IMAGE SEGMENTATION

THROUGH

OPTIMAL TESSELLATION

A thesis submitted for the degree of

Doctor of Philosophy

in the University of London

by

Kevin Luke O'CONNOR

Department of Computing

Imperial College

London

SW7 2BZ

January 1988

ABSTRACT

A major problem in image processing that remains unresolved is how to segment images into foreground and background when the feature distinguishing the two regions is pixel intensity, but the images may be illuminated unevenly. The problem is compounded when the size, shape and location of the objects of interest are not known a priori.

This research investigates the problem and proposes a new approach, showing how it is implemented and presenting the results from applying it to various classes of images.

The technique uses the Dirichlet Tessellation to divide an image into spanning, non-overlapping regions for which threshold values are to be calculated. It is shown that the tessellation is a function of the positions of the generating points of the tiles. A measure of the goodness of a particular tessellation of a given image is defined and a method for finding the positions of the generating points that "optimise" the goodness of tessellation measure is given.

Techniques for computing threshold values for the generating points of the tiles are investigated and a method is advocated. It is then shown how one or more threshold surfaces can be interpolated through the threshold values computed for the generating points, yielding threshold surfaces which are used to threshold the image into two or more regions.

Results from applying the technique are presented and the approach is shown to be effective for the classes of images studied.

TABLE OF CONTENTS

TITLE PAGE	1
ABSTRACT	2
TABLE OF CONTENTS	4
TABLE OF FIGURES	9
TABLE OF TABLES	12
DEDICATION	13
ACKNOWLEDGEMENTS	14
STATEMENT OF ORIGINALITY	15
CHAPTER 1: INTRODUCTION	17
1.1 Introduction	17
1.2 Image Segmentation	19
1.3 Aim of this Work	22
1.4 Contents of Thesis	25
CHAPTER 2: SOME DIGITAL IMAGE PROCESSING FUNDAMENTALS	27
2.1 Introduction	27
2.2 Some Image Characteristics	30
2.2.1 Introduction	30
2.2.2 Intensity Distributions	30
2.2.3 Information Content	33
2.2.4 Variance	37
2.2.5 Moments	39

2.3 Some Forms of Image Enhancement 41

2.3.1 Introduction 41

2.3.2 Histogram Modification 41

2.3.3 Edