence between the observed motion and the measured motion in the sequence. As can be seen working through the vertebrae in table 7.13, the motion is largest at the top and decreases progressively down the spine till at the bottom (C7) where the displacement is nearly zero and the vertebra simply rotates a small amount. Translational motion in the y -direction is generally only about a third of the x value. Overall motion for all three parameters can be described as approximately linear and decreasing towards the beginning and end of sequence. There are a couple exceptions to this in the form of the motion of C5 where this vertebra suddenly rotates faster at the end of the sequence. This measured motion was consistent with what was observed visually in a number of real sequences.

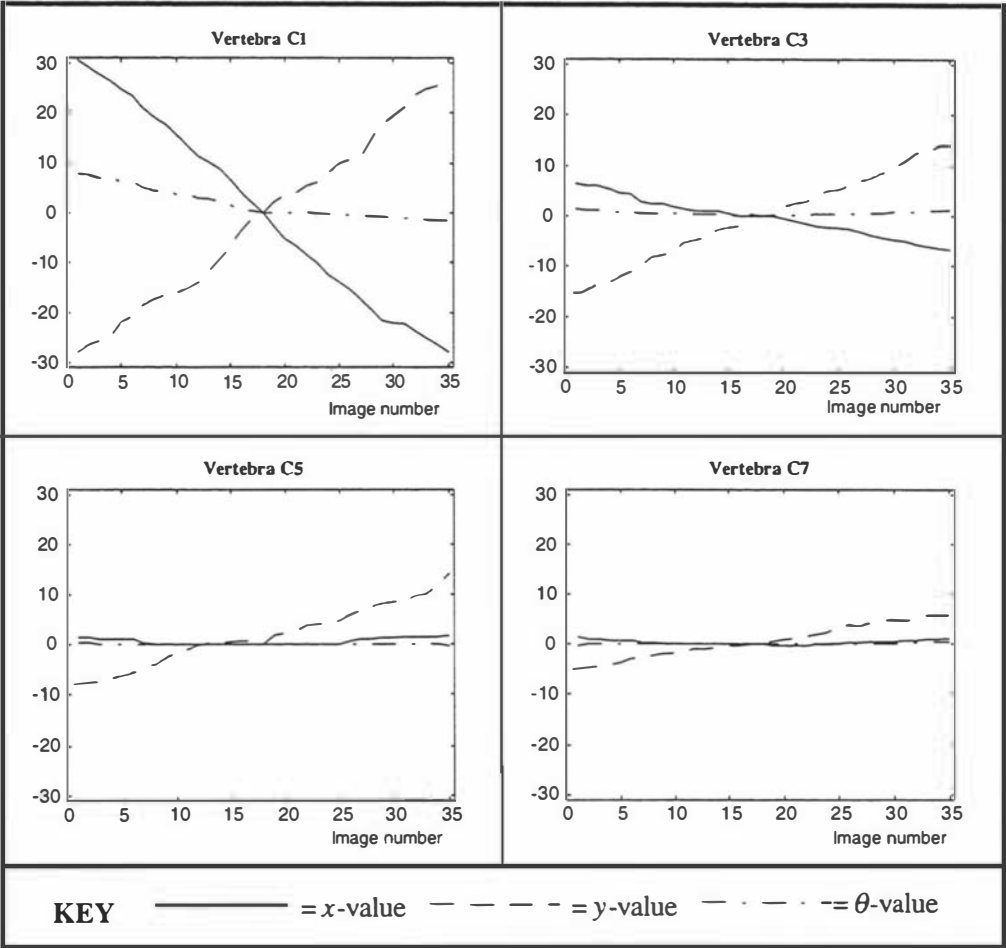


Table 7.13: Typical measured cervical motion

The typical motion in the cervical spine for full range sequences can be summarised by plotting the mean velocity in (x,y,θ) against each region. Figure 7.13 shows the results of this for the jaw, skull and working down the spine from C1 to C7. From the top of the cervical region to the bottom, translational velocity in x and y decreases fairly smoothly until C5 where it reaches a minimum. The rotational velocity on the other hand initially drops rapidly from the skull to C1 but then decreases more slowly reaching its minimum at C7.

In principle the performance of the tracking algorithm should only be effected if adjacent vertebrae move close enough together such that part of one vertebra moves into the field of one or more of the sub-templates that form another vertebra's template. This interference may occur from above and below a given vertebra. If the motion of the two interfering vertebra is similar, then tracking performance will be less effected. However if the motion of the vertebrae is greatly different, then the faster-moving vertebra will tend to drag the template of the slower moving vertebra towards it. How strong this effect will be is highly dependent on the relative selectivity of the interfering areas.

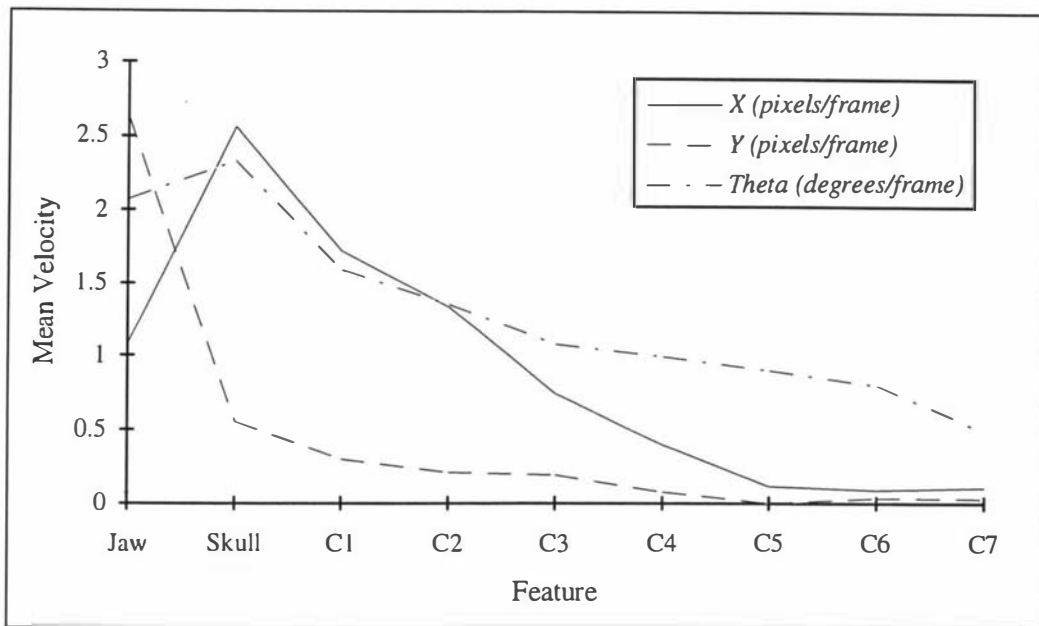


Figure 7.12: Measured mean velocity in the cervical region

Figure 7.14 contains a graph of the mean tracking error for a typical prescribed motion cervical sequence. If the skull, jaw and C4 are initially ignored, then the tracking error for the remaining features is very close to the theoretical limits of the algorithm. Displacement errors are typically between a half to one pixel and rotational errors about 2° . This is not the case for the fourth vertebra (C4) where the tracking error for all three search parameters is about twice the expected value. Experimenting with the placement of the landmark points that define the vertebra's outline template did not significantly improve the results. Consistently C4 did not track the motion well, yet C3 and C5 either side of it did track adequately.

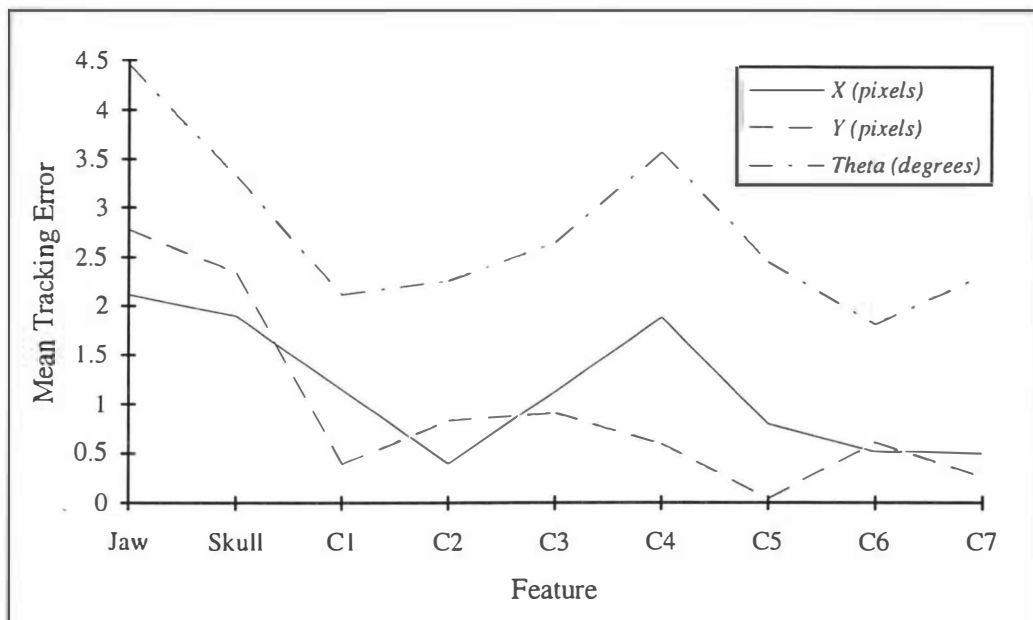


Figure 7.13: Typical mean tracking error in the cervical region

Visually playing the sequence in a *swinging* loop with the vertebral outlines overlaid, the outline for C4 was seen to either stick in one place or slide towards the vertebra above or below it, depending on the placement of the land-mark points. Digitally magnifying this region while the sequences was played revealed that this particular vertebra had a much smoother outline compared to the vertebra either side of it. The *corners* of the vertebral body are much more round (see figure 7.15) resulting in reduced selectivity of the sub-templates and hence poorer overall trackability for the vertebra.

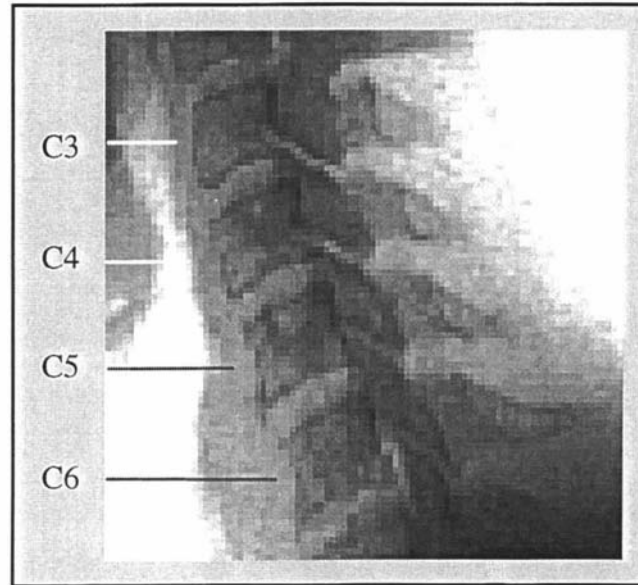


Figure 7.14: Enlarged lower cervical section

One of the slowest moving vertebra C6 has returned the lowest overall tracking error particularly with respect to angle. Even though the vertebra moves at an average velocity of just 1° per frame, a sub-template point placed on the tail of the spinous-process could be reliably matched and hence tracked. This significantly increases the effective object-template width and thus the minimum discernible change in angle.

The skull and jaw are the fastest moving features of those defined yet they display tracking errors two to three times the other features. The reason for this again comes down to the relative selectivity of the sub-template points that form each of these feature object-templates. Points placed on the lower back of the skull and the bottom surface of the jaw (see figure 7.16) are of high contrast but their dimensionality is low as they lie along a very slowly changing arc which is free from any significant changes. In contrast, the only points that can be placed towards the front of the skull and the front top of the jaw are in regions of low contrast but medium dimensionality and thus very poor selectivity overall.

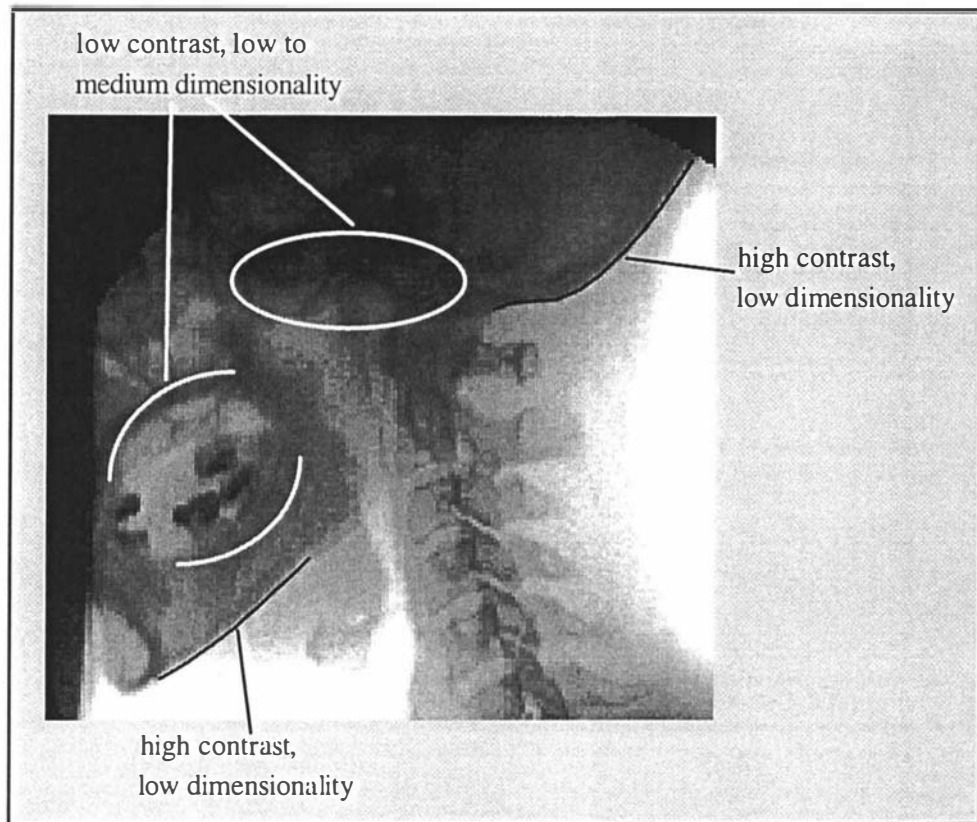


Figure 7.16: Skull and Jaw feature point selectivity

7.5 Summary of Motion-tracking Performance

The motion-tracking algorithm described in section 7.3 has been tested on a wide range of prescribed motion cineradiographic sequences of the cervical spine. The short-term tracking accuracy has been tested by creating sequences with single and double-cycle linear motion, initially independently in the three dimensions, and then combined cyclic motion. This was then followed by extended length sequences designed to specifically test long-term tracking accuracy. Two important issues have been addressed:

- The potential of sub-pixel tracking accuracy
- Mis-match error propagation

The algorithm clearly demonstrated its ability to track motion to a finer resolution than the image pixel grid. This was particularly apparent for the measured orientation of the features. The algorithm was also shown to be reasonably robust with respect to mis-match error propagation. It displayed the ability to compensate for small errors in orientation by making a small correction in either the x or y coordinate.

The final set of test sequences was based on the trajectories of the features measured in real cervical sequences. These measured values (measured by the algorithm and visually verified)

were then used to produce realistic sequences with full vertebral interaction but of known motion.

Overall it can be concluded that with well chosen landmark points to define each vertebral outline, the motion-tracking algorithm is capable of tracking motion in the cervical region of the spine with a displacement error of about ± 1 pixel and an angular error of $\pm 2^\circ$. For the larger vertebra towards the bottom of this region the angular error may be reduced to almost half the above value. Also, since the vertebra increase in size descending down the spine, it would be reasonably expected that the angular tracking error will be reduced even further in these regions.

When the interframe motion is below the trackability of the algorithm (for the cervical region the values are: displacement less than half pixel, rotation below 2°) the position of the centroid of the feature will be tracked but in steps the size of the displacement limit. However orientation will not be maintained.

The most important factor in the success of the motion-tracking algorithm is the placement of the landmark points to define each feature of interest. Poorly chosen points will results in large tracking errors and thus inadequate performance. The next chapter will conclude this research and discuss various ways of automating the landmark placement process and then dynamically monitor the performance of tracking the algorithm as it proceeds.

7.6 References

- [7.1] **Davis P.J.** *Interpolation and Approximation*. pp 67..69, Dover, New York, 1963.
- [7.2] CineMetriX, see the note at the end of chapter 1, Palmerston North, New Zealand.
- [7.3] **Howard, I.P.** *Human Visual Orientation*, John Wiley and Sons, Toronto Canada, pp 23..45, 1982.
- [7.4] **Holman D.** *Digital Image processing Techniques For natural Orientation of Shape*. Undergraduate thesis, Department of Production Technology, Massey University, Palmerston North, New Zealand, 1993.
- [7.5] **Kreyszig E.** *Engineering Mathematics*. 6th edition, pp 978..986, John Wiley and Sons, 1988.
- [7.6] **Jones, M.D.** *Cineradiographic Studies of the Normal Cervical Spine*. Journal of California Medicine, vol. 93, pp 293..296, November 1959.
- [7.7] **Wylie C.R. and Barrett L.C.** *Advanced Engineering Mathematics*. 6th edition, pp 339..344, McGraw-Hill, 1995.

- [7.8] **Van Mameren H., Sanches H., Beursgens J., Drukker J.** *Cervical Spine Motion in the Sagittal Plane II*. Spine, vol. 17, No. 5, pp 467..474, 1992.
- [7.9] **Penning, L.** *Differences in anatomy, motion development and aging of the upper and lower cervical disk segments*. Clinical Biomechanics, vol. 3, pp 37..47, 1988.
- [7.10] **Dvorak J., Froelich D., Penning L., Baumgarten H., Panjabi M.M.** *Functional radiographic diagnosis of the cervical spine: Flexion/extension*. Spine vol. 13, no. 3 pp 748..755, 1988.
- [7.11] **Tufte E.R.** *Envisioning Information*. Graphics Press, 1990.
- [7.12] **Brown H.J.,** *New Frontiers in Virtual Reality*. Limited-Press, 1994.
- [7.13] **Warnecke G.** *The visualisation of the ceaseless atmosphere*. Remote Sensing Applications in Meteorology and Climatology, Vaughan RA, Editor, Lancaster UK; pp 128..133, 1987.
- [7.14] **Pratt W.K.** *Digital Image Processing*, 2th Edition, McGraw-Hill, 1993.
- [7.15] **Tukey J.W.** *Exploratory Data Analysis*. Reading, Adison-Wesley, pp 100..149, 1971.
- [7.16] **Brownrigg D.R.K.,** *The Weighted Median Filter*. Computer ACM 27, pp 807..813, 1984.
- [7.17] **Asano A., Itoh K., and Ichioka Y.** *The Nearest Neighbour Median Filter: Some Deterministic Properties and Implementations*. Pattern Recognition, vol. 23, no. 10, pp 1059..1066, 1990.
- [7.18] **Hodgson R.M., Bailey D.G., Naylor M.J., Ng A.L. and McNeill S.J.** *Properties, implementation and applications of Rank filters*. Image Vision and Computing, vol. 3, no. 1, pp 3..14,1985.

Discussion and Conclusions

8.1 Introduction

The overall aim of the research was to develop practical motion measurement strategies for cineradiographic images of the human spine. The core of this research began from the simple premise that if detecting the edges in a cineradiographic sequence of the human spine could be achieved, then the task of tracking the motion would be straight forward since it would involve a simple binary match to find the positions of the features in subsequent frames (see section 3.1 of chapter 3 under edge-based methods). However, the implementation and extensive testing of standard edge detection algorithms and even the development of a custom algorithm (see chapter 3) did not produce sufficiently consistent results for further work based on the original premise.

The next phase of the research consisted of an investigation into a general framework for motion-tracking and perception, including the wide range of approaches presented in the literature. In particular the technique of adapting optical-flow methods to radiographic images (see section 4.3.6 of chapter 4) was researched. An estimate of the entire velocity field is not required as the only features of interest are the vertebrae. Occlusions were not a problem due to physical constraints on the movement. A template-based match approach was found to be the most appropriate method and computationally has a relatively low cost when performed in the spatial domain using small object-templates.

Central to the template based motion tracking approach is the match statistic that is used to find the best match in subsequent frames. In order to access the performance of a given match statistic a performance measure or figure of merit is needed. Such a metric was developed (see section 5.1.2 of chapter 5). This measure incorporated the effects of both feature match sharpness and dimensionality. The metric was then used to evaluate four match statistics in both intra and interframe matching of cervical cineradiographic images. The simple normalised product correlation match statistic was found to perform best overall, displaying good selectivity consistency and match positional stability.

With the framework of the motion-tracking algorithm complete, the research then turned to the problem of creating cineradiographic test sequences with known motion. Without such sequences any proposed algorithm could only be assessed qualitatively by observing the results

visually. After considering mechanical approaches such as stop-and-shoot and an articulated equivalent phantom, an animation approach based on digital image warping was developed. The final implementation was the result of a number of stages of revision of the original algorithm (see section 6.5.5 of chapter 6). The prescribed motion sequences generated by this algorithm incorporate the effects of image noise and are visually of a quality comparable to real sequences.

Finally in chapter 7, the complete motion-tracking system was described, assembled and tested. Solutions to several practical system implementation problems were also developed, and in particular, an algorithm called 'Fold-and-Match' was developed to determine the orientation of vertebrae in a natural and consistent way. The final motion-tracking algorithm is the result of a pragmatic approach based on using the properties of small-templates with the aim of minimising computational cost. This new algorithm, Geometrically-Locked, Small-Template based or GLST for short, displayed tracking performance that satisfies the first of the system objectives stated in the introduction. Within the resolution and noise limitations of a video cineradiographic imaging system, the measured displacement and orientation of vertebra are accurate and repeatable.

The last three objectives stated in the introduction and repeated below for reference, have been readily met by the development of a system written for Microsoft Windows that incorporates elements of data visualisation complete with an easy to use interface running on standard hardware.

- Provide an opportunity for operator interaction so that derived measurements can be readily related back to what can be directly perceived in a sequence.
- Be easy to use so that researchers (and potentially practitioners) will want to use such a system.
- Run on standard hardware

All the elements of the system were generalised except the video capture card driver. There were no interface standards when this part of the code was written, but this could be easily remedied by using the Video for Windows [8.1] application programming interface (API).

One objective that has only been partially met is that of minimising operator interaction in the motion-tracking process by automating landmark point placement. Typically the operator currently has to place six points to define each vertebra and then specify the motion tracking parameters. In practice the default tracking settings prove adequate. Once these two things have been done, the remainder of the process is fully automatic requiring no user input.

8.2 Algorithm Enhancements

It was concluded in chapter 7 that with well chosen landmark points used to define each vertebral outline, the motion-tracking algorithm is capable of tracking motion in the cervical region of the spine to within a displacement accuracy of about ± 1 pixel and an angular accuracy of $\pm 2^\circ$. These figures could be improved significantly by digitising the images at the full resolution available from a composite video signal. This change would quadruple the amount of storage required relative to that used in this research and necessitate the use of larger sub-template windows to ensure sufficient information is contained within them for a reliable match. The overall computational cost of the algorithm would also be increased by a similar factor. Further enhancements could be made by replacing the analogue camera with a CCD camera with a direct digital output [8.2]. This would provide images at close to the resolution of the sensor rather than the reduced horizontal resolution [8.3] (relative to the vertical resolution) available from an analogue composite video signal. This would also help combat the reduced selectivity of feature points in the horizontal direction.

A potential problem common to all interframe based matching algorithm is the possibility of mis-match error propagation. This phenomenon was accessed in section 7.4.2 of chapter 7 by creating extended length sequences. For an object moving at a speed above the trackability limit this effect does not become significant unless the sequence is very long. However for slow moving objects a gradual lag develops behind the true position in the search space. This was evident in the results obtained for tracking the prescribed motion sequences containing just rotation (see table 7.9 of chapter 7). For cyclic motion this lag did not accumulate and was restored when the motion returned the objects to their original position. The basic reason for this lag is that the interframe motion is so small that none of the sub-templates that form the object-template change position on the integer pixel grid and thus the best match does not move. In the general case, if for a sub-area in the search-space none of sub-templates move, then the actual best position that is recorded is dependent on how the *best* decision is made. The first position in this sub-area will be kept as the best position if a '*keep if greater-than rule*' is used, while the last position in this sub-area will be declared the best position if a '*keep if greater-than or equal too rule*' is used. The first case will tend to produce greater lag than the second one, but the long term effect will be generally unpredictable.

There are two approaches that would help improve the above situation. The simplest of these would be to keep the first frame in the interframe search as the reference until there is a genuine change position measured. This would prevent the motion lag from developing but correspond to sub-sampling in time. This would result in increased coarseness in the plots of the measured position. This is essentially a temporal aliasing effect. A more sophisticated approach would be to keep all the match values in a neighbourhood about the no-motion sub-area and fit a three-dimensional interpolant to estimate the true location to greater precision. As long as the

behaviour of the match in the sub-area is relatively smooth this should also improve the overall resolution of the algorithm in the situation where there is detectable interframe motion.

8.2.1 Adaptive Enhancements

None of the enhancements suggested in the previous section will be of much value if the landmark points placed by the operator are poorly chosen. Ideally, defining the features or the vertebra should be automated and not require any operator input. In the research to date this has not been investigated. A number of adaptive enhancements are possible all based on the methods developed in chapter 5 to access the performance of the match statistics.

The following elements of the algorithm could be made adaptive:

- Tune initial track point placement - At the start of the tracking procedure search locally about the user placed points and adjust their position to ensure that there is good selectivity in both the forward and backward interframes.
- Tune the track points as the algorithm proceeds to minimise feature mis-match and compensate for feature contrast changes.

For a given feature match statistic, the track point placement can be optimised by a combination of shifting the location and changing the size and shape of the sub-templates. Although this appears to be a computationally significant task, in practice, this will not be the case if a little *a priori* knowledge is applied. Firstly the maximum size of the sub-template is governed by how much rotation is expected in the current interframe and how close the nearest interfering feature is. The expected rotation can be estimated from the motion prediction values while the proximity to the nearest feature can be easily determined if all the vertebra have been defined as being of interest, which is usually the case. The shape of the sub-templates can also be significantly constrained using geometric knowledge about where a sub-template is relative to the underlying structure. Due to local point symmetry, sub-templates near the corners of the front of a vertebral body are likely to be best represented by a square window. Sub-templates positioned along the sides of the vertebrae are more likely to be represented by a suitably oriented rectangular window.

Without motion prediction, it was found during the testing of the motion-tracking algorithm that sometimes the best-match for a fast moving feature was detected far from the true match position. This usually occurred over only a few frames in which a new feature was locked onto or by the algorithm randomly moving the object template about in an attempt to find the best match to the previous location. This was due to the relative selectivity of the sub-templates and the effects of nearby strong features. With good motion prediction this problem should be eliminated. There remains the possibility that a mis-match could occur and invalidate the final

results. One solution may be to set a threshold on the minimum selectivity of an object template corresponding to reliable tracking. If this threshold is violated part way through the sequence, then the operator can be warned and asked if they wish to continue.

8.3 Future Work

For a spinal health care practitioner a system that is able to measure the internal motion of the spine in a relatively non-invasive way is of great importance. Such a system has the potential to increase the understanding of the motion of the human spine. It is likely that, it is the patterns of motion or motion signatures that will provide the clue to the diagnosis of patient condition [8.4]. The challenge will be for professionals to identify appropriate operational techniques and to link these measurements both to diagnosis and subsequent treatment. For example, the use of the system may enable a professional to embark upon a program of planned preventive maintenance to avoid impending dysfunction. This could be possible since the motion signatures contain trends which may be extrapolated using suitable statistical analysis to predict potential failure. It is likely that archives of past tests will be most useful in this manner.

Several secondary measures can be derived from basic trajectory data that may also contain the key information both for predicting potential dysfunction and characterising existing conditions. In section 7.2.3 of chapter 7, instantaneous centres of rotation (ICR's) were presented as a way of detecting vertebral instability. More useful and fundamental information may be obtained by determining the acceleration profiles of the vertebrae. When this is combined with an estimate of the relative mass of a vertebra, perhaps determined from the integrated optical density (IOD) of the feature, dynamic force estimate may be possible. This is feasible since the recorded image intensity is a measure of bone density.

Three generic digital image processing based techniques have been developed during this research, they are:

1. Motion-tracking based on the GLST algorithm
2. Generation of prescribed motion sequences using digital image warping
3. Object orientation measurement using the simple Fold-and-Match algorithm

The good performance and low computational cost of the GLST algorithm may make it a suitable candidate for real-time applications. The adaptation of image warping techniques to produce sequences with prescribed motion can be applied to a wide range of situations where an image based motion measurement system is being developed. The robustness of the simple fold-and-match algorithm may provide a better understanding of how the human visual system determines object orientation but this theory needs to be investigated.

So in conclusion, this research has resulted in the development of a set of specialised tools for the measurement of spinal motion from cineradiographic sequences. These tools also include an animation component for creating prescribed motion sequences that can be used for the evaluation of future algorithms.

All that is now required for the realisation of a system of practical use is the implementation of the algorithmic enhancements discussed in this chapter.

8.4 References

- [8.1] Video for Windows API, Microsoft Corporation, 1994.
- [8.2] **Schalkoff R.J.** *Direct Image Processing and Computer Vision.* pp 395..404, John Wiley and Sons, 1987.
- [8.3] **Netravali A.N. and Haskell B.G.** *Digital Pictures - Representation and Compression.* pp 90..112, Applications of Communications Theory, Series Editor Luck R.W., Wiley, 1988.
- [8.4] **Page W.H and Monteith W.** *Bone movement analysis from computer processing of X-ray cinematic video images.* Proceedings of the 4th IEE International Conference on Image Processing and Its Application. pp 381..384, 1992 April 7-9; Maastricht, The Netherlands. IEE London, 1992, ISBN 0 85296 543 5.

Done

Bibliography

- Abdou I.E.** *Quantitative methods of edge-detection*, USCIP Report 830, Image processing Institute, University of Southern California, July 1978.
- Abdou I.F. and Wong K.Y.** *Applied Numerical Analysis*. 2nd Edition, Addison-Wesley, 1978.
- Adelson E.H. and Bergen J.R.** *Spatiotemporal energy models for the perception of motion*. Journal of the Optical Society of America A2, pp 284..299, 1985.
- Aggarwal J. K., Duda R. O. and Rosenfeld A.** (eds.). *Computer Methods in Image Analysis*. Chapter 4: Hu K., Visual Pattern Recognition by Moment Invariants. Los Angeles: IEEE Computer Society, 1977.
- Ahmed, N. and Rao, K.R.,** *Orthogonal Transforms for Digital Signal Processing*. Springer Verlag, New York, 1975.
- Aloimonos Y., Weiss I. and Bandyopadhyay A.** *Active Vision*. Proceedings of the First International Conference on Computer Vision, pp 35..54, 1987.
- Anderberg M.R.** *Cluster Analysis for Application*. New York: Academic Press, 1971.
- Arnold C.F. and Green J.** *Standard Radiography*, Medical Press, 1978.
- Asano A., Itoh K., and Ichioka Y.** *The Nearest Neighbour Median Filter: Some Deterministic Properties and Implementations*. Pattern Recognition, vol. 23, no. 10, pp 1059..1066, 1990.
- Attneave F.** *Some informational aspects of visual perception.*, Psychological Review No. 61, 1954.
- Bajcsy R. and Campos M.** *Active and Exploratory Perception*. CVIGIP: Image Understanding, Vol 56 (1), pp 31..40, July 1992.
- Ballard D.H. and Brown C.M.** *Computer Vision*. Prentice-Hall, 1982.
- Bellman R.E. and Dreyfus S.** *Applied Dynamic Programming*, Princeton, N.J., Princeton University Press, 1962.
- Besl P.J. and Jain R.C.** *Segmentation Through Variable-Order Surface Fitting*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 2, pp 167..92, March 1988.
- Bracewell R.N.** *The Fourier Transform and its application*. McGraw-Hill, pp 56, 1986.
- Brice C.R. and Fennema C.L.** *Scene Analysis using Regions*. In Computer methods in Image Analysis, edited by Aggarwal J.K., Duda, R.O., and Rosenfeld, A., Los Angeles: IEEE Computer Society, 1977.
- Brown H.J.,** *New Frontiers in Virtual Reality*. Limited-Press, 1994.
- Brownrigg D.R.K.,** *The Weighted Median Filter*. Computer ACM 27, pp 807..813, 1984.

- Brunnstrom K. and Eklundh J.** *Active Fixation for Junction Classification.* Proceedings of the Second European Conference Computer Vision, pp 452..459, 1992.
- Burt P.J.** *The pyramid as a structure for efficient computation.* In Multi-resolution image processing and analysis, edited by Rosenfeld, A., Springer Series in Information Sciences, Vol. 12, Springer, New York, 1984.
- Canny J.** *A computational approach to edge detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, p. 679, November 1986.
- Coltman J.W.** *Journal of Radiology*, No. 51, pp 539, 1948.
- Davis L.** *A survey of edge detection techniques.* Computer Graphics and Image Processing, Vol. 4, pp 248..270, 1975.
- Davis P.J.** *Interpolation and Approximation.* pp 67..69, Dover, New York, 1963.
- Delp H.J and Chu C.H.** *Detecting Edge Segments.* IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-15, No. 1, pp 144..152, Jan./Feb. 1981,
- Deriche R.** *Using Canny's criteria to recursively implemented optimal edge detector.* International Journal of Computer Vision, Vol. 1, No. 2, 1987.
- Dimnet J. Pasquet M.H., Krag M.H. and Panjabi M.M.** *Cervical spine motion in the sagittal plane: Kinematics and geometric parameters.* Journal of Biomechanics, Vol. 15, pp 959..969, 1982.
- Dudani S., Breeding K. and McGhee R.** *Aircraft Identification by Moment Invariants.* IEEE Transactions on Computers C-26, No. 1, pp 39..45, Jan. 1977.
- Duntta R., Manmatha R., Williams L. and Risman E.M.** *A Data Set For Quantitative Motion Analysis.* Proceedings IEEE CVPR, San Diego, pp 159..164, 1989.
- Dvorak J., Froelich D., Penning L., Baumgarten H., Panjabi M.M.** *Functional radiographic diagnosis of the cervical spine: Flexion/extension.* Spine vol. 13, no. 3 pp 748..755, 1988.
- Fielding J.W.** *Normal and selected abnormal motion of the cervical spine from the second cervical vertebra to the seventh second cervical vertebra based on cineroentgenography.* Journal of Bone Joint and Surgery, Vol. 47a, pp 1779..1781, 1964.
- Fitzpatrick J.M.** *A method for calculating velocity in time dependent images based on the continuity equation.* IEEE Motion: Representation and Perception, pp 78..81, 1985.
- Fellgett, P.** *Journal of Photographic Science*, Vol 11. No. 1, p 31, 1962.
- Fennema C. and Thomson W.** *Velocity determination in scenes containing several moving objects.* Computer Graphics and Image processing 9, pp 301..315, 1979.
- Fleet D.J. and Jepsen A.D.** *Computation of image velocity from local phase information.* International Journal of Computer Vision 5, pp 77..104, 1990.
- Fleet D.J.** *Measurement Of Image Velocity.* Kluwer Academic Publishers, pp 26..28, 1992.

- Freeman H.** *Computer Processing of Line Drawing Images.* Computer Surveys 6, pp 57..98, March 1974.
- Frei W. and Chen C.** *Fast boundary detection: A generalisation and a new algorithm.* IEEE Transactions on Computing, pp 988..998, Oct. 1977.
- Glazer F., Reynolds G. and Anandan P.** *Scene matching through hierarchical correlation.* Proceedings IEEE CVPR, Washington, pp 432..442, 1983.
- Gonzalez R. and Wintz P.** *Digital Image Processing.* Addison-Wesley, pp 69..72, 1977.
- Goodwin P.N., Quirby E.H. and Morgan R.H.** *Physical Foundation of Radiology*, 4th Edition, Harper and Row Publisher Inc, pp 24..30, 1970.
- Gorbunov V.I. and Melikhov V.S.** *Defektoscopia*, No. 2, (A summary of semiconductor X-ray sensors), pp 28..30, 1978.
- Goshtasby A.** *Piecewise Linear Mapping Functions for Image Registration.* Pattern Recognition, Vol. 19, No. 6, 1986.
- Gurevich G. B.** *Foundations of Theory of Algebraic Invariants.* Groningen, The Netherlands: P Noordhoff, 1964.
- Halmshaw R.** *Industrial Radiology - theory and practice*, Applied Science Publishers, ISBN 0-85334-105-2, pp 122..125, 1982.
- Hannah M.J.** *Bootstrap Stereo.* Proceedings of Image Understanding Workshop, DARPA Conference, pp 201..208, April 1980.
- Hecht S.** *The visual discrimination of intensity and the Weber-Frechner law.* Journal of General Physiology, pp 241..250, 1924.
- Hodgson R.M., Bailey D.G., Naylor M.J., Ng A.L. and McNeill S.J.** *Properties, implementation and applications of Rank filters.* Image Vision and Computing, vol. 3, no. 1, pp 3..14, 1985.
- Holman D.** *Digital Image processing Techniques For natural Orientation of Shape.* Undergraduate thesis, Department of Production Technology, Massey University, Palmerston North, New Zealand, 1993.
- Horowitz N.J. and Pavlidis T.** *Picture segmentation by split-and-merge procedure.* Proceedings of the 2nd IJCPR, pp 424..433, August 1974.
- Horn B.K.P.** *Robot Vision*, MIT Press, Cambridge, 1986
- Horn B.K.P. and Schunck B.G.** *Determining optical flow.* Artificial Intelligence 17, pp 185..204, 1981.
- Hou H.S. and Andrews H.C.** *Cubic Splines for Image Interpolation and Digital Filtering.* IEEE Transactions on Acoustics, Speech, Signal Processing (ASSP), Vol. 26, pp 508..517, 1987.
- Hough P.V.C.** *Method and means of recognising complex patterns.* US Patent no. 3,069,654, 1962.
- Howard, I.P.** *Human Visual Orientation*, John Wiley and Sons, Toronto Canada, pp 23..45, 1982.

- Jain A.K.** *Fundamentals of Digital Image Processing*. Prentice-Hall Information and System Science Series, pp 21..22, 1989.
- Jain A.K. and Ranganath S.** *Image restoration and edge extraction based on 2-D stochastic models*. Proceedings of ICASSP-82, Paris, May 1982.
- Johnson B.M.** *Measurements in Skeletal Radiography*. Wiley, pp 180..195, 1977.
- Jones, M.D.** *Cineradiographic Studies of the Normal Cervical Spine*. Journal of California Medicine, vol. 93, pp 293..296, November 1959.
- Juhl J.H. and Gunmy A.B.** *Esentials of Radiologic Imaging*, 6th Edition. J.B. Lippincott Company, Philadelphia, pp 10..12, 1993.
- Kanamori, H.** *Japanese Journal of Applied Physics*, Vol 7, No. 4, p 414, 1968.
- Kass M., Witkin A. and Terzopoulos D.** *Snakes: Active contour models*. International Journal of Computer Vision 1, pp 321..331, 1988.
- Kitchen L. and Rosenfeld A.** *Edge evaluation using local edge coherence*. IEEE Transactions of Systems, Man, and Cybernetics, Vol. SMC-11, No. 9, pp 597..605, Sept. 1981.
- Kreyszig E.** *Advanced Engineering Mathematics*, 6th Edition, pp 457..504, Wiley, 1988.
- Lawson C.L.** *Software for C^1 Surface Interpolation*. Mathematical Software III, Ed. by Rice J.R., Academic Press, London, pp 161..194, 1977.
- Lee D.T.** *Two Algorithms for Constructing Delaunay Triangulation*. International Journal in Computer Information Science, Vol. 9, pp 219..242, 1980.
- Leipnik R.** *The extended entropy uncertainty principal*. Information Control 3, pp 18..25, 1960.
- Logan B.F. Jnr.** *Information in the zero-crossings of band pass signals*. Bell Systems Technical Journal, No. 56, pp. 487-510, 1977.
- Marr D. and Poggio T.** *A computational theory of human vision*. Proceeding of the Royal Society of London, B 204, pp 301..328, 1979.
- Marr D. and Hildreth E.** *Theory of Edge Detection*. Proceedings of the Royal Society of London, B 207, pp 187..217, 1980.
- Marr D. and Ullman S.** *Directional selectivity and its use in early visual processing*, Proceedings of the Royal Society of London, B 211, pp 151..180, 1981.
- Marr D., Poggio T. and Ullman S.** *Bandpass channels, zero-crossings, and early visual information processing*. Journal Of The Optical Society of America 69, pp 914..916, 1979.
- Martell A.** *Edge-detection Using Heuristic Search Methods*. Proceedings of Computer Graphics and Imaging 1, pp 169..182, August 1972.
- Maxwell E.A.** *General Homogeneous Coordinates in Place of Three Dimensions*. University Press, Cambridge , 1951.
- McIlroy C.D., Lingard R. and Monteith W.** *Hardware for real-time image processing*. IEE Proceedings, Vol. 131, part E, no. 6, November 1984.

- Melissinos A.C.** *Experiments in Modern Physics*, Academic Press, pp 28..32, 1966.
- Morevec H.P.** *Rover Visual Obstacle Avoidance*. Proceedings of IJCAI-7, Vancouver B.C., August 1981.
- Netravali A.N. and Haskell B.G.** *Digital Pictures - Representation and Compression*. pp 90..112, Applications of Communications Theory, Series Editor Luck R.W., Wiley, 1988.
- Nevatia R.** *Locating Object boundaries in textured environments*. IEEE Transactions on Computing, Vol. 25, pp 1170..1180, Nov. 1976.
- Overington I.** *Vision and Acquisition*. Pentech Press, London, 1976.
- Overington I.** *Computer Vision - A unified biologically-inspired approach*. pp 231..239, Elsevier, 1992.
- Paeth A.W.** *A Fast Algorithm for General Raster Rotation*. Graphics Interface '86, pp 77..81, May 1986.
- Page W.H and Monteith W.** *Edge Detection Based on Weber's Law Revisited*. Proceedings of the Image and Vision Computing Conference of New Zealand (IVCNZ '94), pp 2.3.1-2.3.5, August 16-17, 1994.
- Page W.H, Monteith W. and Whitehead L.** *Dynamic Spinal Analysis: Fact or Fiction?* Journal of the Australian Chiropractic Association (JACA), pp 82..85, September 1993.
- Page W.H. and Monteith W.** *Creating Deterministic Motion Using Digital Image Warping*. Proceedings of Digital Image Computing: Techniques and Applications Conference (DICTA'93), pp 485..492, Sydney Australia, 1993.
- Page W.H. and Hoogeeven R.M.** *An Analogue Video Preprocessor*. Proceedings of the Image and Vision Computing Conference of New Zealand (IVCNZ'93), pp 415..421, August 1993.
- Page W.H and Monteith W.** *Bone movement analysis from computer processing of X-ray cinematic video images*. Proceedings of the 4th IEE International Conference on Image Processing and Its Application. pp 381..384, 1992 April 7-9; Maastricht, The Netherlands. IEE London, 1992, ISBN 0 85296 543 5.
- Page W.H and Monteith W.** *Weber's Law and its Application to Dynamic Thresholding in Edge Detection*. Proceedings of the 6th New Zealand Image Processing Workshop, pp 71..76, August, 1991.
- Page W.H and Monteith W.** *A system for the Analysis of Bone Motion in Cinematic X-ray Images*. Proceedings of the 28th National Electronics Conference (NELCON 91), pp. 75..80, August 1991.

- Pahlavan K., Uhlin T. and Eklundh J.O.** *Integrating Primary Ocular Processes*. Proceeding of the Second European Conference on Computer Vision, Vol. 588, pp 526..541, Lecture Notes in Computer Science, Springer-Verlag, May 1992.
- Pavlidis T.** *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.
- Penning, L.** *Differences in anatomy, motion development and ageing of the upper and lower cervical disk segments*. Clinical Biomechanics, vol. 3, pp 37..47, 1988.
- Prewitt J.M.S.** *Object enhancement and extraction*. In Picture Processing and Psychopictorics, Lipkin, B.S., and Rosenfeld, A. editors, New York: Academic Press, 1970.
- Pratt W.K.** *Digital Image Processing*. 2th Edition, New York: Wiley Interscience, 1978.
- Pratt W.K.** *Digital Image Processing*, 6h Edition, McGraw-Hill, 1993.
- Princen J.** *Hough Transform methods for curve detection and parameter estimation*. PhD thesis, University of Surrey, June 1990.
- Read F.J.** *Applied Computational Geometry*, 2nd Edition, Waite-Group, 1988.
- Reynolds R.J.** *Cineradiography by the Indirect Method*. Journal of Radiography, Vol. 31, pp 177..182, 1938.
- Rimey R.D. and Brown C.M.** *Where to Look Next Using a Bayes Net: Incorporating Geometric Relations*. Proceeding of the Second European Conference on Computer Vision, pp 542..550, May 1992.
- Roberts L.G.** *Machine perception of three-dimensional solids*. In Optical and Electro-optical Information Processing, Tippet J.P. et al editors, Cambridge, MA: MIT Press, 1965.
- Robinson G.** *Edge detection by compass gradients*. Computer Graphics and Image Processing, Vol. 6, pp 492..501, 1977.
- Roentgen, W.C.** *Annus Physics Leipzig*, Vol. 64, No. 1, 1898.
- Rosenfeld A. and Kak A.C.** *Digital picture processing*, 2nd edition, Volume II, Academic Press, Orlando, USA, 1982.
- Schutte H., Frydrychowicz S. and Schroder J.** *Scene Matching With Translation Invariant Transforms*. Proceedings of the 5th International Conference on Pattern Recognition. pp 195..198, IEEE, New York, 1980.
- Schalkoff R.J.** *Direct Image Processing and Computer Vision*. pp 395..404, John Wiley and Sons, 1987.
- Schade, O.** *Journal of the Society of Motion Pictures, Television and Engineering*, Vol. 73, No. 2, pp 81..84, 1964.
- Shannon, C.E.** *Journal of Bell System Techniques*, No. 27, p 623, 1948.
- Smith A.R.** *Planar 2-Pass Texture Mapping and Warping*. Computer Graphics, vol. 21, no. 4, pp 263..272, July 1987.

- Smith Y.K.** *Noise and Stochastic Processes*. Cambridge-Press, 1986.
- Smythe D.B.** *A Two-Pass Mesh Warping Algorithm for Object Transformation and Interpolation*. Industrial Light and Magic (ILM) Technical Memo #1030, Computer Graphics Department, Lucasfilm Ltd., 1990.
- Spacek L.A.** *Edge detection and motion detection*. Image Vision Computing, vol. 4, pp 43..51, 1986.
- Stevens A.S.** *Medical Imaging*, Gold-Publishing, 1991.
- Stockman G.C. and Agrawala A.K.** *Equivalence of Hough transform curve detection to template matching*. Communications of the ACM, Vol. 20, pp 820..822, 1977.
- Sturm, R.E., and Morgan, R.H.** *American Journal of Roentgenography*, No. 62, pp 617..622, 1949.
- Tanaka A.M., Kameyama M., Kazama S. and Watanabe O.** *Processing Method for the Rotation of an Image*. U.S. Patent 4,618,991, Hatachi Ltd., October 21, 1986.
- Teves M.C. and Tol, T.** *Philips Technical Review*, No. 14, pp 33, 1952.
- Tufte E.R.** *Envisioning Information*. Graphics Press, 1990.
- Tukey J.W.** *Exploratory Data Analysis*. Reading, Adison-Wesley, pp 100..149, 1971.
- Van Mameren H., Sanches H., Beursgens J., Drukker J.** *Cervical Spine Motion in the Sagittal Plane II*. Spine, vol. 17, No. 5, pp 467..474, 1992.
- Walsh, J.L.** *A closed Set of Orthogonal Functions.*, Proceedings of the London Mathematical Society., vol. 34, pp 241..279, 1932.
- Warnecke G.** *The visualisation of the ceaseless atmosphere*. Remote Sensing Applications in Meteorology and Climatology, Vaughan RA, Editor, Lancaster UK; pp 128..133, 1987.
- Watson D.F.** *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*. The Computer Journal, Vol. 24(2), pp 167..172, 1981.
- Waxman A.M., Wu J. and Bergholm F.** *Convected activation profiles: Receptive fields for real-time measurement of short range visual motion*. Proceedings of CVPR, Ann Arbor, pp 717..723, 1988.
- Williams T.D. and Glazer F.** *Comparison of Feature Operators For Use In Matching Image Pairs*. Image Sequence Processing and Dynamic Scene Analysis. Springer-Verlag, Berlin-Heidelberg, pp 395..423, 1983.
- Wilson H.R. and Bergen J.R.** *A four mechanism model for spatial vision*. Vision Research 19, pp. 19-32, 1979.
- Wolberg G.** *Digital Image Warping*. IEEE Computer Society Press, California, USA, pp 138..145, 1991.
- Wong R. and Hall E.** *Scene Matching With Moment Invariants*, Computer Graphics and Image Processing 8, pp 16..24, 1978.

- Wu J., Brockett R. and Wohn K.** *A contour-based recovery of image flow: Iteration method.* Proceedings IEEE CVPR, San Diego, pp 124..129, 1989.
- Wylie C.R. and Barrett L.C.** *Advanced Engineering Mathematics.* 6th edition, pp 339..344, McGraw-Hill, 1995.
- Young A.A. and Axel L.** *Automatic Spot Detection in Chest Radiographs.* IEEE Transaction on Medical Imaging, Vol. 12, No. 4, pp 505..515, 1994.

Appendix A

Gradient Mask Derivation

A.1 Introduction

This appendix contains the derivation of several of the small-window gradient-masks traditionally used in low-level edge detection algorithms.

A.2 Curve Fitting Problem

In order to compute a partial derivative about any point in an image, a function describing how the intensity varies with position is required. A polynomial function approximation can be found in a real image by initially fitting a low-order function and then testing for overall *goodness-of-fit*. The polynomial order is then increased until a satisfactory fit is obtained. Usually this approach results in a unique function for every pixel in the image from which the gradient can be computed. This variable-order surface fitting method has been used by Besl and Jain [A.1] for the segmentation of images with good success. However an alternative to continuously computing a new function at each location is to assume a specific order of function holds for the image as a whole and compute a generalised algebraic equation that uses weighted pixel intensities to evaluate the local gradient.

A.2.1 Curve Fitting Formulation

Let W be a window ($m \times n$) in the Euclidean image plane to which a 2-D function or surface is to be fitted. It can be shown [A.1] that an order N polynomial surface function, of the general form

$$S_N(x, y) = \sum_{k=0}^N \sum_{l=0}^N c_{kl} x^k y^l \quad \text{eq. A.1}$$

where c_{kl} are the polynomial coefficients and where the total number of coefficients is given by

$$N_c = \sum_{k=0}^N (k+1) \quad \text{eq. A.2}$$

Equation A.1 is linearly parameterisable and can be written in matrix form as:

$$\mathbf{Ax} = \mathbf{b} \quad \text{eq. A.3}$$

where: $\mathbf{b}^T = [I_{11} \ I_{12} \ \cdots \ I_{mn}]$, I_{ij} = intensity at each (x, y) location in W
 $\mathbf{x}^T = [c_1 \ c_2 \ \cdots \ c_{N_c}]$, c_i = polynomial coefficients, and \mathbf{A} = position coefficient matrix

The number of discrete locations in W must be at least as large as the number of polynomial coefficients for there to be a unique solution. Generally there are more rows than columns in \mathbf{A} and thus the system is *over-determined*. A best solution can be sought by choosing vector \mathbf{x} such that the residual vector \mathbf{R} given by equation A.4, is minimised in some way.

$$\mathbf{R} = \mathbf{Ax} - \mathbf{b} \quad \text{eq. A.4}$$

The most commonly used numerical measure of goodness-of-fit is the least-squares criterion. A unique least-squares solution for an arbitrary right-hand side can be obtained using the *Moore-Penrose* generalised inverse [A.2]

$$\mathbf{x} = \mathbf{A}^{\dagger} \mathbf{b} \quad \text{eq. A.5}$$

where $\mathbf{A}^{\dagger} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$

A.2.2 Application to Gradient Masks

Taking an N^{th} order 2-D polynomial

$$S_N(x, y) = c_{00} + c_{10}x + c_{01}y + c_{20}x^2 + c_{02}y^2 + c_{11}xy + \cdots \quad \text{eq. A.6}$$

and partially differentiating with respect to x and y gives

$$\frac{\partial S_N}{\partial x} = c_{10} + 2c_{20}x + c_{11}y + \cdots \quad \text{eq. A.7}$$

$$\frac{\partial S_N}{\partial y} = c_{01} + 2c_{02}y + c_{11}x + \cdots \quad \text{eq. A.8}$$

Evaluating these partial derivatives at the origin (0,0) results in,

$$\left. \frac{\partial S_N}{\partial x} \right|_{(0,0)} = c_{10} \quad \text{and} \quad \left. \frac{\partial S_N}{\partial y} \right|_{(0,0)} = c_{01} \quad \text{eq. A.9}$$

This implies that by choosing the geometric centre as the origin when fitting the surface polynomial to an arbitrary window, the linear coefficients c_{10} and c_{01} represent approximations to the x and y gradients at the centre of the window. This approach can be extended (with caution) to higher order gradients.

If a generalised inverse is used to solve the polynomial fitting problem, then the polynomial coefficients and hence the gradient estimates can be found for any arbitrary \mathbf{b} vector of pixel intensities.

Alternative Orthogonal Formulation

If the window W is square (as is commonly the case) then there is an alternative set of orthogonal axes to those aligned with the pixel grid. These axes are aligned with the diagonals of the grid and due to symmetry should be of equal merit.

Mathematically using the diagonal axes can be realised by rotating the original axes by 45° about the origin of the window [A.3],

$$\begin{bmatrix} x \\ y \end{bmatrix}_{New} = \begin{bmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{Old} \quad \text{eq. A.10}$$

and then using these new co-ordinates to compute the position matrix \mathbf{A}_{45} . From this the polynomial surface coefficient matrix can be obtained using a generalised inverse.

A.2.3 Gradient Mask Results

Shown below in table A.1 are the gradient mask pairs for a 2x2 window fitted with a linear surface and 3x3 window fitted with quadratic surface, using the generalised inverse approach of equation A.5.

Table A.1	2x2 Window	3x3 Window
0° gradient	$\mathbf{G}_0^2 = \frac{1}{4} \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}$	$\mathbf{G}_0^3 = \frac{1}{6} \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$
90° gradient	$\mathbf{G}_{90}^2 = \frac{1}{4} \begin{bmatrix} -1 & -1 \\ +1 & +1 \end{bmatrix}$	$\mathbf{G}_{90}^3 = \frac{1}{6} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$
+45° gradient	$\mathbf{G}_{+45}^2 = \frac{1}{2} \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$	$\mathbf{G}_{+45}^3 = \frac{1}{8} \begin{bmatrix} 0 & +1 & +2 \\ -1 & 0 & +1 \\ -2 & -1 & 0 \end{bmatrix}$
-45° gradient	$\mathbf{G}_{-45}^2 = \frac{1}{2} \begin{bmatrix} -1 & 0 \\ 0 & +1 \end{bmatrix}$	$\mathbf{G}_{-45}^3 = \frac{1}{8} \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & +1 \\ 0 & +1 & +2 \end{bmatrix}$

Table A.1: Small window gradient mask pairs

Several of the well known gradient masks appear in this set. In particular the Robert's Product \mathbf{G}_{45}^2 and \mathbf{G}_{-45}^2 , smoothed Prewitt \mathbf{G}_0^3 and \mathbf{G}_{90}^3 , and the modified Sobel \mathbf{G}_{45}^3 and \mathbf{G}_{-45}^3 (see chapter 3 section 3.2.1 for full details and references).

The output of the modified Sobel masks were combined in a root-mean-square to produce the gradient enhanced images of section 7.2.4 in chapter 7.

A.3 References

- [A.1] **Besl P.J. and Jain R.C.** *Segmentation Through Variable-Order Surface Fitting.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 2, pp 167..92, March 1988.
- [A.2] **Ben-Israel A. and Greville T.N.E.** *Generalised Inverses - Theory and Applications.* Wiley Interscience, pp. 103..158, 1974.
- [A.3] *Numerical Analysis*, 2nd Ed., Schaum's Outline series in Mathematics. McGraw-Hill, pp 405..449, 1982.

Appendix B

Computational Cost Calculations

B.1 Introduction

This appendix contains details of the computational cost calculations for the four match statistics described in chapter 5 and also the complete motion-tracking algorithm described in chapter 7.

B.2 Match Statistic Calculations

The first of the match statistics is the mean absolute difference (*MAD*) given by

$$MAD = \frac{1}{mn} \sum_{i=0}^m \sum_{j=0}^n |A_k(i, j) - A_{k+1}(i, j)| \quad eq. B.1$$

At each pixel in the $m \times n$ window one subtraction is performed, followed by an absolute function. If an absolute function is computationally equated to half an addition and a subtraction equal to an addition, then so far there are $1.5 mn$ additions. The absolute pixel differences are then accumulated, another mn additions and finally the result is divided by the number of pixels, one multiplication and one division. The division is a floating point operation and can be equated to approximately 10 long integer multiplications on modern micro processors. Thus the total computational cost of the *MAD* match is:

$$\text{Additions} = 1.5 mn + mn = 2.5 mn \quad \text{Multiplications} = 1 + 10 = 11$$

The second match statistic is the root-mean-square (*RMS*) and is given by

$$RMS = \frac{1}{mn} \sqrt{\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) - A_{k+1}(i, j)]^2} \quad eq. B.2$$

At each pixel in the $m \times n$ window one subtraction is performed and the result is multiplied by itself to form the square resulting in mn additions and mn multiplications. The squared pixel differences are then accumulated, another mn additions and the square root of the accumulated result taken. The square root is another floating point operation and can be equated to approximately 15 long integer multiplications. Finally the result is divided by the number of pixels, one multiplication and one floating point division equivalent. This gives the estimates of the total computational cost of the *RMS* match as:

$$\text{Additions} = mn + mn = 2mn \quad \text{Multiplications} = mn + 15 + 10 = mn + 25$$

The third match statistics investigated in chapter 5 was the normalised product correlation (*Corr*) which is given by equation B3.

$$\text{Corr} = \frac{\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j) * A_{k+1}(i, j)]}{\sqrt{\left(\sum_{i=0}^m \sum_{j=0}^n [A_k(i, j)]^2 * \sum_{i=0}^m \sum_{j=0}^n [A_{k+1}(i, j)]^2 \right)}} \quad \text{eq. B.3}$$

The top line of this equation requires $m \times n$ multiplications and mn additions. The bottom line requires each value in the target and reference window to be squared and accumulated adding $2 \times (mn \text{ multiplications} + mn \text{ additions})$. A product of these resultant sums is then formed and the square root taken, another $1+15$ long integer multiplication equivalents. Finally the top line result is divided by the bottom line, adding a further 10 multiplications to produce a total computational cost of:

$$\text{Additions} = mn + 2mn = 3mn \quad \text{Multiplications} = mn + 2mn + 16 + 10 = 3mn + 26$$

The forth and final match statistic evaluated in chapter 5 was based on a normalised product correlation of the first six central moment invariants listed below in table B.1.

Second-order moments ($u+v=2$)	$\phi_1 = \mu_{2,0} + \mu_{0,2}$ $\phi_2 = (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2$
Third-order moments ($u+v=3$)	$\phi_3 = (\mu_{3,0} - 3\mu_{1,2})^2 + (\mu_{0,3} - 3\mu_{2,1})^2$ $\phi_4 = (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{0,3} + \mu_{2,1})^2$ $\phi_5 = (\mu_{3,0} - 3\mu_{1,2})(\mu_{3,0} + \mu_{1,2}) * [(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2]$ $+ (\mu_{0,3} - 3\mu_{2,1})(\mu_{0,3} + \mu_{2,1}) * [(\mu_{0,3} + \mu_{2,1})^2 - 3(\mu_{1,2} + \mu_{3,0})^2]$ $\phi_6 = (\mu_{2,0} - \mu_{0,2})[(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2] + 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{0,3} + \mu_{2,1})$

Table B.1: Central moment invariants

The central moments are computed from equation B.5 using the centroid coordinates calculated from the zero and first order moments of equation B.4. If the centroid coordinates are approximated by rounding to the nearest integer then the majority of the remaining operations can be formed in integer arithmetic and not floating point. On this basis the computational cost of second and third order central moments has been calculated and is summarised in table B.2.

$$m_{u,v} = \sum_R \sum f(x, y) x^u y^v \quad u, v = 0, 1, 2, \dots \quad eq. \ B.4$$

$$\mu_{p,q} = \sum_R \sum [(x - \bar{x})^p (y - \bar{y})^q f(x, y)] \quad eq. \ B.5$$

where $\bar{x} = m_{1,0} / m_{0,0}$ and $\bar{y} = m_{0,1} / m_{0,0}$.

Central moment	Computational Cost	
	Additions	Multiplications
μ_{11}	$3 \ mn$	$2 \ mn$
μ_{02}	mn	mn
μ_{20}	mn	mn
μ_{12}	$3 \ mn$	$3 \ mn$
μ_{21}	$3 \ mn$	$3 \ mn$
μ_{03}	mn	$3 \ mn$
μ_{30}	mn	$3 \ mn$

Table B.2: Second and third order central moments - computational cost

Combining the central moments together to produce the first six moment invariants adds further computational cost. The total cost of each moment invariant is summarised in table B.3 below.

Central moment invariant	Total Computational Cost	
	Additions	Multiplications
ϕ_1	$2 \ mn + 1$	$2 \ mn$
ϕ_2	$5 \ mn + 2$	$4 \ mn + 3$
ϕ_3	$5 \ mn + 3$	$9 \ mn + 4$
ϕ_4	$5 \ mn + 3$	$9 \ mn + 2$
ϕ_5	$8 \ mn + 10$	$12 \ mn + 12$
ϕ_6	$8 \ mn + 7$	$12 \ mn + 6$
Total Sum	$33 \ mn + 26$	$48 \ mn + 27$

Table B.3: Central moment invariant total computational cost

The final stage of the moment invariant match statistic calculation is to combine the six invariants in a normalised product correlation as given by equation B.6.

$$MCor = \frac{\sum_{i=1}^6 [\phi_k(i) * \phi_{k+1}(i)]}{\sqrt{\left(\sum_{i=1}^6 [\phi_k(i)]^2 * \sum_{i=1}^6 [\phi_{k+1}(i)]^2 \right)}} \quad eq. \ B.6$$

search space location need to be recomputed. Also, with good motion prediction further reductions can be made as the translation and rotation range that define the sub-region in the search space will be smaller.

Before each interframe match of the motion tracking algorithm is made, the image is median filtered to reduce noise. Only the region of interest containing the features needs to be filtered. This is typically only a third of the total image area. A fast histogram modification [B.1, B.2] based implementation was used. This requires approximately the equivalent of 8 long integer multiplications per pixel. Thus for an image 255x255 pixels the total equivalent computational cost per frame is

$$\text{Multiplications} = 8 \times \underbrace{\left(\frac{1}{3}\right)}_{\text{active fraction}} \times \underbrace{(255 \times 255)}_{\text{image size}} = 173000$$

This is about a third of the equivalent computational cost of a typical single feature search match.

Using the default tracking parameters, no motion prediction and with the algorithm implemented in 16-bit code, it takes approximately 2.1 seconds per feature match (six landmark points per feature) running on a 486 DX50 personal computer. Median filtering the region of interest typically take 1.5 seconds. Thus a complete cervical sequence consisting of 50 frames and with nine features of interest defined, will take 17 minutes to track. The code itself has been written to multi-task so tracking can occur in the background. This slows things down slightly but enables the analysis of the results from a previous sequence while the current sequence is tracked.

B.4 References

- [B.1] **Garibotto G. and Lambarelli L.** *Fast on-line implementation of two-dimensional median filter.* Electronics Letters Vol. 16, pp 24..25, January 1979.
- [B.2] **Ataman E., Aatre V.K. and Wong K.M.** *A fast method for real-time median filtering.* IEEE Transactions on Acoustics, Speech and Signal Processing. Vol 27, pp 13..8, 1979.

Appendix C

Pascal Source Code

This cover page has been placed here to enable quick access to this appendix.

Readers who accept the author's invitation to interact with this volume and experience the two cineradiographic sequences contained in this appendix will find this page most useful.

U.D.H.P. D



Appendix C

Pascal Source Code

This appendix contains the majority of the important pascal language (Borland Pascal 7.0) source code that was written by the author during the course of the research. It is included in its entirety for the following reasons:

☞ Completeness -

Excluding the two units WarpFtns and Opt_Tri which were produced from code fragments of C language source, all the remaining code was written by the author.

☞ Reduce the frustration of other researchers -

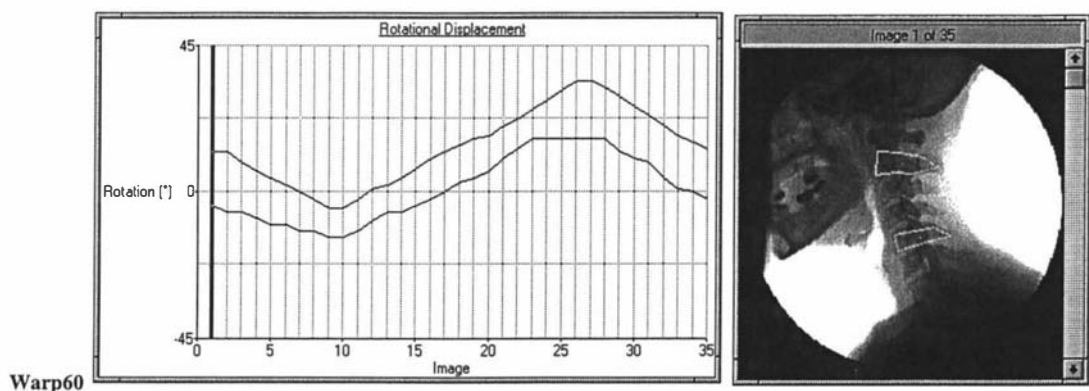
This researcher was frustrated by published code said to be complete that was full of errors. The majority of the errors were incorrect syntax, presumably due to some other person manually re-typing the source code and it not being re-checked. All the code presented in this appendix is the original working code. A special high-lighting macro was written in MS Word to format the plain text in a similar fashion to modern computer language editors. **Bold** is used for key words and *Italics* for comments.

☞ An opportunity to place movie into the thesis -

The research described in this thesis is all about motion yet without a computer it is difficult to demonstrate what is a very visual experience. This is compounded by that fact the motion of interest is contained in cineradiographic X-ray sequences of the human spine, not something of everyday experience. In an attempt to give the reader an opportunity for interaction with this piece of work two short animations have been included. This has been achieved by the age-old method of the cartoon strip.

☺ Reader Interaction Instructions

Holding the volume in its normal position and with fingers held lightly on the bottom right hand edge of this page. Gently release the pressure and allow the pages to flick by while looking at the images. Do the same thing again but this time with the volume on its back and starting from page C-70. The odd page sequence will run in reverse and contains single-cycle linear prescribed motion for C2 and C5. The even page sequence contains full prescribed motion based on trajectories measured in a real sequence.



C.1 Gradient Filters

The following Pascal unit 'GFilters' implements six different gradient filters that form the basis of the low-level edge-detection algorithms described in section 3.2.1 of chapter 3.

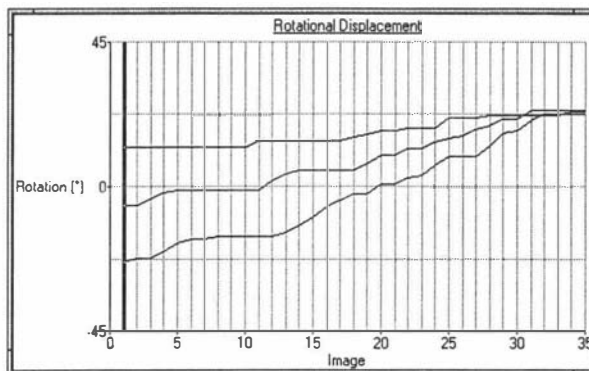
```

unit GFilters;
  { A range of gradient based image filters and support functions. }
interface
  uses
    WinCrt, WinDos, WinProcs, WinTypes,
    PMXProcs, PMXTypes, Files, Globals, Images, PMXUtils,
    ImageWHP, Filters;

  Procedure RobertsGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  Procedure PrewittGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  Procedure SobelGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  Procedure KirschGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  Procedure LaplaceGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  Procedure StochasticGradientImage(InImageH, OutImageH : ImageHandle;
    WSize, SNR : byte; Gain : real);
  Procedure LogScaleImage(InImageH, OutImageH : ImageHandle; MinValue, MaxValue : byte);

implementation
  (=====)
  Procedure RobertsGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
  { Roberts product gradient filters InImageH to produce OutImageH.  if Gain = 0 then
    the image is autoscaled after filtering, else if Gain < 0, filtered image is
    thresholded by this value else Gain > 0, filtered image scaled by the Gain, truncated
    if necessary.
  }
  var
    Thres      : byte;
    srcImage,
    destImage  : ImagePtr;
    destPixelPtr,
    srcPixelPtr : PByte;
    xSum,
    ySum      : Integer;
    destPixel,
    rowByteCount,
    x, y,
    MinVal, MaxVal : Word;
  begin { RobertsGradientImage }
    destImage := GlobalLock(OutImageH);
    srcImage  := GlobalLock(InImageH);
    rowByteCount := Align32(destImage^.header.size.x);
    if Gain < 0.0 then begin { Threshold filtered image with value of gain }
      Gain := abs(Gain);
      if Gain > White then Thres := White else Thres := Trunc(Gain/2);
    end;
    for y := 0 to destImage^.header.size.y - 2 do begin
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
      for x := 0 to destImage^.header.size.x - 2 do begin
        xSum := PByte(srcPixelPtr)^
          - PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
        ySum := PByte(OffsetPointer(srcPixelPtr, 1))^
          - PByte(OffsetPointer(srcPixelPtr, rowByteCount))^;
        destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
        if destPixel > Thres then destPixelPtr^ := White else destPixelPtr^ := Black;
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
      end; { for x }
    end; { for y }
  end { if Gain < 0.0 }
  else if Gain > 0 then begin { filter and scale image by gain, truncate if necessary }
    Gain := Gain / 2; { ensures unity gain response }
  end;

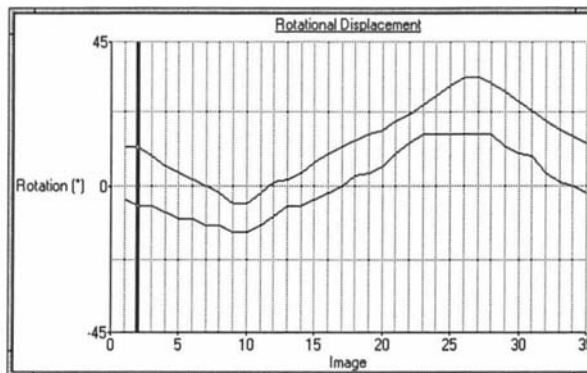
```



```

for y := 0 to destImage^.header.size.y - 2 do begin
  destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
  srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
  for x := 0 to destImage^.header.size.x - 2 do begin
    xSum := PByte(srcPixelPtr)^-PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
    ySum := PByte(OffsetPointer(srcPixelPtr, 1))^
      - PByte(OffsetPointer(srcPixelPtr, rowByteCount))^;
    destPixel := Trunc(Gain * Sqrt(Sqr(longint(xSum)) + Sqr(longint(ySum))));
    if destPixel > White then destPixelPtr^ := White
    else destPixelPtr^ := destPixel;
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
    srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
  end; {for x}
end; {for y}
end {if Gain > 0}
else begin { Gain = 0, -> Auto-scale after filtering }
  MinVal := 2*MaxInt;
  MaxVal := 0;
  for y := 0 to destImage^.header.size.y - 2 do begin { find min and max values }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
    for x := 0 to destImage^.header.size.x - 2 do begin
      xSum := PByte(srcPixelPtr)^
        - PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
      ySum := PByte(OffsetPointer(srcPixelPtr, 1))^
        - PByte(OffsetPointer(srcPixelPtr, rowByteCount))^;
      destPixel := Trunc(Sqrt(Sqr(longint(xSum)) + Sqr(longint(ySum))));
      if destPixel > MaxVal then MaxVal := destPixel;
      if destPixel < MinVal then MinVal := destPixel;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
  writeln(MaxVal, ' ', MinVal);
  for y := 0 to destImage^.header.size.y - 2 do begin { filter and scale the image }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
    for x := 0 to destImage^.header.size.x - 2 do begin
      xSum := PByte(srcPixelPtr)^
        - PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
      ySum := PByte(OffsetPointer(srcPixelPtr, 1))^
        - PByte(OffsetPointer(srcPixelPtr, rowByteCount))^;
      destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      destPixelPtr^ := (longint(White)*(destPixel - MinVal)) div (MaxVal-MinVal);
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
end; { else -> Gain < 0 }
GlobalUnlock(InImageH);
GlobalUnlock(OutImageH);
end; {RobertsGradientImage}
(-----)
Procedure PrewittGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
{ Prewitt gradient filters InImageH to produce OutImageH. If Gain = 0 then the image
  is autoscaled after filtering, else if Gain < 0, filtered image is thresholded by
  this value else Gain > 0, ifiltered image scaled by the Gain, truncated if necessary.)
var
  srcPixel,
  Thres      : Byte;
  srcImage,
  destImage  : ImagePtr;
  destPixelPtr,
  srcPixelPtr : PByte;
  xSum,
  ySum       : Integer;
  destPixel,
  rowByteCount,
  x, y,
  MinVal, MaxVal : Word;

```

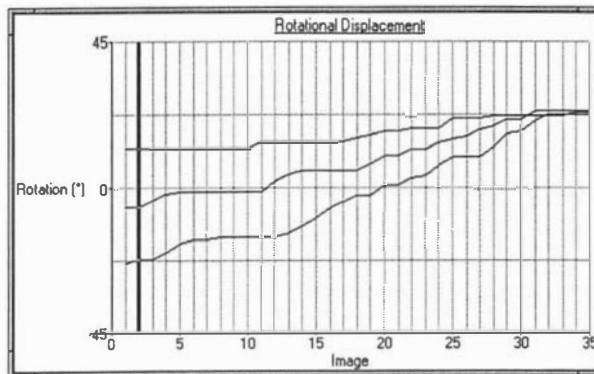



```

Procedure CalcGradient;
begin
  xSum := 0;
  ySum := 0;
  Inc(xSum, Word(PByte(OffsetPointer(srcPixelPtr, - 1))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount - 1))^;
  Inc(xSum, srcPixel);
  Inc(ySum, srcPixel);
  Inc(ySum, Word(PByte(OffsetPointer(srcPixelPtr, - rowByteCount))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount - 1))^;
  Inc(xSum, srcPixel);
  Dec(ySum, srcPixel);
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount + 1))^;
  Dec(xSum, srcPixel);
  Inc(ySum, srcPixel);
  Dec(ySum, Word(PByte(OffsetPointer(srcPixelPtr, rowByteCount))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
  Dec(xSum, srcPixel);
  Dec(ySum, srcPixel);
  Dec(xSum, Word(PByte(OffsetPointer(srcPixelPtr, 1))^));
end; {CalcGradient}

begin {PrewittGradientImage}
  destImage := GlobalLock(OutImageH);
  srcImage := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  if Gain < 0.0 then begin {threshold image after filter}
    Gain := abs(Gain);
    if Gain > White then Thres := White else Thres := Trunc(Gain);
    for y := 1 to destImage^.header.size.y - 2 do begin
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        CalcGradient;
        destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
        if destPixel > 3 * Thres then destPixelPtr^ := White
        else destPixelPtr^ := Black;
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
      end; {for x}
    end; {for y}
  end (if Gain < 0 )
  else if Gain > 0.0 then begin { filter and scale image by gain, truncate if necessary }
    Gain := Gain / 3; { ensures unity response }
    for y := 1 to destImage^.header.size.y - 2 do begin
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        CalcGradient;
        destPixel := Trunc(Gain * Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
        if destPixel > White then destPixelPtr^ := White
        else destPixelPtr^ := destPixel;
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
      end; {for x}
    end; {for y}
  end (else if Gain > 0.0 )
  else begin { Gain = 0.0 -> autoscale }
    MinVal := 2*MaxInt;
    MaxVal := 0;
    for y := 1 to destImage^.header.size.y - 2 do begin { find min and max values }
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        CalcGradient;
        destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
        if destPixel > MaxVal then MaxVal := destPixel;
        if destPixel < MinVal then MinVal := destPixel;
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
      end; {for x}
    end; {for y}
  end; { for y }

```



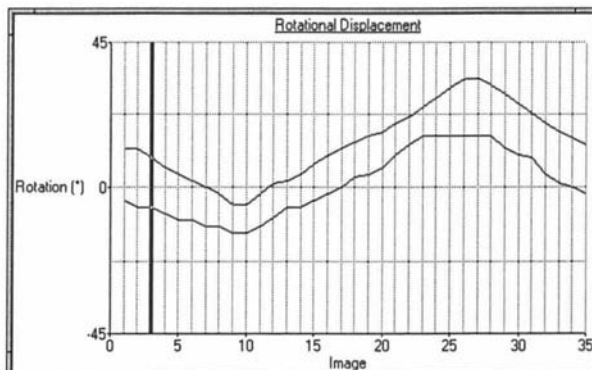
```

for y := 1 to destImage^.header.size.y - 2 do begin
  destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
  srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
  for x := 1 to destImage^.header.size.x - 2 do begin
    CalcGradient;
    destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
    destPixelPtr^ := (longint(White)*(destPixel - MinVal)) div (MaxVal-MinVal);
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
    srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
  end; {for x}
end; {for y}
end; { else Gain = 0.0 }
GlobalUnlock(InImageH);
GlobalUnlock(OutImageH);
end; {PrewittGradientImage}
{-----}
Procedure SobelGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
{ Sobel gradient filters InImageH to produce OutImageH. If Gain = 0 then the image is
  autoscaled after filtering, else if Gain < 0, filtered image is thresholded by this
  value else Gain > 0, filtered image scaled by the Gain, truncated if necessary.
}
var
  srcPixel,
  Thres      : Byte;
  srcImage,
  destImage  : ImagePtr;
  destPixelPtr,
  srcPixelPtr : PByte;
  xSum,
  ySum       : Integer;
  destPixel,
  rowByteCount,
  x,y,
  MinVal, MaxVal : Word;

Procedure CalcGradient;
begin
  xSum := 0;
  ySum := 0;
  Inc(xSum, 2 * Word(PByte(OffsetPointer(srcPixelPtr, - 1))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount - 1))^;
  Inc(xSum, srcPixel);
  Inc(ySum, srcPixel);
  Inc(ySum, 2 * Word(PByte(OffsetPointer(srcPixelPtr, - rowByteCount))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount - 1))^;
  Inc(xSum, srcPixel);
  Dec(ySum, srcPixel);
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount + 1))^;
  Dec(xSum, srcPixel);
  Inc(ySum, srcPixel);
  Dec(ySum, 2 * Word(PByte(OffsetPointer(srcPixelPtr, rowByteCount))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
  Dec(xSum, srcPixel);
  Dec(ySum, srcPixel);
  Dec(xSum, 2 * Word(PByte(OffsetPointer(srcPixelPtr, 1))^));
end; {CalcGradient}

begin {SobelGradientImage}
  destImage := GlobalLock(OutImageH);
  srcImage := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  if Gain < 0.0 then begin { threshold image after filter }
    Gain := abs(Gain);
    if Gain > White then Thres := White else Thres := Trunc(Gain);
    for y := 1 to destImage^.header.size.y - 2 do begin
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        CalcGradient;
        destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
        if destPixel > 5 * Thres then destPixelPtr^ := White

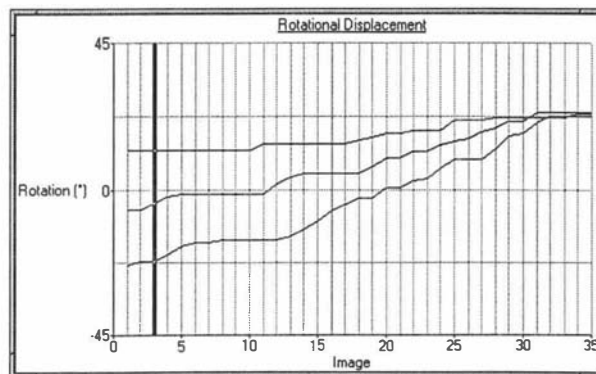
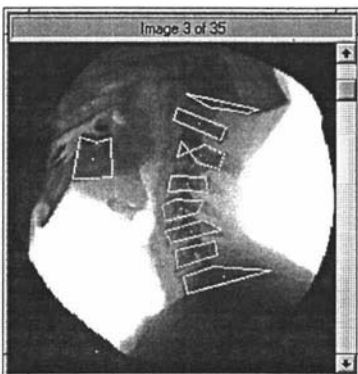
```



```

    else destPixelPtr^ := Black;
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
    srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
  end; {for x}
end; {for y}
end {if Gain < 0.0}
else if Gain > 0.0 then begin { filter and scale image by gain, truncate if necessary}
  Gain := Gain / 5; { ensures unity response }
  for y := 1 to destImage^.header.size.y - 2 do begin
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
    for x := 1 to destImage^.header.size.x - 2 do begin
      CalcGradient;
      destPixel := Trunc(Gain * Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      if destPixel > White then destPixelPtr^ := White
      else destPixelPtr^ := destPixel;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
end {else if Gain > 0.0}
else begin { Gain = 0.0 -> autoscale after filter }
  MinVal := 2*MaxInt;
  MaxVal := 0;
  for y := 1 to destImage^.header.size.y - 2 do begin { find min and max values }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
    for x := 1 to destImage^.header.size.x - 2 do begin
      CalcGradient;
      destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      if destPixel > MaxVal then MaxVal := destPixel;
      if destPixel < MinVal then MinVal := destPixel;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
  writeln(MaxVal, ' ', MinVal);
  for y := 1 to destImage^.header.size.y - 2 do begin { filter and scale the image }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
    for x := 1 to destImage^.header.size.x - 2 do begin
      CalcGradient;
      destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      destPixelPtr^ := (longint(White)*(destPixel - MinVal)) div (MaxVal-MinVal);
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
end; {else Gain = 0.0}
GlobalUnlock(InImageH); GlobalUnlock(OutImageH);
end; {SobelGradientImage}
(-----)
Procedure KirschGradientImage(InImageH, OutImageH : ImageHandle; Gain : real);
{ Kirsch compass gradient filters InImageH to produce OutImageH. If Gain = 0 then the
  image is autoscaled after filtering, else if Gain < 0, filtered image is thresholded
  by this value else Gain > 0, filtered image scaled by the Gain, truncated if necessary
}
var
  srcPixel,
  Thres      : Byte;
  srcImage,
  destImage  : ImagePtr;
  destPixelPtr,
  srcPixelPtr : PByte;
  xSum1, ySum1,
  xSum2, ySum2 : Integer;
  destPixel,
  rowByteCount,
  x, y,
  MinVal, MaxVal : Word;

```



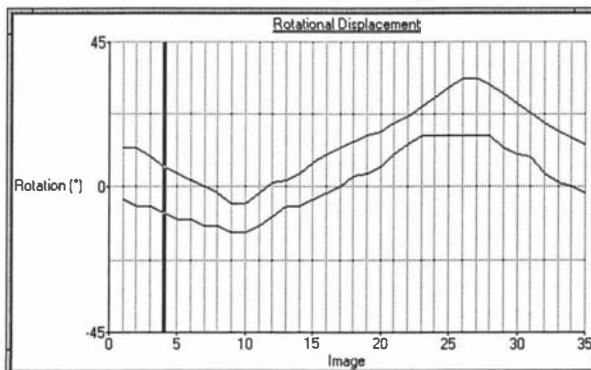

```

Procedure CalcGradient;
begin
  xSum1 := 0;
  ySum1 := 0;
  Inc(xSum1, Word(PByte(OffsetPointer(srcPixelPtr, - 1))^)); {4 Kirsch masks}
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount - 1))^; { 1 0 -1 }
  Inc(xSum1, srcPixel); { 1 0 -1 }
  Inc(ySum1, srcPixel); { 1 0 -1 }
  Inc(ySum1, Word(PByte(OffsetPointer(srcPixelPtr, - rowByteCount))^)); { 1 1 1 }
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount - 1))^; { 0 0 0 }
  Inc(xSum1, srcPixel); { -1 -1 -1 }
  Dec(ySum1, srcPixel);
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount + 1))^;
  Dec(xSum1, srcPixel);
  Inc(ySum1, srcPixel);
  Dec(ySum1, Word(PByte(OffsetPointer(srcPixelPtr, rowByteCount))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^;
  Dec(xSum1, srcPixel);
  Dec(ySum1, srcPixel);
  Dec(xSum1, Word(PByte(OffsetPointer(srcPixelPtr, 1))^));
  xSum1 := abs(xSum1);
  ySum1 := abs(ySum1);
  xSum2 := 0;
  ySum2 := 0;
  Inc(xSum2, Word(PByte(OffsetPointer(srcPixelPtr, - rowByteCount - 1))^)); { 1 1 0 }
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - rowByteCount))^; { 1 0 -1 }
  Inc(xSum2, srcPixel); { 0 -1 -1 }
  Inc(ySum2, srcPixel);
  Inc(ySum2, Word(PByte(OffsetPointer(srcPixelPtr, - rowByteCount + 1))^)); { 0 1 1 }
  srcPixel := PByte(OffsetPointer(srcPixelPtr, - 1))^; {-1 0 1 }
  Inc(xSum2, srcPixel); {-1 -1 0 }
  Dec(ySum2, srcPixel);
  srcPixel := PByte(OffsetPointer(srcPixelPtr, + 1))^;
  Dec(xSum2, srcPixel);
  Inc(ySum2, srcPixel);
  Dec(ySum2, Word(PByte(OffsetPointer(srcPixelPtr, rowByteCount - 1))^));
  srcPixel := PByte(OffsetPointer(srcPixelPtr, rowByteCount))^;
  Dec(xSum2, srcPixel);
  Dec(ySum2, srcPixel);
  Dec(xSum2, Word(PByte(OffsetPointer(srcPixelPtr, rowByteCount + 1))^));
  xSum2 := abs(xSum2);
  ySum2 := abs(ySum2);
  { find largest partial gradient }
  destPixel := xSum1;
  if ySum1 > destPixel then destPixel := ySum1;
  if xSum2 > destPixel then destPixel := xSum2;
  if ySum2 > destPixel then destPixel := ySum2;
end; {CalcGradient}

begin {KirschGradientImage}
  destImage := GlobalLock(OutImageH);
  srcImage := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  if Gain < 0.0 then begin { threshold image after filter }
    Gain := abs(Gain);
    if Gain > White then Thres := White else Thres := Trunc(Gain);
    for y := 1 to destImage^.header.size.y - 2 do begin
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        CalcGradient;
        if destPixel > 5 * Thres then destPixelPtr^ := White
        else destPixelPtr^ := Black;
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
      end; {for x}
    end; {for y}
  end {if Gain < 0.0 }
  else if Gain > 0.0 then begin { filter and scale image by gain, truncate if necessary}
    Gain := Gain / 5; { ensures unity response }
  end;

```

Warp60




```

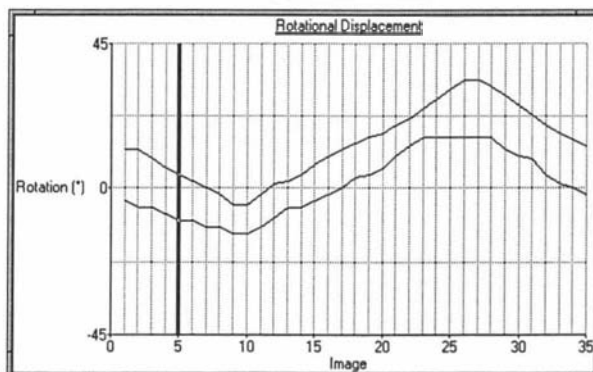
Error      : ErrorCode;

begin (LaplaceGradientImage)
  destImage := GlobalLock(OutImageH);
  srcImage  := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  Error := CreateIntImage(intImageH, destImage^.header.size.x, destImage^.header.size.y);
  if Error <> NoErr then begin
    writeln('Insufficient memory');
    Exit;
  end;
  intImage := GlobalLock(intImageH);
  Width := destImage^.header.size.x;
  MinVal := MaxInt;
  MaxVal := - MinVal;
  for y := 1 to destImage^.header.size.y - 2 do begin
    intPixelPtr := OffsetPointer(@intImage^.data, (longint(y) * Width + 1) * SizeOfInt);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + 1);
    for x := 1 to destImage^.header.size.x - 2 do begin
      destPixel :=
        - PByte(OffsetPointer(srcPixelPtr, - rowByteCount - 1))^      ( -1 -1 -1 )
        - PByte(OffsetPointer(srcPixelPtr, - rowByteCount))^          ( -1  0 -1 )
        - PByte(OffsetPointer(srcPixelPtr, - rowByteCount + 1))^      ( -1 -1 -1 )
        + 8 * PByte(srcPixelPtr)^
        - PByte(OffsetPointer(srcPixelPtr, + 1))^
        - PByte(OffsetPointer(srcPixelPtr, + rowByteCount - 1))^
        - PByte(OffsetPointer(srcPixelPtr, + rowByteCount))^
        - PByte(OffsetPointer(srcPixelPtr, + rowByteCount + 1))^;
      if destPixel > MaxVal then MaxVal := destPixel;
      if destPixel < MinVal then MinVal := destPixel;
      intPixelPtr^ := destPixel;
      intPixelPtr := OffsetPointer(intPixelPtr, SizeOfInt);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; (for x)
  end; (for y)

  If Gain < 0.0 then begin ( find all zero-crossings and set to White )
    Thres := Trunc(Gain); ( internal +/- threshold for zero-crossing detection )
    ( A zero-crossing is declare if in any direction -ve 0 +ve )
    Position[1].min := - Width - 1; Position[1].max := + Width + 1; ( + diagonal )
    Position[2].min := + Width - 1; Position[2].max := - Width + 1; ( - diagonal )
    Position[3].min := - Width; Position[3].max := + Width; ( up/down )
    Position[4].min := - 1; Position[4].max := + 1; ( left/right )
    for y := 1 to destImage^.header.size.y - 2 do begin
      intPixelPtr :=
        OffsetPointer(@intImage^.data, (longint(y) * Width + 1) * SizeOfInt);
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin
        ZeroCross := False;
        i := 1;
        While not(ZeroCross) and (i <= 4) do begin
          MinVal :=
            Integer(PByte(OffsetPointer(intPixelPtr, Position[i].min * SizeOfInt))^);
          MaxVal :=
            Integer(PByte(OffsetPointer(intPixelPtr, Position[i].max * SizeOfInt))^);
          if ((MinVal < -Thres) and (MaxVal > Thres))
            or ((MinVal > Thres) and (MaxVal < -Thres)) then ZeroCross := True;
          inc(i);
        end; (while)
        if ZeroCross then destPixelPtr^ := White else destPixelPtr^ := Black;
        intPixelPtr := OffsetPointer(intPixelPtr, SizeOfInt);
        destPixelPtr := OffsetPointer(destPixelPtr, 1);
      end; (for x)
    end; (for y)
  end ( If Gain < 0.0 )
  else if Gain > 0.0 then begin ( fixed scale by gain )
    for y := 1 to destImage^.header.size.y - 2 do begin
      intPixelPtr := OffsetPointer(@intImage^.data, (longint(y) * Width + 1)*SizeOfInt);
      destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
      for x := 1 to destImage^.header.size.x - 2 do begin

```

Warp60



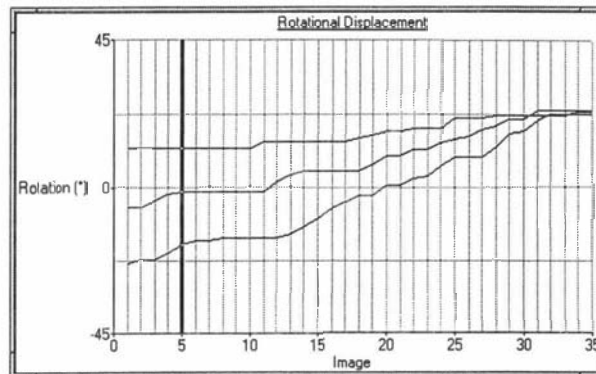
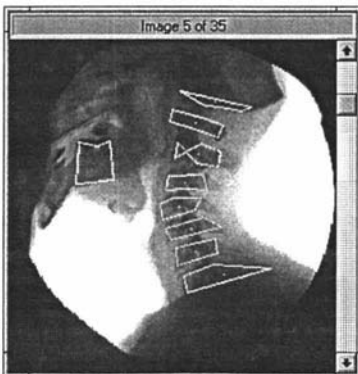
```

destPixel :=
  Trunc((longint(White)*(Gain*integer(intPixelPtr^) - MinVal))/(MaxVal-MinVal));
If destPixel > White then destPixel := White
else if destPixel < Black then destPixel := Black;
destPixelPtr^ := destPixel;
intPixelPtr := OffsetPointer(intPixelPtr, SizeOfInt);
destPixelPtr := OffsetPointer(destPixelPtr, 1);
end; (for x)
end; (for y)
end ( else if Gain > 0.0 )
else begin ( Gain = 0.0 -> autoscale to byte range )
  for y := 1 to destImage^.header.size.y - 2 do begin
    intPixelPtr := OffsetPointer(@intImage^.data, (longint(y) * Width + 1)*SizeOfInt);
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount + 1);
    for x := 1 to destImage^.header.size.x - 2 do begin
      destPixelPtr^ :=
        (longint(White)*(integer(intPixelPtr^) - MinVal)) div (MaxVal-MinVal);
      intPixelPtr := OffsetPointer(intPixelPtr, SizeOfInt);
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
    end; (for x)
  end; (for y)
end; ( else Gain = 0.0)
GlobalUnlock(InImageH);
GlobalUnlock(OutImageH);
DestroyIntImage(intImageH);
end; (LaplaceGradientImage)
)
)
-----
Procedure StochasticGradientImage(InImageH, OutImageH : ImageHandle;
  WSize, SNR : byte; Gain : real);
( Stochastic gradient filters InImageH to produce OutImageH. If Gain = 0 then the
  image is autoscaled after filtering, else if Gain < 0, filtered image is thresholded
  by this value else Gain > 0, filtered image scaled by the Gain, truncated if
  necessary. SNR (signal-to-noise ratio) size 1 and 9 only supported.
)
var
  srcPixel,
  Thres,
  HalfWSize : Byte;
  srcImage,
  destImage : ImagePtr;
  destPixelPtr,
  srcPixelPtr : PByte;
  xSum, ySum,
  i, j : Integer;
  destPixel,
  rowByteCount,
  x, y,
  MinVal, MaxVal : Word;
  wt : Array[-2..+2, -2..+2] of Shortint;

Procedure CalcGradient;
var
  i, j : shortint;
begin
  xSum := 0;
  ySum := 0;
  for i := -HalfWSize to +HalfWSize do begin
    for j := -HalfWSize to +HalfWSize do begin
      inc(xSum, wt[i,j] * Word(PByte(OffsetPointer(srcPixelPtr, i*rowByteCount + j))^));
      inc(ySum, wt[j,i] * Word(PByte(OffsetPointer(srcPixelPtr, i*rowByteCount + j))^));
    end; (for j)
  end; (for i)
  xSum := xSum div 32;
  ySum := ySum div 32;
end; (CalcGradient)

begin (StochasticGradientImage)
  destImage := GlobalLock(OutImageH);
  srcImage := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  if WSize = 3 then begin

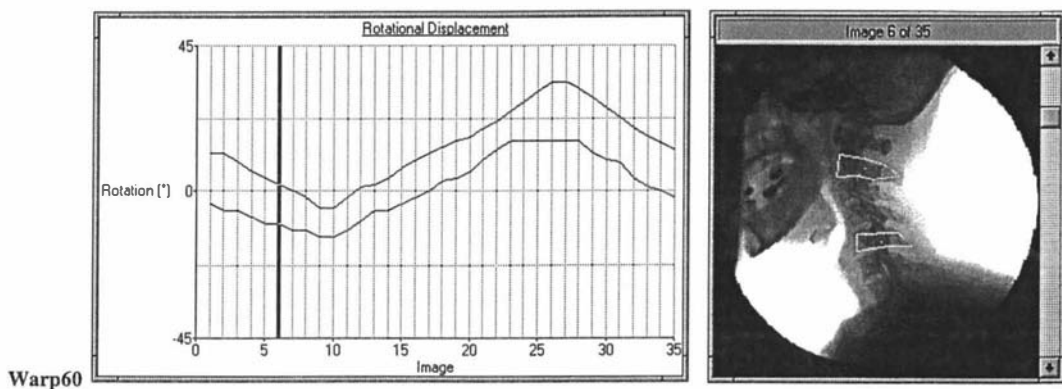
```



```

if SNR = 1 then begin
  wt[-1,-1] := 31; wt[-1,0] := 0; wt[-1,1] := -31;
  wt[0,-1] := 32; wt[0,0] := 0; wt[0,1] := -32;
  wt[1,-1] := 31; wt[1,0] := 0; wt[1,1] := -31;
end { SNR=1 }
else begin { SNR=9 }
  wt[-1,-1] := 25; wt[-1,0] := 0; wt[-1,1] := -25;
  wt[0,-1] := 32; wt[0,0] := 0; wt[0,1] := -32;
  wt[1,-1] := 25; wt[1,0] := 0; wt[1,1] := -25;
end;
end { WSize = 3 }
else begin
  if SNR = 1 then begin
    wt[-2,-2] := 26; wt[-2,-1] := 27; wt[-2,0] := 0; wt[-2,1] := -27; wt[-2,2] := -26;
    wt[-1,-2] := 27; wt[-1,-1] := 29; wt[-1,0] := 0; wt[-1,1] := -29; wt[-1,2] := -27;
    wt[0,-2] := 28; wt[0,-1] := 32; wt[0,0] := 0; wt[0,1] := -32; wt[0,2] := -28;
    wt[1,-2] := 27; wt[1,-1] := 29; wt[1,0] := 0; wt[1,1] := -29; wt[1,2] := -27;
    wt[2,-2] := 26; wt[2,-1] := 27; wt[2,0] := 0; wt[2,1] := -27; wt[2,2] := -26;
  end { SNR=1 }
  else begin { SNR=9 }
    wt[-2,-2] := 9; wt[-2,-1] := 12; wt[-2,0] := 0; wt[-2,1] := -12; wt[-2,2] := -9;
    wt[-1,-2] := 12; wt[-1,-1] := 18; wt[-1,0] := 0; wt[-1,1] := -18; wt[-1,2] := -12;
    wt[0,-2] := 15; wt[0,-1] := 32; wt[0,0] := 0; wt[0,1] := -32; wt[0,2] := -15;
    wt[1,-2] := 12; wt[1,-1] := 18; wt[1,0] := 0; wt[1,1] := -18; wt[1,2] := -12;
    wt[2,-2] := 9; wt[2,-1] := 12; wt[2,0] := 0; wt[2,1] := -12; wt[2,2] := -9;
  end;
end; { else wSize = 5 }
HalfWSize := WSize div 2;
if Gain < 0.0 then begin { threshold image after filter }
  Gain := abs(Gain);
  if Gain > White then Thres := White else Thres := Trunc(Gain);
  for y := HalfWSize to destImage^.header.size.y - 1 - HalfWSize do begin
    destPixelPtr :=
      OffsetPointer(@destImage^.data, longint(y) * rowByteCount + HalfWSize);
    srcPixelPtr :=
      OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + HalfWSize);
    for x := HalfWSize to destImage^.header.size.x - 1 - HalfWSize do begin
      CalcGradient;
      destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      if destPixel > Thres then destPixelPtr^ := White
      else destPixelPtr^ := Black;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; { for x }
  end; { for y }
end { if Gain < 0 }
else if Gain > 0.0 then begin { filter and scale image by gain, truncate if necessary }
  for y := HalfWSize to destImage^.header.size.y - 1 - HalfWSize do begin
    destPixelPtr :=
      OffsetPointer(@destImage^.data, longint(y) * rowByteCount + HalfWSize);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + HalfWSize);
    for x := HalfWSize to destImage^.header.size.x - 1 - HalfWSize do begin
      CalcGradient;
      destPixel := Trunc(Gain * Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
      if destPixel > White then destPixelPtr^ := White;
      destPixelPtr^ := destPixel;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; { for x }
  end; { for y }
end { else if Gain > 0.0 }
else begin { Gain = 0.0 -> autoscale }
  MinVal := 2*MaxInt;
  MaxVal := 0;
  for y := HalfWSize to destImage^.header.size.y-1-HalfWSize do begin
    { Find min and max values }
    destPixelPtr :=
      OffsetPointer(@destImage^.data, longint(y) * rowByteCount + HalfWSize);
    srcPixelPtr :=
      OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + HalfWSize);

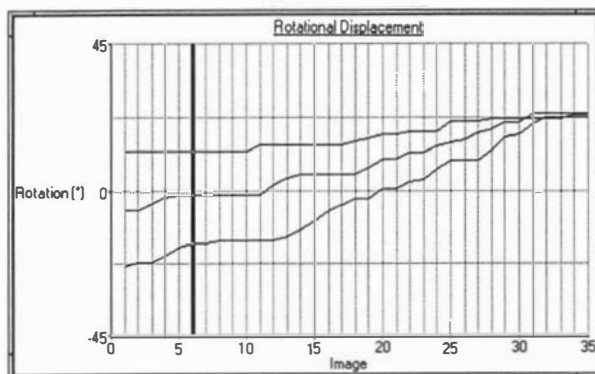
```



```

for x := HalfWSize to destImage^.header.size.x-1-HalfWSize do begin
  CalcGradient;
  destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
  if destPixel > MaxVal then MaxVal := destPixel;
  if destPixel < MinVal then MinVal := destPixel;
  destPixelPtr := OffsetPointer(destPixelPtr, 1);
  srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
end; {for x}
end; {for y}
for y := HalfWSize to destImage^.header.size.y-1-HalfWSize do begin
  destPixelPtr :=
    OffsetPointer(@destImage^.data, longint(y) * rowByteCount + HalfWSize);
  srcPixelPtr :=
    OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + HalfWSize);
  for x := HalfWSize to destImage^.header.size.x-1-HalfWSize do begin
    CalcGradient;
    destPixel := Trunc(Sqrt(Sqr(Longint(xSum)) + Sqr(Longint(ySum))));
    destPixelPtr^ := (longint(White)*(destPixel - MinVal)) div (MaxVal-MinVal);
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
    srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
  end; {for x}
end; {for y}
end; { else Gain = 0.0 }
GlobalUnlock(InImageH);
GlobalUnlock(OutImageH);
end; {StochasticGradientImage}
)-----)
Procedure LogScaleImage(InImageH, OutImageH : ImageHandle; MinValue, MaxValue : byte);
  { Logarithmically scale the values in InImageH to produce OutImageH. For X-ray }
  { images this is equivalent to a relative density image. MinValue and MaxValue set }
  { the dynamic range of OutImage. }
var
  minVal,
  maxVal,
  Value      : byte;
  destPixel  : integer;
  srcImage,
  destImage  : ImagePtr;
  destPixelPtr,
  srcPixelPtr : PByte;
  rowByteCount,
  x, y       : word;
  Log        : Array[0..255] of byte;
begin {LogScaleImage}
  destImage := GlobalLock(OutImageH);
  srcImage := GlobalLock(InImageH);
  rowByteCount := Align32(destImage^.header.size.x);
  for y := 0 to destImage^.header.size.y - 1 do begin { find min and max values }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
    for x := 0 to destImage^.header.size.x - 1 do begin
      Value := PByte(srcPixelPtr)^;
      if Value > maxVal then maxVal := Value;
      if Value < minVal then minVal := Value;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
      srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
  end; {for y}
  for x := minVal to maxVal do begin { compute the logarithmic lookup table }
    log[x] := Trunc(White * (ln(x-minVal+1)) / ln(maxVal-minVal+1));
  end; {for x}
  for y := 0 to destImage^.header.size.y - 1 do begin { log scale the image }
    destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
    srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
    for x := 0 to destImage^.header.size.x - 1 do begin
      destPixel :=
        Trunc(White * longint(log[PByte(srcPixelPtr)^] - minValue)/(maxValue-minValue));
      if destPixel > White then destPixel := White
      else if destPixel < Black then destPixel := Black;
      destPixelPtr^ := destPixel;
      destPixelPtr := OffsetPointer(destPixelPtr, 1);
    end;
  end;

```



```

        srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
    end; {for x}
end; { for y }
end; {LogScaleImage}
(=====)
end. { GFilters }

```

C.2 Canny Edge-detector

The following Pascal unit 'Canny' is an implementation of the Canny edge-detection algorithm described in detail in section 3.2.2 of chapter 3.

```

unit Canny;

interface

    uses
        WinCrt, WinDos, WinProcs, WinTypes,
        PMXProcs, PMXTypes, Files, Globals, Images, PMXUtils,
        ImageWHP, Filters;
    const
        Black = 0;
        Grey = 128;
        White = 255;

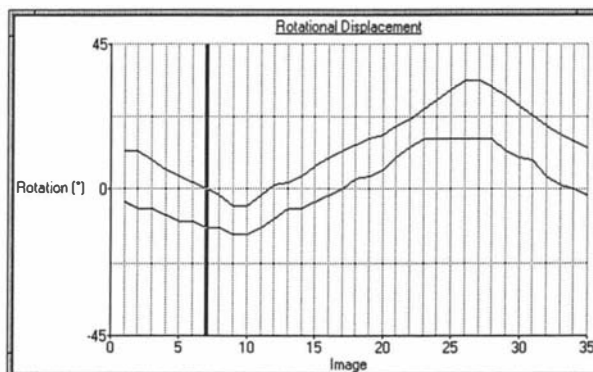
    Function GradGFilterImage(    InImageH : ImageHandle;
                                var GradImageH : RealImageHandle;
                                var DirectionImageH : ImageHandle; StDev : real;
                                FullDirection : boolean) : ErrorCode;

    Function NMSFilterImage(    InGradImageH, DirectionImageH : ImageHandle;
                                var OutGradImageH : ImageHandle): ErrorCode;

    Procedure ThresholdImage2(InImageH, OutImageH : ImageHandle);
    Procedure HysteresisThresholdImage(GradImageH, NMSGGradImageH,ThresImageH : ImageHandle);
    Function ArcTan2(var x, y : real) : real;
    Function ArcTan3(var x, y : real) : real;

implementation
(=====)
Function ArcTan2(var x, y : real) : real;
{ Proper arctan 0->2pi }
    var
        v : real;
    begin
        if x = 0.0 then begin
            if y = 0.0 then v := 0.0
            else v := pi/2 * abs(y)/y;
        end
        else v := y/x;
        if x < 0.0 then ArcTan2 := pi + arcTan(v)
        else if y < 0.0 then ArcTan2 := 2*pi + arcTan(v)
        else ArcTan2 := arcTan(v);
    end;
(-----)
Function ArcTan3(var x, y : real) : real;
{ Mirror Arctan, 0-2pi becomes: 0->pi/2->pi->pi/2->0 }
    var
        v : real;
    begin
        if x = 0.0 then begin
            if y = 0.0 then v := 0.0

```



```

    else v := pi/2 * abs(y)/y;
  end
else v := y/x;
if x < 0.0 then begin
  if y < 0.0 then ArcTan3 := pi - arcTan(v)
  else ArcTan3 := pi + arcTan(v)
  end
else ArcTan3 := arcTan(abs(v));
end;
end;
)-----)
Function GradGFilterImage(InImageH : ImageHandle;
  var GradImageH : RealImageHandle;
  var DirectionImageH : ImageHandle; StDev : real;
  FullDirection : boolean) : ErrorCode;

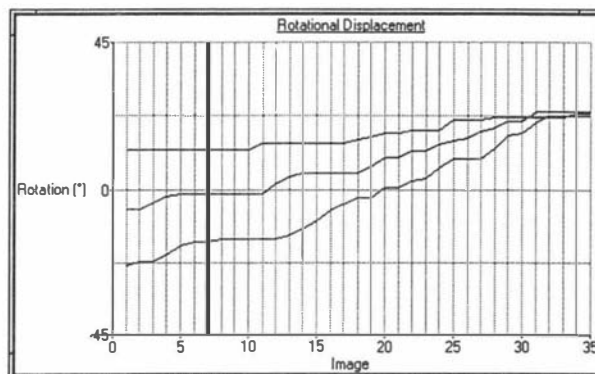
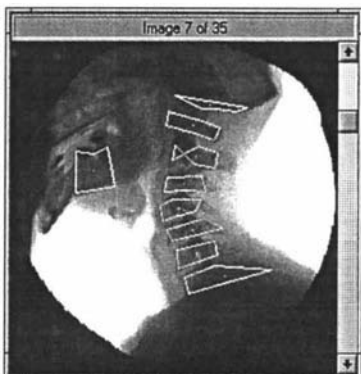
const
  WinRadiusToStDevRatio = 3.0;
label
  ExitPoint;
type
  MaskArray = Array[-12..12,-12..12] of Real;
var
  InPixel,
  WinRadius,
  dx, dy      : integer;
  InImage      : ImagePtr;
  GradImage    : RealImagePtr;
  DirectionImage
               : ImagePtr;
  GradImageCreated
               : boolean;

  TopLeft,
  BottomRight : TPoint;
  InPixelPtr,
  DirectionPixelPtr
               : PByte;

  GradPixelPtr
               : RealPtr;

  xGradPixel,
  yGradPixel,
  TwoS2,
  TwoPiS4      : real;
  rowByteCount,
  x,y           : longint;
  ImageWidth,
  ImageHeight  : Word;
  Error        : ErrorCode;
  Mask         : ^MaskArray;
begin (GradGFilterImage)
  Error := NoErr;
  InImage := GlobalLock(InImageH);
  ImageWidth := InImage^.header.size.x;
  ImageHeight := InImage^.header.size.y;
  GradImageCreated := False;
  Error := CreateRealImage(GradImageH, ImageWidth, ImageHeight);
  if Error <> NoErr then goto ExitPoint;
  GradImageCreated := True;
  GradImage := GlobalLock(GradImageH);
  rowByteCount := Align32(InImage^.header.size.x);
  Error := CreateImage(DirectionImageH, rowByteCount, ImageHeight);
  if Error <> NoErr then goto ExitPoint;
  DirectionImage := GlobalLock(DirectionImageH);
  WinRadius := Round(WinRadiusToStDevRatio * StDev);
  if MaxAvail > (sqr(25) * SizeOfReal) then New(Mask)
  else begin
    Error := 1;
    goto ExitPoint;
  end;
  TopLeft.x := WinRadius;
  TopLeft.y := WinRadius;
  BottomRight.x := ImageWidth-1 - WinRadius;
  BottomRight.y := ImageHeight-1 - WinRadius;

```

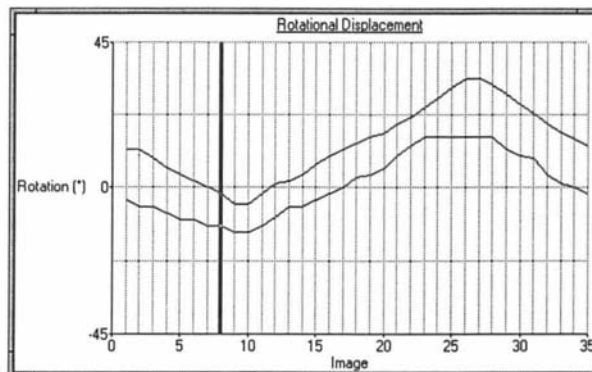



```

TwoS2 := 2 * sqr(StDev);
TwoPiS4 := 2 * pi * sqr(sqr(StDev));
writeln('Gradient of Gaussian Filtering');
writeln('Standard Deviation = ', StDev:1:2, ' Window Size = ', 2 * WinRadius + 1);
for y := -WinRadius to +WinRadius do begin ( Compute convolution mask )
  for x := -WinRadius to +WinRadius do begin
    Mask^[x,y] := -10 * x * exp(-(sqr(x) + sqr(y))/TwoS2)/TwoPiS4;
  end; (for x)
end; (for y)
for y := 0 to ImageHeight-1 do ( initial is output real images )
  for x := 0 to ImageWidth-1 do begin
    RealPtr(OffsetPointer(@GradImage^.data, (y * ImageWidth + x)*SizeOfReal))^ := 0.0;
    PByte(OffsetPointer(@DirectionImage^.data, y * rowByteCount + x))^ := Black;
  end; (for x)
if FullDirection then begin
  for y := TopLeft.y to BottomRight.y do begin
    GradPixelPtr :=
      OffsetPointer(@GradImage^.data, (y * ImageWidth + TopLeft.x) * SizeOfReal);
    DirectionPixelPtr :=
      OffsetPointer(@DirectionImage^.data, y * rowByteCount + TopLeft.x);
    InPixelPtr := OffsetPointer(@InImage^.data, y * rowByteCount + TopLeft.x);
    for x := TopLeft.x to BottomRight.x do begin
      xGradPixel := 0.0;
      yGradPixel := 0.0;
      for dy := -WinRadius to +WinRadius do begin
        for dx := -WinRadius to +WinRadius do begin
          InPixel := PByte(OffsetPointer(InPixelPtr, dy * rowByteCount + dx))^;
          xGradPixel := xGradPixel + InPixel * Mask^[dx,dy];
          yGradPixel := yGradPixel + InPixel * Mask^[dy,dx]; (y-mask=transpose x-mask)
        end; (for dx)
      end; (for dy)
      GradPixelPtr^ := sqrt(sqr(xGradPixel) + sqr(yGradPixel));
      DirectionPixelPtr^ := Byte(Round(ArcTan2(xGradPixel, yGradPixel)/(2*pi)*white));
      InPixelPtr := OffsetPointer(InPixelPtr, 1);
      GradPixelPtr := OffsetPointer(GradPixelPtr, SizeOfReal);
      DirectionPixelPtr := OffsetPointer(DirectionPixelPtr, 1);
    end; (for x)
    write('.');
  end; (for y)
end (if FullDirection)
else begin (Not FullDirection)
  for y := TopLeft.y to BottomRight.y do begin
    GradPixelPtr :=
      OffsetPointer(@GradImage^.data, (y * ImageWidth + TopLeft.x) * SizeOfReal);
    DirectionPixelPtr :=
      OffsetPointer(@DirectionImage^.data, y * rowByteCount + TopLeft.x);
    InPixelPtr := OffsetPointer(@InImage^.data, y * rowByteCount + TopLeft.x);
    for x := TopLeft.x to BottomRight.x do begin
      xGradPixel := 0.0;
      yGradPixel := 0.0;
      for dy := -WinRadius to +WinRadius do begin
        for dx := -WinRadius to +WinRadius do begin
          InPixel := PByte(OffsetPointer(InPixelPtr, dy * rowByteCount + dx))^;
          xGradPixel := xGradPixel + InPixel * Mask^[dx,dy];
          yGradPixel := yGradPixel + InPixel * Mask^[dy,dx]; (y-mask=transpose x-mask)
        end; (for dx)
      end; (for dy)
      GradPixelPtr^ := sqrt(sqr(xGradPixel) + sqr(yGradPixel));
      DirectionPixelPtr^ := Byte(Round(ArcTan3(xGradPixel, yGradPixel)/pi*white));
      InPixelPtr := OffsetPointer(InPixelPtr, 1);
      GradPixelPtr := OffsetPointer(GradPixelPtr, SizeOfReal);
      DirectionPixelPtr := OffsetPointer(DirectionPixelPtr, 1);
    end; (for x)
    write('.');
  end; (for y)
end; (else Not FullDirection)
ExitPoint:
GlobalUnlock(InImageH);
GlobalUnlock(GradImageH);
GlobalUnlock(DirectionImageH);
if (Error <> NoErr) and (GradImageCreated = True) then DestroyRealImage(GradImageH);

```

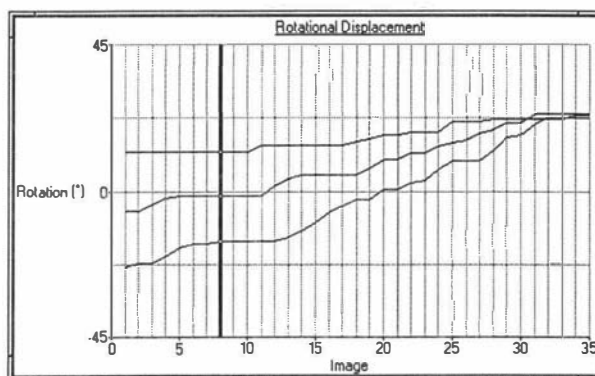
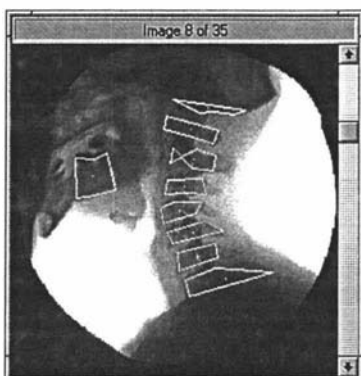
Warp60



```

    Dispose(Mask);
    GradGFilterImage := Error;
end; {GradGFilterImage}
)-----)
Function NMSFilterImage(InGradImageH, DirectionImageH : ImageHandle;
    var OutGradImageH : ImageHandle): ErrorCode;
{ Nonmaxima Suppression of InGradImage using DirectionImageH. }
label
    ExitPoint;
const
    DeltaTheta = 31; {Tolerance on Direction alignment}
var
    InGradImage,
    OutGradImage,
    DirectionImage
        : ImagePtr;
    OutImageCreated
        : boolean;
    InPixelPtr,
    OutPixelPtr,
    DirectionPixelPtr
        : PByte;
    xGradPixel,
    yGradPixel,
    destPixel,
    rowByteCount,
    x,y,
    Sum,
    AveDirection
        : longint;
    Direction
        : byte;
    ImageWidth,
    ImageHeight,
    dx, dy
        : integer;
    Value
        : Word;
    Error
        : ErrorCode;
begin {NMSFilterImage}
    writeln('Non-maxima suppression of gradient');
    Error := NoErr;
    InGradImage := GlobalLock(InGradImageH);
    ImageWidth := InGradImage^.header.size.x;
    ImageHeight := InGradImage^.header.size.y;
    DirectionImage := GlobalLock(DirectionImageH);
    OutImageCreated := False;
    rowByteCount := Align32(InGradImage^.header.size.x);
    Error := CreateImage(OutGradImageH, rowByteCount, ImageHeight);
    if Error <> NoErr then goto ExitPoint;
    OutImageCreated := True;
    OutGradImage := GlobalLock(OutGradImageH);
    for y := 1 to ImageHeight-2 do begin
        InPixelPtr := OffsetPointer(@InGradImage^.data, y * rowByteCount + 1);
        DirectionPixelPtr := OffsetPointer(@DirectionImage^.data, y * rowByteCount + 1);
        OutPixelPtr := OffsetPointer(@OutGradImage^.data, y * rowByteCount + 1);
        for x := 1 to ImageWidth-2 do begin
            {Compute direction to nearest 45 degrees, from 8 corresponding nearest neighbours}
            Direction := (((DirectionPixelPtr^ + 15) mod 255) + 1) div 32;
            case Direction of
                0, 4: if (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, -1))^)
                    and (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, +1))^)
                    then OutPixelPtr^ := InPixelPtr^
                    else OutPixelPtr^ := Black;
                1, 5: if (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, -rowByteCount+1))^)
                    and (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, rowByteCount-1))^)
                    then OutPixelPtr^ := InPixelPtr^
                    else OutPixelPtr^ := Black;
                2, 6: if (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, -rowByteCount))^)
                    and (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, +rowByteCount))^)
                    then OutPixelPtr^ := InPixelPtr^
                    else OutPixelPtr^ := Black;
                3, 7: if (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, -rowByteCount-1))^)

```

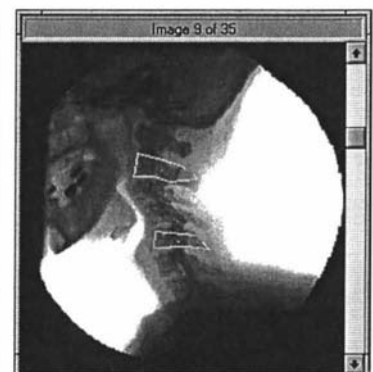
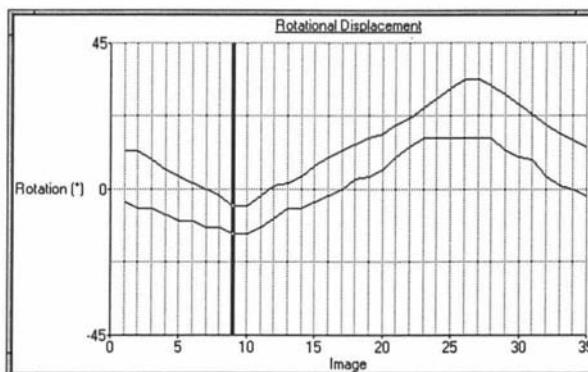


```

        and (InPixelPtr^ >= PByte(OffsetPointer(InPixelPtr, rowByteCount+1))^)
        then OutPixelPtr^ := InPixelPtr^
        else OutPixelPtr^ := Black;
    end; (case)
    InPixelPtr      := OffsetPointer(InPixelPtr, 1);
    OutPixelPtr     := OffsetPointer(OutPixelPtr, 1);
    DirectionPixelPtr := OffsetPointer(DirectionPixelPtr, 1);
end; (for x)
end; (for y)
ExitPoint:
GlobalUnlock(InGradImageH);
GlobalUnlock(DirectionImageH);
GlobalUnlock(OutGradImageH);
if (Error <> NoErr) and (OutImageCreated = True) then DestroyImage(OutGradImageH);
NMSFilterImage := Error;
end; (NMSFilterImage)
(-----)
Procedure ThresholdImage2(InImageH, OutImageH : ImageHandle);
var
    rowByteCount,
    x, y          : Word;
    i,
    destPixel      : integer;
    destImage,
    srcImage       : ImagePtr;
    destPixelPtr,
    srcPixelPtr    : PByte;
    localVariance : longint;
    localMean      : longint;
begin
    destImage := GlobalLock(OutImageH);
    srcImage := GlobalLock(InImageH);
    rowByteCount := Align32(destImage^.header.size.x);
    for y := 1 to destImage^.header.size.y - 2 do begin
        destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
        srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
        for x := 1 to destImage^.header.size.x - 2 do begin
            localMean :=
                (PByte(OffsetPointer(srcPixelPtr, - rowByteCount + x-1))^
                + PByte(OffsetPointer(srcPixelPtr, - rowByteCount + x))^
                + PByte(OffsetPointer(srcPixelPtr, - rowByteCount + x+1))^
                + PByte(OffsetPointer(srcPixelPtr, x-1))^
                + PByte(OffsetPointer(srcPixelPtr, x))^
                + PByte(OffsetPointer(srcPixelPtr, x+1))^
                + PByte(OffsetPointer(srcPixelPtr, rowByteCount + x-1))^
                + PByte(OffsetPointer(srcPixelPtr, rowByteCount + x))^
                + PByte(OffsetPointer(srcPixelPtr, rowByteCount + x+1))^) div 9;
            localVariance := 0;
            for i := 0 to 2 do begin
                inc(localVariance,
                    sqr(PByte(OffsetPointer(srcPixelPtr, - rowByteCount + i))^ - localMean));
                inc(localVariance, sqr(PByte(OffsetPointer(srcPixelPtr, i))^ - localMean));
                inc(localVariance,
                    sqr(PByte(OffsetPointer(srcPixelPtr, rowByteCount + i))^ - localMean));
            end;
            localVariance := Round(sqrt(localVariance)/9);
            if (LocalVariance > 0)
                and (PByte(OffsetPointer(srcPixelPtr, x))^ > (localMean + LocalVariance))
            then PByte(OffsetPointer(destPixelPtr, x))^ := White
            else PByte(OffsetPointer(destPixelPtr, x))^ := Black;
        end; (for x)
    end; (for y)
    GlobalUnlock(InImageH);
    GlobalUnlock(OutImageH);
end; (Threshold2Image)
(-----)
Procedure HysteresisThresholdImage(GradImageH, NMSGradImageH, ThresImageH : ImageHandle);
type
    HistArray = Array[Black..White] of word;
const
    RFitFrac      = 0.30; {Fraction of data to fit Rayleigh distribution}

```

Warp60

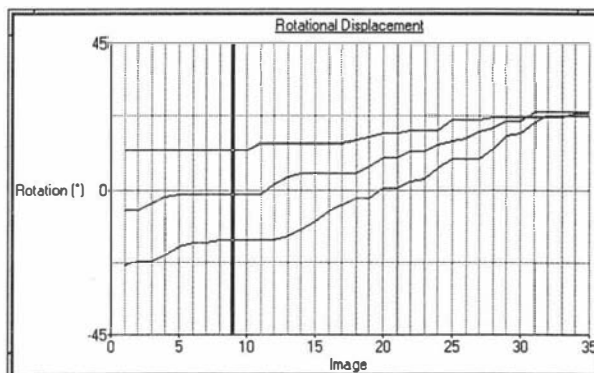
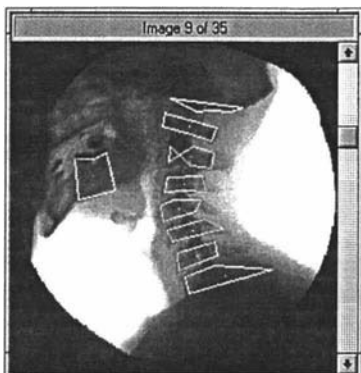


```

RLoThresFrac      = 0.80; {Fraction of Rayleigh distribution for low threshold}
Lo2HiThresRatio = 1.7;  {Ratio of LoThreshold to HiThreshold}
var
  rowByteCount,
  ImageWidth,
  ImageHeight,
  x, y,
  dx
  : integer;
  LoThres,
  HiThres
  : byte;
  Count,
  destPixel
  : integer;
  GradImage,
  NMSGradImage,
  ThresImage
  : ImagePtr;
  destPixelPtr,
  srcPixelPtr
  : PByte;
  Sum
  : longint;
  hist
  : HistArray;
  k1, k2,
  RSum, TRSum
  : real;
  ESPostion
  : Array[0..255] of word;

Procedure FitRayleighDistribution(H : HistArray; FStart,FEnd : byte; var c1,c2 : real);
{ Fits a Rayleigh distribution to the histogram H, between FStart and FEnd }
{ Returns the two distribution coefficients c1 = Normalise, c2 = time constant }
const
  MaxIter = 1000;
  StepSize = 0.0001;
var
  i
  : byte;
  YSum,
  CountOut
  : longint;
  ErrorSum,
  k1, k2
  : real;
Function ComputeError(k : real) : real;
var
  x
  : byte;
  x2
  : longint;
  RSum,
  Sum
  : real;
begin {ComputeError}
  RSum := 0.0;
  for x := FStart to FEnd do RSum := RSum + (k*x*exp(-k*sqr(longint(x))));
  RSum := YSum/RSum;
  Sum := 0.0;
  for x := FStart to FEnd do begin
    x2 := sqr(longint(x));
    Sum := Sum + (RSum*x*k*(1-k*x2)*exp(-k*x2) - hist[x]*(1-k*x2));
  end;
  ComputeError := Sum;
end; {ComputeError}
begin {MAIN: FitRayleighDistribution}
  ySum := 0;
  for i := FStart to FEnd do inc(ySum, h[i]);
  { Find starting values of k that bracket the solution }
  k1 := 0.002;
  k2 := k1;
  ErrorSum := ComputeError(k2);
  if ErrorSum < 0.0 then begin {-ve error}
    k2 := k2 + StepSize;
    if ComputeError(k2) > ErrorSum then {going in the right direction}
      while ComputeError(k2) < 0.0 do k2 := k2 + StepSize
    else {go the other direction}
      while ComputeError(k2) < 0.0 do k2 := k2 - StepSize;
  end {if}

```

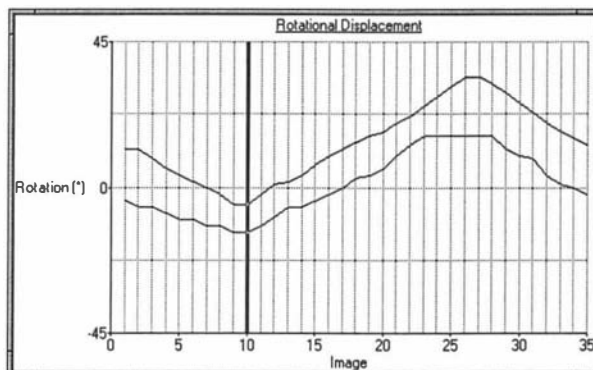


```

else begin (+ve error)
  k2 := k2 + StepSize;
  if ComputeError(k2) < ErrorSum then (going in the right direction)
    while ComputeError(k2) > 0.0 do k2 := k2 + StepSize
  else (go the other direction)
    while ComputeError(k2) > 0.0 do k2 := k2 - StepSize;
  end; (else)
  (Iterate by Bisection to find solution)
  CountOut := 0;
  if ComputeError(k1) < 0.0 then (k1 value -ve)
    Repeat
      inc(CountOut);
      ErrorSum := ComputeError((k1+k2)/2);
      if ErrorSum < 0.0 then k1 := (k1+k2)/2
      else k2 := (k1+k2)/2;
    Until (abs(ErrorSum) < 0.001 * ySum) or (CountOut > MaxIter)
  else (value for k1 +ve)
    Repeat
      inc(CountOut);
      ErrorSum := ComputeError((k1+k2)/2);
      if ErrorSum > 0.0 then k1 := (k1+k2)/2
      else k2 := (k1+k2)/2;
    Until (abs(ErrorSum) < 0.001 * ySum) or (CountOut > MaxIter);
  if Countout > MaxIter then begin
    writeln('Max iterations exceeded ErrorSum = ', ErrorSum);
    readln;
  end;
  ErrorSum := 0.0;
  c2 := (k1+k2)/2;
  c1 := 0.0;
  for i := FStart to FEnd do c1 := c1 + (c2*i*exp(-c2*sqr(longint(i))));
  c1 := ySum/c1;
end; (FitRayleighDistribution)

begin (MAIN: HysteresisThresholdImage)
  GradImage := GlobalLock(GradImageH);
  NMSGGradImage := GlobalLock(NMSGGradImageH);
  ThresImage := GlobalLock(ThresImageH);
  rowByteCount := Align32(ThresImage^.header.size.x);
  ImageWidth := ThresImage^.header.size.x;
  ImageHeight := ThresImage^.header.size.y;
  for x := Black to White do hist[x] := 0;
  (Compute GradImage histogram)
  for y := 0 to ImageHeight - 1 do begin
    srcPixelPtr := OffsetPointer(@GradImage^.data, longint(y) * rowByteCount);
    for x := 0 to ImageWidth - 1 do begin
      inc(hist[PByte(OffsetPointer(srcPixelPtr, x))^]);
    end; (for x)
  end; (for y)
  (Find index for RFitLow fraction of histogram)
  x := 1;
  Sum := 0;
  repeat
    inc(Sum, hist[x]);
    inc(x);
  Until Sum >= RFitFrac * (longint(ImageHeight) * ImageWidth - hist[0]);
  dec(x);
  FitRayleighDistribution(Hist, 1, x, k1, k2);
  ( Compute lower threshold for the distribution)
  RSum := 0.0;
  for x := Black to White do RSum := RSum + k1*k2*x*exp(-k2*sqr(longint(x)));
  TRSum := RSum;
  LoThres := 0;
  RSum := 0.0;
  Repeat
    RSum := RSum + k1*k2*LoThres*exp(-k2*sqr(longint(LoThres)));
    inc(LoThres);
  Until (RSum/TRSum) > RLoThresFrac;
  dec(LoThres);
  (Initialise ThresImage )
  for y := 0 to ImageHeight - 1 do begin

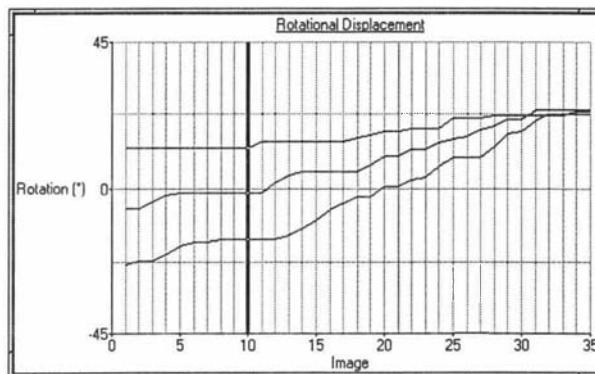
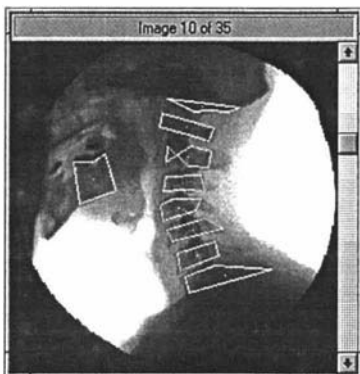
```



```

destPixelPtr := OffsetPointer(@ThresImage^.data, longint(y) * rowByteCount);
for x := 0 to ImageWidth - 1 do
  PByte(OffsetPointer(destPixelPtr, x))^ := Black;
end; {for y}
if Round(Lo2HiThresRatio * LoThres) > White then HiThres := White
else HiThres := Round(Lo2HiThresRatio * LoThres);
writeln('Thresholds: Low = ', LoThres, ' High = ', HiThres);
{Trace edge segments: Start edge-pixel > HiThres, continue until edge-pixel < LoThres}
for y := 0 to ImageHeight - 1 do begin
  srcPixelPtr := OffsetPointer(@NMSGradImage^.data, longint(y) * rowByteCount);
  if y = 0 then begin {first row a special case}
    {Scan along the row and find valid start edge pixels and then extend left & right}
    for x := 0 to ImageWidth - 1 do begin
      if PByte(OffsetPointer(srcPixelPtr, x))^ >= HiThres then begin
        dx := x;
        while (PByte(OffsetPointer(srcPixelPtr, dx))^ > LoThres) and (dx >= 0)
          and (PByte(OffsetPointer(@ThresImage^.data, longint(y) * rowByteCount + dx))^
            <> White)
        do begin {Extend from start edge-pixel to the left if > LoThres}
          PByte(OffsetPointer(@ThresImage^.data, longint(y) * rowByteCount + dx))^ :=
            White;
          dec(dx);
        end; {while}
        dx := x+1;
        while (dx <= ImageWidth-1) and (PByte(OffsetPointer(srcPixelPtr, dx))^ > LoThres)
          and (PByte(OffsetPointer(@ThresImage^.data, longint(y) * rowByteCount + dx))^
            <> White)
        do begin {Extend from start edge-pixel to the right if > LoThres}
          PByte(OffsetPointer(@ThresImage^.data, longint(y) * rowByteCount + dx))^ :=
            White;
          inc(dx);
        end; {while}
      end; {if}
    end; {for x}
  end; {if y=0}
  else begin {all other rows}
    {Scan along all other rows and find valid start edge pixels and then extend
    left and right}
    for x := 0 to ImageWidth - 1 do begin
      if (PByte(OffsetPointer(srcPixelPtr, x))^ > HiThres)
        or (PByte(OffsetPointer(@ThresImage^.data, longint(y-1)*rowByteCount+x))^ = White)
      then begin
        dx := x;
        while (dx >= 0) and (PByte(OffsetPointer(srcPixelPtr, dx))^ > LoThres)
          and (PByte(OffsetPointer(@ThresImage^.data,
            longint(y) * rowByteCount + dx))^ <> White)
        do begin {Extend from start edge-pixel to the left}
          PByte(OffsetPointer(@ThresImage^.data,
            longint(y) * rowByteCount + dx))^ := White;
          dec(dx);
        end; {while}
        dx := x+1;
        while (dx <= ImageWidth-1) and (PByte(OffsetPointer(srcPixelPtr, dx))^ > LoThres)
          and (PByte(OffsetPointer(@ThresImage^.data,
            longint(y) * rowByteCount + dx))^ <> White)
        do begin {Extend from start edge-pixel to the right}
          PByte(OffsetPointer(@ThresImage^.data,
            longint(y) * rowByteCount + dx))^ := White;
          inc(dx);
        end; {while}
      end; {if}
    end; {for x}
  end; {else y > 0}
end; {for y}
GlobalUnlock(GradImageH);
GlobalUnlock(NMSGradImageH);
GlobalUnlock(ThresImageH);
end; {HysteresisThresholdImage}
(=====)
end. {Canny}

```



C.3 Edge Comparison Functions

The following Pascal unit 'Edge_Ops' contains an implementation of Pratt's figure of merit (See equation 3.11 of section 3.2.5 in chapter 3 for full details) for assessing the quality of the output of any edge-detection algorithm with respect to a reference edge-image.

```

unit Edge_Ops;

interface
uses
  WinCrt, WinDos, WinProcs, WinTypes,
  PMXProcs, PMXTypes, Files, Globals, Images, PMXUtils,
  Filters;

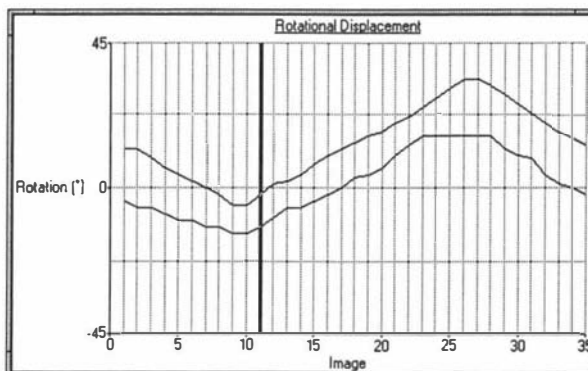
  Function CompareEdges(RefImageH, CompImageH : ImageHandle; var MaxRadius : byte) : real;

implementation
(=====)
  Function Max(i,j : integer) : integer;
  begin
    if i > j then Max := i else Max := j;
  end; {Max}

  Function Min(i,j : integer) : integer;
  begin
    if i < j then Min := i else Min := j;
  end; {Min}
(-----)
  Function CompareEdges(RefImageH, CompImageH : ImageHandle; var MaxRadius : byte) : real;
  { Compares CompImageH binary edge image (where and edge pixel is White) to RefImageH }
  { based on Pratt's figure of merit. The returned value is a percentage, 100% implies }
  { the two images are identical. MaxRadius is the maximum distance from a Comp edge }
  { pixel that had to be scanned to find the nearest reference edge pixel. }
  type
    EdgeArray = Array[1..MaxInt div 4] of TPoint;
  label
    repeatExit;
  var
    minVal,
    maxVal,
    Value,
    Radius      : byte;
    x,y,
    i,
    destPixel   : integer;
    srcImage,
    destImage   : ImagePtr;
    destPixelPtr,
    srcPixelPtr : PByte;
    rowByteCount,
    Width, Height,
    Count,
    RefEdgeCount,
    CompEdgeCount : word;
    Sum           : real;
    EdgeLocated   : boolean;
    CompEdges     : ^EdgeArray;
    TopLeft,
    BottomRight   : TPoint;

  begin {CompareEdges}
    destImage := GlobalLock(CompImageH);
    srcImage  := GlobalLock(RefImageH);
    rowByteCount := Align32(destImage^.header.size.x);

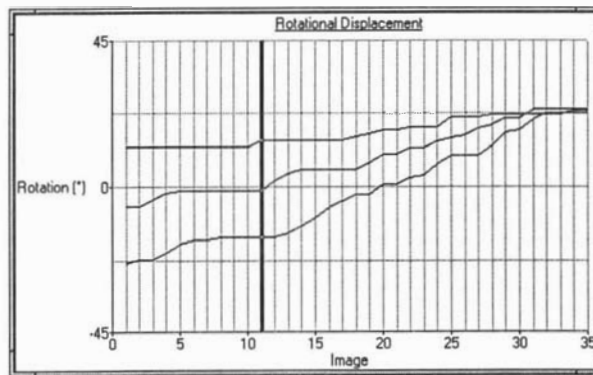
```



```

RefEdgeCount := 0;
CompEdgeCount := 0;
Width := destImage^.header.size.x;
Height := destImage^.header.size.y;
for y := 0 to Height-1 do begin { Count the number of Ref and Comp edges pixels. }
  destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
  srcPixelPtr := OffsetPointer(@srcImage^.data, longint(y) * rowByteCount);
  for x := 0 to Width-1 do begin
    if PByte(srcPixelPtr) = White then inc(RefEdgeCount);
    if PByte(destPixelPtr) = White then inc(CompEdgeCount);
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
    srcPixelPtr := OffsetPointer(srcPixelPtr, 1);
  end; {for x}
end; {for y}
If CompEdgeCount > ((2*MaxInt) div SizeOf(TPoint)) then begin
  writeln('Too many edges pixel: ', CompEdgeCount);
  CompareEdges := -1.0;
  exit;
end;
GetMem(CompEdges, (CompEdgeCount) * sizeof(TPoint));
{ Store the location of each Comp edge pixel. }
CompEdgeCount := 0;
for y := 0 to Height-1 do begin
  destPixelPtr := OffsetPointer(@destImage^.data, longint(y) * rowByteCount);
  for x := 0 to Width-1 do begin
    if PByte(destPixelPtr) = White then
      begin
        inc(CompEdgeCount);
        CompEdges^[CompEdgeCount].x := x;
        CompEdges^[CompEdgeCount].y := y;
      end; {if}
    destPixelPtr := OffsetPointer(destPixelPtr, 1);
  end; {for x}
end; {for y}
{ Take each Comp edge in turn and find the distance ,d, to nearest ref edge. }
{ Accumulate 1/(1+d^2), the error sum as you go. }
Sum := 0;
MaxRadius := 0;
for Count := 1 to CompEdgeCount do begin
  EdgeLocated := False;
  Radius := 0;
  Repeat
    if Radius > MaxRadius then MaxRadius := Radius;
    TopLeft.x := Max(CompEdges^[Count].x - Radius, 0);
    TopLeft.y := Max(CompEdges^[Count].y - Radius, 0);
    BottomRight.x := Min(CompEdges^[Count].x + Radius, Width-1);
    BottomRight.y := Min(CompEdges^[Count].y + Radius, Height-1);
    x := TopLeft.x;
    y := TopLeft.y;
    While not(EdgeLocated) and (x<=BottomRight.x) do begin {top row L2R }
      if PByte(OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + x)) = White
        then EdgeLocated := True;
      inc(x);
    end; {while}
    dec(x);
    if EdgeLocated then goto repeatExit;
    While not(EdgeLocated) and (y<=BottomRight.y) do begin { Right column T2B }
      if PByte(OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + x)) = White
        then EdgeLocated := True;
      inc(y);
    end; {while}
    dec(y);
    if EdgeLocated then goto repeatExit;
    While not(EdgeLocated) and (x>=TopLeft.x) do begin { Bottom row R2L }
      if PByte(OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + x)) = White
        then EdgeLocated := True;
      dec(x);
    end; {while}
    inc(x);
    if EdgeLocated then goto repeatExit;
    While not(EdgeLocated) and (y>=TopLeft.y) do begin { Left column B2T }

```




```

        if PByte(OffsetPointer(@srcImage^.data, longint(y) * rowByteCount + x))^ = White
        then EdgeLocated := True;
        dec(y);
    end; {while}
    inc(y);
    inc(Radius);
    repeatExit:
    Until EdgeLocated;
    sum := sum + 1/(1 + sqr(CompEdges^[Count].x - x) + sqr(CompEdges^[Count].y - y));
    end; {for Count}
    FreeMem(CompEdges, CompEdgeCount * sizeof(TPoint));
    CompareEdges := 100 * sum / Max(RefEdgeCount, CompEdgeCount);
    end; {CompareEdges}
(=====)
end. {Edge_Ops}

```

C.4 Weber's Law based Edge-detection

The following Pascal program 'Weber' contains an implementation of the Weber's law based edge-detection algorithm described in section 3.4 chapter 3. The top of the code describes the five key routines that have been implemented. The main section of the program at the end links the routines together to form the algorithm and provide control of the various internal parameters.

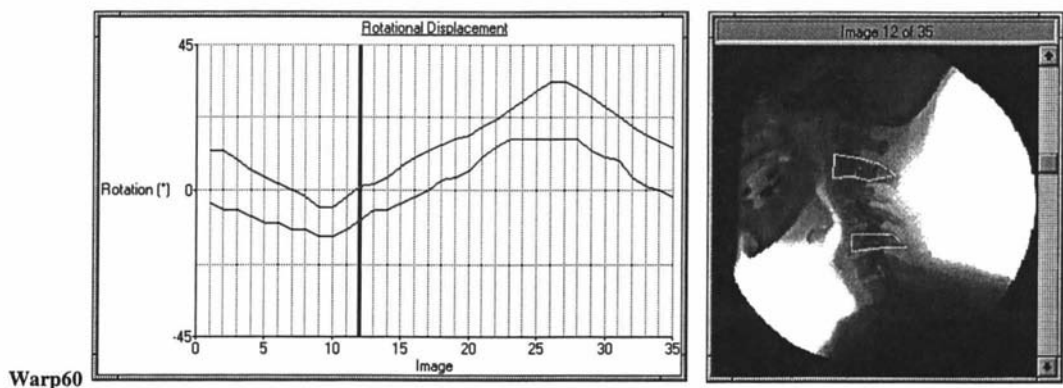
```

Program Weber;
{ Program to implement Webers Law operations.
1. ComputeWeberMap: Generate LAI-ES Maps
2. ComputeWeberEdgeMap: Generate LAI-ES map about edge-template
3. ProfileImage: Profile edge detect an image LowerProfile <= X <= UpperProfile
4. ComputeProfiles: Extract profile from an LAI-ES map based Min, Max, Average
SD of ES.
5. IncludeProfileImage: Generate 2-D profiles, use LAI-ES map directly
(with include ratio)
}
uses
WinCrt, WinDos, WinProcs, WinTypes, Strings,
PMXProcs, PMXTypes, {Files,} Globals, Images, PMXUtils,
Filters, GFilters, Edge_Ops, MLab_ops;

const
MaxES = 255; { Maximum Edge Strength covered }
MaxLAI = 255; { Maximum Local Average Intensity covered }
GradientGain = 0.5; { Gain for gradient filtering = 0.0 -> autoscale }
AverageGain = 1.0; { Gain for all smoothing average filters = 0.0 -> autoscale }
UseRoberts : boolean = True;
LAIWinRadius = 1;

type
TypeArray = array[0..1] of Char;
BitmapInfoRec = record
header : TBitmapInfoHeader;
colors : array[0..255] of TRGBQuad;
end;
ProfileArray = Array[0..MaxLAI] of byte;
var
Header : HeaderRec;
MachineType : longint;
MName : Array[0..20] of Char;
Error : ErrorCode;

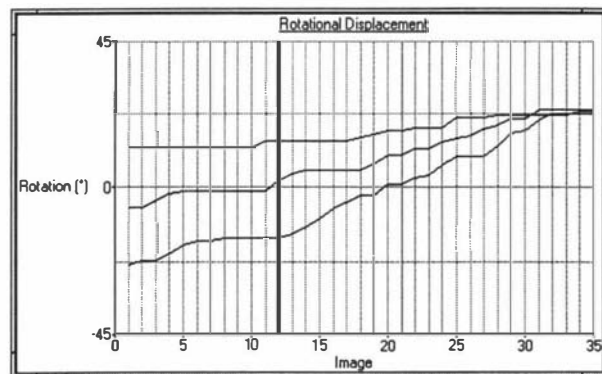
```



```

InFile,
OutFile
    : file;
outTextFile
    : text;
RefImageFile1,
RefImageFile2,
RefEdgeImageFile1,
RefEdgeImageFile2,
EdgeImageFile1,
EdgeImageFile2,
OutImageName1,
OutImageName2,
OutFileName,
OutMatrixName,
OutProfileName1,
OutProfileName2
    : string;
RefImageH, RefEdgeImageH,
EdgeImageH,
ESImageH, LAIImageH
    : ImageHandle;
RefImage, RefEdgeImage,
EdgeImage,
ESImage, LAIImage
    : ImagePtr;
ImageWeberMapH,
EdgeWeberMapH
    : MatrixHandle;
ImageWeberMap,
EdgeWeberMap
    : MatrixPtr;
LowerProfile,
AverageProfile,
UpperProfile,
SDProfile
    : ProfileArray;
KProfile
    : Array[0..MaxES] of word;
srcPtr,
destPtr
    : PByte;
ImageWidth,
ImageHeight,
rowByteCount
    : word;
x, y,
ImageSize
    : longint;
ES, LAI,
LAIStart, LAIEnd,
ESStart, ESEnd,
Pos,
Radius
    : byte;
Figure,
MaxFigure
    : real;
BitmapInfo,
BitmapInfoTemp
    : BitmapInfoRec;
FileInfo
    : TBitmapFileHeader;
TopLeft, BottomRight
    : TPoint;
(-----ComputeWeberMap-----)
Procedure ComputeWeberMap(WeberMapH : MatrixHandle; Roberts : boolean);
( Computes the Weber map ES vs LAI for the given RefImage. )
var
    ESMaX      : byte;
    WeberMap   : MatrixPtr;
begin (ComputeWeberMap)
    ESMaX := Black;
    if Roberts then begin
        RobertsGradientImage(RefImageH, ESImageH, GradientGain); (Compute edge-strength)
        ExtendImage(ESImageH, 1);
        (Compute local average intensity)
        Smooth2x2FilterImage(RefImageH, LAIImageH, AverageGain);
        ExtendImage(LAIImageH, 1);
    end

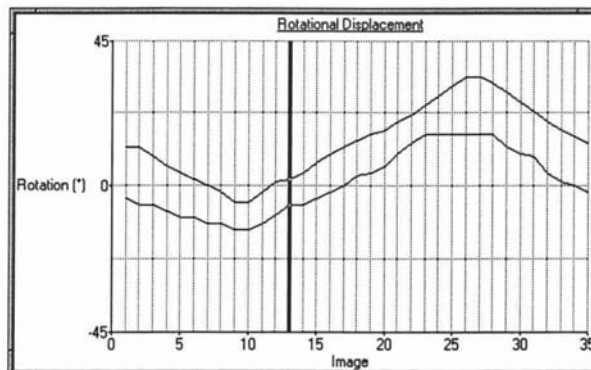
```



```

else begin
  SobelGradientImage(RefImageH, ESImageH, GradientGain); (Compute edge-strength)
  ExtendImage(ESImageH, 1);
  (Compute local average intensity)
  SmoothFilterImage(RefImageH, LAIImageH, LAIWinRadius, AverageGain);
  ExtendImage(LAIImageH, 1);
end; (else)
ESImage := GlobalLock(ESImageH);
LAIImage := GlobalLock(LAIImageH);
WeberMap := GlobalLock(WeberMapH);
for x := 0 to word(MaxLAI) * MaxES - 1 do (Initialise ImageWeberMap)
  RealPtr(OffsetPointer(@WeberMap^.data, x * SizeOfReal))^ := 0.0;
writeln('Forming LAI-ES map');
for y := 0 to ImageHeight-1 do begin
  for x := 0 to ImageWidth-1 do begin
    ES := PByte(OffsetPointer(@ESImage^.data, y * rowByteCount + x))^;
    if ES > ESMAX then ESMAX := ES;
    if ES > MaxES then ES := MaxES;
    LAI := PByte(OffsetPointer(@LAIImage^.data, y * rowByteCount + x))^;
    if LAI > MaxLAI then LAI := MaxLAI;
    RealPtr(OffsetPointer(@WeberMap^.data, (longint(ES) * MaxLAI + LAI) * SizeOfReal))^
      := RealPtr(OffsetPointer(@WeberMap^.data,
        (longint(ES) * MaxLAI + LAI) * SizeOfReal))^ + 1;
  end; (for x)
  write('.');
end; (for y)
writeln;
writeln('Max Edge-Strength = ', ESMAX);
GlobalUnlock(ESImageH); GlobalUnlock(LAIImageH);
GlobalUnlock(WeberMapH);
end; (ComputeWeberMap)
(-----ComputeWeberEdgeMap-----)
Procedure ComputeWeberEdgeMap(WeberMapH : MatrixHandle; EdgeGrowth : byte;
  Roberts : boolean);
  (-Computes Weber map ES vs LAI for the given RefImage about the specified edge image.)
var
  WeberMap : MatrixPtr;
begin (ComputeWeberEdgeMap)
  writeln('Computing Weber Edge map');
  ThresholdImage(RefEdgeImageH, EdgeImageH, 254);
  ExtendImage(EdgeImageH, 3);
  for x := 1 to EdgeGrowth do begin
    writeln('Growing edge layer ', x);
    RankFilterImage(EdgeImageH, ESImageH, 9);
    ExtendImage(ESImageH, 1);
    NoFilterImage(ESImageH, EdgeImageH);
  end; (for x)
  if Roberts then begin
    RobertsGradientImage(RefImageH, ESImageH, GradientGain); (Compute edge-strength)
    ExtendImage(ESImageH, 1);
    (Compute local average intensity)
    Smooth2x2FilterImage(RefImageH, LAIImageH, AverageGain);
    ExtendImage(LAIImageH, 1);
  end
  else begin (3x3 operator)
    writeln('Sobel filtering');
    SobelGradientImage(RefImageH, ESImageH, GradientGain); (Compute edge-strength)
    ExtendImage(ESImageH, 1);
    (Compute local average intensity)
    SmoothFilterImage(RefImageH, LAIImageH, LAIWinRadius, AverageGain);
    ExtendImage(LAIImageH, 1);
  end; (else)
  EdgeImage := GlobalLock(EdgeImageH);
  ESImage := GlobalLock(ESImageH);
  LAIImage := GlobalLock(LAIImageH);
  WeberMap := GlobalLock(WeberMapH);
  for x := 0 to word(MaxLAI+1) * (MaxES+1) - 1 do (initialise matrix)
    RealPtr(OffsetPointer(@WeberMap^.data, x * SizeOfReal))^ := 0.0;
  writeln('Forming LAI-ES edge map');
  for y := 0 to ImageHeight-1 do begin
    for x := 0 to ImageWidth-1 do begin

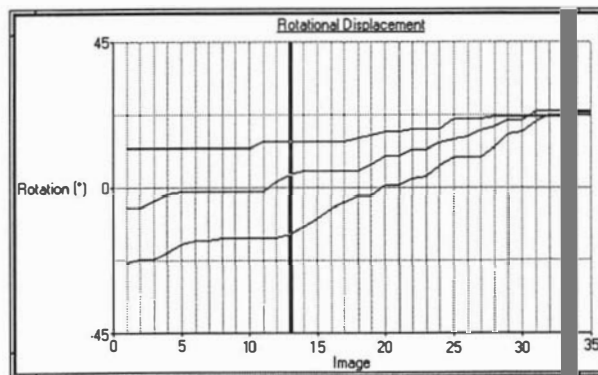
```



```

if PByte(OffsetPointer(@EdgeImage^.data, y * rowByteCount + x))^ = White then begin
  ES := PByte(OffsetPointer(@ESImage^.data, y * rowByteCount + x))^;
  if ES > MaxES then ES := MaxES;
  LAI := PByte(OffsetPointer(@LAIImage^.data, y * rowByteCount + x))^;
  if LAI > MaxLAI then LAI := MaxLAI;
  RealPtr(OffsetPointer(@WeberMap^.data, (longint(ES)*MaxLAI + LAI)*SizeOfReal))^
    := RealPtr(OffsetPointer(@WeberMap^.data,
      (longint(ES) * MaxLAI + LAI)* SizeOfReal))^ + 1;
end; (if)
end; (for x)
write('.');
end; (for y)
writeln;
GlobalUnlock(EdgeImageH); GlobalUnlock(LAIImageH);
GlobalUnlock(WeberMapH);
end; (ComputeWeberEdgeMap)
)-----ProfileImage-----)
Procedure ProfileImage(var LowerProfile, UpperProfile : ProfileArray; Roberts : boolean);
{ Extracts the edges from RefImage based on the given Profiles. }
{ An edge is declared if LowerProfile[LAI] <= ES[LAI] <= UpperProfile[LAI]. }
var
  ES, LAI
    : byte;
begin (ProfileImage)
  writeln('Applying 1-D Profile');
  if Roberts then begin
    RobertsGradientImage(RefImageH, ESImageH, GradientGain); {Compute edge-strength}
    ExtendImage(ESImageH, 1);
    {Compute local average intensity}
    Smooth2x2FilterImage(RefImageH, LAIImageH, AverageGain);
    ExtendImage(LAIImageH, 1);
  end
  else begin {3x3 operator}
    SobelGradientImage(RefImageH, ESImageH, GradientGain); {Compute edge-strength}
    ExtendImage(ESImageH, 1);
    {Compute local average intensity}
    SmoothFilterImage(RefImageH, LAIImageH, LAIWinRadius, AverageGain);
    ExtendImage(LAIImageH, 1);
  end; (else)
  EdgeImage := GlobalLock(EdgeImageH);
  ESImage := GlobalLock(ESImageH);
  LAIImage := GlobalLock(LAIImageH);
  for y := 0 to ImageHeight-1 do begin
    for x := 0 to ImageWidth-1 do begin
      ES := PByte(OffsetPointer(@ESImage^.data, y * rowByteCount + x))^;
      LAI := PByte(OffsetPointer(@LAIImage^.data, y * rowByteCount + x))^;
      if (ES >= LowerProfile[LAI]) and (ES <= UpperProfile[LAI]) then
        PByte(OffsetPointer(@EdgeImage^.data, y * rowByteCount + x))^ := White
      else PByte(OffsetPointer(@EdgeImage^.data, y * rowByteCount + x))^ := Black;
    end; (for x)
  end; (for y)
  GlobalUnlock(EdgeImageH); GlobalUnlock(ESImageH); GlobalUnlock(LAIImageH);
end; (ProfileImage)
)-----ProfileImage2D-----)
Procedure ProfileImage2D(WeberMapH : MatrixHandle; Roberts : boolean);
{ Extracts the edges from RefImage based on the given ES-LAI WeberMap. }
{ An edge is declared if WeberMap[LAI,ES] > 0. }
const
  CountThres = 0; {minimum pixel count to declare an edge pixel }
var
  ES, LAI
    : byte;
  WeberMap : MatrixPtr;
begin (ProfileImage)
  writeln('Applying 2-D Profile');
  if Roberts then begin
    RobertsGradientImage(RefImageH, ESImageH, GradientGain); {Compute edge-strength}
    ExtendImage(ESImageH, 1);
    {Compute local average intensity}
    Smooth2x2FilterImage(RefImageH, LAIImageH, AverageGain);

```



Warp30

```

    ExtendImage(LAIImageH, 1);
  end (if Roberts)
else begin (3x3 operator)
  SobelGradientImage(RefImageH, ESImageH, GradientGain); (Compute edge-strength)
  ExtendImage(ESImageH, 1);
  (Compute local average intensity)
  SmoothFilterImage(RefImageH, LAIImageH, LAIWinRadius, AverageGain);
  ExtendImage(LAIImageH, 1);
end; (else)
EdgeImage := GlobalLock(EdgeImageH);
ESImage := GlobalLock(ESImageH);
LAIImage := GlobalLock(LAIImageH);
WeberMap := GlobalLock(WeberMapH);
for y := 0 to ImageHeight-1 do begin
  for x := 0 to ImageWidth-1 do begin
    ES := PByte(OffsetPointer(@ESImage^.data, y * rowByteCount + x))^;
    if ES > MaxES then ES := MaxES;
    LAI := PByte(OffsetPointer(@LAIImage^.data, y * rowByteCount + x))^;
    if RealPtr(OffsetPointer(@WeberMap^.data,
      (longint(ES) * MaxLAI + LAI) * SizeOfReal))^ > CountThres
    then PByte(OffsetPointer(@EdgeImage^.data, y * rowByteCount + x))^ := White
    else PByte(OffsetPointer(@EdgeImage^.data, y * rowByteCount + x))^ := Black;
  end; (for x)
end; (for y)
GlobalUnlock(EdgeImageH); GlobalUnlock(ESImageH); GlobalUnlock(LAIImageH);
GlobalUnlock(WeberMapH);
end; (ProfileImage2D)
(-----ComputeProfiles-----)
Procedure ComputeProfiles(
  WeberMapH : MatrixHandle;
  var LowerProfile, UpperProfile, AverageProfile, SDProfile : ProfileArray);
{ Computes the Min, Max and Average and Standard Deviation profiles
  (Profile[LAI] = min(ES), max(ES) Ave(ES), SDave(ES)) for the given Weber (LAI-ES) map.
}
type
  LargeProfileArray = Array[0..255] of longint;
var
  SumProfile,
  VarianceProfile : ^LargeProfileArray;
  CountProfile : ^ProfileArray;
  Count : longint;
  WeberMap : MatrixPtr;
begin (ComputeProfiles)
  new(SumProfile);
  new(VarianceProfile);
  new(CountProfile);
  WeberMap := GlobalLock(WeberMapH);
  writeln('Computing 1-D Profiles from LAI-ES map');
  for x := 0 to MaxLAI do begin ( initialise profiles )
    LowerProfile[x] := White;
    UpperProfile[x] := Black;
    AverageProfile[x] := 0;
    CountProfile^[x] := 0;
    SDProfile[x] := 0;
    SumProfile^[x] := 0;
    VarianceProfile^[x] := 0;
  end; (for x)
  for y := 0 to MaxLAI do begin ( Compute Min, Max and Sum profiles )
    for x := 0 to MaxES do begin
      Count :=
        Round(RealPtr(OffsetPointer(@WeberMap^.data, (x * MaxLAI + y) * SizeOfReal))^);
      if Count > 0 then begin (a pixel at this LAI-ES position was recorded)
        if LowerProfile[y] = White then LowerProfile[y] := x;
        if x > UpperProfile[y] then UpperProfile[y] := x;
        inc(SumProfile^[y], x);
        inc(CountProfile^[y]);
      end; (if Count > 0)
    end; (for x)
    if UpperProfile[y] = Black (no value recorded)
    then UpperProfile[y] := White;
  end; (for y)
end;

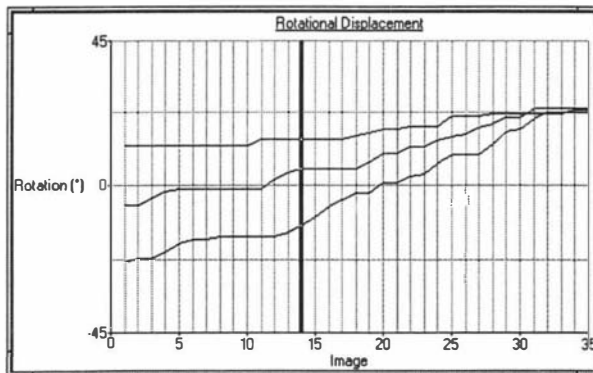
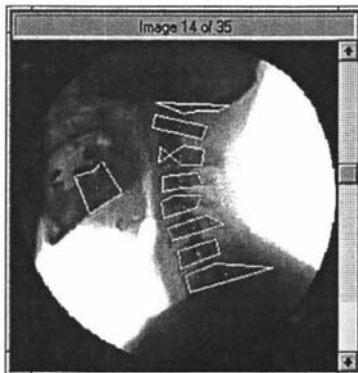
```



```

for x := 0 to MaxLAI do begin (Average SumProfile)
  if CountProfile^[x] = 0 {no value recorded}
    then AverageProfile[x] := White
  else AverageProfile[x] := SumProfile^[x] div CountProfile^[x];
end; {for x}
for y := 0 to MaxLAI do begin (Compute VarianceProfile)
  for x := 0 to MaxES do begin
    Count :=
      Round(RealPtr(OffsetPointer(@WeberMap^.data, (x * MaxLAI + y) * SizeOfReal))^);
    if Count > 0 then inc(VarianceProfile^[y], Round(sqr(x - AverageProfile[y])));
  end; {for x}
end; {for y}
for x := 0 to MaxLAI do begin(Compute SDProfile for VarianceProfile)
  if CountProfile^[x] = 0 then SDProfile[x] := 0
  else SDProfile[x] := Round(Sqrt(VarianceProfile^[x]) / CountProfile^[x]);
end; {for x}
GlobalUnlock(EdgeImageH); GlobalUnlock(ESImageH); GlobalUnlock(LAIImageH);
Dispose(SumProfile);
Dispose(VarianceProfile);
Dispose(CountProfile);
end; (ComputeProfiles)
)-----TuneProfile-----)
Procedure TuneProfile(Roberts : boolean);
const
  TuneSDRatio : integer = 4;
  Offset = 4;
var
  c : char;
begin
  { Tune the Average profile to maximise Pratt's figure of merit. Tune range +/- 2SD }
  LAIStart := Black;
  while AverageProfile[LAIStart] = White do inc(LAIStart);
  LAIEnd := White-1;
  while AverageProfile[LAIEnd] = White do dec(LAIEnd);
  writeln('LAI(Start,End) = (' ,LAIStart,',',LAIEnd,')');
  for LAI := LAIStart to LAIEnd do begin
    Pos := AverageProfile[LAI];
    if (AverageProfile[LAI] - TuneSDRatio*SDProfile[LAI] - Offset) < 0 then ESStart := 0
    else ESStart := Round(AverageProfile[LAI] - TuneSDRatio * SDProfile[LAI] - Offset);
    if (AverageProfile[LAI] + TuneSDRatio * SDProfile[LAI] + Offset) > MaxES then
      ESEnd := MaxES
    else ESEnd := Round(AverageProfile[LAI] + TuneSDRatio * SDProfile[LAI] + Offset);
    MaxFigure := 0.0;
    write(LAI,': (' ,ESStart,' ',ESEnd,')');
    for ES := ESStart to ESEnd do begin
      AverageProfile[LAI] := ES;
      ProfileImage(AverageProfile, UpperProfile, Roberts);
      ExtendImage(EdgeImageH, 2);
      Figure := CompareEdges(RefEdgeImageH, EdgeImageH, Radius);
      if Figure > MaxFigure then begin
        MaxFigure := Figure;
        Pos := ES;
      end; {if}
    end; {for ES}
    writeln('MaxF = ',MaxFigure:2:1,');
    AverageProfile[LAI] := Pos;
    if Keypressed then begin
      c := Readkey;
      writeln('Do you want to quit? (Y/N)');
      repeat until keypressed;
      c := readkey;
      c := upcase(c);
      if c = 'Y' then exit;
    end;
  end; {for LAI}
end; (TuneProfile)
)-----LoadRefImage-----)
Procedure LoadRefImage(ImageFile : string);
begin
  Assign(inFile, ImageFile);

```

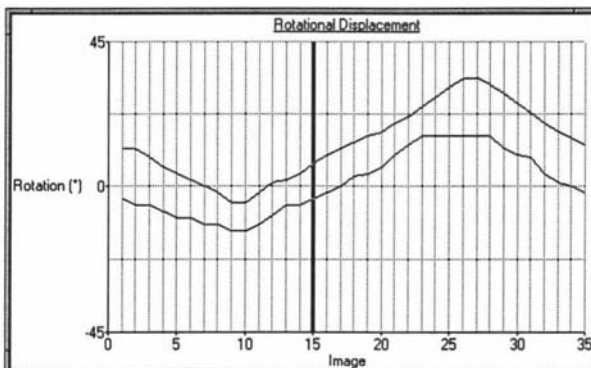


```

if IOResult <> 0 then begin
  Writeln('Error Reading Input File');
  Exit;
end;
Reset(inFile, 1);
if IOResult <> 0 then begin
  Writeln('Error Reading Input File');
  Exit;
end;
Error := FileRead(TFileRec(inFile), @fileInfo, SizeOf(fileInfo), 0);
if Error <> 0 then Exit;
Error := FileRead(TFileRec(inFile), @bitmapInfo, SizeOf(bitmapInfo), 0);
ImageWidth := bitmapInfo.header.biWidth;
ImageHeight := bitmapInfo.header.biHeight;
writeln('Image Size: ', ImageWidth, 'x', ImageHeight);
rowByteCount := Align32(ImageWidth);
Error := CreateImage(RefImageH, rowByteCount, ImageHeight);
if Error <> NoErr then begin
  writeln('Insufficient memory');
  Exit;
end;
RefImage := GlobalLock(RefImageH);
Error := FileRead(TFileRec(inFile), @RefImage^.data,
  longint(rowByteCount) * ImageHeight, 0);

if Error <> NoErr then begin
  writeln('Error reading RefImage');
  Exit;
end;
Close(inFile);
GlobalUnlock(RefImageH);
end; (LoadRefImage)
(-----LoadRefEdgeImage-----)
Procedure LoadRefEdgeImage(ImageFile : string);
begin
  Error := CreateImage(EdgeImageH, rowByteCount, ImageHeight);
  if Error <> NoErr then begin
    writeln('Insufficient memory');
    Exit;
  end;
  EdgeImage := GlobalLock(EdgeImageH);
  Error := CreateImage(RefEdgeImageH, rowByteCount, ImageHeight);
  if Error <> NoErr then begin
    writeln('Insufficient memory');
    Exit;
  end;
  RefEdgeImage := GlobalLock(RefEdgeImageH);
  Assign(inFile, ImageFile);
  If IOResult <> 0 then begin
    Writeln('Error Reading RefEdgeImage File');
    Exit;
  end;
  Reset(inFile, 1);
  If IOResult <> 0 then begin
    Writeln('Error Reading RefEdgeImage File');
    Exit;
  end;
  Error := FileRead(TFileRec(inFile), @fileInfo, SizeOf(fileInfo), 0);
  if Error <> 0 then Exit;
  Error := FileRead(TFileRec(inFile), @bitmapInfoTemp, SizeOf(bitmapInfo), 0);
  if Error <> 0 then Exit;
  Error := FileRead(TFileRec(inFile), @RefEdgeImage^.data,
    longint(rowByteCount) * ImageHeight, 0);
  if Error <> NoErr then begin
    writeln('Error reading RefEdgeImage');
    Exit;
  end;
  Close(inFile);
  GlobalUnlock(RefEdgeImageH);
end; (LoadRefEdgeImage)

```



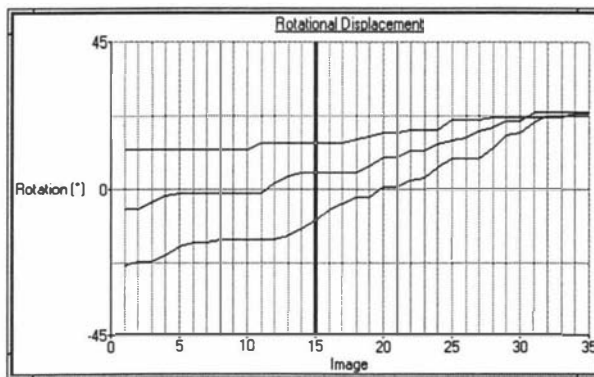
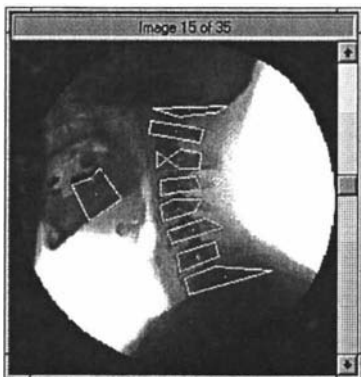
```

(-----SaveImage-----)
Procedure SaveImage(var ImageH : ImageHandle; ImageFile : string;
                    InvertPalette : boolean);

var
    Image      : ImagePtr;
    rowByteCount : word;
    FileInfo    : TBitmapFileHeader;
    BitmapInfo  : BitmapInfoRec;
begin
    Image := GlobalLock(ImageH);
    rowByteCount := Align32(Image^.header.size.x);
    { setup output file BitMap info }
    TypeArray(fileInfo.bfType) := 'BM';
    fileInfo.bfSize :=
        SizeOf(fileInfo) + SizeOf(bitmapInfo) + longint(rowByteCount) * ImageHeight;
    fileInfo.bfReserved1 := 0;
    fileInfo.bfReserved2 := 0;
    fileInfo.bfOffBits := SizeOf(fileInfo) + SizeOf(bitmapInfo);
    bitmapInfo.header.biSize := SizeOf(bitmapInfo.header);
    bitmapInfo.header.biWidth := Image^.header.size.x;
    bitmapInfo.header.biHeight := Image^.header.size.y;
    bitmapInfo.header.biPlanes := 1;
    bitmapInfo.header.biBitCount := 8;
    bitmapInfo.header.biCompression := 0;
    bitmapInfo.header.biSizeImage := 0; { non-compressed }
    bitmapInfo.header.biXPelsPerMeter := 0;
    bitmapInfo.header.biYPelsPerMeter := 0;
    bitmapInfo.header.biClrUsed := 0;
    bitmapInfo.header.biClrImportant := 0;
    if Not(InvertPalette) then begin
        for x := 0 to 255 do
            with bitmapInfo.colors[x] do begin
                rgbBlue := x;
                rgbGreen := x;
                rgbRed := x;
                rgbReserved := 0;
            end; (with)
        end (if)
    else begin
        for x := 0 to 255 do
            with bitmapInfo.colors[x] do begin
                rgbBlue := 255-x;
                rgbGreen := 255-x;
                rgbRed := 255-x;
                rgbReserved := 0;
            end; (with)
        end; (else)
    Assign(outFile, ImageFile);
    if IOResult <> 0 then begin
        Writeln('Error writing OutImage File');
        Exit;
    end;
    Rewrite(outFile, 1);
    if IOResult <> 0 then begin
        Writeln('Error writing OutImage File');
        Exit;
    end;
    Error := FileWrite(TFileRec(outFile), @fileInfo, SizeOf(fileInfo), 0);

    if Error <> 0 then Exit;
    Error := FileWrite(TFileRec(outFile), @bitmapInfo, SizeOf(bitmapInfo), 0);
    if Error <> 0 then Exit;
    Error := FileWrite(TFileRec(outFile), @Image^.data,
        longint(rowByteCount) * Image^.header.size.y, 0);
    if Error <> NoErr then begin
        writeln('Error writing OutImage');
        Exit;
    end;
    Close(outFile);
    GlobalUnlock(ImageH);
end; (SaveEdgeOutImage)

```



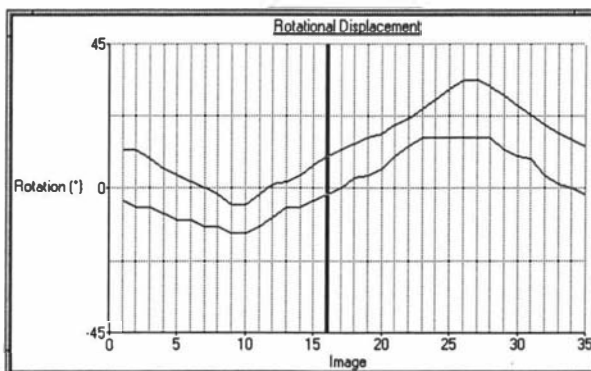

```

(-----SaveWeberImageArray-----)
Procedure SaveWeberImageArray(ArrayFile : string);
begin
  OutFileName := OutMatrixName + '.mat';
  MachineType := 0; { found this value by reading a MATLAB file. }
  Assign(outFile, OutFileName);
  if IOResult <> 0 then begin
    writeln('File Error');
    Exit;
  end;
  Rewrite(outFile, 1);
  if IOResult <> 0 then begin
    writeln('File Error');
    Exit;
  end;
  ImageWeberMap := GlobalLock(ImageWeberMapH);
  with Header do begin
    mattype := MachineType;
    matrows := ImageWeberMap^.size.x;
    matcols := ImageWeberMap^.size.y;
    imagef := 0;
    namelen := Length(OutMatrixName) + 1; {null terminated}
  end; {with Header}
  {Write out header information}
  Error := FileWrite(TFileRec(outFile), @Header, SizeOf(Header), 0);
  if Error <> NoErr then begin
    writeln('File write Error');
    Exit;
  end;
  StrPCopy(MName, OutMatrixName); { convert pas-string to nul-string }
  Error := FileWrite(TFileRec(outFile), @MName, header.namelen, 0);
  if Error <> NoErr then begin
    writeln('File write Error');
    Exit;
  end;
  Error := FileWrite(TFileRec(outFile), @ImageWeberMap^.data, ImageWeberMap^.dataSize, 0);
  if Error <> NoErr then begin
    writeln('File write Error');
    Exit;
  end;
  Close(outFile);
end; {SaveWeberImageArray}

(-----SaveProfile-----)
Procedure SaveProfile(Profile : ProfileArray; ProfileFile : string);
begin
  Assign(outTextFile, ProfileFile);
  if IOResult <> 0 then begin
    Writeln('Error writing OutImage File');
    Exit;
  end;
  Rewrite(outTextFile);
  if IOResult <> 0 then begin
    Writeln('Error writing OutImage File');
    Exit;
  end;
  for x := Black to White do
    writeln(outTextFile, Profile[x]);
  Close(outTextFile);
end; {SaveProfile}

(-----SaveAllProfiles-----)
Procedure SaveAllProfiles(ProfileFile : string);
begin
  Assign(outTextFile, ProfileFile);
  if IOResult <> 0 then begin
    Writeln('Error writing OutImage File');
    Exit;
  end;
  Rewrite(outTextFile);
  if IOResult <> 0 then begin
    Writeln('Error writing OutImage File');
    Exit;
  end;

```



```

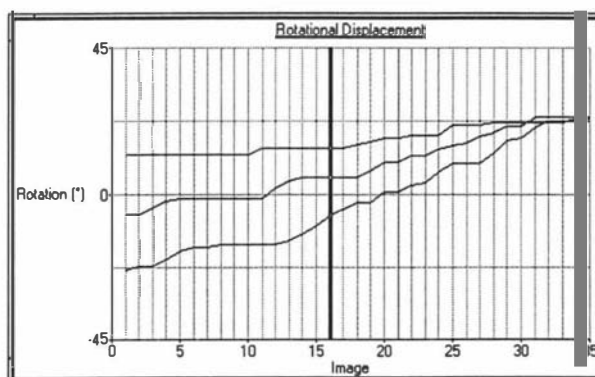
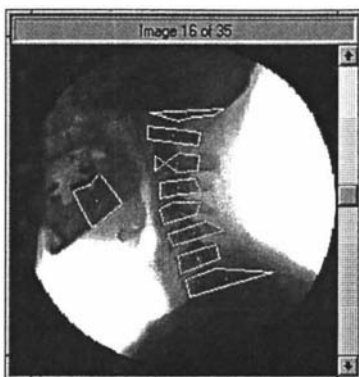
    end;
    for x := Black to White do
        writeln(outTextFile, LowerProfile[x], ' ', AverageProfile[x],
            ' ', UpperProfile[x], ' ', SDProfile[x]);
    Close(outTextFile);
end; (SaveAllProfiles)

(=====Weber MAIN=====)
begin (Weber MAIN)
    RefImageFile1      := ('e:\phd\edges\lo_level.ops\wt.bmp');
    RefEdgeImageFile1 := ('e:\phd\edges\hi_level.ops\neck_sec.tn\415-1.bmp';
    RefImageFile2      := ('e:\phd\edges\lo_level.ops\wt_can_e .bmp';
    RefEdgeImageFile2 := ('e:\phd\edges\hi_level.ops\neck_sec.tn\marhil\sec-edg2.bmp';

    OutImageName1      := '415-1-s.bmp';
    OutProfileName1    := 'bk_prof.txt';
    RefImageFile2      := 'e:\phd\edges\lo_level.ops\weber\wt_med.bmp';
    RefEdgeImageFile2 := 'e:\phd\edges\lo_level.ops\wt_e.bmp';
    OutImageName2      := 'wt_tune.bmp';
    OutProfileName2    := 'wt_prof.txt';
    OutMatrixName      := 'test_web';    { Name must not start with a number! }
    OutFileName        := OutMatrixName + '.mat';

    LoadRefImage(RefImageFile1);
    { ExtendImage(RefImageH, 3); }
    { LoadRefEdgeImage(RefEdgeImageFile1); }
    Error := CreateImage(ESImageH, rowByteCount, ImageHeight);
    if Error <> NoErr then begin
        writeln('Insufficient memory');
        Exit;
    end;
    ESImage := GlobalLock(ESImageH);
    Error := CreateImage(LAIImageH, rowByteCount, ImageHeight);
    if Error <> NoErr then begin
        writeln('Insufficient memory');
        Exit;
    end;
    LAIImage := GlobalLock(LAIImageH);
    GlobalUnlock(RefImageH); GlobalUnlock(EdgeImageH); GlobalUnlock(ESImageH);
    Error := CreateMatrix(EdgeWeberMapH, MaxLAI+1, MaxES+1);
    if Error <> NoErr then begin
        writeln('Mem error');
        exit;
    end;
    Error := CreateMatrix(ImageWeberMapH, MaxLAI+1, MaxES+1);
    if Error <> NoErr then begin
        writeln('Mem error');
        exit;
    end;
    ( **** Select appropriate functions to perform desired PROCESSING **** )
    (*
    UseRoberts := False(True);
    ComputeWeberEdgeMap(EdgeWeberMapH, 0, UseRoberts);
    ProfileImage2D(EdgeWeberMapH, UseRoberts);
    *)
    (*
    ComputeProfiles(EdgeWeberMapH, LowerProfile, UpperProfile, AverageProfile, SDProfile);
    ProfileImage(AverageProfile, UpperProfile, UseRoberts);
    *)
    { PointFilterImage(EdgeImageH, LAIImageH); }
    DestroyImage(LAIImageH);
    { HistogramImage(RefImageH, LAIImageH); }
    TopLeft.x := 95;
    TopLeft.y := 50;
    BottomRight.x := TopLeft.x + 70;
    BottomRight.y := TopLeft.y + 110;
    writeln('Copy');
    Error := CopySectionOfImage(RefImageH, LAIImageH, TopLeft, BottomRight);
    if Error <> NoErr then writeln('Error in Copy');
    { NoFilterImage(EdgeImageH, LAIImageH); }
    ExtendImage(LAIImageH, 1);
    SaveImage((Edge)LAIImageH, OutImageName1, False);
    (*

```



```

    ( Tune profile for first image )
    ComputeWeberEdgeMap(EdgeWeberMapH, 0);
    ComputeProfiles(EdgeWeberMapH, LowerProfile, UpperProfile, AverageProfile, SDProfile);
    SaveAllProfiles('bk_prof2.txt');
    ProfileImage(AverageProfile, UpperProfile);
    TuneProfile;
    ExtendImage(EdgeImageH, 2);
    EdgeImage := GlobalLock(EdgeImageH);
    SaveEdgeOutImage(OutImageName1);
    SaveProfile(AverageProfile, OutProfileName1);
    ( Tune profile for second image )
    DestroyImage(RefImageH); DestroyImage(EdgeImageH);
    LoadRefImage(RefImageFile2);
    LoadRefEdgeImage(RefEdgeImageFile2);
    ComputeWeberEdgeMap(EdgeWeberMapH, 0);
    ComputeProfiles(EdgeWeberMapH, LowerProfile, UpperProfile, AverageProfile, SDProfile);
    ProfileImage(AverageProfile, UpperProfile);
    TuneProfile;
    ExtendImage(EdgeImageH, 1);
    EdgeImage := GlobalLock(EdgeImageH);
    SaveEdgeOutImage(OutImageName2);
    SaveProfile(AverageProfile, OutProfileName2);
*)
GlobalUnlock(ImageWeberMapH);
DestroyImage(RefImageH); DestroyImage(EdgeImageH);
DestroyImage(ESImageH); DestroyImage(LAIImageH);
DestroyMatrix(ImageWeberMapH); DestroyMatrix(EdgeWeberMapH);
writeln('Finished'); readln;
DoneWinCrt;
(=====)
end. (Weber)

```

C.5 Match-statistic Performance Support

The following Pascal program 'Match' contains support code and functions for the match statistics described in section 5.1.1 of chapter 5 and summarised in table 5.1. It also contains an implementation of the selectivity measure given by equation 5.7 used to compare the various statistics in both intraframe and interframe matches.

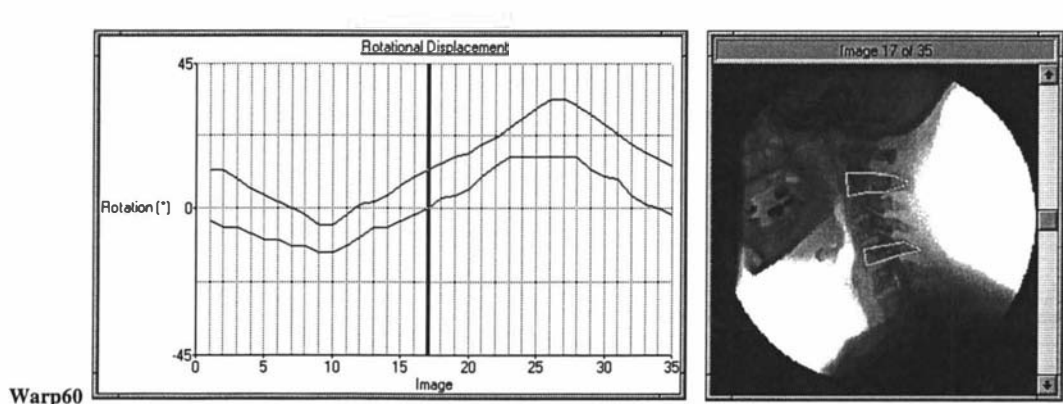
This program was used to generate all the results of section 5.3 of chapter 5.

```

Program Match;
(Implements a range of Match statistics and support routines for for interframe comparisons.)
uses
    WinCrt, WinDos, WinProcs, WinTypes, Strings,
    PMXProcs, PMXTypes, PMXUtils,
    Points, Files, Globals, Images,
    Maths2, MLab_ops, Filters;

label
    ExitPoint;
const
    MemErr = 10;
    AbsMeanDiff = 1; (Mean of sum of absolute differences)
    RMSDiff = 2; (Root mean-square of differences)
    NormCorr = 3; (Normalised area correlation)
    Moment = 4; (Moment invariants)

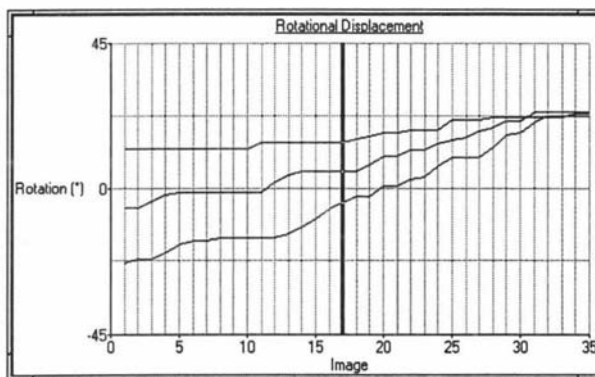
```



```

RT1  = 1;           {Rotation, translation invariant moment 1}
RT2  = RT1 + 1;     {Rotation, translation invariant moment 1..2}
RT3  = RT2 + 1;     {Rotation, translation invariant moment 1..3}
RT4  = RT3 + 1;     {Rotation, translation invariant moment 1..4}
RT6  = RT4 + 1;     {Rotation, translation invariant moment 1..6}
RTS2 = RT6 + 1;     {Rotation, translation and scale invariant moment 1..2}
RTS4 = RTS2 + 1;    {Rotation, translation and scale invariant moment 1..4}
RTS6 = RTS4 + 1;    {Rotation, translation and scale invariant moment 1..6}
var
  imageNo,
  pointGroupNo,
  rowByteCount      : word;
  PointNo,
  WindowRadius,
  SearchRadius,
  MomentType,
  MatchType,
  NoShells
  : byte;
  SelectivityBase   : integer;
  fileNo            : longint;
  ImagePath,
  CMXDocPath,
  CMXDocName,
  DataDocName,
  FileType,
  S
  : string;
  Backward          : boolean;
  DocName,
  CMXDocPathP,
  CMXDocNameP
  : array[0..63] of char;
  Name
  : Str255;
  inFile, outFile   : TFileRec;
  FileFound,
  Append
  : boolean;
  Error
  : ErrorCode;
  DataSize
  : Longint;
  DocumentInfo
  : DocInfoRec;
  docID
  : DocIDRec;
  PointInfoH
  : PointInfoHandle;
  path
  : PathStr;
  searchInfo
  : TSearchRec;
  pointGroupName
  : PointGroupNameStr;
  trackInfo
  : TrackInfoRec;
  ImageSize,
  Point
  : TPoint;
  Image1H, Image2H,
  Image3H, Image4H
  : ImageHandle;
  Image1, Image2,
  Image3, Image4
  : ImagePtr;
  MatchMapH,
  SelectivityH,
  SelMinMapH,
  SelAveMapH
  : MatrixHandle;
  MatchMap,
  Selectivity,
  SelMinMap,
  SelAveMap
  : MatrixPtr;
  MinSel, AveSel
  : real;
  DataName
  : NameStr;
{-----ComputeMatchMap-----}
Procedure ComputeMatchMap(SourceImageH, TargetImageH : ImageHandle;
  SearchRadius, WindowRadius, MatchType : byte;
  Centre : TPoint;
  var MatchMapH : MatrixHandle);
{ About the Point coords a feature is defined of size WindowRadius. This feature is
  then matched in a local area of radius SearchRadius using the specified MatchType.
  Match strength values are then stored in MatchMap. }
var
  i,j,
  TargetPixel,
  SourcePixel
  : byte;
  (Counter)

```



```

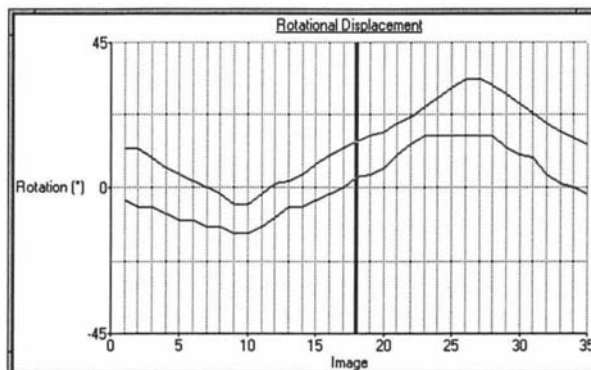
SourceSum,
TargetSum,
SumMatch
      : longInt;

SourceSumD,
TargetSumD,
MatchSumD
      : double;

SourceImage,
TargetImage
      : ImagePtr;
ImageData
      : BytePtr;
MatrixMap
      : MatrixPtr;
CentrePixel
      : BytePtr;
rowByteCount
      : longint;
WindowPixelCount,
SearchWidth,
dx, dy, x, y
      : integer;
MaxDiff,
MinDiff,
cx, cy
      : real;      (x and y centre of area coords)
sBM
      : Array[0..3, 0..3] of longint; (Source Base moments 0..3)
sCM
      : Array[0..3, 0..3] of real;      (Source Central moments 0..3)
tBM
      : Array[0..3, 0..3] of longint; (Target Base moments 0..3)
tCM
      : Array[0..3, 0..3] of real;      (Target Central moments 0..3)
sMI
      : Array[1..7] of real;            (First seven source moment invariants)
tMI
      : Array[1..7] of real;            (First seven target moment invariants)
begin
SourceImage := GlobalLock(SourceImageH);
rowByteCount := Align32(SourceImage^.header.size.x);
MatchMap := GlobalLock(MatchMapH);
WindowPixelCount := sqr(2*WindowRadius + 1);
SearchWidth := 2*SearchRadius + 1;
if TargetImageH = 0 ( selfmatch)
  then TargetImage := GlobalLock(SourceImageH)
  else TargetImage := GlobalLock(TargetImageH);
case MatchType of
  AbsMeanDiff : begin
    for y := -SearchRadius to SearchRadius do begin
      for x := -SearchRadius to SearchRadius do begin
        SumMatch := 0;
        for dy := -WindowRadius to WindowRadius do begin
          for dx := -WindowRadius to WindowRadius do begin
            SourcePixel :=
              BytePtr(OffsetPointer(@SourceImage^.data,
                                    centre.x + dx + (centre.y + dy) * rowByteCount))^;
            TargetPixel :=
              BytePtr(OffsetPointer(@TargetImage^.data,
                                    centre.x + x+dx + (centre.y + y+dy)*rowByteCount))^;
            SumMatch := SumMatch + abs(integer(SourcePixel) - TargetPixel);
          end; (for dx)
        end; (for dy)
        RealPtr(OffsetPointer(@MatchMap^.data,
                              ((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^
          := SumMatch/WindowPixelCount/White;
      end; (for x)
    end; (for y)
  end; (AbsMeanDiff)
  RMSDiff : begin
    for y := -SearchRadius to SearchRadius do begin
      for x := -SearchRadius to SearchRadius do begin
        SumMatch := 0;
        for dy := -WindowRadius to WindowRadius do begin
          for dx := -WindowRadius to WindowRadius do begin
            SourcePixel :=
              BytePtr(OffsetPointer(@SourceImage^.data,
                                    centre.x + dx + (centre.y + dy) * rowByteCount))^;
            TargetPixel :=
              BytePtr(OffsetPointer(@TargetImage^.data,
                                    centre.x + x+dx + (centre.y+y+dy)*rowByteCount))^;
            SumMatch := SumMatch + abs(sqr(word(SourcePixel)) - sqr(TargetPixel));
          end; (for dx)
        end; (for dy)
      end; (for x)
    end; (for y)
  end; (RMSDiff)
end;

```

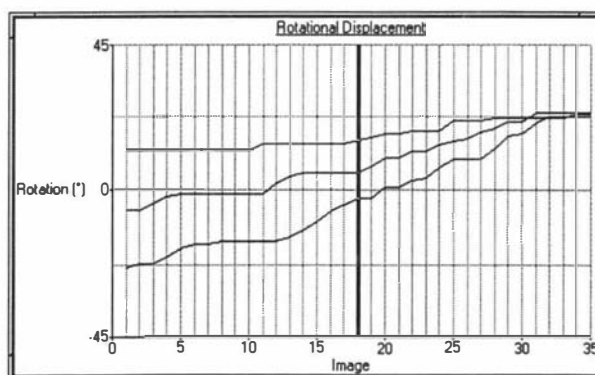
Warp60



```

    RealPtr(OffsetPointer(@MatchMap^.data,
        ((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^
        := sqrt(SumMatch)/WindowPixelCount/White;
    end; {for x}
    end; {for y}
    end; {RMSDiff}
    NormCorr : begin
        for y := -SearchRadius to SearchRadius do begin
            for x := -SearchRadius to SearchRadius do begin
                SourceSum := 0;
                TargetSum := 0;
                SumMatch := 0;
                for dy := -WindowRadius to WindowRadius do begin
                    for dx := -WindowRadius to WindowRadius do begin
                        SourcePixel :=
                            BytePtr(OffsetPointer(@SourceImage^.data,
                                centre.x + dx + (centre.y + dy) * rowByteCount))^;
                        TargetPixel :=
                            BytePtr(OffsetPointer(@TargetImage^.data,
                                centre.x + x+dx + (centre.y+y+dy)*rowByteCount))^;
                        SumMatch := SumMatch + integer(SourcePixel) * TargetPixel;
                        inc(TargetSum, sqr(longint(TargetPixel)));
                        inc(SourceSum, sqr(longint(SourcePixel)));
                    end; {for dx}
                end; {for dy}
                if (SourceSum = 0) or (TargetSum = 0) then
                    RealPtr(OffsetPointer(@MatchMap^.data,
                        ((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ := 0
                else
                    RealPtr(OffsetPointer(@MatchMap^.data,
                        ((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^
                        := SumMatch/(sqr(SourceSum) * TargetSum);
                end; {for x}
            end; {for y}
        end; {NormCorr}
    Moment : begin
        {Calc source base moments}
        for i := 0 to 3 do {initialise sBM}
            for j := 0 to 3 do
                sBM[i,j] := 0;
            end;
        end;
        for dy := -WindowRadius to WindowRadius do begin
            for dx := -WindowRadius to WindowRadius do begin
                SourcePixel :=
                    BytePtr(OffsetPointer(@SourceImage^.data,
                        centre.x + dx + (centre.y + dy) * rowByteCount))^;
                inc(sBM[0,0], SourcePixel);
                inc(sBM[0,1], dy*SourcePixel);
                inc(sBM[0,2], sqr(dy)*SourcePixel);
                inc(sBM[0,3], dy*sqr(dy)*SourcePixel);
                inc(sBM[1,0], dx*SourcePixel);
                inc(sBM[1,1], dx*dy*SourcePixel);
                inc(sBM[1,2], dx*sqr(dy)*SourcePixel);
                inc(sBM[2,0], dx*sqr(dx)*SourcePixel);
                inc(sBM[2,1], dx*dy*sqr(dx)*SourcePixel);
                inc(sBM[3,0], dx*sqr(dx)*dx*SourcePixel);
            end; {for dx}
        end; {for dy}
        cx := sBM[1,0]/sBM[0,0]; cy := sBM[0,1]/sBM[0,0]; {Calc centre of area coords}
        if MomentType >= RTS2 then begin {Calc source normalised central moments}
            sCM[0,0] := 1;
            sCM[0,1] := 0;
            sCM[0,2] := (sBM[0,2] - cy*sBM[0,1])/sqr(sBM[0,0]);
            sCM[0,3] :=
                (sBM[0,3] - 3*cy*sBM[0,2] + 2*sqr(cy)*sBM[0,1])/(sqr(sBM[0,0])*sqrt(sBM[0,0]));
            sCM[1,0] := 0;
            sCM[1,1] := (sBM[1,1] - cy*sBM[1,0])/sqr(sBM[0,0]);
            sCM[1,2] := (sBM[1,2] - 2*cy*sBM[1,1] - cx*sBM[0,2] + 2*sqr(cy)*sBM[1,0])
                /(sqr(sBM[0,0])*sqrt(sBM[0,0]));
            sCM[1,3] := 0;
            sCM[2,0] := (sBM[2,0] - cx*sBM[1,0])/sqr(sBM[0,0]);
            sCM[2,1] := (sBM[2,1] - 2*cx*sBM[1,1] - cy*sBM[2,0] + 2*sqr(cx)*sBM[1,0])
                /(sqr(sBM[0,0])*sqrt(sBM[0,0]));
        end;
    end;

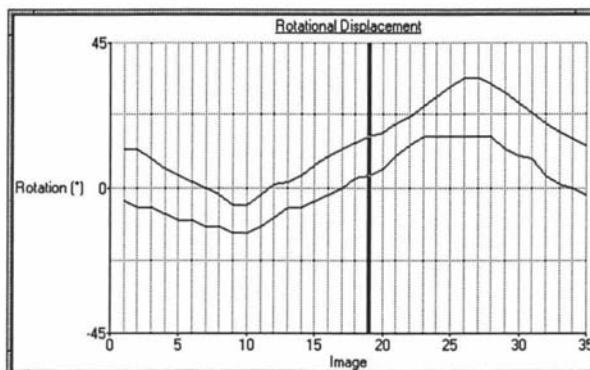
```



```

SCM[2,2] := 0;
SCM[3,0] := (SBM[3,0] - 3*cx*SBM[2,0] + 2*sqr(cx)*SBM[1,0])
            / (sqr(SBM[0,0])*sqrt(SBM[0,0]));
end
else begin {Calc source non-normalised central moments}
  sCM[0,0] := SBM[0,0];
  sCM[0,1] := 0;
  sCM[0,2] := (SBM[0,2] - cy*SBM[0,1]);
  sCM[0,3] := (SBM[0,3] - 3*cy*SBM[0,2] + 2*sqr(cy)*SBM[0,1]);
  sCM[1,0] := 0;
  sCM[1,1] := (SBM[1,1] - cy*SBM[1,0]);
  sCM[1,2] := (SBM[1,2] - 2*cy*SBM[1,1] - cx*SBM[0,2] + 2*sqr(cy)*SBM[1,0]);
  sCM[2,0] := (SBM[2,0] - cx*SBM[1,0]);
  sCM[2,1] := (SBM[2,1] - 2*cx*SBM[1,1] - cy*SBM[2,0] + 2*sqr(cx)*SBM[1,0]);
  sCM[3,0] := (SBM[3,0] - 3*cx*SBM[2,0] + 2*sqr(cx)*SBM[1,0]);
end; {else}
{Calc source moment invariants }
SMI[1] := sCM[2,0] + sCM[0,2];
SMI[2] := sqr(sCM[2,0]-sCM[0,2]) + 4*sqr(sCM[1,1]);
SMI[3] := sqr(sCM[3,0]-3*sCM[1,2]) + sqr(3*sCM[2,1] + sCM[0,3]);
SMI[4] := sqr(sCM[3,0]+sCM[1,2]) + sqr(sCM[2,1] + sCM[0,3]);
SMI[5] := (sCM[3,0] - 3*sCM[1,2])*(sCM[3,0]+sCM[1,2])*(sqr(sCM[3,0]+sCM[1,2])
-3*sqr(sCM[2,1]+sCM[0,3])) + (3*sCM[2,1]-sCM[0,3])*(sCM[2,1]
+ sCM[0,3])*(3*sqr(sCM[3,0]+sCM[1,2])-sqr(sCM[2,1]+sCM[0,3]));
SMI[6] := (sCM[2,0]-sCM[0,2])*(sqr(sCM[3,0]+sCM[1,2])-sqr(sCM[2,1]+sCM[0,3]))
+ 4*sCM[1,1]*(sCM[3,0]+sCM[1,2])*(sCM[2,1]+sCM[0,3]);
SMI[7] := (3*sCM[1,2]-sCM[3,0])*(sCM[3,0]+sCM[1,2])*(sqr(sCM[3,0]+sCM[1,2])
-3*sqr(sCM[2,1]+sCM[0,3])) + (3*sCM[2,1]-sCM[0,3])*(sCM[2,1]
+ sCM[0,3])*(3*sqr(sCM[3,0]+sCM[1,2])-sqr(sCM[2,1]+sCM[0,3]));
for y := -SearchRadius to SearchRadius do begin
  for x := -SearchRadius to SearchRadius do begin
    {compute target base moments}
    for i := 0 to 3 do {initialise tBM}
      for j := 0 to 3 do
        tBM[i,j] := 0;
    for dy := -WindowRadius to WindowRadius do begin
      for dx := -WindowRadius to WindowRadius do begin
        TargetPixel :=
          BytePtr(OffsetPointer(@TargetImage^.data,
                                centre.x + x+dx + (centre.y + y+dy) *rowByteCount))^;
        inc(tBM[0,0], TargetPixel);
        inc(tBM[0,1], dy*TargetPixel);
        inc(tBM[0,2], sqr(dy)*TargetPixel);
        inc(tBM[0,3], sqr(dy)*dy*TargetPixel);
        inc(tBM[1,0], dx*TargetPixel);
        inc(tBM[1,1], dx*dy*TargetPixel);
        inc(tBM[1,2], dx*sqr(dy)*TargetPixel);
        inc(tBM[2,0], sqr(dx)*TargetPixel);
        inc(tBM[2,1], sqr(dx)*dy*TargetPixel);
        inc(tBM[3,0], sqr(dx)*dx*TargetPixel);
      end; {for dx}
    end; {for dy}
    cx := tBM[1,0]/tBM[0,0]; cy := tBM[0,1]/tBM[0,0]; {Calc centre of area coords}
    if MomentType >= RTS2 then begin {Calc Normalise target central moments}
      tCM[0,0] := 1;
      tCM[0,1] := 0;
      tCM[0,2] := (tBM[0,2] - cy*tBM[0,1])/sqr(tBM[0,0]);
      tCM[0,3] := (tBM[0,3] - 3*cy*tBM[0,2] + 2*sqr(cy)*tBM[0,1])
                  / (sqr(tBM[0,0])*sqrt(tBM[0,0]));
      tCM[1,0] := 0;
      tCM[1,1] := (tBM[1,1] - cy*tBM[1,0])/sqr(tBM[0,0]);
      tCM[1,2] := (tBM[1,2] - 2*cy*tBM[1,1] - cx*tBM[0,2] + 2*sqr(cy)*tBM[1,0])
                  / (sqr(tBM[0,0])*sqrt(tBM[0,0]));
      tCM[2,0] := (tBM[2,0] - cx*tBM[1,0])/sqr(tBM[0,0]);
      tCM[2,1] := (tBM[2,1] - 2*cx*tBM[1,1] - cy*tBM[2,0] + 2*sqr(cx)*tBM[1,0])
                  / (sqr(tBM[0,0])*sqrt(tBM[0,0]));
      tCM[3,0] := (tBM[3,0] - 3*cx*tBM[2,0] + 2*sqr(cx)*tBM[1,0])
                  / (sqr(tBM[0,0])*sqrt(tBM[0,0]));
    end
  else begin {Calc non-normalised central target moments}
    tCM[0,0] := tBM[0,0];
    tCM[0,1] := 0;

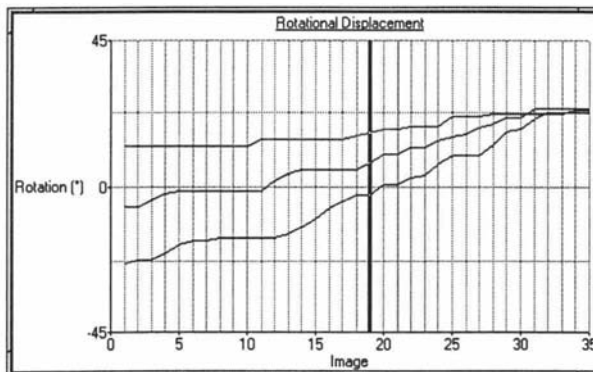
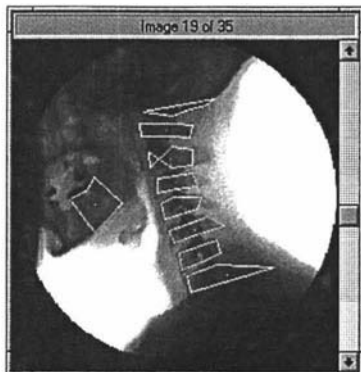
```



```

tCM[0,2] := (tBM[0,2] - cy*tBM[0,1]);
tCM[0,3] := (tBM[0,3] - 3*cy*tBM[0,2] + 2*sqr(cy)*tBM[0,1]);
tCM[1,0] := 0;
tCM[1,1] := (tBM[1,1] - cy*tBM[1,0]);
tCM[1,2] := (tBM[1,2] - 2*cy*tBM[1,1] - cx*tBM[0,2] + 2*sqr(cy)*tBM[1,0]);
tCM[2,0] := (tBM[2,0] - cx*tBM[1,0]);
tCM[2,1] := (tBM[2,1] - 2*cx*tBM[1,1] - cy*tBM[2,0] + 2*sqr(cx)*tBM[1,0]);
tCM[3,0] := (tBM[3,0] - 3*cx*tBM[2,0] + 2*sqr(cx)*tBM[1,0])
end;
( Calc target moment invariants )
tMI[1] := tCM[2,0] + tCM[0,2];
tMI[2] := sqr(tCM[2,0]-tCM[0,2]) + 4*sqr(tCM[1,1]);
tMI[3] := sqr(tCM[3,0]-3*tCM[1,2]) + sqr(3*tCM[2,1] + tCM[0,3]);
tMI[4] := sqr(tCM[3,0]+tCM[1,2]) + sqr(tCM[2,1] + tCM[0,3]);
tMI[5] := (tCM[3,0]-3*tCM[1,2])*(tCM[3,0]+tCM[1,2])*(sqr(tCM[3,0]+tCM[1,2])
-3*sqr(tCM[2,1]+tCM[0,3])) + (3*tCM[2,1]-tCM[0,3])*(tCM[2,1]
+ tCM[0,3])*(3*sqr(tCM[3,0]+tCM[1,2])-sqr(tCM[2,1]+tCM[0,3]));
tMI[6] := (tCM[2,0]-tCM[0,2])*(sqr(tCM[3,0]+tCM[1,2])-sqr(tCM[2,1]+tCM[0,3]))
+ 4*tCM[1,1]*(tCM[3,0]+tCM[1,2])*(tCM[2,1]+tCM[0,3]);
tMI[7] := (3*tCM[1,2]-tCM[3,0])*(tCM[3,0]+tCM[1,2])*(sqr(tCM[3,0]+tCM[1,2])
-3*sqr(tCM[2,1]+tCM[0,3])) + (3*tCM[2,1]-tCM[0,3])*(tCM[2,1]
+ tCM[0,3])*(3*sqr(tCM[3,0]+tCM[1,2])-sqr(tCM[2,1]+tCM[0,3]));
case MomentType of
RT2: RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
4*abs(ln(abs(sMI[1]))-ln(abs(tMI[1])))
+ 0.25*abs(ln(abs(sMI[2]))-ln(abs(tMI[2])));
RT4: RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
4*abs(ln(abs(sMI[1]))-ln(abs(tMI[1])))
+ abs(ln(abs(sMI[2]))-ln(abs(tMI[2])))
+ abs(ln(abs(sMI[3]))-ln(abs(tMI[3])))
+ abs(ln(abs(sMI[4]))-ln(abs(tMI[4])));
RT6: RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
abs(ln(abs(sMI[1]))-ln(abs(tMI[1])))
+ abs(ln(abs(sMI[2]))-ln(abs(tMI[2])))
+ abs(ln(abs(sMI[3]))-ln(abs(tMI[3])))
+ abs(ln(abs(sMI[4]))-ln(abs(tMI[4])))
+ abs(ln(abs(sMI[5]))-ln(abs(tMI[5])))
+ abs(ln(abs(sMI[6]))-ln(abs(tMI[6])));
RTS2: RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
abs(sMI[1]-tMI[1]) + abs(sMI[2]-tMI[2]);
RTS4: RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
abs(sMI[1]-tMI[1]) + abs(sMI[2]-tMI[2])
+ abs(sMI[3]-tMI[3]) + abs(sMI[4]-tMI[4]);
RTS6: begin (Combine first 6 moment invariants using a normalised correlation)
MatchSumD := 0;
SourceSumD := 0;
TargetSumD := 0;
for i := 1 to 6 do begin
MatchSumD := MatchSumD + abs(sMI[i]*tMI[i]);
SourceSumD := SourceSumD + sqr(sMI[i]);
TargetSumD := TargetSumD + sqr(tMI[i]);
end; (for i)
RealPtr(OffsetPointer(@MatchMap^.data,
((SearchRadius-y)*SearchWidth + x+SearchRadius) * SizeOfReal))^ :=
MatchSumD / sqrt(SourceSumD * TargetSumD);
end; (RTS6)
end; (case)
end; (for x)
end; (for y)
end; (Moment)
end; (Case MatchType of)
end; (ComputeMatchMap)
(=====)
end. (Matching)

```



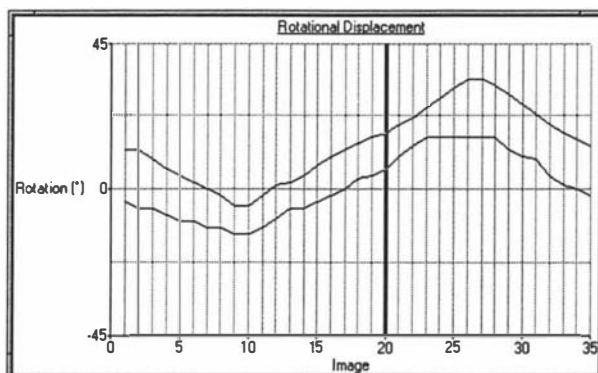

```

(-----CalcMatchSelectivity-----)
Procedure CalcMatchSelectivity(
    MatchMapH : MatrixHandle; CentreBase : integer; NumShells : byte;
    var MinSel, AveSel : real; var MatchPosition : TPoint);
{ Calculates the selectivity from the MatchMap. This measure computes the minimum
  difference between the centre match value and values in a square annulus of radius
  1..NumShells. These minima are normalised by the maximum difference across all shell
  annulus and averaged.
    SelValue = (min(M0-M1)/Mmax + min(M0-M2)/Mmax + min(M0-M3)/Mmax + ...)/NumShells

  If CentreBase = 0 then the centre is taken as the middle of MatchMapH
  If CentreBase < 0 then the location of the minimum is used as centre
  if CentreBase > 0 then the location of the maximum is used as the centre.
}
const
    MaxNumShells = 5;    { number of shells scanned to compute selectivity }
var
    ShellNo      : byte;
    x,y, i,j     : integer;
    CenValue,    { Value at the Centre of MatchMap }
    Value
    : real;
    MinDiff      : Array[1..MaxNumShells] of real;
    {Min difference between the centre value and values in each shell }
    MaxDiff,     {Max difference between the centre value and values across all 4 shells}
    Max, Min
    : real;
    SumDiff      {Sum of the differences between the centre value for the current shell}
    : Array[1..MaxNumShells] of real;
    MatchMap     : MatrixPtr;
    Centre       : TPoint;
begin
    if NumShells > MaxNumShells then begin
        writeln('Error: too many shells, max of 5');
        readln;
        exit;
    end;
    MatchMap := GlobalLock(MatchMapH);
    x := MatchMap^.size.x;
    y := MatchMap^.size.y;
    if CentreBase = 0 then begin { use middle of MatchMap as base centre}
        Centre.x := x div 2;
        Centre.y := y div 2;
    end
    else if CentreBase > 0 then begin { Find maximum value coords }
        Max := -1;
        for i := 0 to y-1 do
            for j := 0 to x-1 do begin
                Value := RealPtr(OffsetPointer(@MatchMap^.data, (j + i * x) * SizeOfReal))^;
                if Value > Max then begin
                    Max := Value;
                    Centre.x := j; Centre.y := i
                end; {if}
            end; {for j}
        end {elseif}
    else begin { Find minimum value coords }
        Min := 1E9;
        for i := 0 to y-1 do
            for j := 0 to x-1 do begin
                Value := RealPtr(OffsetPointer(@MatchMap^.data, (j + i * x) * SizeOfReal))^;
                if Value < Min then begin
                    Min := Value;
                    Centre.x := j; Centre.y := i
                end; {if}
            end; {for j}
        end;
    MatchPosition.x := Centre.x;
    MatchPosition.y := Centre.y;
    if (Centre.x >= NumShells) and (Centre.x < x-1-NumShells)
       and (Centre.y >= NumShells) and (Centre.y < y-1-NumShells) then begin

```

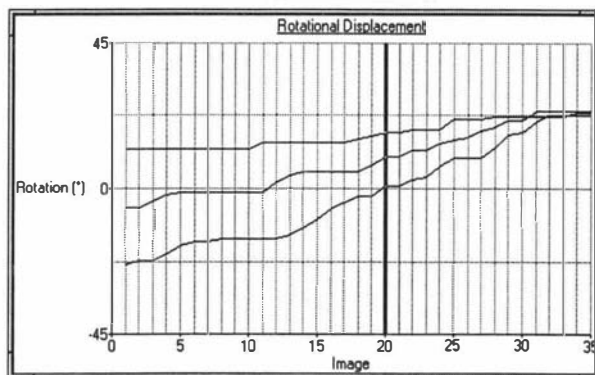
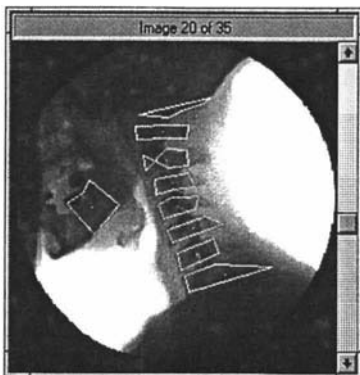
Warp60



```

for i := 1 to NumShells do MinDiff[i] := 1E9;
MaxDiff := 0;
CenValue :=
  RealPtr(OffsetPointer(@MatchMap^.data, (Centre.x + Centre.y * y) * SizeOfReal))^;
for ShellNo := 1 to NumShells do begin
  SumDiff[ShellNo] := 0;
  for i := -ShellNo to ShellNo-1 do begin
    { Top: left to right }
    Value :=
      abs(CenValue - RealPtr(OffsetPointer(@MatchMap^.data,
        (Centre.x+i + (Centre.y-ShellNo) * y) * SizeOfReal))^);
    SumDiff[ShellNo] := SumDiff[ShellNo] + Value;
    if Value < MinDiff[ShellNo] then MinDiff[ShellNo] := Value;
    if Value > MaxDiff then MaxDiff := Value;
    { Right: top to bottom }
    Value :=
      abs(CenValue - RealPtr(OffsetPointer(@MatchMap^.data,
        (Centre.x+ShellNo + (Centre.y+i) * y) * SizeOfReal))^);
    SumDiff[ShellNo] := SumDiff[ShellNo] + Value;
    if Value < MinDiff[ShellNo] then MinDiff[ShellNo] := Value;
    if Value > MaxDiff then MaxDiff := Value;
    { Bottom: left to right }
    Value :=
      abs(CenValue - RealPtr(OffsetPointer(@MatchMap^.data,
        (Centre.x+i+1 + (Centre.y+ShellNo)*y) * SizeOfReal))^);
    SumDiff[ShellNo] := SumDiff[ShellNo] + Value;
    if Value < MinDiff[ShellNo] then MinDiff[ShellNo] := Value;
    if Value > MaxDiff then MaxDiff := Value;
    { Right: top to bottom }
    Value :=
      abs(CenValue - RealPtr(OffsetPointer(@MatchMap^.data,
        (Centre.x-ShellNo + (Centre.y+i+1)*y) * SizeOfReal))^);
    SumDiff[ShellNo] := SumDiff[ShellNo] + Value;
    if Value < MinDiff[ShellNo] then MinDiff[ShellNo] := Value;
    if Value > MaxDiff then MaxDiff := Value;
  end; {for i}
end; {for ShellNo}
MinSel := 0;
AveSel := 0;
for ShellNo := 1 to NumShells do begin
  MinSel := MinSel + MinDiff[ShellNo];
  AveSel := AveSel + SumDiff[ShellNo]/(8*ShellNo)
end; {for i}
If MaxDiff = 0.0 then begin
  MinSel := 0;
  AveSel := 0;
end
else begin
  MinSel := MinSel / (MaxDiff * NumShells);
  AveSel := AveSel / (MaxDiff * NumShells);
end;
end {if Centre within MapMatch Area}
else begin
  MinSel := -0.02;
  AveSel := -0.02;
end;
end; {CalcMatchSelectivity}
(-----ReadBackwardAndForwardImages-----)
Procedure ReadBackwardAndForwardImages;
{ Read the previous and next images about the centre image
  previous->Image3H, Next->Image4H }
var
  Name : Str255;
begin
  StrPCopy(Name, '');
  writeln('loading images');
  CMXDocName := 'cervical';
  {Find the document}
  StrPCopy(CMXDocPathP, CMXDocPath);
  fileNo := 0;
  LStrCat(LStrCat(@Name, CMXDocPathP), '\doc?????.cmx');

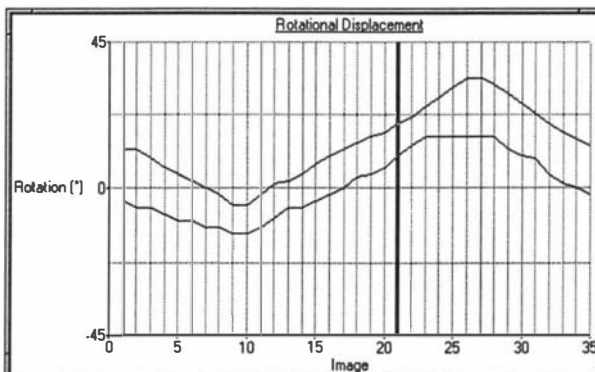
```



```

FindFirst(@Name, faReadOnly, searchInfo);
error := BPErrors(DosError);
if Error <> NoErr then begin
  writeln('Error finding CMX documents, path incorrect ', Name);
  Exit;
end;
FileFound := False;
while (Error = NoErr) and not(FileFound) do begin
  LStrCat(LStrCat(StrPCopy(@Name, CMXDocPathP), '/'), @searchInfo.name);
  Error := FileOpen(TFileRec(inFile), @name, Version, Nil, False);
  if Error <> NoErr then begin
    writeln('Error opening input file ', name);
    Exit;
  end;
  Error := ReadDocID(TFileRec(inFile), docID);
  if Error <> NoErr then begin
    writeln('Error reading input file info header ', Error);
    FileClose(TFileRec(inFile), Error);
    Exit;
  end;
  StrPCopy(CMXDocNameP, CMXDocName);
  if LStrCmpi(DocID.name, CMXDocNameP) = 0 then begin
    FileFound := True;
  end
  else FindNext(searchInfo);
  Error := BPErrors(DosError);
  FileClose(TFileRec(inFile), Error);
end; {while}
if Error <> NoErr then begin
  writeln('Error finding matching CMX file ', CMXDocName);
  Exit;
end;
Error := FileOpen(TFileRec(inFile), @name, Version, Nil, False);
if Error <> NoErr then begin
  writeln('Error opening input file', name);
  Exit;
end;
Error :=
  FileReadData(TFileRec(inFile), DocInfoDataID, @DocumentInfo, SizeOf(DocumentInfo), nil, 0);
if Error <> NoErr then begin
  writeln('Error reading input file info header ', Error);
  FileClose(TFileRec(inFile), Error);
  Exit;
end;
Error := ReadImage(Image2H, InFile, ImageDataID + (DocumentInfo.ImageCount div 2) - 1);
if Error <> NoErr then begin
  writeln('Error reading centre image');
  Exit;
end;
Error := DecompressImage(Image2H, Image3H); {Image3H created by decompress}
if Error <> NoErr then begin
  writeln('Error decompressing image');
  Exit;
end;
GradientFilterImage(Image3H, Image2H, 6.0/5);
NoFilterImage(Image2H, Image3H);
Error := ReadImage(Image2H, InFile, ImageDataID + (DocumentInfo.ImageCount div 2));
if Error <> NoErr then begin
  writeln('Error reading centre image');
  Exit;
end;
Error := DecompressImage(Image2H, Image1H); {Image1H created by decompress}
if Error <> NoErr then begin
  writeln('Error decompressing image');
  Exit;
end;
if Error <> NoErr then begin
  writeln('Error decompressing image');
  Exit;
end;
GradientFilterImage(Image1H, Image2H, 6.0/5);

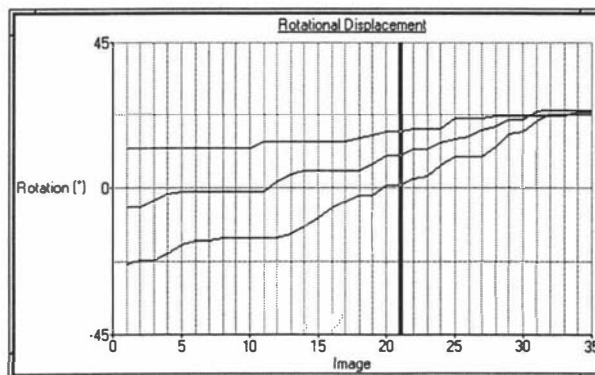
```



```

NoFilterImage(Image2H, Image1H);
Error := ReadImage(Image2H, InFile, ImageDataID + (DocumentInfo.ImageCount div 2)+1);
if Error <> NoErr then begin
  writeln('Error reading centre image');
  Exit;
end;
Error := DecompressImage(Image2H, Image4H); {Image4H created by decompress}
if Error <> NoErr then begin
  writeln('Error decompressing image');
  Exit;
end;
GradientFilterImage(Image4H, Image2H, 6.0/5 );
NoFilterImage(Image2H, Image4H);
FileClose(TFileRec(InFile), Error);
if Error <> NoErr then begin
  Writeln('Error closing input file');
  Exit;
end;
end; { ReadBackwardAndForwardImages}
{-----MatchMarkedPoints-----}
Procedure MatchMarkedPoints;
var
  MatchCoord : TPoint;
begin
  for pointGroupNo := 0 to GetPointGroupCount(pointInfoH)-1 do begin
    GetPointGroupName(pointInfoH, pointGroupNo, pointGroupName);
    for PointNo := 0 to GetPointCount(pointInfoH, @pointGroupName)-1 do begin
      GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
      writeln('Processing Name:', pointGroupName, ' No:',
        PointNo, ' (x,y)=', Point.x, ',', Point.y);
      ComputeMatchMap(Image1H, 0, SearchRadius, WindowRadius, MatchType, Point, MatchMapH);
      CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells, MinSel, AveSel, MatchCoord);
      RealPtr(@Selectivity^.data)^ := MinSel;
      RealPtr(OffsetPointer(@Selectivity^.data, SizeOfReal))^ := AveSel;
    end; {for PointNo}
    DataName := StrPas(pointGroupName);
    if PointGroupNo = 0 then begin
      Error := SaveMat(DataDocName, DataName, MatchMapH, 0, False);
      Error := SaveMat(DataDocName, DataName + '_sel', SelectivityH, 0, True);
    end
    else begin
      Error := SaveMat(DataDocName, DataName, MatchMapH, 0, True);
      Error := SaveMat(DataDocName, DataName + '_sel', SelectivityH, 0, True);
    end;
  end; {for pointGroupNo}
end; {MatchMarkedPoints}
{-----MatchAllPoints-----}
Procedure MatchAllPoints;
var
  x, y,
  Clearance : longint;
  MatrixSize : Tpoint;
  MatchCoord : TPoint;
  OutMapH : MatrixHandle;
begin
  ReadBackwardAndForwardImages;
  Clearance := WindowRadius + SearchRadius;
  MatrixSize.x := ImageSize.x-(2*Clearance+1);
  MatrixSize.y := ImageSize.y-(2*Clearance+1);
  Error := CreateMatrix(SelMinMapH, MatrixSize.x, MatrixSize.y, DoublePrecision);
  if Error <> NoErr then begin
    Writeln('Error creating Matrix SelMin');
    Exit;
  end;
  SelMinMap := GlobalLock(SelMinMapH);
  Error := CreateMatrix(SelAveMapH, MatrixSize.x, MatrixSize.y, DoublePrecision);
  if Error <> NoErr then begin
    Writeln('Error creating Matrix SelAve');
    exit;
  end;
  SelAveMap := GlobalLock(SelAveMapH);

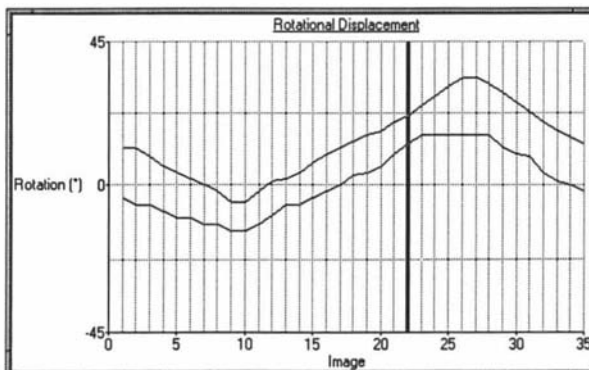
```



```

writeln('Matching previous image ');
for y := 0 to MatrixSize.y-1 do begin
  for x := 0 to MatrixSize.x-1 do begin
    Point.x := x+Clearance;
    Point.y := y+Clearance;
    ComputeMatchMap(Image1H, Image3H, SearchRadius,
      WindowRadius, MatchType, Point, MatchMapH);
    CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
      MinSel, AveSel, MatchCoord);
    RealPtr(OffsetPointer(@SelMinMap^.data,
      (x + (MatrixSize.y-1-y)*MatrixSize.x) * SizeOfDouble))^ := MinSel;
    RealPtr(OffsetPointer(@SelAveMap^.data,
      (x + (MatrixSize.y-1-y)*MatrixSize.x) * SizeOfDouble))^ := AveSel;
  end; (for x)
end; (for y)
Error := ConvertMatrix(SelMinMapH, OutMapH, BytePrecision, True);
if Error <> NoErr then begin
  writeln('ConvertError');
  exit;
end;
Error := SaveMat(FileType + 'minp.mat', 'Sel_min', OutMapH, 0, False);
DestroyMatrix(OutMapH);
Error := ConvertMatrix(SelAveMapH, OutMapH, BytePrecision, True);
Error := SaveMat(FileType + 'avep.mat', 'Sel_ave', OutMapH, 0, False);
DestroyMatrix(OutMapH);
writeln('Matching current image');
for y := 0 to MatrixSize.y-1 do begin
  for x := 0 to MatrixSize.x-1 do begin
    Point.x := x+Clearance;
    Point.y := y+Clearance;
    ComputeMatchMap(Image1H, Image1H, SearchRadius, WindowRadius,
      MatchType, Point, MatchMapH);
    CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
      MinSel, AveSel, MatchCoord);
    RealPtr(OffsetPointer(@SelMinMap^.data,
      (x + (MatrixSize.y-1-y)*MatrixSize.x) * SizeOfReal))^ := MinSel;
    RealPtr(OffsetPointer(@SelAveMap^.data,
      (x + (MatrixSize.y-1-y)*MatrixSize.x) * SizeOfReal))^ := AveSel;
  end; (for x)
  write('.');
end; (for y)
Error := ConvertMatrix(SelMinMapH, OutMapH, BytePrecision, True);
Error := SaveMat(FileType + 'minc.mat', 'Sel_min', OutMapH, 0, False);
DestroyMatrix(OutMapH);
Error := ConvertMatrix(SelAveMapH, OutMapH, BytePrecision, True);
Error := SaveMat(FileType + 'avec.mat', 'Sel_ave', OutMapH, 0, False);
DestroyMatrix(OutMapH);
writeln('Matching next image');
for y := 0 to MatrixSize.y-1 do begin
  for x := 0 to MatrixSize.x-1 do begin
    Point.x := x+Clearance;
    Point.y := y+Clearance;
    ComputeMatchMap(Image1H, Image4H, SearchRadius, WindowRadius,
      MatchType, Point, MatchMapH);
    CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
      MinSel, AveSel, MatchCoord);
    RealPtr(OffsetPointer(@SelMinMap^.data,
      (x + (MatrixSize.y-1 - y)*MatrixSize.x) * SizeOfReal))^ := MinSel;
    RealPtr(OffsetPointer(@SelAveMap^.data,
      (x + (MatrixSize.y-1 - y)*MatrixSize.x) * SizeOfReal))^ := AveSel;
  end; (for x)
  write('.');
end; (for y)
Error := ConvertMatrix(SelMinMapH, OutMapH, BytePrecision, True);
Error := SaveMat(FileType + 'minn.mat', 'Sel_min', OutMapH, 0, False);
DestroyMatrix(OutMapH);
Error := ConvertMatrix(SelAveMapH, OutMapH, BytePrecision, True);
Error := SaveMat(FileType + 'aven.mat', 'Sel_ave', OutMapH, 0, False);
DestroyMatrix(OutMapH);
DestroyMatrix(SelMinMapH);
DestroyMatrix(SelAveMapH);
end; (MatchAllPoints)

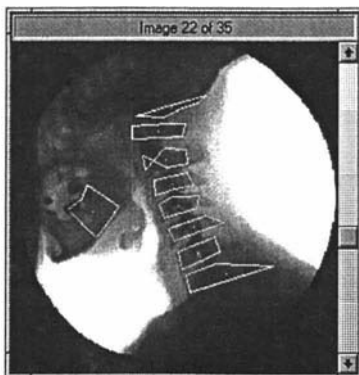
```



```

(-----CalcSelRadEffects-----)
Procedure CalcSelRadEffects;
var
  SelRadius,
  PointGroupCount
    : byte;
  MinSelRadMapH,
  AveSelRadMapH
    : MatrixHandle;
  MinSelRadmap,
  AveSelRadMap
    : MatrixPtr;
  MatchCoord : TPoint;
begin
  PointGroupCount := GetPointGroupCount(pointInfoH);
  Error := CreateMatrix(MinSelRadMapH, 5, PointGroupCount, DoublePrecision);
  MinSelRadMap := GlobalLock(MinSelRadMapH);
  Error := CreateMatrix(AveSelRadMapH, 5, PointGroupCount, DoublePrecision);
  AveSelRadMap := GlobalLock(AveSelRadMapH);
  for pointGroupNo := 0 to PointGroupCount-1 do begin
    GetPointGroupName(pointInfoH, pointGroupNo, pointGroupName);
    for PointNo := 0 to GetPointCount(pointInfoH, @pointGroupName)-1 do begin
      for SelRadius := 1 to 5 do begin
        GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
        writeln('Processing Name:', pointGroupName,
          ' No:', PointNo, ' (x,y)=', Point.x, ',', Point.y);
        ComputeMatchMap(ImagelH, 0, SearchRadius, WindowRadius,
          MatchType, Point, MatchMapH);
        CalcMatchSelectivity(MatchMapH, SelectivityBase, SelRadius,
          MinSel, AveSel, MatchCoord);
        RealPtr(OffsetPointer(@MinSelRadMap^.data,
          (pointGroupNo + (SelRadius-1)*PointGroupCount)*SizeOfReal))^ := MinSel;
        RealPtr(OffsetPointer(@AveSelRadMap^.data,
          (pointGroupNo + (SelRadius-1)*PointGroupCount)*SizeOfReal))^ := AveSel;
      end; (for SelRadius)
    end; (for PointNo)
  end; (for pointGroupNo)
  Error := SaveMat('cor5_min.mat', 'Sel_min', MinSelRadMapH, 0, False);
  Error := SaveMat('cor5_ave.mat', 'Sel_ave', AveSelRadMapH, 0, False);
  DestroyMatrix(MinSelRadMapH);
  DestroyMatrix(AveSelRadMapH);
end; (CalcSelRadEffects)
(-----CalcSelWindowEffects-----)
Procedure CalcSelWindowEffects;
  { Calculates the selectivity for various window radius }
var
  PointGroupCount
    : byte;
  MinSelRadMapH,
  AveSelRadMapH
    : MatrixHandle;
  MinSelRadmap,
  AveSelRadMap
    : MatrixPtr;
  MatchCoord : TPoint;
begin
  PointGroupCount := GetPointGroupCount(pointInfoH);
  Error := CreateMatrix(MinSelRadMapH, 6, PointGroupCount, DoublePrecision);
  MinSelRadMap := GlobalLock(MinSelRadMapH);
  Error := CreateMatrix(AveSelRadMapH, 6, PointGroupCount, DoublePrecision);
  AveSelRadMap := GlobalLock(AveSelRadMapH);
  for pointGroupNo := 0 to PointGroupCount-1 do begin
    GetPointGroupName(pointInfoH, pointGroupNo, pointGroupName);
    for PointNo := 0 to GetPointCount(pointInfoH, @pointGroupName)-1 do begin
      for WindowRadius := 1 to 6 do begin
        GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
        writeln('Processing Name:', pointGroupName,
          ' No:', PointNo, ' (x,y)=', Point.x, ',', Point.y);
        ComputeMatchMap(ImagelH, 0, SearchRadius, WindowRadius,
          MatchType, Point, MatchMapH);
        CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
          MinSel, AveSel, MatchCoord);
      end;
    end;
  end;

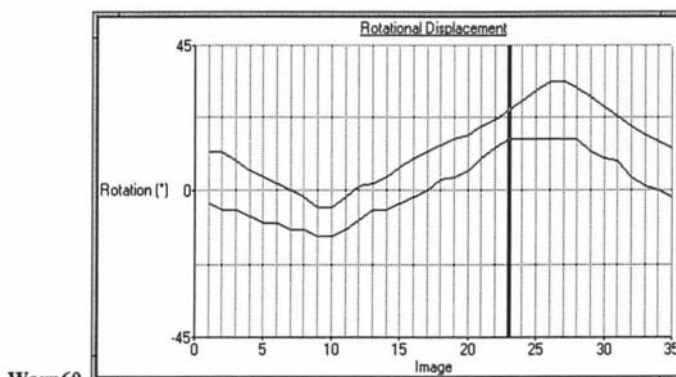
```



```

    RealPtr(OffsetPointer(@MinSelRadMap^.data,
        (pointGroupNo + (WindowRadius-1)*PointGroupCount)*SizeOfReal))^ := MinSel;
    RealPtr(OffsetPointer(@AveSelRadMap^.data,
        (pointGroupNo + (WindowRadius-1)*PointGroupCount)*SizeOfReal))^ := AveSel;
end; (for WindowRadius)
end; (for PointNo)
end; (for pointGroupNo)
Error := SaveMat('rms_wrad.mat', 'rms_min', MinSelRadMapH, 0, False);
Error := SaveMat('rms_wrad.mat', 'rms_ave', AveSelRadMapH, 0, True);
DestroyMatrix(MinSelRadMapH);
DestroyMatrix(AveSelRadMapH);
end; (CalcSelWindowEffects)
(-----CalcSelWindowEffectsBetweenImages-----)
Procedure CalcSelWindowEffectsBetweenImages;
{ Calculates the selectivity for various window radius, between images }
var
    PointGroupCount,
    MaxWinRadius
        : byte;
    SelWinRadmapH,
    OutMatrixH,
    xCoordsH,
    yCoordsH
        : MatrixHandle;
    SelWinRadmap,
    xCoords,
    yCoords
        : MatrixPtr;
    MatchCoord
        : TPoint;
begin
    MaxWinRadius := 8;
    ReadBackwardAndForwardImages;
    PointGroupCount := GetPointGroupCount(pointInfoH);
    Error := CreateMatrix(SelWinRadMapH, MaxWinRadius, 6*PointGroupCount, DoublePrecision);
    SelWinRadMap := GlobalLock(SelWinRadMapH);
    Error := CreateMatrix(xCoordsH, MaxWinRadius, 3*PointGroupCount, DoublePrecision);
    xCoords := GlobalLock(xCoordsH);
    Error := CreateMatrix(yCoordsH, MaxWinRadius, 3*PointGroupCount, DoublePrecision);
    yCoords := GlobalLock(yCoordsH);
    for pointGroupNo := 0 to PointGroupCount-1 do begin
        GetPointGroupName(pointInfoH, pointGroupNo, pointGroupName);
        for PointNo := 0 to GetPointCount(pointInfoH, @pointGroupName)-1 do begin
            { Match Previous frame }
            writeln('Previous Frame');
            for WindowRadius := 1 to MaxWinRadius do begin
                GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
                ComputeMatchMap(ImagelH, Image3H, SearchRadius, WindowRadius,
                    MatchType, Point, MatchMapH);
                CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells, MinSel,
                    AveSel, MatchCoord);
                writeln('Processing Name:', pointGroupName, ' (x,y)=', Point.x, ', ', Point.y,
                    ', ', WindowRadius, ' ', MatchCoord.x, ', ', MatchCoord.y);
                RealPtr(OffsetPointer(@SelWinRadMap^.data, ((0*pointGroupCount) + PointGroupNo
                    + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := MinSel;
                RealPtr(OffsetPointer(@SelWinRadMap^.data, ((3*PointGroupCount) + PointGroupNo
                    + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := AveSel;
                RealPtr(OffsetPointer(@xCoords^.data, (0 * PointGroupCount + PointGroupNo
                    + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.x;
                RealPtr(OffsetPointer(@yCoords^.data, (0 * PointGroupCount + PointGroupNo
                    + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.y;
            end; (for WindowRadius)
            { Match Centre frame }
            writeln('Middle Frame');
            for WindowRadius := 1 to MaxWinRadius do begin
                GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
                ComputeMatchMap(ImagelH, ImagelH, SearchRadius, WindowRadius,
                    MatchType, Point, MatchMapH);
                CalcMatchSelectivity(MatchMapH, (SelectivityBase) 0, NoShells,
                    MinSel, AveSel, MatchCoord);
                writeln('Processing Name:', pointGroupName, ' (x,y)=', Point.x, ', ', Point.y,
                    ', ', WindowRadius, ' ', MatchCoord.x, ', ', MatchCoord.y);
            end; (for WindowRadius)
        end; (for pointGroupNo)
    end;

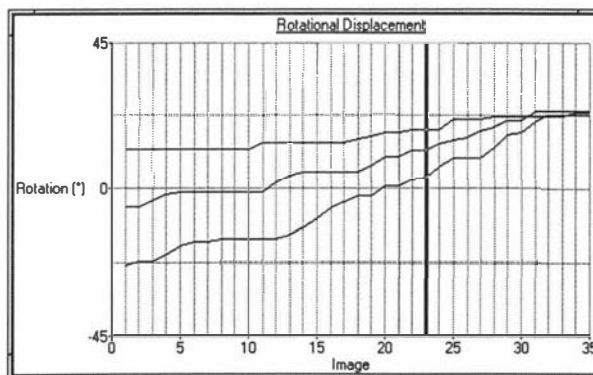
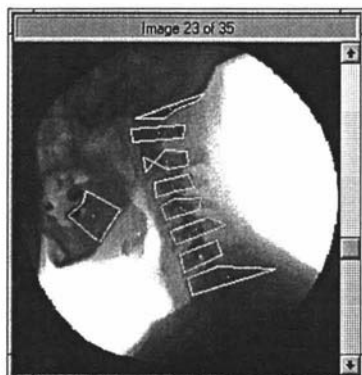
```



```

    RealPtr(OffsetPointer(@SelWinRadMap^.data, ((1 * pointGroupCount) + PointGroupNo
        + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := MinSel;
    RealPtr(OffsetPointer(@SelWinRadMap^.data, ((4 * PointGroupCount) + PointGroupNo
        + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := AveSel;
    RealPtr(OffsetPointer(@xCoords^.data, (1 * PointGroupCount + PointGroupNo
        + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.x;
    RealPtr(OffsetPointer(@yCoords^.data, (1 * PointGroupCount + PointGroupNo
        + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.y;
    end; {for WindowRadius}
    { Match next frame }
    writeln('Next Frame');
    for WindowRadius := 1 to MaxWinRadius do begin
        GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
        ComputeMatchMap(Image1H, Image4H, SearchRadius, WindowRadius,
            MatchType, Point, MatchMapH);
        CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
            MinSel, AveSel, MatchCoord);
        writeln('Processing Name:', pointGroupName, ' (x,y)=', Point.x, ',', Point.y,
            ', ', WindowRadius, ', ', MatchCoord.x, ', ', MatchCoord.y);
        RealPtr(OffsetPointer(@SelWinRadMap^.data, ((2 * pointGroupCount) + PointGroupNo
            + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := MinSel;
        RealPtr(OffsetPointer(@SelWinRadMap^.data, ((5 * PointGroupCount) + PointGroupNo
            + (WindowRadius-1)*6*PointGroupCount)*SizeOfReal))^ := AveSel;
        RealPtr(OffsetPointer(@xCoords^.data, (2 * PointGroupCount + PointGroupNo
            + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.x;
        RealPtr(OffsetPointer(@yCoords^.data, (2 * PointGroupCount + PointGroupNo
            + (WindowRadius-1) * 3*PointGroupCount)*SizeOfReal))^ := MatchCoord.y;
    end; {for WindowRadius}
    end; {for PointNo}
end; {for pointGroupNo}
Error := SaveMat('RTS6drad.mat', 'dyna_sel', SelWinRadMapH, 0, False);
Error := SaveMat('RTS6drad.mat', 'dyna_x', xCoordsH, 0, True);
Error := SaveMat('RTS6drad.mat', 'dyna_y', yCoordsH, 0, True);
DestroyMatrix(SelWinRadMapH);
DestroyMatrix(xCoordsH);
DestroyMatrix(yCoordsH);
end; {CalcSelWindowEffectsBetweenImages}
{-----MatchPointsBetweenImages-----}
Procedure MatchPointsBetweenImages(Backward : boolean);
var
    MatchCoord : TPoint;
begin
    writeln(' MatchPointsBetweenImages');
    ReadBackwardAndForwardImages;
    for pointGroupNo := 0 to GetPointGroupCount(pointInfoH)-1 do begin
        GetPointGroupName(pointInfoH, pointGroupNo, pointGroupName);
        for PointNo := 0 to GetPointCount(pointInfoH, @pointGroupName)-1 do begin
            GetPoint(pointInfoH, @pointGroupName, pointNo, Point);
            writeln('Processing Name:', pointGroupName,
                ' No:', PointNo, ' (x,y)=', Point.x, ',', Point.y);
            if Backward then
                ComputeMatchMap(Image1H, Image3H, SearchRadius, WindowRadius,
                    MatchType, Point, MatchMapH)
            else ComputeMatchMap(Image1H, Image4H, SearchRadius, WindowRadius,
                MatchType, Point, MatchMapH);
            CalcMatchSelectivity(MatchMapH, SelectivityBase, NoShells,
                MinSel, AveSel, MatchCoord);
            RealPtr(@Selectivity^.data)^ := MinSel;
            RealPtr(OffsetPointer(@Selectivity^.data, SizeOfReal))^ := AveSel;
        end; {for PointNo}
        DataName := StrPas(pointGroupName);
        if PointGroupNo = 0 then begin
            Error := SaveMat(DataDocName, DataName, MatchMapH, 0, False);
            Error := SaveMat(DataDocName, DataName + '_sel', SelectivityH, 0, True);
        end
        else begin
            Error := SaveMat(DataDocName, DataName, MatchMapH, 0, True);
            Error := SaveMat(DataDocName, DataName + '_sel', SelectivityH, 0, True);
        end;
    end; {for pointGroupNo}
end; {MatchPointsBetweenImages}

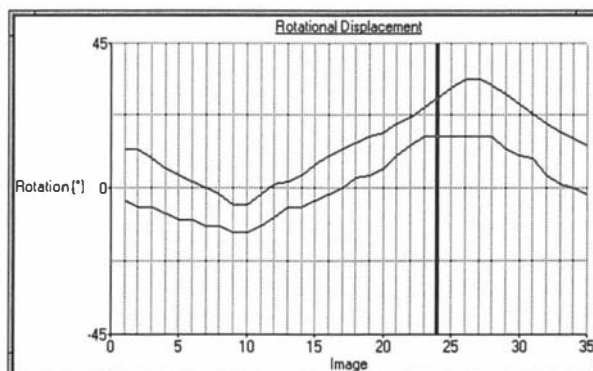
```




```

(===== Main Mapping =====)
begin
  ScreenSize.x := 65;
  ScreenSize.y := 25;
  (Define run parameters)
  WindowRadius := 5;
  SearchRadius := 7;
  MatchType := (AbsMeanDiff)(RMSDiff)(NormCorr)Moment;
  MomentType := RTS6;
  str(WindowRadius, S);
  case MatchType of
    AbsMeanDiff : FileType := 'abs' + s;
    RMSDiff : FileType := 'rms' + s;
    NormCorr : FileType := 'cor' + s;
    Moment :
      case MomentType of
        RTS2 : FileType := 'RTS2';
        RTS4 : FileType := 'RTS4';
        RT2 : FileType := 'RT2';
        RTS4 : FileType := 'RT4';
      end; (case)
  end; (case)
  NoShells := 3; (Number of shells for selectivity measure)
  SelectivityBase := +1; { 0=use centre, -ve use min, +ve use max, coords }
  CMXDocPath := 'e:\cmxdocs';
  CMXDocName := 'Ref1'; (CMX reference document with marked points)
  DataDocName := 'rts6_67.mat';
  (Find the document)
  StrPCopy(CMXDocPathP, CMXDocPath);
  fileNo := 0;
  LStrCat(LStrCat(@Name, CMXDocPathP), '\doc?????.cmx');
  FindFirst(@Name, faReadOnly, searchInfo);
  error := BPErrors(DosError);
  if Error <> NoErr then begin
    writeln('Error finding CMX documents, path incorrect', Name);
    goto ExitPoint;
  end;
  FileFound := False;
  while (Error = NoErr) and not(FileFound) do begin
    LStrCat(LStrCat(StrPCopy(@Name, CMXDocPathP), '/'), @searchInfo.name);
    Error := FileOpen(TFileRec(inFile), @name, Version, Nil, False);
    if Error <> NoErr then begin
      writeln('Error opening input file ', name);
      goto ExitPoint;
    end;
    Error := ReadDocID(TFileRec(inFile), docID);
    if Error <> NoErr then begin
      writeln('Error reading input file info header ', Error);
      FileClose(TFileRec(inFile), Error);
      goto ExitPoint;
    end;
    StrPCopy(CMXDocNameP, CMXDocName);
    if LStrcmpi(DocID.name, CMXDocNameP) = 0 then begin
      FileFound := True;
    end
    else FindNext(searchInfo);
    Error := BPErrors(DosError);
    FileClose(TFileRec(inFile), Error);
  end; (while)
  if Error <> NoErr then begin
    writeln('Error finding matching CMX file ', CMXDocName);
    goto ExitPoint;
  end;
  Error := FileOpen(TFileRec(inFile), @name, Version, Nil, False);
  if Error <> NoErr then begin
    writeln('Error opening input file', name);
    goto ExitPoint;
  end;
  Error :=
    FileReadData(TFileRec(inFile), DocInfoDataID, @DocumentInfo, SizeOf(DocumentInfo), nil, 0);

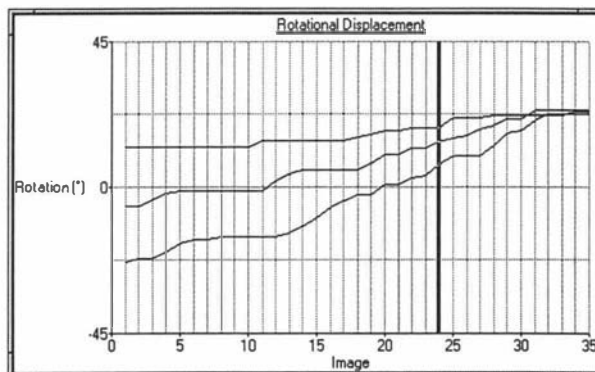
```



```

if Error <> NoErr then begin
  writeln('Error reading input file info header ', Error);
  FileClose(TFileRec(inFile), Error);
  goto ExitPoint;
end;
Error := CreatePointInfo(PointInfoH, DocumentInfo.imageCount);
if Error <> NoErr then begin
  writeln('Error creating PointsInfo');
  FileClose(TFileRec(inFile), Error);
  goto ExitPoint;
end;
Error := ReadPointInfo(PointInfoH, TFileRec(inFile), PointInfoDataID);
if Error <> NoErr then begin
  writeln('Error reading Point information = ', Error);
  FileClose(TFileRec(inFile), Error);
  goto ExitPoint;
end;
ImageSize.x := DocumentInfo.imageSize.x;
rowByteCount := Align32(ImageSize.x); { zero stuffed to longint alignment }
ImageSize.y := DocumentInfo.imageSize.y;
writeln('Image Size: ', ImageSize.x, 'x', ImageSize.y);
Error := CreateImage(Image2H, rowByteCount, ImageSize.y);
if Error <> NoErr then begin
  writeln('OutOfMem');
  goto ExitPoint;
end;
Image2 := GlobalLock(Image2H);
{read the middle image}
Error := ReadImage(Image2H, InFile, ImageDataID + (DocumentInfo.ImageCount div 2));
if Error <> NoErr then begin
  writeln('Error reading centre image');
  goto ExitPoint;
end;
Error := DecompressImage(Image2H, Image1H); {Image1H created by decompress}
if Error <> NoErr then begin
  writeln('Error decompressing image');
  goto ExitPoint;
end;
GlobalUnlock(Image2H);
FileClose(TFileRec(inFile), Error);
if Error <> NoErr then begin
  Writeln('Error closing input file');
  goto ExitPoint;
end;
{$I-}
Error := CreateMatrix(MatchMapH, 2*SearchRadius+1, 2*SearchRadius+1, DoublePrecision);
if Error <> NoErr then begin
  Writeln('Error creating Matrix');
  goto ExitPoint;
end;
MatchMap := GlobalLock(MatchMapH);
Error := CreateMatrix(SelectivityH, 2, 1, DoublePrecision);
if Error <> NoErr then begin
  Writeln('Error Selectivity Matrix');
  goto ExitPoint;
end;
Selectivity := GlobalLock(SelectivityH);
MatchMarkedPoints; {Compute feature-strength maps and selectivity about marked points}
MatchAllPoints; {Compute selectvity of all points in the image}
str(WindowRadius, S);
FileType := 'RTS6_' + S;
SelectivityBase := +1;
CalcSelRadEffects; {Compute the effects of number of shells on Selectivity measure}
CalcSelWindowEffects; {Compute selectivity for various window sizes }
{Compute selectivity for various window sizes for previous, current and next images}
CalcSelWindowEffectsBetweenImages;
{Compute feature-strength maps and selectivity about marked points between frames}
MatchPointsBetweenImages(Backward);
Exitpoint:
  if Error <> NoErr then readln;
  DestroyPointInfo(PointInfoH);

```



```

DestroyImage(Image1H);
DestroyImage(Image2H);
DestroyImage(Image3H);
DestroyImage(Image4H);
DestroyMatrix(MatchMapH);
DoneWinCrt;
(=====)
end. (Matching)

```

C.6 Image Rotation

The following Pascal unit 'Rotate' implements an image rotation function based on the three-pass shear scanline algorithm described in section 6.4.2 of chapter 6. This function was used to generate the results of column one of table 6.1.

```

unit Rotate;
{Routines to produce rotated antialiased images using the three-pass shear method.}
interface
(=====INTERFACE=====)
uses
  WinProcs, WinTypes,
  PMXTypes, PMXProcs, PMXUtils, Globals;

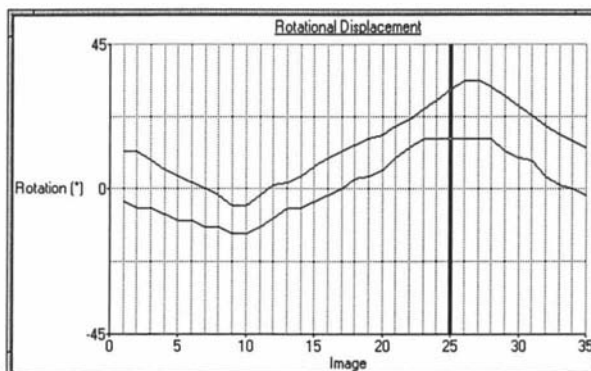
  Procedure SkewLine(
    Source : PByte;      {Source image data}
    SrcLen, : integer;    {Source image scaline length}
    DestLen : integer;    {Destination image scaline length}
    Start : double;       {Real starting position of first pixel}
    Offset : integer;     {Offset to next scanline}
    Dest : PByte;         {Destination image data}
  { Skew scanline in Source (length SrcLen) into Dest (length DestLen) starting at position
    Start. Offset between scanlines = 1 for rows or images width for columns.
  }

  Function RotateImage(
    InImageH : ImageHandle;
    Theta : double;
    var OutImageH : ImageHandle) : ErrorCode;
  { Rotates image InImageH about its centre by angle Theta (in radians) to produce
    OutImageH. The angle is assumed to be in the range of +/- Pi. NOTE: Does not work
    for negative angles, need to correct offset to skew line when angle -ve.
  }

implementation
(=====IMPLEMENTATION=====)

(----- SkewLine -----)
  Procedure SkewLine(
    Source : PByte;      {Source image data}
    SrcLen, : integer;    {Source image scaline length}
    DestLen : integer;    {Destination image scaline length}
    Start : double;       {Real starting position of first pixel}
    Offset : integer;     {Offset to next scanline}
    Dest : PByte;         {Destination image data}
  { Skew scanline in Source (length SrcLen) into Dest (length DestLen) starting at position
    Start. Offset between scanlines = 1 for rows or images width for columns.
  }

```

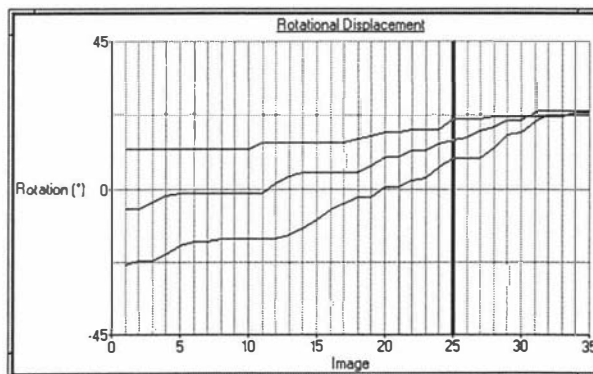
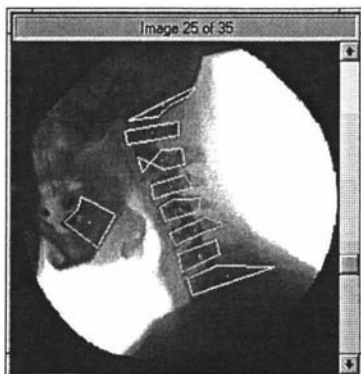


```

var
  i,
  IntStart,
  Limit      : integer;
  F, G, W1, W2 : double;

begin (Procedure: SkewLine)
  (-Process left end of output: either prepare for clipping or add padding)
  IntStart := Trunc(Start);      (Integer index)
  if IntStart < 0 then           (Negative skew)
    (Advance input pointer for clipping)
    Source := OffsetPointer(Source, -IntStart * longint(offset));
  Limit := MinL(SrcLen + IntStart, DestLen);  (Find index for end edge (valid range))
  i := 0;
  while i < IntStart do begin   (Visit all null output pixels at start edge and pad with 0)
    Dest^ := 0;
    Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
    inc(i); (Increment index)
  end; (while)
  F := Abs(Start-IntStart);      (Weight for end straddle)
  G := 1.0 - F;                 (Weight for start straddle)
  if F = 0.0 then begin         (Simple integer shift, no interpolation)
    while i < Limit do begin    (Visit all pixels in valid range)
      Dest^ := Source^;        (Copy input to output)
      Source := OffsetPointer(Source, Offset); (Advance input pointer)
      Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
      inc(i); (Increment index)
    end; (while)
  end (if F=0.0)
  else begin (Fractional shift: interpolate)
    if Start > 0.0 then begin
      W1 := F; (Weight for start pixel)
      W2 := G; (Weight for end pixel)
      Dest^ := Round(G * Source^); (First pixel special case)
      Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
      inc(i); (Increment index)
    end
    else begin (Start <= 0.0 -> negative skew)
      W1 := G; (Weight for start pixel)
      W2 := F; (Weight for end pixel)
      if Limit < DestLen then dec(Limit);
    end;
    while i < Limit do begin   (Visit all pixels in valid range)
      (Linear interpolation)
      Dest^ := Round(W1*Source^ + W2*PByte(OffsetPointer(Source, Offset))^);
      Source := OffsetPointer(Source, Offset); (Advance input pointer)
      Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
      inc(i); (Increment index)
    end; (while)
    if i < DestLen then begin
      Dest^ := Round(W1 * Source^); (Last pixel special case)
      Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
      inc(i); (Increment index)
    end;
  end; (else Fractional shift...)
  while i < DestLen do begin   (Visit all remaining pixels at end edge and pad with 0)
    Dest^ := 0;
    Dest := OffsetPointer(Dest, Offset); (Advance output pointer)
    inc(i); (Increment index)
  end; (while)
end; (Procedure: SkewLine)
(----- RotateImage -----)
Function RotateImage(
  InImageH : ImageHandle;
  Theta    : double;
  var OutImageH : ImageHandle) : ErrorCode;
( Rotates image InImageH about its centre by angle Theta (in radians) to produce
  OutImageH. The angle is assumed to be in the range of +/- Pi. )
label
  ExitPoint;

```



```

var
  Error      : ErrorCode;
  x, y       : longint;
             {Row and column counters }
  InImageWidth, InImageHeight,
  TmpImageWidth, {Temporary image width = final output image width }
  OutImageHeight, OutImageWidth
             : word;
  TmpImageH   : ImageHandle;
             {Temporary image for intermediate results}
  InImage,
  TmpImage,
  OutImage    : ImagePtr;
             {Pointers to Input, Temporary, and Output images}

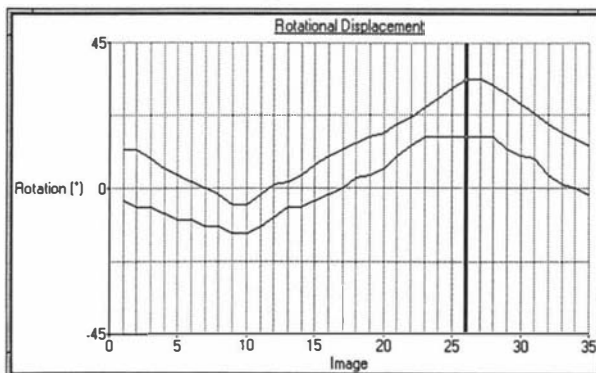
  SrcImagePtr,
  DestImagePtr : PByte;
             {Pointers to Source and Destination image pixels }

  Cosine,
  Sine,
  Tangent,
  Offset     : double;
             {Cosine of rotation angle}
             {Sine of rotation angle}
             {Tangent of rotation half the angle}
             {Offset from top of image}

{ Data flow: InImage -> OutImage -> TmpImage -> OutImage,
+1 added to dimensions due to last fractional pixel. }
begin {Function: RotateImage}
  if abs(Theta) > PI then begin
    Error := 2;
    goto ExitPoint;
  end;
  InImage := GlobalLock(InImageH);
  InImageWidth := InImage^.header.size.x;
  InImageHeight := InImage^.header.size.y;
  Sine := Sin(Theta);
  Cosine := Cos(Theta);
  Tangent := Sin(Theta/2) / Cos(Theta/2);
  TmpImageWidth := InImageWidth + abs(Trunc(InImageHeight * Tangent)) + 1;
  OutImageHeight := Trunc(InImageWidth * abs(Sine) + InImageHeight * Cosine) + 1;
  OutImageWidth := Trunc(InImageHeight * abs(Sine) + InImageWidth * Cosine) + 1;
  Error := CreateImage(TmpImageH, TmpImageWidth, OutImageHeight);
  if Error <> NoErr then goto ExitPoint;
  TmpImage := GlobalLock(TmpImageH);
  Error := CreateImage(OutImageH, OutImageWidth, OutImageHeight);
  if Error <> NoErr then goto ExitPoint;
  OutImage := GlobalLock(OutImageH);
  {-First pass: Skew x (horizontal scanlines)}
  for y := 0 to InImageHeight-1 do begin {Visit each row in InImage}
    SrcImagePtr := OffsetPointer(@InImage^.data, y * InImageWidth);
    DestImagePtr := OffsetPointer(@OutImage^.data, y * TmpImageWidth);
    SkewLine(SrcImagePtr, InImageWidth, TmpImageWidth, y * Tangent, 1, DestImagePtr);
  end; {for y}
  {-Second pass: Skew y (vertical scanlines). Use TmpImage for intermediate storage }
  Offset := (InImageWidth-1) * Sine; {Offset from top of image}
  for x := 0 to TmpImageWidth-1 do begin {Visit each column in OutImage}
    SrcImagePtr := OffsetPointer(@OutImage^.data, x);
    DestImagePtr := OffsetPointer(@TmpImage^.data, x);
    SkewLine(SrcImagePtr, InImageHeight, OutImageHeight,
      Offset-x*Sine, TmpImageWidth, DestImagePtr);
  end; {for x}
  {-Third pass: Skew x (horizontal scanlines) }
  for y := 0 to OutImageHeight-1 do begin {Visit each row in InImage }
    SrcImagePtr := OffsetPointer(@TmpImage^.data, y * TmpImageWidth);
    DestImagePtr := OffsetPointer(@OutImage^.data, y * OutImageWidth);
    SkewLine(SrcImagePtr, TmpImageWidth, OutImageWidth,
      (y - Offset) * Tangent, 1, DestImagePtr);
  end; {for y}
  ExitPoint:
    DestroyImage(TmpImageH);
    if Error <> NoErr then DestroyImage(OutImageH);
    RotateImage := Error;
  end; {Function: RotateImage}
  (=====)
end. {Unit: Rotate}

```

Warp60



C.7 Two-pass Mesh-warp Image Functions

The following Pascal unit 'WarpFtns' contains all the functions of the two-pass mesh-warp image resampling algorithm that was used as the basis of the generation of the prescribed motion cineradiographic sequences of chapter 7.

```

unit WarpFtns;
  {Routines to perform 2-D Mesh warping of images }

  {$r warpftns.res}

INTERFACE
(=====INTERFACE=====)
uses
  WinCrt, WinTypes, WinProcs, PMXTypes, PMXProcs, PMXUtils, Utils,
  Globals, Resource, MLab_ops, Images;

const
  {Extra error types}
  MemoryErr      = 11;
  DivByZeroErr   = 12;
  OutOfRangeErr  = 13;
type
  Real = Single;
  RealLine = Array[0..MaxWord div RSize - RSize] of Real;
  PRealLine = ^RealLine;
  ByteLine = Array[0..MaxWord-WSize] of Byte;
  PByteLine = ^ByteLine;
  RealImageHandle = THandle;
  RealImageRec = record
    DataSize : Longint;
    Size      : TPoint;
    Data      : Array[0..0] of Real;
  end;
  RealImagePtr = ^RealImageRec;
  MeshHandle   = RealImageHandle;
  MeshPtr      = RealImagePtr;

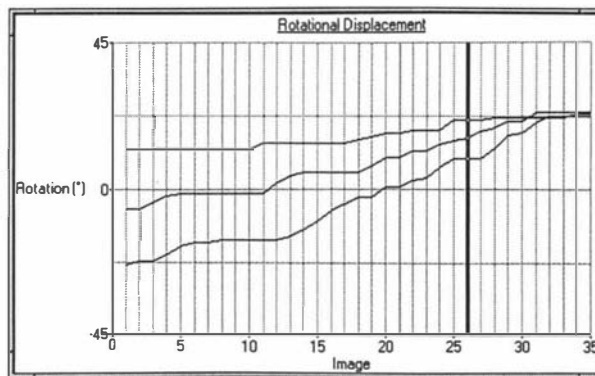
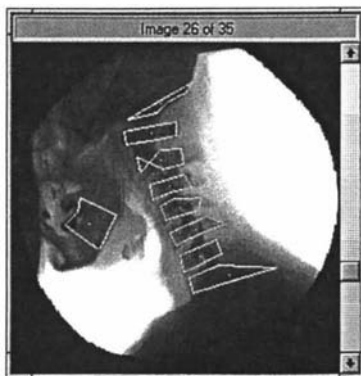
Function CreateRealImage(
  var RealImageH : RealImageHandle;
  width, height : Word) : ErrorCode;
Procedure DestroyRealImage(var RealImageH : RealImageHandle);

Function InterpolateCubic( {Generalised cubic spline interpolator.}
  XDataIn, YDataIn : PRealLine; LengthIn : word;
  XDataOut, YDataOut : PRealLine; LengthOut : word) : ErrorCode;

Function InterpolateLinear( {Generalised linear spline interpolator.}
  XDataIn, YDataIn : PRealLine; LengthIn : word;
  XDataOut, YDataOut : PRealLine; LengthOut : word) : ErrorCode;

Function MeshWarpImage(
  InImageH           {Input image handle}
                    : ImageHandle;
  var OutImageH      {Output image handle}
                    : ImageHandle;
  XMeshInH, YMeshInH, {Handle to x and y coords of input image mesh}
  XMeshOutH, YMeshOutH {Handle x and y coords of output image mesh}
                    : MeshHandle;
  RegularisedOutMesh : Boolean) : ErrorCode;
{A function to resamples InImageH to produce OutImageH based on the forward mesh maps
defined by the relationship between (XMeshIn, XMeshOut) and (YMeshIn, YMeshOut).}

```



```

Function ResampleScanline( {Generalised image ByteLine resampling.}
    ForwardMap      {Table for point samples of the forward mapping function}
    : PRealLine;
    InpLine,        {Pointer to Input scanline to be resampled}
    OutLine          {Pointer to the resulting Output scanline}
    : PByte;
    LineLength,      {Input scanline length}
    LineOffset       {Offset between scanlines}
    : Word) : ErrorCode;

Implementation
(=====IMPLEMENTATION=====)

(-----CreateRealImage-----)
Function CreateRealImage(
    var RealImageH : RealImageHandle;
    width, height : word) : ErrorCode;
{--Allocates memory for a real image "RealImageH" of size "width" x "height" of real}
{If an error occurs "RealImageH" will be 0. Returns the code of any error that occurs.}
)
var
    Error      : ErrorCode;      {Code of any error that has occurred}
    RealImage : RealImagePtr;    {Pointer to RealArray}
begin {CreateRealImage}
    Error := CreateHandle(RealImageH,
        SizeOf(RealImageRec) + (Longint(width) * height-1) * RSize, 0);
    if Error = NoErr then begin
        RealImage := GlobalLock(RealImageH);
        RealImage^.DataSize := Longint(width) * height * RSize;
        SetPoint(RealImage^.size, width, height);
        GlobalUnlock(RealImageH);
    end; {if}
    CreateRealImage := Error;
end; {CreateRealImage}

(-----DestroyRealImage-----)
Procedure DestroyRealImage(var RealImageH : RealImageHandle);
{--Destroys the "RealArrayH" and sets it to 0}
begin {DestroyRealImage}
    DestroyHandle(RealImageH);
end; {DestroyRealImage}

(-----SolveTridiagonalMatrix-----)
Function SolveTridiagonalMatrix(
    BandA, BandB, BandC, VectorD : PRealLine;
    Length : word) : ErrorCode;
{ Gauss Elimination with backsubstitution for solving a tridiagonal matrix of size
  "Length", with bands "BandA, BandB, BandC" and column vector "VectorD"
}
)
var
    i      : integer; {counter}
    Pivot  : Real;
    TempVector : PRealLine;
begin {-SolveTridiagonalMatrix}
    if MaxAvail < Length * RSize then begin
        SolveTridiagonalMatrix := MemoryErr;
        Exit;
    end;
    GetMem(TempVector, Length * RSize);
    Pivot := BandB^[0];
    VectorD^[0] := VectorD^[0] / Pivot;
    for i := 1 to Length-1 do begin {Perform Forward substitution}
        TempVector^[i] := BandC^[i-1]/Pivot;
        Pivot := BandB^[i] - BandA^[i] * TempVector^[i];
        if Pivot = 0 then begin
            SolveTridiagonalMatrix := DivByZeroErr;
            FreeMem(TempVector, Length * RSize);
            Exit;
        end; {if}
        VectorD^[i] := (VectorD^[i] - VectorD^[i-1] * BandA^[i]) / Pivot;
    end; {for i}

```

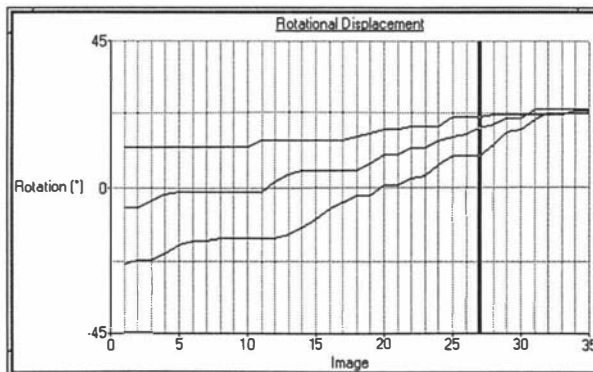
Warp60



```

for i := Length-2 downto 0 do (Perform Backsubstitution)
  VectorD[i] := VectorD[i] - VectorD[i+1] * TempVector[i+1];
  SolveTridiagonalMatrix := NoErr;
  FreeMem(TempVector, Length * RSize);
end; (-Function: SolveTridiagonalMatrix)
(----- ComputeYDerivs -----)
Function ComputeYDerivs(XData, YData : PRealLine;
  var YDerivs : PRealLine;
  Length : Word) : ErrorCode;
(-Compute the 1st derivatives "YDerivs" of data in "XData, YData" of "Length" entries.
The not-a-knot condition is used. If successful returns NoErr else MemoryErr for
insufficient memory or DivByZeroErr for divide by zero, indicating at least one
undefined derivative.
)
var
  i : integer; (Counter )
  BandA, BandB, BandC : PRealLine; (The three bands of the tridiagonal matrix)
  DeltaX0, DeltaX1, Slope0, Slope1 : Real; (Change in xdata between the current and the next segment)
  UndefinedSlope : boolean; (True if any segment slope is undefined)
begin (-Function ComputeYDerivs)
  if MaxAvail < (3 * longint(Length) * RSize) then begin
    ComputeYDerivs := MemoryErr;
    Exit;
  end;
  (-Allocate memory for tridiagonal bands A,B,C.)
  GetMem(BandA, Length * RSize);
  GetMem(BandB, Length * RSize);
  GetMem(BandC, Length * RSize);
  (-Initialise first row.)
  DeltaX0 := XData[1] - XData[0];
  DeltaX1 := XData[2] - XData[1];
  if (DeltaX0 <> 0) and (DeltaX1 <> 0) then begin (First slope defined)
    UndefinedSlope := False;
    Slope0 := (YData[1] - YData[0]) / DeltaX0;
    Slope1 := (YData[2] - YData[1]) / DeltaX1;
    BandA[0] := 0; (Not strictly necessary)
    BandB[0] := DeltaX0 * (DeltaX0 + DeltaX1);
    BandC[0] := Sqr(DeltaX0 + DeltaX1);
    YDerivs[0] := Slope0 * (3*DeltaX0*DeltaX1 + 2*Sqr(DeltaX1)) + Slope1*Sqr(DeltaX0);
    (-Initialise tridiagonal bands A,B,C and column vector YDerivs)
    i := 1;
    repeat
      DeltaX0 := XData[i] - XData[i-1];
      DeltaX1 := XData[i+1] - XData[i];
      if (DeltaX0 <> 0) and (DeltaX1 <> 0) then begin (Slope defined)
        Slope0 := (YData[i] - YData[i-1]) / DeltaX0;
        Slope1 := (YData[i+1] - YData[i]) / DeltaX1;
        BandA[i] := DeltaX1;
        BandB[i] := 2 * (DeltaX0 + DeltaX1);
        BandC[i] := DeltaX0;
        YDerivs[i] := 3 * (Slope0*DeltaX1 + Slope1*DeltaX0);
      end (if the slope defined)
      else UndefinedSlope := True;
      inc(i);
    until (i = Length-1) or (UndefinedSlope);
    (-Initialise last row)
    if not UndefinedSlope then begin
      BandA[Length-1] := Sqr(DeltaX0 + DeltaX1);
      BandB[Length-1] := DeltaX0 * (DeltaX0 + DeltaX1);
      BandC[Length-1] := 0; (Not strictly necessary)
      YDerivs[Length-1] :=
        Slope0 * Sqr(DeltaX1) + Slope1 * (3*DeltaX0 * DeltaX1 + 2*Sqr(DeltaX0));
      (-Solve for the tridiagonal matrix: YDerivs = YDerivs * inverse(Tridiag matrix).)
      ComputeYDerivs := SolveTridiagonalMatrix(BandA, BandB, BandC, YDerivs, Length);
    end
    else ComputeYDerivs := DivByZeroErr;
  end (if SlopeDefined)

```

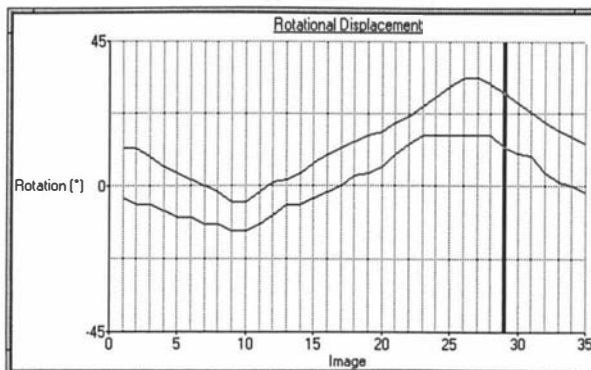



```

else ComputeYDerivs := DivByZeroErr; { ie. atleast one of the derivatives is undefined}
  (-Deallocate tridiagonal bands)
  FreeMem(BandA, Length * RSize);
  FreeMem(BandB, Length * RSize);
  FreeMem(BandC, Length * RSize);
end; {Function: ComputeYDerivs}
(----- InterpolateCubic -----)
Function InterpolateCubic(
  XDataIn, YDataIn : PRealLine; LengthIn : word;
  XDataOut, YDataOut : PRealLine; LengthOut : word) : ErrorCode;
{ Generalised Interpolating cubic spline function for irregularly-spaced points
  INPUT:  "YDataIn" is a list of irregular data points ("LengthIn" entries)
          Their x-coordinates are specified in "XDataIn"
  OUTPUT: "YDataOut" a cubic spline sampled according to "XDataOut" ("LengthOut" entries)
  Assumes that "XDataIn", "XDataOut" entries are monotonically increasing.
  If successful returns NoErr, else MemoryErr for insufficient memory,
  DivByZeroErr for divide by zero: indicating entries Not monotonically
  increasing, OutOfRangeErr if XDataOut does not lie fully in the range of
  XDataIn.
}
var
  i,
  InDataIndex      (Index for input data in current interval)
                  : integer;
  YDerivs          (YData 1st derivatives)
                  : PRealLine;
  LeftEndPoint,    (Left end point of the interval)
  SamplePoint,     (Sample point in the interval)
  RightEndPoint,   (Right end point of the interval)
  Coeff0, Coeff1,  (Constant and linear coefficients of the spline)
  Coeff2, Coeff3,  (Quadratic and cubic coefficients of the spline)
  DeltaX, DeltaY,  (Current interval differences for X and Y)
  LeftToSample     (Difference in current interval from LeftEndPoint to SamplePoint)
                  : Real;
  Error            : ErrorCode;

begin {Function: InterpolateCubic}
  if MaxAvail < (LengthIn * RSize) then begin
    InterpolateCubic := MemoryErr;
    Exit;
  end;
  (-Allocate memory for 1st Y Derivatives)
  GetMem(YDerivs, LengthIn * RSize);
  (Compute 1st derivatives of each point)
  Error := ComputeYDerivs(XDataIn, YDataIn, YDerivs, LengthIn);
  if Error = NoErr then begin {All derivatives defined}
    (Error check data ranges)
    if (XDataOut^[0] < XDataIn^[0]) or (XDataOut^[LengthOut-1] > XDataIn^[LengthIn-1])
    then begin
      InterpolateCubic := OutOfRangeErr;
      FreeMem(YDerivs, LengthIn * RSize);
      Exit;
    end;
    (Force coefficient initialisation)
    RightEndPoint := XDataOut^[0] - 1;
    InDataIndex := 0;
    for i := 0 to LengthOut-1 do begin
      (Check if in new interval)
      SamplePoint := PReal(OffsetPointer(XDataOut, i * RSize))^;
      if (SamplePoint > RightEndPoint) then begin
        (Find the interval that contains the SamplePoint)
        while (InDataIndex < LengthIn-2)
          and (SamplePoint > XDataIn^[InDataIndex]) do inc(InDataIndex);
        if SamplePoint < XDataIn^[InDataIndex] then dec(InDataIndex);
        LeftEndPoint := XDataIn^[InDataIndex]; {Update left end point}
        (Compute Spline coefficients)
        DeltaX := 1.0 / (XDataIn^[InDataIndex+1] - XDataIn^[InDataIndex]);
        DeltaY := DeltaX * (YDataIn^[InDataIndex+1] - YDataIn^[InDataIndex]);
        Coeff0 := YDataIn^[InDataIndex];
        Coeff1 := YDerivs^[InDataIndex];

```

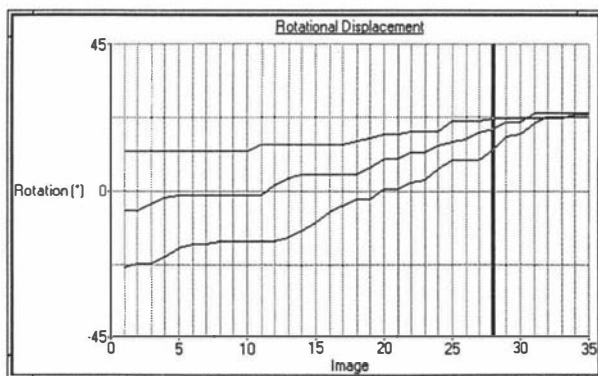
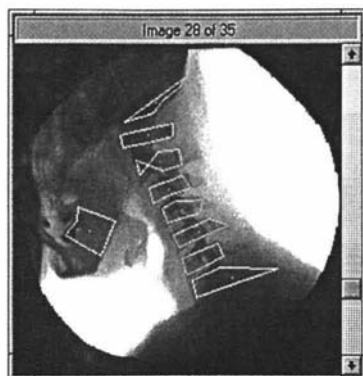


```

    Coeff2 := DeltaX
            * (3.0 * DeltaY - 2.0 * YDerivs^[InDataIndex] - YDerivs^[InDataIndex+1]);
    Coeff3 := Sqr(DeltaX)
            * (-2.0 * DeltaY + YDerivs^[InDataIndex] + YDerivs^[InDataIndex+1]);
    end; {if SamplePoint > ...}
    {-Use Horner's rule to calculate the cubic polynomial}
    LeftToSample := SamplePoint - LeftEndPoint;
    YDataOut^[i]
        := ((Coeff3*LeftToSample + Coeff2) * LeftToSample + Coeff1)*LeftToSample + Coeff0;
    end; {for i}
end; {if Error = NoErr}
FreeMem(YDerivs, LengthIn * RSize);
InterpolateCubic := Error;
end; {Function: InterpolateCubic}
(----- InterpolateLinear -----)
Function InterpolateLinear(
    XDataIn, YDataIn : PRealLine; LengthIn : word;
    XDataOut, YDataOut : PRealLine; LengthOut : word) : ErrorCode;
{ Generalised Interpolating linear spline function for irregularly-spaced points
INPUT: "YDataIn" is a list of irregular data points ("LengthIn" entries)
      Their x-coordinates are specified in "XDataIn"
OUTPUT: "YDataOut" a linear spline sampled according to "XDataOut" (LengthOut entries)
      Assumes that "XDataIn", "XDataOut" entries are monotonically increasing.
      If successful returns NoErr, else DivByZeroErr for divide by zero indicating
      entries not monotonically increasing or OutOfRangeErr if "XDataOut" does not
      lie fully in the range of "XDataIn".
}
var
    i,
    InDataIndex      (Index for input data in current interval)
                    : integer;
    LeftEndPoint,    (Left end point of the interval)
    SamplePoint,     (Sample point in the interval)
    RightEndPoint,   (Right end point of the interval)
    Coeff0, Coeff1,  (Constant and linear coefficients of interpolant)
    LeftToSample     (Difference in current interval from LeftEndPoint to SamplePoint)
                    : Real;
    Error            : ErrorCode;

begin {Function: InterpolateLinear}
    Error := NoErr;
    if (XDataOut^[0] < XDataIn^[0]) or (XDataOut^[LengthOut - 1] > XDataIn^[LengthIn - 1])
    then begin
        InterpolateLinear := OutOfRangeErr;
        Exit;
        end;
    {Force coefficient initialisation}
    RightEndPoint := XDataOut^[0] - 1;
    InDataIndex := 0;
    for i := 0 to LengthOut-1 do begin
        {Check if in new interval}
        SamplePoint := XDataOut^[i];
        if (SamplePoint > RightEndPoint) then begin
            {Find the interval that contains the SamplePoint}
            while (InDataIndex < LengthIn-2)
                and (SamplePoint > XDataIn^[InDataIndex]) do inc(InDataIndex);
            if SamplePoint < XDataIn^[InDataIndex] then dec(InDataIndex);
            LeftEndPoint := XDataIn^[InDataIndex]; {Update left point}
            {Compute linear coefficients}
            Coeff0 := YDataIn^[InDataIndex];
            Coeff1 := (YDataIn^[InDataIndex+1] - Coeff0)
                    / ((XDataIn^[InDataIndex+1] - LeftEndPoint));
            end; {if SamplePoint > ...}
            LeftToSample := SamplePoint - LeftEndPoint;
            YDataOut^[i] := Coeff0 + LeftToSample * Coeff1;
        end; {for i}
        InterpolateLinear := Error;
    end; {Function: InterpolateLinear}

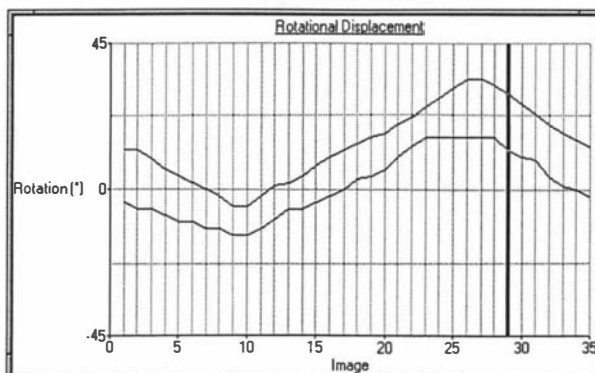
```



```

(----- ResampleScanline-----)
Function ResampleScanline(
    ForwardMap      (Table for point samples of the forward mapping function)
    : PRealLine;
    InpLine,        (Pointer to Input scanline to be resampled)
    OutLine         (Pointer to the resulting Output scanline)
    : PByte;
    LineLength,     (Input ByteLine length)
    LineOffset      (Offset between scanlines)
    : Word) : ErrorCode;
{ Resamples an Input ByteLine "InLine" based on the spacially varying "ForwardMap" table
  to produce an anti-aliased output scan line "OutLine" of the same length, "LineLength".
  The LineOffset = 1 for horizontal scanlines and LineLength vertical scanlines.
  Implementation based on Fant's resampling algorithm, Box filtering for minification,
  linear interpolation for maxification. If successful returns NoErr else MemoryErr.
}
var
    OutIndex,      (Output line pixel counter)
    InIndex        (Input line pixel counter)
    : Longint;
    Accumulator,   (Accumulated weighted contribution)
    Value,         (Calculated new pixel value)
    InverseScaleFactor,
    InSegment,     (Proportion of input pixel available)
    OutSegment     (Proportion of output pixel available)
    : Real;
    InPosition     (Input line index for each output pixel)
    : PRealLine;
begin (-Function: ResampleScanline)
    if MaxAvail < (LineLength * RSize) then begin
        ResampleScanline := MemoryErr;
        Exit;
    end;
    GetMem(InPosition, longint(LineLength) * RSize);
    {Precompute input index positions for each output pixel }
    InIndex := 0;
    for OutIndex := 0 to LineLength-1 do begin
        while ForwardMap^[InIndex+1] < OutIndex do Inc(InIndex);
        InPosition^[OutIndex] := InIndex
            + (OutIndex - ForwardMap^[InIndex]) / (ForwardMap^[InIndex+1] - ForwardMap^[InIndex]);
    end; {for OutIndex}
    {-Initialise}
    InSegment := 1.0;           {Entire input pixel is available}
    OutSegment := InPosition^[1]; {Number of input pixel that maps onto output pixel 1}
    InverseScaleFactor := OutSegment;
    Accumulator := 0.0;        {Clear accumulator}
    {-Compute all output pixels}
    InIndex := 0;
    OutIndex := 0;
    while (OutIndex < LineLength) do begin
        {Use linear interpolation for reconstruction}
        if InIndex < LineLength - 1 then
            Value := InSegment * PByte(OffsetPointer(InpLine, InIndex * LineOffset))^
                + (1-InSegment) * PByte(OffsetPointer(InpLine, (InIndex+1) * LineOffset))^;
        if (InSegment < OutSegment) then begin {Input pixel entirely consumed before output}
            Accumulator := Accumulator + Value * InSegment; {Accumulate weighted contribution}
            OutSegment := OutSegment - InSegment; {Input segment portion has been filled}
            InSegment := 1.0; {New input pixel will be available}
            inc(InIndex); {Index into next input pixel}
        end (if)
        else begin {-Input pixel NOT entirely consumed before output pixel}
            Accumulator := Accumulator + Value * OutSegment; {Accumulate weighted contribution}
            PByte(OffsetPointer(OutLine, OutIndex * LineOffset))^
                := Trunc(Accumulator / InverseScaleFactor);
            {-Initialise output with normalised accumulator}
            Accumulator := 0.0; {Clear accumulator for next output pixel}
            InSegment := InSegment - OutSegment; {Output segment portion of input been filled}
            inc(OutIndex); {Index into next output pixel}
        end (if)
    end;
end;

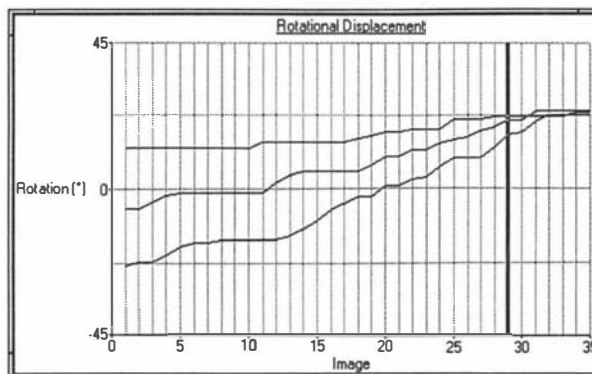
```



```

    if OutIndex < LineLength-1 then
        InverseScaleFactor := InPosition^[OutIndex+1] - InPosition^[OutIndex];
        OutSegment := InverseScaleFactor;          {Initialise spacially varying size factor}
    end; {else}
end; {while OutIndex <}
FreeMem(InPosition, LineLength * RSize);
ResampleScanline := NoErr;
end; {-Function: ResampleScanline}
(----- MeshWarpImage -----)
Function MeshWarpImage(
    InImageH                      {Input image handle}
    : ImageHandle;
    var OutImageH                 {Output image handle}
    : ImageHandle;
    XMeshInH, YMeshInH,          {Handle to x and y coords of input image mesh}
    XMeshOutH, YMeshOutH         {Handle x and y coords of output image mesh}
    : MeshHandle;
    RegularisedOutMesh : Boolean) : ErrorCode;
{-Mesh Warps the InImage given its X and Y Mesh coords to produce the OutImage given the
new output X and Y mesh coords. Warp performed in two passes and is based on Smythe's
algorithm. If XMeshInH or YMeshInH are zero, input mesh assumed to be a regular one
pixel grid the size of size of InImage. If RegularisedOutMesh is true it is assumed
that the output mesh grids represent the resampling maps directly and thus ImageH is
resampled based on them.
}
label
    MeshWarpImageExit;
var
    BufferSize,                    {Size of interpolated line and index buffers}
    x, y,                        {General x and y coord counters}
    Time,                        {Execution time store}
    address                      {mesh/image element address counter}
    : Longint;
    imageWidth, imageHeight,     {Size of input and output images}
    meshHeight, meshWidth,      {Height and width of mesh tables}
    rowByteCount                {Number of byte in a mesh row}
    : Word;
    InImage, OutImage,          {Input and output images}
    InterImage                  {Intermediate image}
    : ImagePtr;
    InImageData, OutImageData,  {Pointers to Input, Output and}
    InterImageData              {Intermediate image data.}
    : PByte;
    InterImageH                 {Handle to intermediate image}
    : ImageHandle;
    XMeshIn, YMeshIn,           {x and y coords of input mesh}
    XMeshOut, YMeshOut          {x and y coords of output mesh}
    : MeshPtr;
    XMeshInData, YMeshInData,   {Pointers to X and Y input mesh values}
    XMeshOutData, YMeshOutData  {Pointers to X and Y output mesh values}
    : PReal;
    Source, Dest                {Source and destination pointers for ByteLines}
    : PByte;
    XRow1, YRow1,               {1st row and column buffer}
    XRow2, YRow2,               {2nd row and column buffer}
    Map1, Map2,                 {Mapping function from sampled splines}
    Index                       {Stores indices for sampled splines}
    : PRealLine;
    XRow1H, YRow1H,             {Handles for 1st row and column buffer}
    XRow2H, YRow2H,             {Handles for 2nd row and column buffer}
    Map1H, Map2H,               {Handles for mapping function from sampled splines}
    IndexH                      {Handle for Index}
    : THandle;
    InTableH,                   {Handles to input, intermediate and output image tables}
    InterTableH,
    OutTableH
    : RealImageHandle;
    InTable,                    {Input image table}
    InterTable,                 {Intermediate image table}
    OutTable                    {Output image table}
    : RealImagePtr;

```



Warp30

```

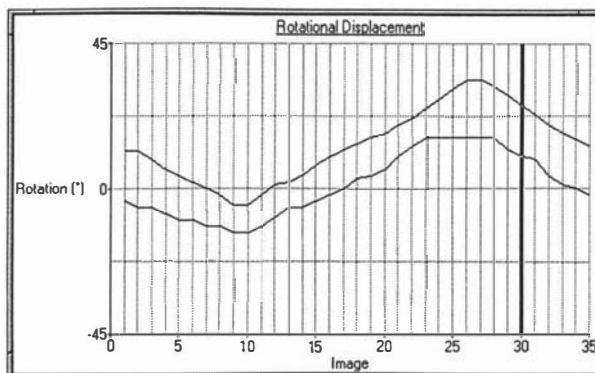
InTableData,           (Pointers to the data in each table)
InterTableData,
OutTableData

      : PReal;

Error      : General error)
      : ErrorCode;
InMeshSpecified      (True if an input mesh has been specified, False assumes an
                      input mesh of 1-pixel regular spacing.)
      : Boolean;
begin (Function: MeshWarpImage )
  (Check if an input mesh has been specified )
  if (XMeshInH = 0) or (YMeshInH = 0) then InMeshSpecified := False
  else InMeshSpecified := True;
  (Compute buffer sizes and allocate them in memory )
  InImage := GlobalLock(InImageH);
  imageWidth := InImage^.header.size.x;
  imageHeight := InImage^.header.size.y;
  InImageData := @InImage^.data;
  (Initialise handles)
  InterImageH := 0; OutImageH := 0;
  InTableH := 0; InterTableH := 0; OutTableH := 0;
  XRow1H := 0; YRow1H := 0; XRow2H := 0; YRow2H := 0;
  IndexH := 0; Map1H := 0; Map2H := 0;
  (Create output image)
  Error := CreateImage(OutImageH, imageWidth, imageHeight);
  if Error <> NoErr then goto MeshWarpImageExit;
  OutImage := GlobalLock(OutImageH);
  OutImageData := @OutImage^.data;
  (Create intermediate image)
  Error := CreateImage(InterImageH, imageWidth, imageHeight);
  if Error <> NoErr then goto MeshWarpImageExit;
  InterImage := GlobalLock(InterImageH);
  InterImageData := @InterImage^.data;
  (Point to meshes)
  XMeshIn := GlobalLock(XMeshInH); YMeshIn := GlobalLock(YMeshInH);
  XMeshOut := GlobalLock(XMeshOutH); YMeshOut := GlobalLock(YMeshOutH);
  meshWidth := XMeshOut^.size.x;
  meshHeight := XMeshOut^.size.y;
  if imageWidth > imageHeight then BufferSize := imageWidth else BufferSize := imageHeight;
  BufferSize := (BufferSize + 1) * RSize;
  if RegularisedOutMesh then begin
    Error := CreateHandle(Map1H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    Map1 := GlobalLock(Map1H);
  end (if)
  else begin (Not RegularisedOutMesh)
    (Create input table)
    Error := CreateRealImage(InTableH, meshWidth, imageHeight);
    if Error <> NoErr then goto MeshWarpImageExit;
    InTable := GlobalLock(InTableH);
    InTableData := @InTable^.data; (Get start address of table data)
    (Create index, row and map buffers)
    Error := CreateHandle(IndexH, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    Index := GlobalLock(IndexH);
    Error := CreateHandle(xRow1H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    xRow1 := GlobalLock(xRow1H);
    Error := CreateHandle(yRow1H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    yRow1 := GlobalLock(yRow1H);
    Error := CreateHandle(xRow2H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    xRow2 := GlobalLock(xRow2H);
    Error := CreateHandle(yRow2H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    yRow2 := GlobalLock(yRow2H);
    Error := CreateHandle(Map1H, BufferSize, 0);
    if Error <> NoErr then goto MeshWarpImageExit;
    Map1 := GlobalLock(Map1H);
    Error := CreateHandle(Map2H, BufferSize, 0);
  end
end

```

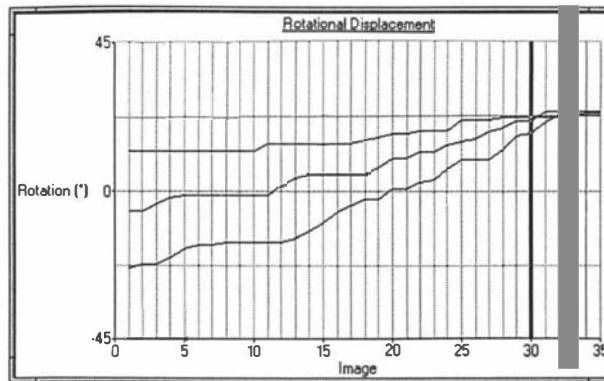
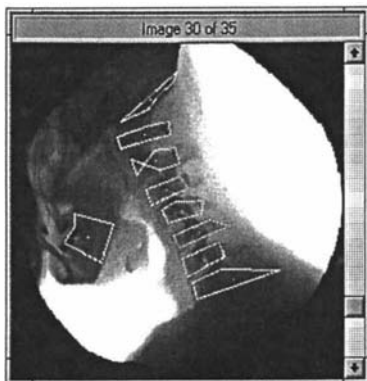
Warp60



```

    if Error <> NoErr then goto MeshWarpImageExit;
    Map2 := GlobalLock(Map2H);
end; (else not RegularisedOutMesh)
(-FIRST PASS (Phase1): Create input and intermediate image tables for )
( x-intercepts of vertical splines. )
Time := GetTickCount;
if InMeshSpecified then begin      (Point to start of input mesh data)
    XMeshInData := @XMeshIn^.data;
    YMeshInData := @YMeshIn^.data;
end;
XMeshOutData := @XMeshOut^.data;   (Point to start of output mesh data)
YMeshOutData := @YMeshOut^.data;
if not(RegularisedOutMesh) then begin
    writeln(' First Pass: Horizontal warp ');
    for y := 0 to imageHeight-1 do Index^[y] := y; (Indices to sample vertical splines)
    if InMeshSpecified then begin
        (Create intermediate table)
        Error := CreateRealImage(InterTableH, meshWidth, imageHeight);
        if Error <> NoErr then goto MeshWarpImageExit;
        InterTable := GlobalLock(InterTableH);
        InterTableData := @InterTable^.data; (Get start address of table data)
        rowByteCount := meshWidth * RSize;
        x := 0;
        repeat (Visit each vertical spline)
            address := x * RSize;
            for y := 0 to meshHeight-1 do begin
                xRow1^[y] := PReal(OffsetPointer(XMeshInData, address))^;
                yRow1^[y] := PReal(OffsetPointer(YMeshInData, address))^;
                xRow2^[y] := PReal(OffsetPointer(XMeshOutData, address))^;
                Inc(address, rowByteCount);
            end; (for y)
            (Scan convert vertical splines of input and intermediate image)
            Error := InterpolateLinear(yRow1, xRow1, meshHeight, Index, Map1, imageHeight);
            if (Error = NoErr) and
                (InterpolateLinear(yRow1, xRow2, meshHeight, Index, Map2, imageHeight) = NoErr)
            then begin
                address := x * RSize;
                for y := 0 to imageHeight-1 do begin (Store resampled rows back in Table cols)
                    PReal(OffsetPointer(InTableData, address))^ := Map1^[y];
                    PReal(OffsetPointer(InterTableData, address))^ := Map2^[y];
                    Inc(address, rowByteCount);
                end; (for y)
            end; (if NoErr)
            Inc(x);
        until (x >= meshWidth) or (Error <> NoErr);
    end (if InMeshSpecified)
else begin (InMesh NOT specified, hence assumed regular 1-pixel)
    rowByteCount := meshWidth * RSize;
    for x := 0 to meshWidth-1 do begin (Visit each vertical spline of the OutMeshes)
        address := x * RSize; (Element address)
        for y := 0 to imageHeight-1 do begin (Store resampled cols back into Table cols)
            PReal(OffsetPointer(InTableData, address))^ := x; (Regular grid)
            Inc(address, rowByteCount);
        end; (for y)
    end; (for x)
    InterTableData := XMeshOutData; (Point intermediate table at xMeshOut)
end; (else no InMeshSpecified, hence a regular grid used)
(-FIRST PASS (Phase2): Warp in X direction using input and intermediate
( tables to produce the intermediate image )
if Error = NoErr then begin
    for x := 0 to imageWidth-1 do Index^[x] := x; (Indices to sample horizontal splines)
    for y := 0 to imageHeight-1 do begin (Visit each row)
        (Fit a spline to x-intercepts and resample over all columns)
        address := y * meshWidth * RSize;
        for x := 0 to meshWidth-1 do begin
            xRow1^[x] := PReal(OffsetPointer(InTableData, address))^;
            xRow2^[x] := PReal(OffsetPointer(InterTableData, address))^;
            Inc(address, RSize);
        end; (for x)
        Error := InterpolateLinear(xRow1, xRow2, meshWidth, Index, Map1, imageWidth);
        if Error <> NoErr then goto MeshWarpImageExit;
    end; (for y)
end;

```

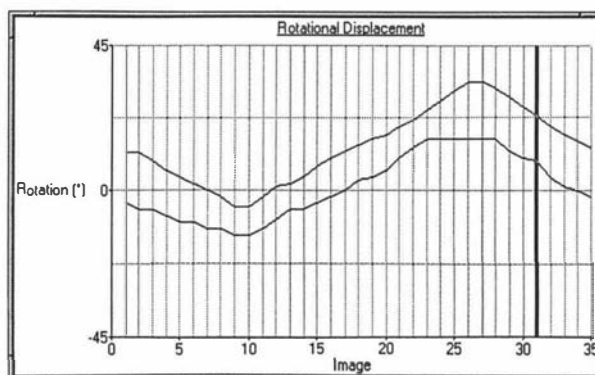


```

    (Resample input image row based on map1 into intermediate image)
    Source := PByte(OffsetPointer(InImageData, y * imageWidth));
    Dest := PByte(OffsetPointer(InterImageData, y * imageWidth));
    Error := ResampleScanline(Map1, Source, Dest, imageWidth, 1);
    if Error <> NoErr then goto MeshWarpImageExit;
end; (for y)
end (if NoErr)
else goto MeshWarpImageExit;
DestroyRealImage(InTableH);
DestroyRealImage(InterTableH);
end (if not RegularisedOutMesh)
else begin (RegularisedOutMesh)
writeln(' First Pass: Horizontal warp (regularised)');
GlobalUnlock(Map1H);
for y := 0 to imageHeight-1 do begin (Visit each row)
    for x := 0 to meshWidth-1 do
        Map1^[x] := PReal(OffsetPointer(XMeshOutData, (x + y * imageWidth) * RSize))^;
        (Resample input image row based on map1 into intermediate image)
    Source := PByte(OffsetPointer(InImageData, y * imageWidth));
    Dest := PByte(OffsetPointer(InterImageData, y * imageWidth));
    Error := ResampleScanline(Map1, Source, Dest, imageWidth, 1);
    if Error <> NoErr then goto MeshWarpImageExit;
    end; (for y)
    Map1 := GlobalLock(Map1H);
end; (else RegularisedOutMesh)
(-SECOND PASS (phase 1): creates tables for y-intercepts of horizontal splines )
( of intermediate and output images )
if not(RegularisedOutMesh) then begin
writeln(' Second Pass: Vertical warp');
Error := CreateRealImage(InterTableH, imageWidth, meshHeight);
if Error <> NoErr then goto MeshWarpImageExit;
InterTable := GlobalLock(InterTableH);
Error := CreateRealImage(OutTableH, imageWidth, meshHeight);
if Error <> NoErr then goto MeshWarpImageExit;
OutTable := GlobalLock(OutTableH);
InterTableData := @InterTable^.data; (Get start address of table data)
OutTableData := @OutTable^.data;
for x := 0 to imageWidth-1 do Index^[x] := x; (Indices to sample horizontal splines)
y := 0;
if InMeshSpecified then begin
    y := 0;
    repeat (Scan convert horizontal splines of intermediate and output images)
        address := y * meshWidth * RSize;
        for x := 0 to meshWidth-1 do begin
            yRow1^[x] := PReal(OffsetPointer(YMeshInData, address))^;
            xRow2^[x] := PReal(OffsetPointer(XMeshOutData, address))^;
            yRow2^[x] := PReal(OffsetPointer(YMeshOutData, address))^;
            Inc(address, RSize);
        end; (for x)
        Error := InterpolateLinear(xRow2, yRow1, meshWidth, Index, Map1, imageWidth);
        if (Error = NoErr) then begin
            Error := InterpolateLinear(xRow2, yRow2, meshWidth, Index, Map2, imageWidth);
        end; (if NoErr)
        address := y * meshWidth * RSize;
        for x := 0 to meshWidth-1 do begin (Store resampled rows )
            PReal(OffsetPointer(InterTableData, address))^ := Map1^[x];
            PReal(OffsetPointer(OutTableData, address))^ := Map2^[x];
            Inc(address, RSize);
        end; (for x)
        Inc(y);
    until (y >= meshHeight) or (Error <> NoErr);
end (if InMeshSpecified)
else begin (Regular 1-pixel InMesh grid assumed)
    repeat (Scan convert horizontal splines of intermediate and output images)
        address := y * meshWidth * RSize;
        for x := 0 to meshWidth-1 do begin
            yRow1^[x] := y; (Regular grid)
            xRow2^[x] := PReal(OffsetPointer(XMeshOutData, address))^;
            yRow2^[x] := PReal(OffsetPointer(YMeshOutData, address))^;
            Inc(address, RSize);
        end; (for x)

```

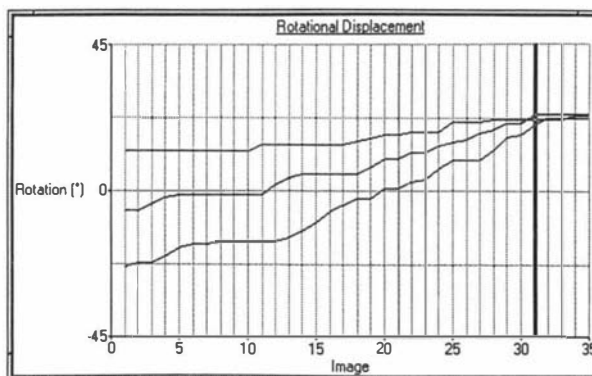
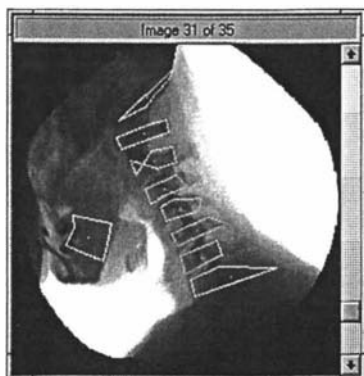
Warp60



```

Error := InterpolateLinear(xRow2, yRow1, meshWidth, Index, Map1, imageWidth);
if (Error = NoErr) then
  Error := InterpolateLinear(xRow2, yRow2, meshWidth, Index, Map2, imageWidth);
address := y * meshWidth * RSize;
for x := 0 to meshWidth-1 do begin {Copy interpolated values back}
  PReal(OffsetPointer(InterTableData, address))^ := Map1^[x];
  PReal(OffsetPointer(OutTableData, address))^ := Map2^[x];
  Inc(address, RSize);
end; {for x}
inc(y);
until (y = meshHeight) or (Error <> NoErr);
end; {else not InMeshSpecified}
(-SECOND PASS (Phase 2): Warp y using intermediate and output tables)
if Error = NoErr then begin
  for y := 0 to imageHeight-1 do Index^[y] := y;
  rowByteCount := longint(meshWidth) * RSize;
  for x := 0 to imageWidth-1 do begin {Visit each vertical spline}
    address := x * RSize;
    for y := 0 to meshHeight-1 do begin {Store each column as a row for splining}
      XRow1^[y] := PReal(OffsetPointer(InterTableData, address))^;
      YRow1^[y] := PReal(OffsetPointer(OutTableData, address))^;
      Inc(address, rowByteCount);
    end; {for y}
    {Scan convert vertical splines of input and intermediate image}
    Error := InterpolateLinear(XRow1, YRow1, meshHeight, Index, Map1, imageHeight);
    if Error <> NoErr then goto MeshWarpImageExit;
    {Resample intermediate image columns based on Map1}
    Source := PByte(OffsetPointer(InterImageData, x));
    Dest := PByte(OffsetPointer(OutImageData, x));
    Error := ResampleScanline(Map1, Source, Dest, imageHeight, imageWidth);
    if Error <> NoErr then goto MeshWarpImageExit;
  end; {for x}
end; {if NoErr}
end {if not RegularisedOutMesh}
else begin {RegularisedOutMesh}
  writeln(' Second Pass: Vertical warp (regularised)');
  rowByteCount := longint(meshWidth) * RSize;
  for x := 0 to imageWidth-1 do begin {Visit each vertical spline}
    {Store each column as a row for splining}
    address := x * RSize;
    for y := 0 to imageHeight-1 do begin
      Map1^[y] := PReal(OffsetPointer(YMeshOutData, address))^;
      Inc(address, rowByteCount);
    end; {for y}
    {Resample intermediate image columns based on Map1}
    Source := PByte(OffsetPointer(InterImageData, x));
    Dest := PByte(OffsetPointer(OutImageData, x));
    Error := ResampleScanline(Map1, Source, Dest, imageHeight, imageWidth);
    if Error <> NoErr then goto MeshWarpImageExit;
  end; {for x}
end; {else RegularisedOutMesh}
Time := GetTickCount - Time;
writeln(' Execution Time = ', (Time / 1000):2:1, ' sec');
MeshWarpImageExit:
  DestroyImage(InterImageH);
  if (Error <> NoErr) then DestroyImage(OutImageH);
  else GlobalUnlock(OutImageH);
  GlobalUnlock(InImageH);
  GlobalUnlock(XMeshInH); GlobalUnlock(YMeshInH);
  GlobalUnlock(XMeshOutH); GlobalUnlock(YMeshOutH);
  DestroyRealImage(InTableH);
  DestroyRealImage(InterTableH);
  DestroyRealImage(OutTableH);
  DestroyHandle(IndexH);
  DestroyHandle(XRow1H); DestroyHandle(YRow1H);
  DestroyHandle(XRow2H); DestroyHandle(YRow2H);
  DestroyHandle(Map1H); DestroyHandle(Map2H);
  MeshWarpImage := Error;
end; {Function: MeshWarpImage}
(=====)
end. {Unit: WarpFtns}

```



Warp30

C.8 Optimal Triangulation

The following Pascal unit '*Opt_Tri*' contains a function called *Triangulate* that performs optimal Delaunay triangulation of an arbitrary set of two-dimensional scatter points. This function was used to perform the segmentation of the reference image used in the generation of the prescribed motion cineradiographic sequences described of chapter 7.

```

unit Opt_Tri;
{Delaunay Triangulation for a set of 2-D scatter points. Can be easily extended to 3-D.}

INTERFACE
(=====INTERFACE=====)
uses
  WinDos, WinTypes, WinProcs,
  PMXTypes, PMXProcs, PMXUtils, Utils, Globals,
  MLab_ops, WarpFtns;

const
  BigNum      = 1E37;           {A large value for intialisation}
  Epsilon     = 1E-5;          {Smallest no-zero value}
  TSize       = (1 * 2) * 75;  {Tempory storage size factor}
  Range       = 10.0;          {Factor for radius of control points}
  MaxPoints   = (MaxWord div 3) div RSize; {Maximum number of control points}
  DimensionCount = 2;          {Currently hardwired}

type
  Real          = Single;
  ControlPointArrayPtr = ^ControlPointArray;
  ControlPointArray = Array[0..MaxPoints-1, 0..DimensionCount-1] of Real;
  IndexArrayPtr   = ^IndexArray;
  IndexArray      = Array[0..MaxPoints-1] of Integer;
  TriIndicesArrayPtr = ^TriIndicesArray;
  TriIndicesArray  = Array[0..MaxPoints-1, 0..2] of Integer;
  CentresArrayPtr  = ^CentresArray;
  CentresArray     = Array[0..MaxPoints-1, 0..DimensionCount] of Real;

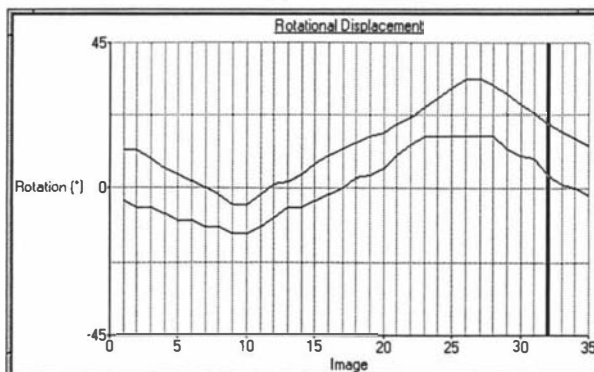
Function Triangulate(  PointsH      {Handle to the input 2-D control points}
                      : THandle;
                      pointCount   {Number of input control points}
                      : Integer;
                      var TriIndicesH {Handle to output triangles and their point indices}
                      : THandle;
                      var TriCount   {Number of triangles in the tessellation}
                      : Integer) : ErrorCode;

implementation
(=====)
Function Triangulate(  PointsH      {Handle to the input 2-D control points}
                      : THandle;
                      pointCount   {Number of input control points}
                      : Integer;
                      var TriIndicesH {Handle to output triangles and their point indices}
                      : THandle;
                      var TriCount   {Number of triangles in the tessellation}
                      : Integer) : ErrorCode;
{-Performs Delaunay optimal triangulation (triangular tessellation) on a set of 2-D real
control points stored in PointH. The output is an list of triangles (TriIndicesH) with
indices to their corresponding position in PointsH. The actual size of PointH needs to
be two entries larger than pointCount for intermediate working.

Based on C-code by Dave Watson, algorithm described in -
  Watson, D.F., 1981, Computing the n-dimensional Delaunay tessellation with
  application to Voronoi polytopes: The Computer J., 24(2), p. 167-172.
}

```

Warp60



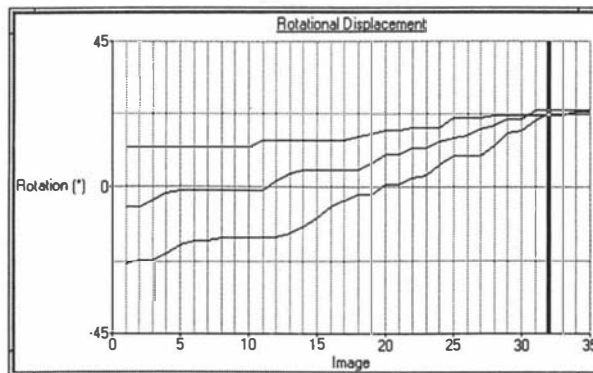
Additional information about this algorithm can be found in -

CONTOURING: A guide to the analysis and display of spatial data, by David F. Watson,
Pergamon Press, 1992, ISBN 0 08 040286 0

```

}
label
  TriangulateExit,
  Corner1, Corner2, Corner3;
var
  Points          (Pointer to x-y data points)
  : ControlPointArrayPtr;
  xx, yy,         (x and y intermediate values)
  MaxDiff         (Absolute max control point coord difference)
  : Real;
  MinMaxPoints    (Holds the min and max x and y point values)
  : Array[0..1, 0..1] of Real;
  CalcMat         (Calculation matrix)
  : Array[0..1, 0..2] of Real;
  CentresH,       (Handle to circle centres of each point)
  indexH,         (Handle control point index array)
  a3sH            (Handle to triangle output indices array)
  : THandle;
  Centres         (Pointer to circle centres of each point)
  : CentresArrayPtr;
  a3s,            (Pointer to triangle output indices array)
  TriIndices      (Pointer to output triangles and their point indices)
  : TriIndicesArrayPtr;
  iIndex          (A single triangle index)
  : Array[0..2] of Integer;
  index           (Pointer control point index array)
  : IndexArrayPtr;
  Tmp             (Temporary array to store point indices)
  : Array[0..TSize, 0..1] of integer;
  dim,            (Number of dimensions 2 or 3)
  dimMinus1,      (Number of dimensions minus one)
  i0, i1, i2, i3, (General loop counters)
  i4, i5, i6, i7,
  i8, i9, i10,
  storeSize,      (Estimated number of array elements for calculation)
  storeCount      (Actual number of array elements used in calcs)
  : integer;
  Error           (General error holder)
  : ErrorCode;
begin
  Error := NoErr;
  if pointCount > MaxPoints then begin
    writeln('Too many points, max = ', MaxPoints);
    Triangulate := OutOfRangeErr;
    Exit;
  end; (if pointCount)
  if PointsH <> 0 then Points := GlobalLock(PointsH)
  else begin
    writeln('No points specified');
    Triangulate := OutOfRangeErr;
    Exit;
  end; (else PointsH = 0)
  (Initialise handles)
  CentresH := 0;
  indexH   := 0;
  a3sH     := 0;
  dim := DimensionCount; (Numver of dimensions currently hardwired)
  dimMinus1 := dim - 1;
  for i0 := 0 to TSize-1 do begin (Initialise Tmp)
    tmp[i0, 0] := 0;
    tmp[i0, 1] := 0;
  end; (for i0)
  for i0 := 0 to 2 do (Initialise iIndex)
    iIndex[i0] := 0;
  end;
  for i0 := 0 to dimMinus1 do begin (Initialise MinMaxPoints)
    MinMaxPoints[0, i0] := - BigNum;
    MinMaxPoints[1, i0] := + BigNum;
  end; (for i0)

```

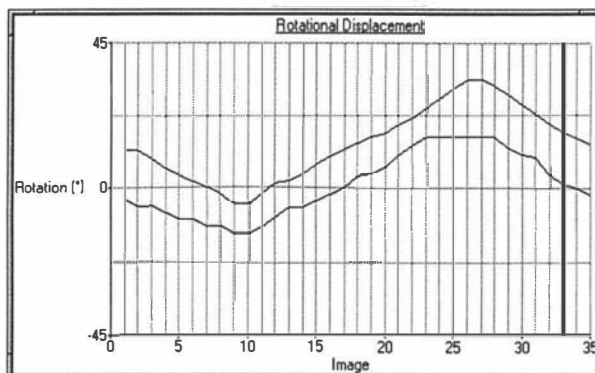


```

for i0 := 0 to dimMinus1 do {Initialise calculation matrix}
  for i1 := 0 to dim do CalcMat[i0, i1] := -Range;
for i0 := 0 to dimMinus1 do CalcMat[i0, i0] := 5 * Range;
for i0 := 0 to pointCount-1 do {Find points with min and max, x and y values}
  for i1 := 0 to dimMinus1 do begin
    if MinMaxPoints[0, i1] < Points^[i0, i1]
    then MinMaxPoints[0, i1] := Points^[i0, i1];
    if MinMaxPoints[1, i1] > Points^[i0, i1]
    then MinMaxPoints[1, i1] := Points^[i0, i1];
  end; {for i1}
MaxDiff := 0;
for i0 := 0 to dimMinus1 do begin {Find max point difference from the min-max points}
  MinMaxPoints[0, i0] := MinMaxPoints[0, i0] - MinMaxPoints[1, i0];
  if MaxDiff < MinMaxPoints[0, i0] then MaxDiff := MinMaxPoints[0, i0];
end; {for i0}
MaxDiff := MaxDiff * Epsilon; {Scale max point difference by epsilon}
RandSeed := 367; {Initialise random gen}
for i0 := 0 to pointCount-1 do {Add random perturbation to points}
  for i1 := 0 to dimMinus1 do
    Points^[i0, i1] := Points^[i0, i1] + MaxDiff * (0.5 - Random(MaxWord)/MaxWord);
for i0 := 0 to dim do
  for i1 := 0 to dimMinus1 do
    Points^[pointCount + i0, i1] :=
      MinMaxPoints[1, i1] + CalcMat[i1, i0] * MinMaxPoints[0, i1];
storeSize := dim * (pointCount + dim); {Estimate storage allocation needed}
Error := CreateHandle(indexH, storeSize * ISize, 0);
if Error <> NoErr then begin
  writeln('Not enough memory to create indexH');
  goto TriangulateExit;
end;
index := GlobalLock(indexH);
for i0 := 0 to storeSize-1 do {Initialise index array}
  index^[i0] := i0;
Error := CreateHandle(a3sH, storeSize * (3 * ISize), 0);
if Error <> NoErr then begin
  writeln('Not enough memory to create a3sH');
  goto TriangulateExit;
end;
a3s := GlobalLock(a3sH);
Error := CreateHandle(CentresH, storeSize * (3 * RSize), 0);
if Error <> NoErr then begin
  writeln('Not enough memory to create CentresH');
  goto TriangulateExit;
end;
Centres := GlobalLock(CentresH);
for i0 := 0 to storeSize-1 do {Initialise a3s and Centres}
  for i1 := 0 to dim do begin
    a3s^[i0, i1] := 0;
    Centres^[i0, i1] := 0;
  end; {for i1}
for i0 := 0 to dim do a3s^[0, i0] := pointCount + i0;
{Initialise Centres}
Centres^[0, dim] := BigNum;
for i0 := 0 to dimMinus1 do Centres^[0, i0] := 0;
{-Start triangulation}
storeCount := 1;
i4 := 1;
for i0 := 0 to pointCount-1 do begin
  i1 := -1;
  i7 := -1;
  i9 := 0;
  i10 := 0;
  while i10 < storeCount do begin
    inc(i1);
    while a3s^[i1, 0] < 0 do inc(i1);
    xx := Centres^[i1, dim];
    for i2 := 0 to dimMinus1 do begin
      xx := xx - sqr(Points^[i0, i2] - Centres^[i1, i2]);
      if xx < 0 then goto Corner3;
    end; {for i2}
    dec(i9);
  end;
end;

```

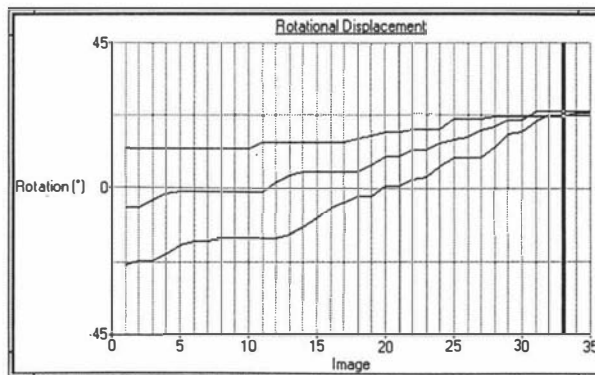
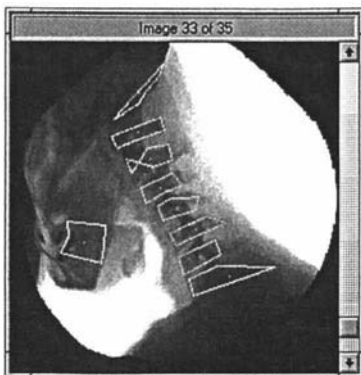
Warp60



```

dec(i4);
index^[i4] := i1;
i2 := 0;
while i2 < (dim+1) do begin
  iIndex[0] := 0;
  if iIndex[0] = i2 then inc(iIndex[0]);
  for i3 := 1 to dimMinus1 do begin
    iIndex[i3] := iIndex[i3-1] + 1;
    if iIndex[i3] = i2 then inc(iIndex[i3]);
  end; {for i3}
  if i7 > dimMinus1 then begin
    i8 := i7;
    i3 := 0;
    while i3 <= i8 do begin
      for i5 := 0 to dimMinus1 do
        if a3s^[i1, iIndex[i5]] <> Tmp[i3, i5] then goto Corner1;
      for i6 := 0 to dimMinus1 do
        Tmp[i3, i6] := Tmp[i8, i6];
      dec(i7);
      goto Corner2;
    end;
  end;
  inc(i3);
end; {while i3}
end; {if i7}
inc(i7);
if i7 > (TSize-1) then begin
  writeln('Temporary storage exceeded- increase TSize');
  Error := memoryErr;
  goto TriangulateExit;
end; {if i7}
for i3 := 0 to dimMinus1 do Tmp[i7, i3] := a3s^[i1, iIndex[i3]];
Corner2:
  inc(i2);
end; {while i2}
a3s^[i1, 0] := -1;
Corner3:
  inc(i10);
end; {while i10}
for i1 := 0 to i7 do begin
  if not(false and (tmp[i1, 0] < pointCount)) then begin {}
    for i2 := 0 to dimMinus1 do begin
      CalcMat[i2, dim] := 0;
      for i3 := 0 to dimMinus1 do begin
        CalcMat[i2, i3] := Points^[Tmp[i1, i2], i3] - Points^[i0, i3];
        CalcMat[i2, dim] := CalcMat[i2, dim]
          + CalcMat[i2, i3] * (Points^[Tmp[i1, i2], i3] + Points^[i0, i3]) / 2;
      end; {for i3}
    end; {for i2}
    xx := CalcMat[0, 0] * CalcMat[1, 1] - CalcMat[1, 0] * CalcMat[0, 1];
    Centres^[index^[i4], 0]
      := (CalcMat[0, 2] * CalcMat[1, 1] - CalcMat[1, 2] * CalcMat[0, 1]) / xx;
    Centres^[index^[i4], 1]
      := (CalcMat[0, 0] * CalcMat[1, 2] - CalcMat[1, 0] * CalcMat[0, 2]) / xx;
    Centres^[index^[i4], dim] := 0;
    for i2 := 0 to dimMinus1 do begin
      Centres^[index^[i4], dim]
        := Centres^[index^[i4], dim] + sqr(Points^[i0, i2] - Centres^[index^[i4], i2]);
      a3s^[index^[i4], i2] := Tmp[i1, i2];
    end; {for i2}
    a3s^[index^[i4], dim] := i0;
    inc(i4);
    inc(i9);
  end; {if not..}
end; {for i1}
inc(storeCount, i9);
end; {for i0}
(Create output triangle indices array)
Error := CreateHandle(TriIndicesH, storeCount * (3 * ISize), 0);
if Error <> NoErr then begin
  writeln('Not enough memory to create TriIndicesH');
  goto TriangulateExit;
end;

```



```

TriIndices := GlobalLock(TriIndicesH);
  {Continue with the Triangulation}
  i0 := -1;
  triCount := 0;
  for i10 := 0 to storeCount-1 do begin
    inc(i0);
    while a3s^[i0, 0] < 0 do inc(i0);
    if a3s^[i0, 0] < pointCount then begin
      for i1 := 0 to dimMinus1 do for i2 := 0 to dimMinus1 do
        CalcMat[i1, i2] := Points^[a3s^[i0, i1], i2] - Points^[a3s^[i0, dim], i2];
      xx := CalcMat[0,0] * CalcMat[1,1] - CalcMat[0,1] * CalcMat[1,0];
      if abs(xx) > Epsilon then begin {A physical triangle exits so copy to TriIndices}
        TriIndices^[triCount, 0] := a3s^[i0, 0];
        if xx < 0 then begin
          TriIndices^[triCount, 1] := a3s^[i0, 2];
          TriIndices^[triCount, 2] := a3s^[i0, 1];
        end {if xx < 0}
        else begin
          TriIndices^[triCount, 1] := a3s^[i0, 1];
          TriIndices^[triCount, 2] := a3s^[i0, 2];
        end; {else}
        Inc(triCount);
      end; {if abs..}
    end; {if a3s}
  end; {for i10}
  {Reduce the size of TriIndices to the actual size required.}
  Error := ResizeHandle(TriIndicesH, triCount * (3 * ISize), 0);
  if Error <> NoErr then begin
    writeln('Error resizing TriIndicesH');
  end;
TriangulateExit:
  DestroyHandle(indexH);
  DestroyHandle(a3sH);
  DestroyHandle(CentresH);
  if Error <> NoErr then DestroyHandle(TriIndicesH);
  GlobalUnlock(PointsH);
  Triangulate := Error;
end; {Triangulate}
(=====)
end. (Opt_Tri)

```

C.8 Prescribed Motion by Image Warping

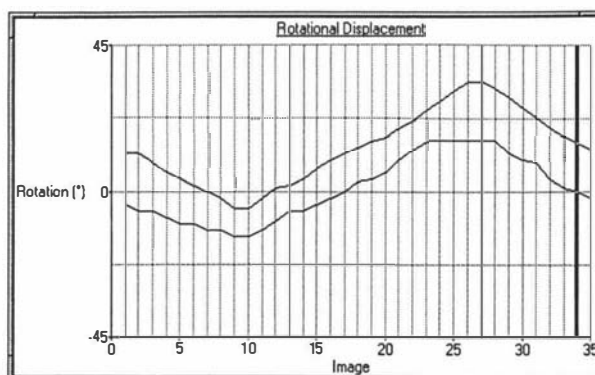
The following Pascal program 'CineWarp' was used to generate the prescribed motion cineradiographic sequences utilised in chapter 7 in the evaluation of the motion-tracking algorithm. Based on a trajectory file (defining the motion of the defined features) the reference image is warped to produce the animation.

```

Program CineWarp;
  {-Program to animate veterbra from a single cineradiographic image using Digital Image
  Warping Techniques. Animation based on trajectory file data.
  }
  uses
    WinCrt, WinDos, WinProcs, WinTypes, Strings,
    Globals, Resource, Files, PMXTypes, PMXProcs, PMXUtils, Points, Utils,
    WarpFtns, Images, Filters, MLab_ops, Opt_Tri;

  label
    CineWarpExit;                                {Exit goto point for the program}

```



```

const
  TabChar = 0;                                (Ascii TAB character)
  TriPatchStr = '>';                          (Triangle string representation)
  TrajectorySpecFileExt = '.tjs';             (File extension of a trajectory specification file)
  TrajectoryDataFileExt = '.tjd';            (File extension of a trajectory data file)
  CMXDocPath = 'd:\cmxdocs';                 (Path to CMX documents)
  InFileName = CMXDocPath + '\ccref1.cmx';    (File containing corrected ref x-ray data
                                              with outline points)

  Version = $100;                             (Current CMX patient file version)
  MaxEdgePointCount = 32;                     (Max number of edge points allowed)
  MaxObjectCount = 16;                       (Max number of objects that can be animated)
  MaxOutlinePointCount = 24 * MaxEdgePointCount; (Max number of points forming an object)
  MaxImageCount = 75;                        (Max number of images in a sequence)
  MaxTriPatchCount = MaxInt div 64;          (Max number of TriPatches in an image)
  MaxScanlineCount = 127;                   (Max number of scanlines in an object)
  BPCount = 10;                             (Number of points defined on each axis of the boundary)

  (Program control.)
  AddTriPatchesToPointGroups = (True)False;
  AddTriPatchesToImages = (True)False;
  SaveOutputMeshes = (True)False;
  DefaultTrajectoryFileName = 'Warp63';      (Name of the trajectory file)

type
  RPoint = record (Real precision x,y coords)
    x, y : Real;
  end;

  ObjectIndexRec = record (Structure for objects defined in the image)
    objectNum (The number of the object, 0 implies boundary object)
      : byte;
    centroid (The centroid of the object)
      : RPoint;
  end; (ObjectIndexRec)
  ObjectIndexArrayPtr = ^ObjectIndexArray;
  ObjectIndexArray = Array[0..MaxObjectCount-1] of ObjectIndexRec;
  XYThetaRec = record
    x, y, (x and y translation in pixels)
    Theta (rotation in radians)
      : Real
  end; (XYTheta)
  ObjectsPositionArray = Array[0..MaxObjectCount-1] of XYThetaRec;
  TrajectoryArray = Array[0..MaxImageCount-1] of ObjectsPositionArray;
  TrajectoryArrayPtr = ^TrajectoryArray;
  TriPatchRec = record (Triangular planar patches in x and y affine mapping ip's to op's)
    ipt (Input coords of the three points forming the patch)
      : Array[0..2] of TPoint;
    opt (Output coords of the three points forming the patch)
      : Array[0..2] of RPoint;
    apt (Object association of the three points)
      : Array[0..2] of ObjectIndexRec;
    ax, bx, cx, dx, (x-patch coeffs of the fitted TriPatch)
    ay, by, cy, dy (y-patch coeffs of the fitted TriPatch)
      : Real;
  end; (TriPatchRec)
  TriPatchArray = Array[0..MaxTriPatchCount-1] of TriPatchRec;
  TriPatchArrayPtr = ^TriPatchArray;
  HorizontalScanlineRec = record
    y, xStart, xEnd : word;
  end;
  TriPatchScanlineArrayPtr = ^TriPatchScanlineArray;
  TriPatchScanlineArray = Array[0..MaxScanlineCount-1] of HorizontalScanlineRec;
  OutlinePointArray = Array[0..MaxOutlinePointCount-1] of TPoint;

  (Define file components)
  fPathStr = Array[0..fsPathName] of Char;
  fDirStr = Array[0..fsDirectory] of Char;
  fNameStr = Array[0..fsFileName] of Char;
  fExtStr = Array[0..fsExtension] of Char;

var
  x, y, (General column and row counters for each image)
  ImageBytes, (Total size of each image in bytes)
  fileNum (Number of the CMX doc file)
    : Longint;

```

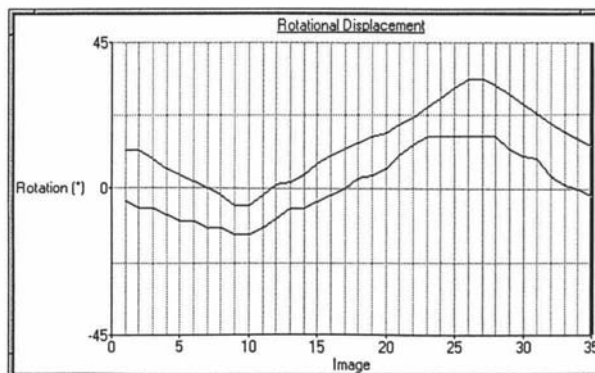


```

i, j,                (General counters)
imageNum,            (Counter for each image in the animation)
lineNum,             (Scan line counter)
rowByteCount,       (Width of an image scanline, accounts for longint alignment)
pointGroupNum,      (Current PointGroup number)
outlinePointCount,  (Number of points in the current tri-patch outline)
outlinePointNum     (Counter for points in the current tri-patch outline)
: Word;
imageCount,          (Number of image in the animated sequence)
objectCount,         (Number of objects to animate)
objectNum,           (Counter for number of objects)
EdgePointCount,     (Number of edge point in the current object)
EdgePointNum,       (Counter of edge point in the current object)
controlPointCount,  (Number of control points defined )
triPatchCount,      (Number of triangular patches)
triPatchNum,        (TriPatch counter)
pointNum            (Point counter)
: integer;
FileComment          (Warp file comment)
: Str255;
  searchInfo          (File search info)
: TSearchRec;
ObjectIndicesH,      (Handle for the structure defining the vertebra to be animated)
ControlPointsH,      (Handle for the control points)
TriIndicesH,         (Handle to the tri-indices array to the control points)
TrajectoryH,         (Handle for the trajectory path of the vertebra)
TriPatchesH          (Handle for TriPatches)
: THandle;
ObjectIndices        (Defines objects interms of horizontal and vertical lines)
: ObjectIndexArrayPtr;
ControlPoints        (Pointer to the control points)
: ControlPointArrayPtr;
TriIndices           (Pointer to the tri-indices array to the control points)
: TriIndicesArrayPtr;
Trajectory           (Trajectory of the four vertebra to be animated)
: TrajectoryArrayPtr;
ObjectInclude        (Defines whether an object will be included in the animation)
: Array[0..MaxObjectCount-1] of boolean;
TriPatches           (An array of triangular patches tessalating the image)
: TriPatchArrayPtr;
TriPatchScanlines    (Structure to hold the scanlines of a single tri-patch)
: TriPatchScanlineArray;
OutlinePoints        (All points forming a tri-patch outline )
: OutlinePointArray;
imageSize,           (Image dimensions, x and y)
meshSize,            (Mesh dimensions, width and height)
point               (General TPoint)
: TPoint;
rePoint             (General RPoint)
: RPoint;
obPoint             (General ObjectIndexRec var)
: ObjectIndexRec;
InImageH,            (Handle for the input image)
OutImageH,           (Handle for the Output image)
InterImageH          (Handle for the Intermediate image)
: ImageHandle;
InImage, OutImage,   (Images pointers)
InterImage
: ImagePtr;
inFile, outFile      (General input and output file)
: TFileRec;
path,                (Path var to CMX docs)
outFileName,         (General output file name)
AnimationFileName,   (Name of the output warp animation CMX file)
TrajectoryFileName   ( )
: fPathStr;
Name                 (Name component of a file path after fileSplit)
: fNameStr;
Directory            (Directory component of a path after fileSplit)
: fDirStr;

```

Warp60

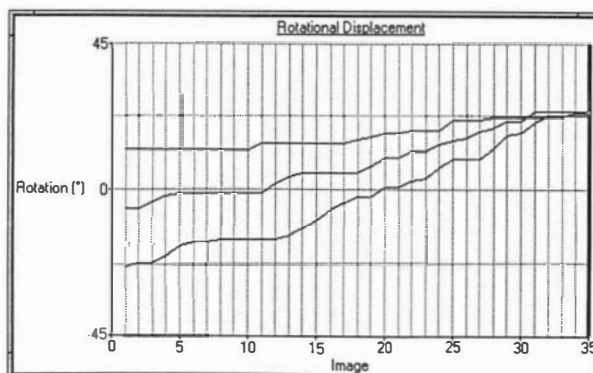


```

Extension          (Extension component of a path after fileSplit)
                   : fExtnStr;
dataFile           : Text;
s, fileName        : String;
Error              : ErrorCode;
  DocumentInfo     : DocInfoRec;
  docID            : DocIDRec;
  PatData          : PatientIDRec;
  ss               : FileStr;
OptData            : OperatorStr;
DataSize           : Longint;
PointInfoH,        (Handles to point data)
PointInfo2H        : PointInfoHandle;
pointInfo          : pointInfoPtr;
PointGroupName     : PointGroupNameStr;
Sorted             : Boolean;
  (-----FormTrajectory-----)
Function FormTrajectory(trajecoryFileName : fPathStr) : ErrorCode;
  (-Reads the specified trajectory specification file and based on the specified
  parameters generates the trajectories for each object in the animation sequence.
  Returns a handle to the trajectory along with the number of images and a comment
  about the form of the trajectory. Also produces a trajectory data file containing
  the actual trajectory. )
var
  trajectoryFile   (Trajectory file)
                   : Text;
  Trajectory       (Trajectory of the four vertebra to be animated)
                   : TrajectoryArrayPtr;
  tForm            (Trajectory form, N=Normal or L=List)
                   : Char;
  tPLim, tNLim,    (Positive and negative limits for the current object param)
  tCycles,         (Number of cycles for the current object param)
  dFrame,          (Number of frames per cycle for the current object param)
  value,           (Value of the current parameter for the current object)
  dValue,          (Value step size for the current object param)
  x, y, theta      (Trajectory values)
                   : Real;
  tFrames,         (Number of frames in the sequence)
  tType,           (Type of sequence for the current object param, 1=Linear, 2=Sine)
  frameNum,        (Frame counter)
  objectNum,       (Object counter)
  paramNum         (Current object parameter counter)
                   : Byte;
  VertebraName     : String;

begin
  Error := NoErr;
  TrajectoryH := 0;
  Assign(trajectoryFile, trajectoryFileName);
  Reset(trajectoryFile);
  if IOResult <> 0 then Error := fileErr;
  if Error = NoErr then readln(trajectoryFile, FileComment);
  if IOResult <> 0 then Error := fileErr;
  writeln(FileComment);
  if Error = NoErr then readln(trajectoryFile, tForm, tFrames);
  if IOResult <> 0 then Error := fileErr;
  tForm := UpCase(tForm);
  if (Error <> NoErr) or (tFrames > MaxImageCount) then begin
    writeln('Too many frames specified in trajectory ', tFrames);
    TrajectoryH := 0;
    Error := OutofRangeErr;
  end (if tFrames)
  else begin
    imageCount := tFrames;
    Error := CreateHandle(TrajectoryH, imageCount * Sizeof(ObjectsPositionArray), 0);
  end; (else)
end;

```




```

if Error = NoErr then begin
  Trajectory := GlobalLock(TrajectoryH);
  objectNum := 0;
  while not(Eof(trajjectoryFile)) do begin
    if tForm = 'N' then begin {Normal definition file}
      read(trajjectoryFile, tType);
      if tType = 0 then ObjectInclude[objectNum] := False
      else begin
        ObjectInclude[objectNum] := True;
        for paramNum := 1 to 3 do begin
          if paramNum > 1 then read(trajjectoryFile, tType);
          case tType of
            1 : {Linear sequence}
              begin
                read(trajjectoryFile, tCycles, tNLim, tPLim);
                if paramNum = 3 then begin {Convert limits to radians}
                  tPLim := pi * tPLim / 180; tNLim := pi * tNLim / 180;
                end; {if tParam}
                dFrame := (tFrames-1) / (tCycles); {Number of frames per cycle}
                dValue := - 1/ dFrame; {initialise dValue}
                for frameNum := 0 to tFrames-1 do begin
                  dValue := dValue + 1 / dFrame;
                  if dValue > 1.0 then dValue := dValue - 1.0; {pin dValue to range 0-1}
                  {Piecewise periodic map dValue: 0.0->0.25->0.75->1.0
                   becomes 0.0->1.0->-1.0->0.0}
                  if (dValue < 0.25) then value := 4 * dValue
                  else if (dValue < 0.75) then value := 1.0 - 4 * (dValue - 0.25)
                  else value := -1.0 + 4 * (dValue - 0.75);
                  {Scale to limits}
                  value := (tPLim + tNLim)/2 + (tPLim - tNLim)/2 * value;

                  case paramNum of
                    1 : Trajectory^[frameNum, objectNum].x := value;
                    2 : Trajectory^[frameNum, objectNum].y := value;
                    3 : Trajectory^[frameNum, objectNum].theta := value;
                  end; {case tParam}
                end; {for frameNum}
              end; {case 1}
            2 : {Sinusodial sequence}
              begin
                read(trajjectoryFile, tCycles, tNLim, tPLim);
                if paramNum = 3 then begin {Convert limits to radians}
                  tPLim := pi * tPLim / 180;
                  tNLim := pi * tNLim / 180;
                end; {if tParam}
                dFrame := (tFrames-1) / (tCycles);
                for frameNum := 0 to tFrames-1 do begin
                  value :=
                    (tPLim + tNLim)/2 + (tPLim - tNLim)/2 * Sin(2*pi * frameNum/dFrame);
                  case paramNum of
                    1 : Trajectory^[frameNum, objectNum].x := value;
                    2 : Trajectory^[frameNum, objectNum].y := value;
                    3 : Trajectory^[frameNum, objectNum].theta := value;
                  end; {case tParam}
                end; {for frameNum}
              end; {case 2}
            end; {case tType}
          end; {for paramNum}
          end; {else tType > 0}
          readln(trajjectoryFile); {move to next line}
          Inc(objectNum);
        end {if tForm}
      else begin {List File definition}
        objectNum := 0;
        while not(EOF(trajjectoryFile)) do begin
          readln(trajjectoryFile, VertebraName);
          for frameNum := 0 to tFrames-1 do begin
            with Trajectory^[frameNum, objectNum] do begin
              readln(trajjectoryFile, x, y, theta);
              theta := theta/180*pi; {Convert to radians}
              if IOResult <> 0 then begin
                writeln('Error reading ',objectNum,' ', frameNum); readln;
                FormTrajectory := IOResult;
                Close(trajjectoryFile);
                Exit;
              end; {if Error}
            end; {with Trajectory}
          end; {for frameNum}
          ObjectInclude[objectNum] := True;
          inc(objectNum);
        end; {while not EOF}
      end; {else List definition}
    end; {while not Eof}
    Close(trajjectoryFile);
    objectCount := objectNum;
  end {if trajectory created}

```

```

else writeln('Error not enough memory for trajectory');
GlobalUnlock(TrajectoryH);
FormTrajectory := Error;
end; (FormTrajectory)
(-----FormControlPoints-----)
Function FormControlPoints : ErrorCode;
(-Get the edge points for each object and add them to the controlPoints list and compute
the centroid of each object and add it to the ObjectIndices list along the its object
number. Finally add evenly spaced boundary control points based on BPCount.
)
var
Error          : ErrorCode;
x, y,          : (x and y coord counters)
count,         : (general counter)
edgePointCount, (Number of edge points in the current pointGroup)
edgePointNum,  : (Current Edge point number)
pointGroupNum  : (Current Point group number)
: Integer;
xSum, ySum     : (x and y coord sum used in object centroid calculation)
: longint;
centroid       : (Centroid of the current object)
: RPoint;
begin (MAIN: FormControlPoints)
Error := NoErr;
pointGroupNum := 0;
repeat (-Range check the number of edge points in each object)
GetPointGroupName(PointInfoH, pointGroupNum, PointGroupName);
edgePointCount := GetEdgePointCount(PointInfoH, @PointGroupName);
if EdgePointCount >= MaxEdgePointCount then Error := OutOfRangeErr;
inc(pointGroupNum);
until (pointGroupNum = ObjectCount) or (Error <> NoErr);
if Error = NoErr then begin
controlPointCount := 0;
for pointGroupNum := 0 to objectCount-1 do begin
if ObjectInclude[pointGroupNum] then begin (add the object)
xSum := 0; (Initialise object centroid sums)
ySum := 0;
GetPointGroupName(PointInfoH, pointGroupNum, PointGroupName);
edgePointCount := GetEdgePointCount(PointInfoH, @PointGroupName);
for edgePointNum := 0 to edgePointCount-1 do begin
GetPoint(PointInfoH, @PointGroupName, edgePointNum, point);
ControlPoints^[controlPointCount, 0] := point.x;
ControlPoints^[controlPointCount, 1] := point.y;
ObjectIndices^[controlPointCount].objectNum := pointGroupNum+1;
Inc(controlPointCount);
inc(xSum, point.x); (Add x and y values to sums)
inc(ySum, point.y);
end; (for edgePointNum)
centroid.x := xSum / edgePointCount;
centroid.y := ySum / edgePointCount;
for count := (controlPointCount-edgePointCount) to controlPointCount-1 do
ObjectIndices^[count].centroid := centroid; (fill in object centroid)
end; (if ObjectInclude)
end; (for pointGroupNum)
(-Add in four corner boundary control points)
ControlPoints^[controlPointCount, 0] := 0;
ControlPoints^[controlPointCount, 1] := 0;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
ControlPoints^[controlPointCount, 0] := imageSize.x-1;
ControlPoints^[controlPointCount, 1] := 0;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
ControlPoints^[controlPointCount, 0] := 0;
ControlPoints^[controlPointCount, 1] := imageSize.y-1;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
ControlPoints^[controlPointCount, 0] := imageSize.x-1;
ControlPoints^[controlPointCount, 1] := imageSize.y-1;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
(-Add in evenly spaced boundary point on each axis)
count := imageSize.x div (BPCount-1);
x := count;
while x < (imageSize.x - count) do begin (Define x axis mid boundary control points)
ControlPoints^[controlPointCount, 0] := x;
ControlPoints^[controlPointCount, 1] := 0;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
ControlPoints^[controlPointCount, 0] := x;
ControlPoints^[controlPointCount, 1] := imageSize.y-1;
ObjectIndices^[controlPointCount].objectNum := 0; (boundary object)
Inc(controlPointCount);
Inc(x, count);
end; (while x)
(Define y axis mid-boundary control points)
count := imageSize.y div (BPCount-1);

```

```

y := count;
while y < (imageSize.y - count) do begin
  ControlPoints^[controlPointCount, 0] := 0;
  ControlPoints^[controlPointCount, 1] := y;
  ObjectIndices^[controlPointCount].objectNum := 0; {boundary object}
  Inc(controlPointCount);
  ControlPoints^[controlPointCount, 0] := imageSize.x-1;
  ControlPoints^[controlPointCount, 1] := y;
  ObjectIndices^[controlPointCount].objectNum := 0; {boundary object}
  Inc(controlPointCount);
  Inc(y, count);
end; {while y}
end; {if NoErr}
FormControlPoints := Error;
end; {Function: FormControlPoints}
)-----FitTriangularPatch-----)
Function FitTriangularPatch(var TriPatch : TriPatchRec) : ErrorCode;
(-Computes the affine coefficients that map the input triangle formed by ipt's to the
 output triangle formed by opt's. The coefficients of the triangular mapping are returned
 in ax,bx,cx,dx for x and ay,by,cy,dy for y planes.

      ax * in.x + bx * in.y + cx * out.x + dx = 0
      ay * in.x + by * in.y + cy * out.y + dy = 0

)
var
  diff1, diff2,      {Point coords differences}
  diff3
      : Real;

begin {FitTriangularPatch}
  Error := NoErr;
  with TriPatch do begin
    {Calculate the x patch coefficients.}
    diff1 := opt[1].x - opt[2].x;
    diff2 := longint(ipt[1].y) - ipt[2].y;
    diff3 := longint(ipt[1].x) - ipt[2].x;
    ax := ipt[0].y * diff1
        - opt[0].x * diff2 + (ipt[1].y * opt[2].x - opt[1].x * ipt[2].y);
    bx := -(ipt[0].x * diff1 - opt[0].x * diff3
        + (ipt[1].x * opt[2].x - ipt[2].x * opt[1].x));
    cx := ipt[0].x * diff2 - ipt[0].y * diff3
        + (longint(ipt[1].x) * ipt[2].y - longint(ipt[2].x) * ipt[1].y);
    dx := -(ipt[0].x * (ipt[1].y * opt[2].x - ipt[2].y * opt[1].x)
        - ipt[0].y * (ipt[1].x * opt[2].x - ipt[2].x * opt[1].x)
        + opt[0].x * (longint(ipt[1].x) * ipt[2].y - longint(ipt[2].x) * ipt[1].y));

    {Calculate the y patch coefficients.}
    diff1 := opt[1].y - opt[2].y;
    diff2 := longint(ipt[1].y) - ipt[2].y;
    diff3 := longint(ipt[1].x) - ipt[2].x;
    ay := ipt[0].y * diff1 - opt[0].y * diff2
        + (ipt[1].y * opt[2].y - opt[1].y * ipt[2].y);
    by := -(ipt[0].x * diff1 - opt[0].y * diff3
        + (ipt[1].x * opt[2].y - ipt[2].x * opt[1].y));
    cy := ipt[0].x * diff2 - ipt[0].y * diff3
        + (longint(ipt[1].x) * ipt[2].y - longint(ipt[2].x) * ipt[1].y);
    dy := -(ipt[0].x * (ipt[1].y * opt[2].y - ipt[2].y * opt[1].y)
        - ipt[0].y * (ipt[1].x * opt[2].y - ipt[2].x * opt[1].y)
        + opt[0].y * (longint(ipt[1].x) * ipt[2].y - longint(ipt[2].x) * ipt[1].y));

    end; {with TriPatch}
  FitTriangularPatch := Error;
end; {FitTriangularPatch}
)-----TransformPoint-----)
Procedure TransformPoint(
  inPoint      {Input point coords}
      : TPoint;
  centrePoint  {Centroid of transform system}
      : RPoint;
  XYTheta      {x,y and theta defining the transform}
      : XYThetaRec;
  var outPoint {Transformed coords}
      : RPoint;
)
(-Transforms the InPoint to get the OutPoint about the CentrePoint. The transform
 involves simple translation and rotation about the centre point.
)
begin {TransformPoint}
  with XYTheta do begin
    outPoint.x := (inPoint.x - centrePoint.x) * Cos(Theta)
        - (inPoint.y - centrePoint.y) * Sin(Theta) + x + centrePoint.x;
    outPoint.y := (inPoint.x - centrePoint.x) * Sin(Theta)
        + (inPoint.y - centrePoint.y) * Cos(Theta) + y + centrePoint.y;
    end; {with XYTheta}
  end; {TransformPoint}
)-----LineProc-----)
Procedure LineProc(x,y : integer; lParam : Longint); Export;
begin (-Used by LineDDA in interpolating object outlines)

```

```

OutlinePoints[outlinePointNum].x := x;
OutlinePoints[outlinePointNum].y := y;
inc(outlinePointNum);
end; (LinProc)
)-----SaveOutputMesh-----)
Function SaveOutputMesh(xMeshH, yMeshH : meshHandle; meshNumber : integer) : ErrorCode;
(-Saves the current x and y output mesh grids as bitmap image files: xMesh_n.bmp,
 ymesh_n.bmp where n = image number
)
type
  TypeArray = array[0..1] of Char;
  BitmapInfoRec = record
    header : TBitmapInfoHeader;
    colors : array[0..255] of TRGBQuad;
  end;
var
  x, y, (Counters)
  address, (Mesh element address)
  meshWidth, meshHeight, (Mesh width and height)
  rowByteCount (Number of bytes per row in BMP)
  : longint;
  xBuffer, yBuffer (x and y line buffers)
  : Array[0..1023] of byte;
  xMeshfile, yMeshfile (x and y mesh files)
  : TFileRec;
  bitmapInfo : BitmapInfoRec;
  fileInfo : TBitmapFileHeader;
  xMesh, yMesh (Pointers to x and y mesh)
  : MeshPtr;
  xMeshData, yMeshData (Pointer to x and y mesh data)
  : PReal;
  s : string;
  Error (General Error var)
  : ErrorCode;

begin (SaveOutputMesh)
  if (xMeshH > 0) and (yMeshH > 0) then begin (Mesh grids are defined)
    writeln(' Saving Mesh grids');
    xMesh := GlobalLock(xMeshH);
    yMesh := GlobalLock(yMeshH);
    xMeshData := @xMesh^.data;
    yMeshData := @yMesh^.data;
    meshWidth := xMesh^.size.x;
    meshHeight := xMesh^.size.y;
    Str(meshNumber, s);
    Assign(File(xMeshFile), 'Meshx_' + s + '.bmp');
    Assign(File(yMeshFile), 'Meshy_' + s + '.bmp');
    Rewrite(File(xMeshFile), 1);
    if IOResult <> 0 then writeln('Error xMeshFile');
    Rewrite(File(yMeshFile), 1);
    if IOResult <> 0 then writeln('Error yMeshFile');
    TypeArray(fileInfo.bfType) := 'BM';
    fileInfo.bfSize := SizeOf(fileInfo) + SizeOf(bitmapInfo) + meshWidth * meshHeight;
    fileInfo.bfReserved1 := 0;
    fileInfo.bfReserved2 := 0;
    fileInfo.bfOffBits := SizeOf(fileInfo) + SizeOf(bitmapInfo);
    Error := FileWrite(xMeshFile, @fileInfo, SizeOf(fileInfo), 0);
    if Error <> NoErr then writeln('1 Error writing output file1 info ', Error);
    Error := FileWrite(yMeshFile, @fileInfo, SizeOf(fileInfo), 0);
    if Error <> NoErr then writeln('2 Error writing output file2 info ', Error);
    bitmapInfo.Header.biSize := SizeOf(bitmapInfo.Header);
    bitmapInfo.Header.biWidth := meshWidth;
    bitmapInfo.Header.biHeight := meshHeight;
    bitmapInfo.Header.biPlanes := 1;
    bitmapInfo.Header.biBitCount := 8;
    bitmapInfo.Header.biCompression := 0;
    bitmapInfo.Header.biSizeImage := meshWidth * meshHeight;
    bitmapInfo.Header.biXPelsPerMeter := 0;
    bitmapInfo.Header.biYPelsPerMeter := 0;
    bitmapInfo.Header.biClrUsed := 0;
    bitmapInfo.Header.biClrImportant := 0;
    for x := 0 to 255 do
      with bitmapInfo.Colors[x] do begin
        rgbBlue := x;
        rgbGreen := x;
        rgbRed := x;
        rgbReserved := 0;
      end; (with)
    Error := FileWrite(xMeshFile, @bitmapInfo, SizeOf(bitmapInfo), 0);
    if Error <> NoErr then writeln('3 Error writing Output File1 ', Error);
    Error := FileWrite(yMeshFile, @bitmapInfo, SizeOf(bitmapInfo), 0);
    if Error <> NoErr then writeln('4 Error writing Output File2 ', Error);
    rowByteCount := Align32(meshWidth);
    address := 0;
    for y := 0 to meshHeight-1 do begin
      for x := 0 to meshWidth-1 do begin
        xBuffer[x] :=

```

```

        Round(255 * PReal(OffsetPointer(xMeshData, address))^ / meshWidth);
    yBuffer[x] :=
        Round(255 * PReal(OffsetPointer(yMeshData, address))^ / meshHeight);
    Inc(address, RSize);
end; (for x)
Error := FileWrite(xMeshFile, @xBuffer, rowByteCount, 0);
if Error <> NoErr then writeln('Error writing Output File1');
Error := FileWrite(yMeshFile, @yBuffer, rowByteCount, 0);
if Error <> NoErr then writeln('Error writing Output File2');
end; (for y)
Close(file(xMeshFile));
Close(file(yMeshFile));
GlobalUnlock(xMeshH);
GlobalUnlock(yMeshH);
end (if mesh grids defined)
else Error := memoryErr;
SaveOutputmesh := Error;
end; (SaveOutputMesh)
(-----WarpSequence-----)
Function WarpSequence(InImageH, OutImageH : THandle;
    ImageCount : Byte;
    TrajectoryH : THandle) : ErrorCode;
(-Beginning from the InImageH produce an animated sequence of ImageCount images given the
trajectoryH data.
)
label
    WarpSequenceExit;
var
    meshSize          (Input and output image mesh dimensions, x,y)
    xMeshInH, yMeshInH (Handles for input image x and y meshes)
    xMeshOutH, yMeshOutH (Handles for output image x and y meshes)
    : MeshHandle;
    xMeshIn, yMeshIn,
    xMeshOut, yMeshOut (Input image x and y meshes)
    : MeshPtr;
    Image              (General image pointer)
    : ImagePtr;
    Trajectory          (Trajectory of objects to move)
    : TrajectoryArrayPtr;
    ImageNum,
    rowByteCount        (Counter for images)
    (Number of bytes in a mesh row)
    : Word;
    x, y,
    address              (Row and column counters)
    (Element address)
    : Longint;
    Centroid,
    InPoint              (Centroid of current object)
    (Input point to be transformed)
    : TPoint;
    OutPoint            (Output transformed point)
    : RPoint;
    Error               (General error)
    : ErrorCode;
(-----InitialiseOutputMesh-----)
Procedure InitialiseOutputMesh;
(-Initialise output mesh grid to a regular 1-pixel spacing.)
var
    x, y,
    address              (Counters)
    (Element address)
    : longint;
    xData, yData
    : PReal;
begin (InitialiseOutputMesh)
    if (xMeshInH <> 0) and (yMeshInH <> 0) then begin (initialise the input meshes)
        xData := @xMeshIn^.data;
        yData := @yMeshIn^.data;
        address := 0;
        for y := 0 to meshSize.y-1 do begin
            for x := 0 to meshSize.x-1 do begin
                PReal(OffsetPointer(xData, address))^ := x;
                PReal(OffsetPointer(yData, address))^ := y;
                Inc(address, RSize);
            end; (for x)
        end; (for y)
    end; (if input mesh grids defined)
    xData := @xMeshOut^.data;
    yData := @yMeshOut^.data;
    address := 0;
    for y := 0 to meshSize.y-1 do begin (Initialise output meshes)
        for x := 0 to meshSize.x-1 do begin
            PReal(OffsetPointer(xData, address))^ := x;
            PReal(OffsetPointer(yData, address))^ := y;
            Inc(address, RSize);
        end; (for x)
    end; (for y)
end; (InitialiseOutputMesh)
(-----AddTriPatchesToImage-----)

```

```

Function AddTriPatchesToImage : ErrorCode;
  (Draws the TriPatch outlines on outImageH.)
  const
    Colour = 127; (greyscale index of tripatch line colour)
  var
    address,           (Address in image of current element)
    imageWidth         (Width of the image)
    OutImage           : longint;
    OutImage           : ImagePtr;
    OutImageData       (Pointer to output image data)
    OutImageData       : PByte;
    Error              (General error)
    Error              : ErrorCode;
begin
  Error := NoErr;
  writeln(' Adding TriPatch outlines to image');
  OutImage := GlobalLock(OutImageH);
  imageWidth := OutImage^.header.size.x;
  OutImageData := @OutImage^.data;
  for triPatchNum := 0 to triPatchCount-1 do begin
    with TriPatches^[triPatchNum] do begin
      (Interpolate all the outline points of the triPatch)
      outlinePointNum := 0;
      LineDDA(Round(opt[0].x), Round(opt[0].y),
              Round(opt[1].x), Round(opt[1].y), @LineProc, Nil); (point 0-1)
      LineDDA(Round(opt[1].x), Round(opt[1].y),
              Round(opt[2].x), Round(opt[2].y), @LineProc, Nil); (point 1-2)
      LineDDA(Round(opt[2].x), Round(opt[2].y),
              Round(opt[0].x), Round(opt[0].y), @LineProc, Nil); (point 2-0)
      outlinePointCount := outlinePointNum;
      if outlinePointCount > MaxOutlinePointCount then begin
        writeln('ERROR: Too many TriPatch outline points');
        Error := OutOfRangeErr;
      end; (if)
      if Error = NoErr then begin
        for outlinePointNum := 0 to outlinePointCount-1 do begin
          address := OutlinePoints[outlinePointNum].x
                    + OutlinePoints[outlinePointNum].y * imageWidth;
          PByte(OffsetPointer(OutImageData, address))^ := Colour;
        end; (for outlinePointNum)
      end; (if NoErr)
    end; (with TriPatches)
  end; (for triPatchNum)
  GlobalUnlock(OutImageH);
  AddTriPatchesToImage := Error;
end; (AddTriPatchesToImage)
  (-----SmoothOutputMesh-----)
Function SmoothOutputMesh : ErrorCode;
  (-Smooths the output x and y mesh grids using a 3x3 box filter)
  var
    Error              : ErrorCode;
    x, y               (Counters)
    meshWidth          : longint;
    meshWidth          ( )
    meshWidth          : Word;
    xBuffer1H, xBuffer2H, (Handles to two xMesh line buffers)
    yBuffer1H, yBuffer2H (Handles to two yMesh line buffers)
    THandle            : THandle;
    xBuffer1, xBuffer2, (Pointers to two xMesh line buffers)
    yBuffer1, yBuffer2, (Pointers to two xMesh line buffers)
    tmpBuffer,          (Temporary line buffer)
    xSrc, ySrc          (xMesh and yMesh source pointers)
    xSrc, ySrc          : PRealLine;
begin (SmoothOutputMesh)
  MeshWidth := MeshSize.x;
  Error := CreateHandle(xBuffer1H, MeshWidth, 0);
  if Error = NoErr then Error := CreateHandle(xBuffer2H, MeshWidth, 0);
  Error := CreateHandle(yBuffer1H, MeshWidth, 0);
  if Error = NoErr then Error := CreateHandle(yBuffer2H, MeshWidth, 0);
  if Error = NoErr then begin
    xBuffer1 := GlobalLock(xBuffer1H);
    xBuffer2 := GlobalLock(xBuffer2H);
    yBuffer1 := GlobalLock(yBuffer1H);
    yBuffer2 := GlobalLock(yBuffer2H);
    (Copy first two lines of x and y meshes into buffers)
    xSrc := PRealLine(@xMeshOut^.data);
    ySrc := PRealLine(@yMeshOut^.data);
    for x := 0 to MeshWidth - 1 do begin
      xBuffer1[x] := xSrc[x];
      xBuffer2[x] := xSrc[MeshWidth + x];
      yBuffer1[x] := ySrc[x];
      yBuffer2[x] := ySrc[MeshWidth + x];
    end; (for x)
    (Smooth the meshes)
    for y := 1 to MeshSize.y - 2 do begin
      xSrc := PRealLine(OffsetPointer(@xMeshOut^.data, (y * MeshSize.x) * RSize));

```

```

ySrc := PRealLine(OffsetPointer(@yMeshOut^.data, (y * MeshSize.x) * RSize));
for x := 1 to MeshWidth-2 do begin
  xSrc^[x] := (x average)
    (xBuffer1^[x-1] + xBuffer1^[x] + xBuffer1^[x+1]
    + xBuffer2^[x-1] + xBuffer2^[x] + xBuffer2^[x+1]
    + xSrc^[MeshWidth + x-1] + xSrc^[MeshWidth + x] + xSrc^[MeshWidth + x+1]) / 9;
  ySrc^[x] := (y average)
    (yBuffer1^[x-1] + yBuffer1^[x] + yBuffer1^[x+1]
    + yBuffer2^[x-1] + yBuffer2^[x] + yBuffer2^[x+1]
    + ySrc^[MeshWidth + x-1] + ySrc^[MeshWidth + x] + ySrc^[MeshWidth + x+1]) / 9;
end; (for x)
(Swap buffer pointers)
tmpBuffer := xBuffer1; xBuffer1 := xBuffer2;
xBuffer2 := tmpBuffer; tmpBuffer := yBuffer1;
yBuffer1 := yBuffer2;
yBuffer2 := tmpBuffer;
for x := 0 to MeshWidth-1 do begin (Buffer the next mesh x and y lines)
  xBuffer2^[x] := xSrc^[MeshWidth + x];
  yBuffer2^[x] := ySrc^[MeshWidth + x];
end; (for x)
end; (for y)
(Destroy buffers)
DestroyHandle(xBuffer1H); DestroyHandle(xBuffer2H);
DestroyHandle(yBuffer1H); DestroyHandle(yBuffer2H);
end; (if NoErr)
SmoothOutputMesh := Error;
end; (SmoothOutputMesh)

(-----InterpolateOutputMesh-----)
Function InterpolateOutputMesh : ErrorCode;
(-Interpolates output mesh to adjust for object motion.
  Uses the triPatch coefficients to calculate the new internal patch coords.
)
var
  count,
  lineCount,          (Number of scanlines in the current triPatch)
  meshWidth,          (Width of the images)
  meshHeight,         (Height of the images)
  xMin, xMax          (Min and max x coords of a triPatch scanline)
  : Word;
  xNew, yNew          (The new x and y mesh coordinates)
  : Real;
  x, y,               (x and y counters)
  address             (Address of cureent mesh element)
  : Longint;
begin (InterpolateOutputMesh)
  meshWidth := MeshSize.x;
  meshHeight := MeshSize.y;
  for triPatchNum := 0 to triPatchCount-1 do begin
    with TriPatches^[triPatchNum] do begin
      (Interpolate all the outline points of the triPatch)
      outlinePointNum := 0;
      LineDDA(ipt[0].x, ipt[0].y, ipt[1].x, ipt[1].y, @LineProc, Nil); (point 0-1)
      LineDDA(ipt[1].x, ipt[1].y, ipt[2].x, ipt[2].y, @LineProc, Nil); (point 1-2)
      LineDDA(ipt[2].x, ipt[2].y, ipt[0].x, ipt[0].y, @LineProc, Nil); (point 2-0)
      outlinePointCount := outlinePointNum;
      if outlinePointCount > MaxOutlinePointCount then begin
        writeln('ERROR: Too many TriPatch outline points');
        Error := OutOfRangeErr;
      end; (if)
      if Error = NoErr then begin
        count := 0;
        (Find the start and end xCoord of each scanline in the current TriPatch)
        for y := ipt[2].y to ipt[0].y do begin
          xMin := MaxInt;
          xMax := 0;
          for outlinePointNum := 0 to outlinePointCount-1 do begin
            if OutlinePoints[outlinePointNum].y = y then begin
              (On the same horizontal scanline)
              x := OutlinePoints[outlinePointNum].x;
              if x > xMax then xMax := x;
              if x < xMin then xMin := x;
            end; (if on the same horizontal scanline)
          end; (for outlinePointNum)
          TriPatchScanLines[count].y := y;
          TriPatchScanLines[count].xStart := xMin;
          TriPatchScanLines[count].xEnd := xMax;
          Inc(count);
        end; (for y)
        lineCount := count;
        (Interpolate the new internal coords of the current TriPatch)
        for count := 0 to lineCount-1 do begin
          y := TriPatchScanLines[count].y;
          address := (TriPatchScanLines[count].xStart + y * meshWidth) * RSize;
          for x := TriPatchScanLines[count].xStart
            to TriPatchScanLines[count].xEnd do begin
            PReal(OffsetPointer(@xMeshOut^.data, address))^

```

```

:= -(ax * x + bx * y + dx)/cx;
PReal(OffsetPointer(@yMeshOut^.data, address))^
:= -(ay * x + by * y + dy)/cy;
Inc(address, RSize); {increment to next element}
end; {for x}
end; {for count}
end; {if NoErr}
end; {with TriPatches}
end; {for triPatchNum}
InterpolateOutputMesh := Error;
end; {InterpolateOutputMesh}
{-----Main: WarpSequence-----}
begin {MAIN: WarpSequence}
  {Get image dimensions}
  Image := GlobalLock(InImageH);
  MeshSize := Image^.header.Size;
  GlobalUnlock(InImageH);
  {Initialise handles}
  xMeshInH := 0; yMeshInH := 0;
  xMeshOutH := 0; yMeshOutH := 0;
  {Create output image meshes}
  Error := CreateRealImage(xMeshOutH, MeshSize.x, MeshSize.y);
  if Error <> NoErr then goto WarpSequenceExit;
  xMeshOut := GlobalLock(xMeshOutH);
  Error := CreateRealImage(yMeshOutH, MeshSize.x, MeshSize.y);
  if Error <> NoErr then goto WarpSequenceExit;
  yMeshOut := GlobalLock(yMeshOutH);
  Trajectory := GlobalLock(TrajectoryH);
  for imageNum := 0 to ImageCount-1 do begin
    for triPatchNum := 0 to triPatchCount-1 do begin
      with TriPatches^[triPatchNum] do begin {Update current triPatch output points}
        for pointNum := 0 to 2 do begin
          if apt[pointNum].objectNum > 0 then {Move the object point}
            TransformPoint(ipt[pointNum], apt[pointNum].centroid,
              Trajectory^[imageNum, apt[pointNum].objectNum - 1], opt[pointNum]);
        end; {for pointNum}
      end; {with TriPatches}
      {Determine the coefficients for current TriPatch}
      Error := FitTriangularPatch(TriPatches^[triPatchNum]);
      if Error <> NoErr then begin
        writeln('Error fitting triangles ', Error); readln;
      end; {if Err}
    end; {for triPatchNum}
    InitialiseOutputMesh; {Initialise meshes to a regular grid}
    Error := InterpolateOutputMesh; {Interpolate output mesh to adjust for object motion}
    if Error <> NoErr then begin
      writeln('Error Interpolating meshes: ', Error);
      readln;
    end; {if Err}
    {Force mesh boundaries to image border. Prevents out of range errors during warping.}
    rowByteCount := longint(meshSize.x) * RSize;
    address := 0;
    for y := 0 to meshSize.y-1 do begin {Left vertical}
      PReal(OffsetPointer(@xMeshOut^.data, address))^ := 0;
      PReal(OffsetPointer(@yMeshOut^.data, address))^ := y;
      Inc(address, rowByteCount);
    end; {for y}
    address := (meshSize.x-1) * RSize;
    for y := 0 to meshSize.y-1 do begin {Right vertical}
      PReal(OffsetPointer(@xMeshOut^.data, address))^ := meshSize.x-1;
      PReal(OffsetPointer(@yMeshOut^.data, address))^ := y;
      Inc(address, rowByteCount);
    end; {for y}
    address := 0;
    for x := 0 to meshSize.x-1 do begin {Bottom horizontal}
      PReal(OffsetPointer(@xMeshOut^.data, address))^ := x;
      PReal(OffsetPointer(@yMeshOut^.data, address))^ := 0;
      Inc(address, RSize);
    end; {for x}
    address := longint(meshSize.y - 1) * meshSize.x * RSize;
    for x := 0 to meshSize.x-1 do begin {Top horizontal}
      PReal(OffsetPointer(@xMeshOut^.data, address))^ := x;
      PReal(OffsetPointer(@yMeshOut^.data, address))^ := meshSize.y-1;
      Inc(address, RSize);
    end; {for x}
    if SaveOutputMeshes then begin
      Error := SaveOutputMesh(xMeshOutH, yMeshOutH, imageNum);
      if Error <> NoErr then writeln('Error Saving output mesh grids');
    end; {if SaveOutputMeshes}
    Error := SmoothOutputMesh; {Box 3x3 filter output meshes}
    if Error <> NoErr then begin
      writeln('Error smoothing output meshes: ', Error); readln;
    end; {if Err}
    GlobalUnlock(xMeshOutH);
    GlobalUnlock(yMeshOutH);
    writeln('Mesh Warping image: ', imageNum);
    Error := MeshWarpImage(InImageH, OutImageH,

```



```

                                xMeshInH, yMeshInH, xMeshOutH, yMeshOutH, False);
case Error of
  MemoryErr      : writeln('Insufficient memory to warp image');
  DivByZeroErr   : writeln('Divide-by-zero: an image may NOT be warped correctly');
  OutOfRangeErr  : writeln('Out-of-range:, an image has not been created');
end;
if AddTriPatchesToImages then begin
  Error := AddTriPatchesToImage;
  if Error <> NoErr then writeln('Error adding TriPatches to image');
end; {AddTriPatchesToImages}
if (Error = NoErr) then Error := CompressImage(OutImageH, InterImageH);
if Error <> NoErr then begin
  writeln('Error Compressing image ', imageNum);
  goto WarpSequenceExit;
end {if Err}
else begin
  Error := FileOpen(outFile, @AnimationFileName, Version, Nil, False);
  if Error <> NoErr then writeln('Error opening output file: ', Error)
  else Error := WriteImage(InterImageH, outFile, ImageDataID + imageNum);
  if Error <> NoErr then writeln('writing image ', imageNum);
  FileClose(outFile, Error);
  DestroyImage(InterImageH);
end; {else NoErr}
DestroyImage(OutImageH);
end; {for ImageNum}
WarpSequenceExit:
GlobalUnlock(TrajectoryH);
DestroyRealImage(xMeshOutH); DestroyRealImage(yMeshOutH);
WarpSequence := Error;
end; {WarpSequence}
(=====)
begin {MAIN Program: CineWarp}
  if ParamCount > 0 then begin
    fileName := ParamStr(1); {Get the complete trajectory specification file name}
    StrPCopy(Path, fileName);
    FileSplit(Path, Directory, Name, Extension);
    LStrCpy(TrajectoryFileName, Name);
  end {if ParamCount}
  else StrCopy(TrajectoryFileName, DefaultTrajectoryFileName);
  ScreenSize.x := 60;
  ScreenSize.y := 40;
  if Error = NoErr then Error := CreateHandle(TriPatchesH, Sizeof(TriPatchArray), 0);
  if Error = NoErr then TriPatches := GlobalLock(TriPatchesH);
  {-Read trajectory file and form object trajectory}
  StrCopy(outFileName, TrajectoryFileName); {Form the file name}
  LStrCat(outFileName, TrajectorySpecFileExtn); {add the extension}
  Error := FormTrajectory(outFileName);
  if Error = NoErr then Trajectory := GlobalLock(TrajectoryH)
  else writeln('Error forming trajectory ', Error);
  {-Open the reference patient file and get the info }
  if Error = NoErr then
    Error := FileOpen(inFile, PathPtr(StringPtr(InFileName)), Version, Nil, True);
  if Error = NoErr then
    Error := FileReadData(inFile, docInfoDataID, @DocumentInfo, SizeOf(DocInfoRec), Nil, 0);
  if Error = NoErr then begin
    ImageSize.x := DocumentInfo.imageSize.x;
    rowByteCount := Align32(ImageSize.x);
    ImageSize.y := DocumentInfo.imageSize.y;
    writeln('Image Size: ', ImageSize.x, 'x', ImageSize.y);
  end; {if NoErr}
  {-Get patient information}
  if Error = NoErr then Error := ReadPatientID(inFile, PatData);
  if Error = NoErr then Error := ReadDocID(inFile, docID);
  if Error = NoErr then
    Error := FileReadData(inFile, OperatorDataID, @OptData, Sizeof(OperatorStr), nil, 0);
  if Error = NoErr then Error := CreatePointInfo(PointInfoH, 0);
  if Error = NoErr then Error := ReadPointInfo(PointInfoH, InFile, PointInfoDataID);
  if Error <> NoErr then FileClose(inFile, Error)
  else ObjectCount := GetPointGroupCount(pointInfoH);
  {Create images}
  if Error = NoErr then Error := CreateImage(InterImageH, rowByteCount, ImageSize.y);
  if Error = NoErr then InterImage := GlobalLock(InterImageH);
  if Error = NoErr then {Get centre image}
    Error := ReadImage(InterImageH, inFile, ImageDataID + (DocumentInfo.ImageCount div 2));
  if Error = NoErr then FileClose(inFile, Error);
  if Error = NoErr then
    Error := DecompressImage(InterImageH, InImageH); {InImage created by decompress}
  {-Find first available document number}
  if Error = NoErr then begin
    fileNum := 0;
    while (Error <> abortErr) and (fileNum <= 99999) do begin
      LStrCat(LStrCat(LStrCpy(@outFileName, CMXDocPath), '\doc%05lu', '.cmx');
      WVSPrintF(@path, @outFileName, fileNum);
      Inc(fileNum);
      FindFirst(@path, faReadOnly, searchInfo);
      error := BPErr(DosError);
    end; {while}
  end;
end;

```

```

if (fileNum > 99999) then writeln('File count exceeded ', Error)
else begin
  StrCopy(outFileName, path);
  Error := NoErr;
end; (else)
end; (if NoErr)
StrCopy(AnimationFileName, outFileName);
(Create and open the output file )
if Error = NoErr then Error := FileCreate(@path, cmxFileChar, Version);
if Error = NoErr then
  Error := FileOpen(OutFile, @path, Version, Nil, False);
if Error = NoErr then Error := WritePatientID(OutFile, @PatData, False);
LStrCat(LStrCat(DocID.name, ' '), TrajectoryFileName);
if Error = NoErr then Error := WriteDocID(OutFile, @docID, true);
if Error = NoErr then begin
  DocumentInfo.imageNo := imageCount div 2; (current image the centre image)
  DocumentInfo.imageCount := imageCount; (Number of images)
end; (if NoErr)
if Error = NoErr then (Add DocumentInfo to new file)
  Error := FileWriteData(OutFile, DocInfoDataID, @DocumentInfo, SizeOf(DocInfoRec), False, 0);
if Error = NoErr then (Add operator info to new file)
  Error :=
    FileWriteData(OutFile, OperatorDataID, @OptData, SizeOf(OperatorStr), False, 0);
DestroyImage(InterImageH);
(Create and write pointInfo to out file )
if Error = NoErr then Error := CreatePointInfo(PointInfo2H, DocumentInfo.imageCount);
(Create an ObjectIndices Array to hold the control point...)
if Error = NoErr then
  Error := CreateHandle(ObjectIndicesH, MaxInt div SizeOf(ObjectIndexRec), 0);
if Error = NoErr then ObjectIndices := GlobalLock(ObjectIndicesH);
(Create an array to hold the control points)
if Error = NoErr then
  Error := CreateHandle(ControlPointsH, MaxInt div SizeOf(ObjectIndexRec), 0);
if Error = NoErr then ControlPoints := GlobalLock(ControlPointsH);
(-Assemble control points from objects)
if Error = NoErr then Error := FormControlPoints;
if Error <> NoErr then begin
  writeln('Error Forming Control Points ', Error);
  goto CineWarpExit;
end;
GlobalUnlock(ControlPointsH);
GlobalUnlock(TriIndicesH);
(-Form the triangular tessellation of the control points)
if Error = NoErr then
  Error := Triangulate(ControlPointsH, controlPointCount, TriIndicesH, triPatchCount);
if Error <> NoErr then begin
  writeln('Error Triangulation ', Error);
  goto CineWarpExit;
end;
TriIndices := GlobalLock(TriIndicesH);
if Error = NoErr then begin
  (Copy pointGroups to output file structure )
  for PointGroupNum := 0 to objectCount-1 do begin
    if ObjectInclude[pointGroupNum] then begin (add the object)
      GetPointGroupName(pointInfoH, PointGroupNum, PointGroupName);
      Error := AddPointGroup(pointInfo2H, @PointGroupName);
      if Error <> NoErr then writeln('Error Adding new point group');
      EdgePointCount := GetEdgePointCount(pointInfoH, @PointGroupName);
      if EdgePointCount > MaxEdgePointCount then writeln('Error MaxEdgePoints exceeded');
      for EdgePointNum := 0 to EdgePointCount-1 do begin
        GetPoint(pointInfoH, @PointGroupName, EdgePointNum, Point);
        Error := AddPoint(pointInfo2H, @PointGroupName, Point);
        if Error <> NoErr then writeln('Error Adding new point');
      end; (for EdgePointNo)
    end; (if ObjectInclude)
  end; (for PointGroupNum)
  (Add in the triangulation groups)
  for triPatchNum := 0 to triPatchCount-1 do begin
    if AddTriPatchesToPointGroups then begin
      Str(triPatchNum, s);
      StrCopy(pointGroupName, StrPCopy(ss, triPatchStr + s));
      Error := AddPointGroup(pointInfo2H, @pointGroupName);
      if Error <> NoErr then writeln('Error adding point group');
    end; (if AddTriPatchesToPointGroups)
    for pointNum := 0 to 2 do begin (Extract each point and its info)
      Point.x := Round(ControlPoints^[TriIndices^[triPatchNum, pointNum], 0]);
      Point.y := Round(ControlPoints^[TriIndices^[triPatchNum, pointNum], 1]);
      TriPatches^[triPatchNum].ipt[pointNum] := point;
      TriPatches^[triPatchNum].opt[pointNum].x := point.x;
      TriPatches^[triPatchNum].opt[pointNum].y := point.y;
      TriPatches^[triPatchNum].apt[pointNum]
        := ObjectIndices^[TriIndices^[triPatchNum, pointNum]];
      if AddTriPatchesToPointGroups then begin
        Error := AddPoint(pointInfo2H, @pointGroupName, Point);
        if Error <> NoErr then writeln('Error adding TriPoint');
      end; (if AddTriPatchesToPointGroups)
    end; (for pointNum)
  end; (for triPatchNum)
end; (for pointNum)

```

```

    (Reorder the tri-points in descending order of y)
  repeat (Sorted)
    Sorted := True;
    for pointNum := 0 to 1 do begin (Extract each point and its info)
      with TriPatches^[triPatchNum] do begin
        if ipt[pointNum+1].y > ipt[pointNum].y then begin
          (Swap order of the tri-point data)
          point := ipt[pointNum];
          ipt[pointNum] := ipt[pointNum+1];
          ipt[pointNum+1] := point;
          rePoint := opt[pointNum];
          opt[pointNum] := opt[pointNum+1];
          opt[pointNum+1] := rePoint;
          obPoint := apt[pointNum];
          apt[pointNum] := apt[pointNum+1];
          apt[pointNum+1] := obPoint;
          Sorted := False;
        end; (if not in order)
      end; (with TriPatches)
    end; (for pointNum)
  until Sorted;
end; (for triPatchNum)
DestroyHandle(ControlPointsH);
DestroyHandle(ObjectIndicesH);
DestroyHandle(TriIndicesH);
end; (if NoErr)
if Error = NoErr then Error := WritePointInfo(PointInfo2H, OutFile, PointInfoDataID);
if Error <> NoErr then begin
  writeln('Error writing pointinfo ', Error);
  FileClose(outFile, Error);
  goto CineWarpExit;
end; (Error)
DestroyPointInfo(PointInfo2H);
(-Output trajectory to data file)
StrCopy(outFileName, TrajectoryFileName); (Form the file name)
LStrCat(outFileName, TrajectoryDataFileExtn); (add the extension)
Assign(dataFile, outFileName);
Rewrite(dataFile);
for objectNum := 0 to GetPointGroupCount(pointInfoH)-1 do begin
  if ObjectInclude[objectNum] then begin
    GetPointGroupName(pointInfoH, objectNum, PointGroupName);
    write(dataFile, ' ', PointGroupName, ', ');
  end; (if ObjectInclude)
end; (for objectNum)
writeln(dataFile);
for objectNum := 0 to GetPointGroupCount(pointInfoH)-1 do begin
  if ObjectInclude[objectNum] then
    write(dataFile, 'x(pixels)', y(pixels)', 'theta(degrees)', ');
end; (for objectNum)
writeln(dataFile);
for imageNum := 0 to imageCount-1 do begin
  for objectNum := 0 to GetPointGroupCount(pointInfoH)-1 do begin
    if ObjectInclude[objectNum] then
      with Trajectory^[imageNum, objectNum] do
        write(dataFile, 'x:2:1', 'y:2:1', '(', theta * 180/pi):2:1, ');
    end; (for objectNum)
  end; (for imageNum)
  Close(dataFile);
  DestroyPointInfo(PointInfoH);
  (Add comment to the output file)
  if Error = NoErr then Error := FileWriteData(TFileRec(outFile), docCommentDataID,
    @FileComment, Strlen(FileComment), False, 0);
  FileClose(outFile, Error);
  if Error <> NoErr then writeln('Error closing file');
  (Start Image Warping, InImage contains the reference image)
  GlobalUnlock(InImageH);
  GlobalUnlock(OutImageH);
  Error := WarpSequence(InImageH, OutImageH, imageCount, TrajectoryH);
  if Error <> NoErr then writeln('Error warping ');
CineWarpExit: (-EXIT Block-)
  if Error <> NoErr then begin
    writeln('ERROR: ', Error); readln;
  end;
  DestroyImage(InImageH); DestroyImage(OutImageH);
  DestroyHandle(TrajectoryH);
  DestroyHandle(TriPatchesH);
  writeln('FINISHED');
  DoneWinCrt;
(=====)
end. (Program: CineWarp)

```

C.10 Motion-Tracking

The following Pascal unit 'Tracking' implements the motion-tracking algorithm developed through chapters 4 to 6 and then described and tested in detail in chapter 7. An important part of the algorithm is the feature orientation calculation. This is based on the *Fold-and-Match* algorithm described in section 7.2.2 of chapter 7 and contained in the *TrackFirst* function that begins at the bottom of page C-84.

```

unit Tracking;

interface
(=====)
uses
    PMXProcs, PMXTypes, PMXUtils, WinProcs, WinTypes,
    Dialogs, Images, Globals, Maths, Points, Resource, Utils;
Function BeginTrack(window : HWND) : ErrorCode;
Procedure EndTrack(window : HWND);
Function Track(window : HWND) : ErrorCode;

implementation
(=====IMPLEMENTATION=====)
type
    CentreLInfoPtr = ^CentreLInfoRec;
    CentreLInfoRec = record
        xSum : Longint;
        ySum : Longint;
        count : Word;
    end;

    TrackDInfoPtr = ^TrackDInfoRec;
    TrackDInfoRec = record
        pointInfoChanged : Boolean;
        trackOptionsChanged : Boolean;
        window : HWND;
    end;

    TrackingInfoPtr = ^TrackingInfoRec;
    TrackingInfoRec = record
        filter : Boolean;
        unused : Boolean;
        filterImageInfo : FilterImageInfoRec;
        destImage : ImageHandle;
        srcImage : ImageHandle;
        imageNoOffset : Integer;
        pointList : PointListHandle;
        srcPointWindowList : PointWindowListHandle;
        bestMatch : Real;
        orientation : Real;
        centre : RealPoint;
        destPointList : RealPointListHandle;
        pointList1 : RealPointListHandle;
        pointList2 : RealPointListHandle;
        trackingPointList : TrackingPointListHandle;
        destImageFilter : Word;
        destImageNo : Word;
        maxPointCount : Word;
        pointCount : Word;
        pointGroupIndex : Word;
        pointGroupNo : Word;
        pointWindowRecSize : Word;
        rotCount : Word;
        rotNo : Word;
        srcImageFilter : Word;
        srcImageNo : Word;
        startPointGroupNo : Word;
    end;

    Procedure CentreLineProc(x, y : Integer; lInfo : CentreLInfoPtr); export; forward;
    Function TrackFirst(window : HWND; wInfo : DocWInfoPtr;
        trackingInfo : TrackingInfoPtr) : ErrorCode; forward;
    Function TrackNext(window : HWND; wInfo : DocWInfoPtr;
        trackingInfo : TrackingInfoPtr) : ErrorCode; forward;

    (-----BeginTrack-----)
    Function BeginTrack(window : HWND) : ErrorCode;
    {Prepares everything for the start of the motion-tracking procedure}
    var
        wInfo : DocWInfoPtr;
        error : ErrorCode;
        dInfo : TrackDInfoRec;

```

```

trackingInfo : TrackingInfoPtr;
pointCount : Word;
pointGroupCount : Word;
pointGroupNo : Word;

begin (BeginTrack)
error := SendMessage(window, getDocDataMsg, pointInfoDataID, Longint(nil));
if error = noErr then begin
dInfo.window := window;
case DialogBoxParam(hInstance, PChar(trackDlogID), applWindow,
@TrackDialogProc, Longint(@dInfo)) of
-1 :
error := dialogErr;
id_Cancel :
error := abortErr;
end; (case)
end; (if)
wInfo := DocWInfoPtr(SendMessage(window, lockWInfoMsg, 1, 0));
if error = noErr then
error := CreateHandle(wInfo^.trackingInfo, SizeOf(trackingInfoRec), gmem_ZeroInit);
if error = noErr then begin
trackingInfo := GlobalLock(wInfo^.trackingInfo);
trackingInfo^.filter := true;
trackingInfo^.startPointGroupNo := $FFFF;
pointGroupCount := 0;
for pointGroupNo := 0 to GetPointGroupCount(wInfo^.pointInfo) - 1 do
if not PointGroupMarked(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo), trackMark)
then begin
pointCount := GetEdgePointCount(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo));
if pointCount > trackingInfo^.maxPointCount then
trackingInfo^.maxPointCount := pointCount;
if pointCount > 0 then begin
if trackingInfo^.startPointGroupNo = $FFFF then
trackingInfo^.startPointGroupNo := pointGroupNo;
Inc(pointGroupCount);
end; (if)
end; (if)
error := CreateHandle(trackingInfo^.pointList,
SizeOf(TPoint) * (trackingInfo^.maxPointCount + 1), 0);
if error = noErr then
error := CreateHandle(trackingInfo^.pointList1,
SizeOf(RealPoint) * (trackingInfo^.maxPointCount + 2), 0);
if error = noErr then
error := CreateHandle(trackingInfo^.pointList2,
SizeOf(RealPoint) * (trackingInfo^.maxPointCount + 2), 0);
if error = noErr then begin
SendMessage(wInfo^.messageControl, setMaxMsg,
wInfo^.imageCount * pointGroupCount - 1, 0);
trackingInfo^.destImageNo := wInfo^.pointImageNo;
trackingInfo^.srcImageNo := wInfo^.pointImageNo;
trackingInfo^.pointGroupNo := trackingInfo^.startPointGroupNo;
wInfo^.showPoints := wInfo^.trackOptions.display;
EnableWindow(wInfo^.stopButton, true);
end; (if)
if error = noErr then
error := PostBackgroundMessage(window, trackMsg, 0, 0);
if error <> noErr then
EndTrack(window);
end; (if)
SendMessage(window, lockWInfoMsg, 0, 0);
BeginTrack := error;
end; (BeginTrack)
(-----CentreLineProc-----)
Procedure CentreLineProc(x, y : Integer; lInfo : CentreLInfoPtr);
{ Used in the oritation calculation to find the axis line
}
begin (CentreLineProc)
Inc(lInfo^.xSum, x);
Inc(lInfo^.ySum, y);
Inc(lInfo^.count);
end; (CentreLineProc)
(-----EndTrack-----)
Procedure EndTrack(window : HWnd);
{ Called at the conclusion of the tracking procedure to tidyup and store the results.
}
var
wInfo : DocWInfoPtr;
image : ImageHandle;
trackingInfo : TrackingInfoPtr;
begin (EndTrack)
RemoveBackgroundMessage(window, trackMsg);
wInfo := DocWInfoPtr(SendMessage(window, lockWInfoMsg, 1, 0));
if wInfo^.trackingInfo <> 0 then begin
trackingInfo := GlobalLock(wInfo^.trackingInfo);
SetDiscard(trackingInfo^.destImage, true);
SetDiscard(trackingInfo^.srcImage, true);

```

```

DestroyHandle(trackingInfo^.pointList);
DestroyHandle(trackingInfo^.srcPointWindowList);
DestroyHandle(trackingInfo^.destPointList);
DestroyHandle(trackingInfo^.pointList1);
DestroyHandle(trackingInfo^.pointList2);
DestroyHandle(trackingInfo^.trackingPointList);
if (trackingInfo^.destImageFilter <> noFilter)
or (trackingInfo^.srcImageFilter <> noFilter) then begin
    image := EndFilterImage(trackingInfo^.filterImageInfo);
    DestroyImage(image);
end; (if)
SendMessage(wInfo^.messageControl, setMsgPosMsg, 0, 0);
end; (if)
DestroyHandle(wInfo^.trackingInfo);
SendMessage(window, lockWInfoMsg, 0, 0);
end; (EndTrack)
{-----Track-----}
Function Track(window : HWnd) : ErrorCode;
{Control routine for running the tracking algorithm.}
var
    wInfo : DocWInfoPtr;
    error : ErrorCode;
    trackingInfo : TrackingInfoPtr;
begin (Track)
    error := noErr;
    wInfo := DocWInfoPtr(SendMessage(window, lockWInfoMsg, 1, 0));
    trackingInfo := GlobalLock(wInfo^.trackingInfo);
    if trackingInfo^.destImageNo = wInfo^.imageCount then
        EndTrack(window)
    else begin
        if trackingInfo^.destImageNo = trackingInfo^.srcImageNo then
            error := TrackFirst(window, wInfo, trackingInfo)
        else
            error := TrackNext(window, wInfo, trackingInfo);
            if (error = noErr) and (wInfo^.trackingInfo <> 0) then
                error := PostBackgroundMessage(window, trackMsg, 0, 0);
            if error <> noErr then
                EndTrack(window);
                ErrorAlert(error, nil);
            end; (else)
        if wInfo^.trackingInfo <> 0 then
            GlobalUnlock(wInfo^.trackingInfo);
            SendMessage(window, lockWInfoMsg, 0, 0);
            Track := error;
        end; (Track)
    end;
{-----TrackFilter-----}
Function TrackFilter(window : HWnd; var trackingInfo : TrackingInfoRec; imageNo : Word;
    var image : ImageHandle; var imageFilter : Word) : ErrorCode;
{-Control routine for filtering the images using either a median filter during tracking
or a gradient enhancement filter for visualisation.}
var
    error : ErrorCode;
    filteredImage : ImageHandle;
begin (TrackFilter)
    error := noErr;
    if image = 0 then begin
        error := SendMessage(window, getDocDataMsg, imageDataID + imageNo, Longint(@image));
        if (error = noErr) and trackingInfo.filter then begin
            Inc(imageNo);
            SendMessage(window, setMsgMsg, filterImageStrID, Longint(@imageNo));
            imageFilter := rankFilter;
            error := BeginFilterImage(trackingInfo.filterImageInfo, image, rankFilter, 5);
            end; (if)
        end; (if)
    else if FilterImage(trackingInfo.filterImageInfo) then begin
        filteredImage := EndFilterImage(trackingInfo.filterImageInfo);
        if imageFilter = rankFilter then begin
            SetDiscard(image, true);
            image := filteredImage;
            imageFilter := gradientFilter;
            error := BeginFilterImage(trackingInfo.filterImageInfo, image, gradientFilter, 2);
            end; (if)
        else begin
            DestroyImage(image);
            image := filteredImage;
            imageFilter := noFilter;
            end; (else)
        end; (if)
    TrackFilter := error;
end; (TrackFilter)
{-----TrackFirst-----}
Function TrackFirst(window : HWnd; wInfo : DocWInfoPtr;
    trackingInfo : TrackingInfoPtr) : ErrorCode;
{-Initialise the start of the tracking procedure and includes the the feature orientation
calculation.}

```

```

var
  error : ErrorCode;
  pointList : PointListPtr;
  orientation : Real;
  centre : RealPoint;
  pointList1 : RealPointListPtr;
  pointList2 : RealPointListPtr;
  pointCount : Word;
  pointGroupNo : Word;
  -----CalcCentre-----
Procedure CalcCentre;
  ( Calculate current point group centroid )
  var
    centreLInfo : CentreLInfoRec;
    pointNo : Word;
  begin (CalcCentre)
    if pointCount = 1 then begin
      GetPoint(wInfo^.pointInfo, PointGroupNamePtr(trackingInfo^.pointGroupNo),
        0, pointList^[0]);
      PointToRealPoint(pointList^[0], centre);
    end (if)
    else begin
      GetPoint(wInfo^.pointInfo, PointGroupNamePtr(trackingInfo^.pointGroupNo),
        0, pointList^[0]);
      centreLInfo.xSum := 0;
      centreLInfo.ySum := 0;
      centreLInfo.count := 0;
      for pointNo := 1 to pointCount do begin
        GetPoint(wInfo^.pointInfo, PointGroupNamePtr(trackingInfo^.pointGroupNo),
          pointNo mod pointCount, pointList^[pointNo]);
        LineDDA(pointList^[pointNo - 1].x, pointList^[pointNo - 1].y,
          pointList^[pointNo].x, pointList^[pointNo].y,
          @CentreLineProc, @centreLInfo);
        end; (for)
      centre.x := centreLInfo.xSum / centreLInfo.count;
      centre.y := centreLInfo.ySum / centreLInfo.count;
    end; (else)
  end; (CalcCentre)
  -----CalcOrientation-----
Procedure CalcOrientation;
  ( Calculate current point group orientation to the horizontal )
  var
    angle : Real;
    currentArea : Real;
    currentLength : Real;
    lastArea : Real;
    maxAngle : Real;
    maxLength : Real;
    minAngle : Real;
    nextArea : Real;
    stepAngle : Real;
    direction : RealPoint;
    endPoint : RealPoint;
    startPoint : RealPoint;
    i : Word;
    j : Word;
    pointList1Count : Word;
    pointList2Count : Word;

  Procedure CalcArea;
  var
    distance : Real;
    maxDistance1 : Real;
    maxDistance2 : Real;
    minDistance1 : Real;
    minDistance2 : Real;
    linePoint : RealPoint;
    point : RealPoint;
    vector : RealPoint;
    i, j : Word;
  begin (CalcArea)
    nextArea := 0;
    linePoint := startPoint;
    if RealEqualZero(@direction.x) then
      i := Round((endPoint.y - startPoint.y) / direction.y)
    else
      i := Round((endPoint.x - startPoint.x) / direction.x);
    vector.x := -direction.y;
    vector.y := direction.x;
    for i := 0 to i do begin
      minDistance1 := maxReal;
      maxDistance1 := -maxReal;
      for j := 0 to pointList1Count - 2 do begin
        point := linePoint;
        if LineIntersectsSegment(point, @vector, @pointList1^[j], @pointList1^[j + 1])
        then begin
          LineToLength(@linePoint, @point, distance);
          if distance < minDistance1 then

```

```

        minDistance1 := distance;
        if distance > maxDistance1 then
            maxDistance1 := distance;
        end; {if}
    end; {for}
minDistance2 := maxReal;
maxDistance2 := -maxReal;
for j := 0 to pointList2Count - 2 do begin
    point := linePoint;
    if LineIntersectsSegment(point, @vector, @pointList2^[j], @pointList2^[j + 1])
    then begin
        LineToLength(@linePoint, @point, distance);
        if distance < minDistance2 then
            minDistance2 := distance;
        if distance > maxDistance2 then
            maxDistance2 := distance;
        end; {if}
    end; {for}
    if minDistance2 > minDistance1 then
        minDistance1 := minDistance2;
    if maxDistance2 < maxDistance1 then
        maxDistance1 := maxDistance2;
    if minDistance1 < maxDistance1 then
        nextArea := nextArea + maxDistance1 - minDistance1;
    linePoint.x := linePoint.x + direction.x;
    linePoint.y := linePoint.y + direction.y;
end; {for}
end; {CalcArea}

Procedure CalcLineSegment;
var
    distance : Real;
    maxDistance : Real;
    minDistance : Real;
    point : TPoint;
    pointNo : Word;
begin {CalcLineSegment}
    minDistance := 0;
    maxDistance := 0;
    for pointNo := 0 to GetPointCount(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo)) - 1
    do begin
        GetPoint(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo), pointNo, point);
        distance := direction.x * (point.x - centre.x)
            + direction.y * (point.y - centre.y);
        if distance < minDistance then
            minDistance := distance
        else if distance > maxDistance then
            maxDistance := distance;
        end; {for}
        startPoint.x := centre.x + minDistance * direction.x;
        startPoint.y := centre.y + minDistance * direction.y;
        endPoint.x := centre.x + maxDistance * direction.x;
        endPoint.y := centre.y + maxDistance * direction.y;
    end; {CalcLineSegment}

Procedure SplitShape;
var
    point : RealPoint;
    point1 : RealPoint;
    point2 : RealPoint;
    pointNo : Word;
begin {SplitShape}
    pointNo := 0;
    pointList1Count := 0;
    repeat
        PointToRealPoint(pointList^[pointNo], pointList1^[pointList1Count]);
        Inc(pointList1Count);
        point := centre;
        PointToRealPoint(pointList^[pointNo], point1);
        Inc(pointNo);
        PointToRealPoint(pointList^[pointNo], point2);
    until LineIntersectsSegment(point, @direction, @point1, @point2);
    if not EqualRealPoint(@point1, @point) then begin
        pointList1^[pointList1Count] := point;
        Inc(pointList1Count);
    end; {if}
    pointList2^[0] := point;
    pointList2Count := 1;
    repeat
        PointToRealPoint(pointList^[pointNo], pointList2^[pointList2Count]);
        Inc(pointList2Count);
        point := centre;
        PointToRealPoint(pointList^[pointNo], point1);
        Inc(pointNo);
        PointToRealPoint(pointList^[pointNo], point2);
    until LineIntersectsSegment(point, @direction, @point1, @point2);
    if not EqualRealPoint(@point1, @point) then begin

```



```

    pointList2^[pointList2Count] := point;
    Inc(pointList2Count);
  end; (if)
  pointList2^[pointList2Count] := pointList2^[0];
  Inc(pointList2Count);
  PointToRealPoint(pointList^[pointNo], point1);
  if not EqualRealPoint(@point, @point1) then begin
    pointList1^[pointList1Count] := point;
    Inc(pointList1Count);
  end; (if)
  while pointNo <= pointCount do begin
    PointToRealPoint(pointList^[pointNo], pointList1^[pointList1Count]);
    Inc(pointNo);
    Inc(pointList1Count);
  end; (while)
end; (SplitShape)

begin (MAIN: CalcOrientation)
  case pointCount of
    1 :
      orientation := 0;
    2 : begin
      i := 1;
      direction.x := pointList^[0].x - pointList^[i].x;
      direction.y := pointList^[0].y - pointList^[i].y;
      VectorToAngle(@direction, orientation);
    end; (2)
  else begin
    minAngle := -pi / 2 - maxStepAngle;
    maxAngle := pi / 2 + maxStepAngle;
    stepAngle := maxStepAngle;
    for i := 0 to Round((Ln(maxStepAngle) - Ln(minStepAngle)) / Ln(10)) do begin
      angle := minAngle;
      lastArea := maxReal;
      currentArea := maxReal;
      maxLength := 0;
      for j := 0 to Round((maxAngle - minAngle) / stepAngle) do begin
        AngleToVector(@angle, direction);
        SplitShape;
        CalcLineSegment;
        CalcArea;
        if RealLessThan(@lastArea, @currentArea)
          and RealLessThan(@nextArea, @currentArea)
          and RealLessThan(@maxLength, @currentLength) then begin
          orientation := angle - stepAngle;
          maxLength := currentLength;
        end; (if)
        startPoint := pointList2^[0];
        endPoint := pointList2^[pointList2Count - 2];
        lastArea := currentArea;
        currentArea := nextArea;
        LineToLength(@startPoint, @endPoint, currentLength);
        angle := angle + stepAngle;
      end; (for)
      maxAngle := orientation + stepAngle;
      if maxAngle > pi / 2 then
        maxAngle := pi / 2;
      minAngle := orientation - stepAngle;
      if minAngle < -pi / 2 then
        minAngle := -pi / 2;
      stepAngle := stepAngle / 10;
    end; (for)
    orientation := (maxAngle + minAngle) / 2;
  end; (else)
end; (case)
end; (CalcOrientation)
{-----}
begin (MAIN: TrackFirst)
  pointGroupNo := trackingInfo^.pointGroupNo;
  pointCount := GetEdgePointCount(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo));
  ClearTrackInfo(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo));
  wInfo^.changed := wInfo^.changed or pointInfoChangedBit;
  pointList := GlobalLock(trackingInfo^.pointList);
  pointList1 := GlobalLock(trackingInfo^.pointList1);
  pointList2 := GlobalLock(trackingInfo^.pointList2);
  CalcCentre;
  CalcOrientation;
  GlobalUnlock(trackingInfo^.pointList);
  GlobalUnlock(trackingInfo^.pointList1);
  GlobalUnlock(trackingInfo^.pointList2);
  SetTrackInfo(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo),
    trackingInfo^.destImageNo, @centre, @orientation);
  error := noErr;
  if NextPointGroupNo(wInfo^.pointInfo, trackingInfo^) then begin
    DestroyHandle(trackingInfo^.pointList);
    DestroyHandle(trackingInfo^.pointList1);
    DestroyHandle(trackingInfo^.pointList2);
  end;
end;

```

```

if wInfo^.trackOptions.display then
  SendMessage(window, setImageMsg, trackingInfo^.destImageNo, 1);
trackingInfo^.maxPointCount := 0;
for pointGroupNo := 0 to GetPointGroupCount(wInfo^.pointInfo) - 1 do
  if not PointGroupMarked(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo), trackMark)
  then begin
    pointCount := GetPointCount(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo));
    if pointCount > trackingInfo^.maxPointCount then
      trackingInfo^.maxPointCount := pointCount;
    end; (if)
  trackingInfo^.pointWindowRecSize :=
    SizeOf(PointWindowRec) - SizeOf(Byte)
    + SizeOf(Byte) * Sqr(2 * wInfo^.trackOptions.windowRadius + 1);
  error := CreateHandle(trackingInfo^.srcPointWindowList,
    trackingInfo^.pointWindowRecSize*trackingInfo^.maxPointCount, 0);
  if error = noErr then
    error := CreateHandle(trackingInfo^.destPointList,
      SizeOf(RealPoint) * trackingInfo^.maxPointCount, 0);
  if error = noErr then
    CreateTrackPointList;
  if error = noErr then begin
    trackingInfo^.destImageNo := wInfo^.pointImageNo - 1;
    trackingInfo^.imageNoOffset := -1;
    trackingInfo^.pointCount :=
      GetPointCount(wInfo^.pointInfo, PointGroupNamePtr(trackingInfo^.startPointGroupNo));
    trackingInfo^.pointGroupNo := trackingInfo^.startPointGroupNo;
    trackingInfo^.rotCount :=
      Round(2 * wInfo^.trackOptions.rotRange / wInfo^.trackOptions.rotStep) + 1;
    trackingInfo^.rotNo := Word(-1);
    trackingInfo^.srcImageNo := wInfo^.pointImageNo;
    end; (if)
  end; (if)
  if error = noErr then
    SendMessage(wInfo^.messageControl, setMessagePosMsg,
      LoWord(SendMessage(wInfo^.messageControl, getMessagePosMsg, 0, 0)) + 1, 0);
  TrackFirst := error;
end; (TrackFirst)
(-----TrackNext-----)
Function TrackNext(window : HWND; wInfo : DocWInfoPtr;
  trackingInfo : TrackingInfoPtr) : ErrorCode;
(- Performs the main core of the motion-tracking algorithm. It does this in the
  background using timer events.
)
var
  windowPixelPtr : BytePtr;
  pixelPtr : BytePtr;
  error : ErrorCode;
  destImage : ImagePtr;
  srcImage : ImagePtr;
  x : Integer;
  y : Integer;
  sum : Longint;
  pointGroupName : PointGroupNameStr;
  pointWindowList : PointWindowListPtr;
  pointWindow : PointWindowPtr;
  direction : Real;
  match : Real;
  centre : RealPoint;
  offset : RealPoint;
  vector : RealPoint;
  destPointList : RealPointListPtr;
  paramList : record
    pointGroupName : PChar;
    imageNo : Word;
  end;
  destPoint : TPoint;
  srcTrackInfo : TrackInfoRec;
  trackingPointList : TrackingPointListPtr;
  trackOptions : TrackOptionsRec;
  destSum : Word;
  length : Word;
  pixelNo : Word;
  pointCount : Word;
  pointGroupIndex : Word;
  pointGroupNo : Word;
  pointNo : Word;
  rowByteCount : Word;
  windowDiameter : Word;
begin (TrackNext)
  error := noErr;
  if (trackingInfo^.srcImage = 0) or (trackingInfo^.srcImageFilter <> noFilter) then
    error := TrackFilter(window, trackingInfo^, trackingInfo^.srcImageNo,
      trackingInfo^.srcImage, trackingInfo^.srcImageFilter)
  else if (trackingInfo^.destImage = 0) or (trackingInfo^.destImageFilter <> noFilter) then
    error := TrackFilter(window, trackingInfo^, trackingInfo^.destImageNo,
      trackingInfo^.destImage, trackingInfo^.destImageFilter)
  else begin
    pointWindowList := GlobalLock(trackingInfo^.srcPointWindowList);

```

```

trackingPointList := GlobalLock(trackingInfo^.trackingPointList);
trackOptions := wInfo^.trackOptions;
pointCount := trackingInfo^.pointCount;
pointGroupIndex := trackingInfo^.pointGroupIndex;
pointGroupNo := trackingInfo^.pointGroupNo;
rowByteCount := Align32(wInfo^.bitmapInfo.width);
windowDiameter := 2 * trackOptions.windowRadius + 1;
if trackingInfo^.rotNo = Word(-1) then begin
  GetPointGroupName(wInfo^.pointInfo, pointGroupNo, pointGroupName);
  paramList.pointGroupName := @pointGroupName;
  paramList.imageNo := trackingInfo^.destImageNo + 1;
  SendMessage(window, setMsg, imageTrackStrID, Longint(@paramList));
  srcImage := GlobalLock(trackingInfo^.srcImage);
  pointWindow := PointWindowPtr(pointWindowList);
  GetTrackInfo(wInfo^.pointInfo,
    PointGroupNamePtr(pointGroupNo), trackingInfo^.srcImageNo, srcTrackInfo);
  for pointNo := 0 to pointCount - 1 do begin
    direction :=
      srcTrackInfo.orientation + trackingPointList^[pointGroupIndex + pointNo].direction;
    PinAngle2Pi(direction);
    AngleToVector(@direction, vector);
    SetPoint(destPoint, Round(srcTrackInfo.centre.x +
      trackingPointList^[pointGroupIndex + pointNo].length * vector.x),
      Round(srcTrackInfo.centre.y +
        trackingPointList^[pointGroupIndex + pointNo].length * vector.y));
    pointWindow^.sum := 0;
    pixelPtr :=
      OffsetPointer(@srcImage^.data, (destPoint.y + trackOptions.windowRadius)
        * rowByteCount + destPoint.x - trackOptions.windowRadius);
    windowPixelPtr := @pointWindow^.data;
    for y := 1 to windowDiameter do begin
      for x := 1 to windowDiameter do begin
        Inc(pointWindow^.sum, pixelPtr^);
        windowPixelPtr^ := pixelPtr^;
        Inc(PChar(windowPixelPtr));
        pixelPtr := OffsetPointer(pixelPtr, 1);
      end; (for)
      pixelPtr := OffsetPointer(pixelPtr, -(rowByteCount + windowDiameter));
    end; (for)
    pointWindow := PointWindowPtr(windowPixelPtr);
  end; (for)
  GlobalUnlock(trackingInfo^.srcImage);
  trackingInfo^.bestMatch := 0;
  trackingInfo^.orientation :=
    srcTrackInfo.orientation - pi * trackOptions.rotRange / 180;
  PinAngle2Pi(trackingInfo^.orientation);
  trackingInfo^.centre := srcTrackInfo.centre;
  trackingInfo^.rotNo := 0;
end (if)
else begin
  destImage := GlobalLock(trackingInfo^.destImage);
  destPointList := GlobalLock(trackingInfo^.destPointList);
  if trackingInfo^.rotNo < trackingInfo^.rotCount then begin
    centre := trackingInfo^.centre;
    for pointNo := 0 to pointCount - 1 do begin
      direction := trackingInfo^.orientation + trackingPointList^[pointGroupIndex
        + pointNo].direction;
      PinAngle2Pi(direction);
      AngleToVector(@direction, vector);
      destPointList^[pointNo].x :=
        centre.x + trackingPointList^[pointGroupIndex + pointNo].length * vector.x;
      destPointList^[pointNo].y :=
        centre.y + trackingPointList^[pointGroupIndex + pointNo].length * vector.y;
    end; (for)
    offset.x := trackOptions.transStep;
    offset.y := 0;
    length := 1;
    while length <= 2 * trackOptions.transRadius do begin
      for pixelNo := 1 to length do begin
        match := 0;
        pointWindow := PointWindowPtr(pointWindowList);
        for pointNo := 0 to pointCount - 1 do begin
          SetPoint(destPoint, Round(destPointList^[pointNo].x),
            Round(destPointList^[pointNo].y));
          sum := 0;
          destSum := 0;
          pixelPtr :=
            OffsetPointer(@destImage^.data, (destPoint.y + trackOptions.windowRadius)
              * rowByteCount + destPoint.x - trackOptions.windowRadius);
          windowPixelPtr := @pointWindow^.data;
          for y := 1 to windowDiameter do begin
            for x := 1 to windowDiameter do begin
              Inc(sum, Abs(pixelPtr^ * Longint(windowPixelPtr^)));
              Inc(destSum, pixelPtr^);
              pixelPtr := OffsetPointer(pixelPtr, 1);
              Inc(PChar(windowPixelPtr));
            end; (for)
            pixelPtr := OffsetPointer(pixelPtr, -(rowByteCount + windowDiameter));
          end; (for)
        end; (for)
      end; (for)
    end; (while)
  end; (if)
end; (else)

```

```

    end; (for)
    if destSum * Longint(pointWindow^.sum) <> 0 then
        match := match + sum / (destSum * Longint(pointWindow^.sum));
        pointWindow := PointWindowPtr(windowPixelPtr);
    end; (for)
    if match > trackingInfo^.bestMatch then begin
        SetTrackInfo(wInfo^.pointInfo, PointGroupNamePtr(pointGroupNo),
            trackingInfo^.destImageNo, @centre, @trackingInfo^.orientation);
        trackingInfo^.bestMatch := match;
    end; (if)
    centre.x := centre.x + offset.x;
    centre.y := centre.y + offset.y;
    for pointNo := 0 to pointCount - 1 do begin
        destPointList^[pointNo].x := destPointList^[pointNo].x + offset.x;
        destPointList^[pointNo].y := destPointList^[pointNo].y + offset.y;
    end; (for)
    end; (for)
    direction := offset.x;
    offset.x := -offset.y;
    offset.y := direction;
    if not RealEqualZero(@offset.x) then
        Inc(length);
    end; (while)
    trackingInfo^.orientation :=
        trackingInfo^.orientation + pi * trackOptions.rotStep / 180;
    PinAngle2Pi(trackingInfo^.orientation);
    Inc(trackingInfo^.rotNo);
end (if)
else begin
    if NextPointGroupNo(wInfo^.pointInfo, trackingInfo^) then begin
        if wInfo^.trackOptions.display then
            SendMessage(window, setImageMsg, trackingInfo^.destImageNo, 1);
        if trackingInfo^.filter then
            DestroyImage(trackingInfo^.srcImage)
        else
            SetDiscard(trackingInfo^.srcImage, true);
        if trackingInfo^.destImageNo = 0 then begin
            if trackingInfo^.filter then
                DestroyImage(trackingInfo^.destImage)
            else
                SetDiscard(trackingInfo^.destImage, true);
            trackingInfo^.destImage := 0;
            trackingInfo^.srcImage := 0;
            trackingInfo^.destImageNo := wInfo^.pointImageNo + 1;
            trackingInfo^.imageNoOffset := 1;
            trackingInfo^.srcImageNo := wInfo^.pointImageNo;
        end (if)
        else begin
            trackingInfo^.srcImage := trackingInfo^.destImage;
            trackingInfo^.destImage := 0;
            trackingInfo^.srcImageNo := trackingInfo^.destImageNo;
            trackingInfo^.destImageNo :=
                trackingInfo^.destImageNo + trackingInfo^.imageNoOffset;
        end; (else)
        trackingInfo^.pointGroupIndex := 0;
        trackingInfo^.pointGroupNo := trackingInfo^.startPointGroupNo;
    end (if)
    else
        Inc(trackingInfo^.pointGroupIndex, trackingInfo^.pointCount);
        trackingInfo^.pointCount :=
            GetPointCount(wInfo^.pointInfo, PointGroupNamePtr(trackingInfo^.pointGroupNo));
        trackingInfo^.rotNo := Word(-1);
        SendMessage(wInfo^.messageControl, setMsgPosMsg,
            LoWord(SendMessage(wInfo^.messageControl, getMsgPosMsg, 0, 0)) + 1, 0);
    end; (else)
    GlobalUnlock(trackingInfo^.destImage);
    GlobalUnlock(trackingInfo^.destPointList);
end; (else)
GlobalUnlock(trackingInfo^.srcPointWindowList);
GlobalUnlock(trackingInfo^.trackingPointList);
if trackingInfo^.destImageNo = wInfo^.imageCount then
    EndTrack(window);
end; (else)
TrackNext := error;
end; (TrackNext)
(=====)
end. (UNIT: Tracking)

```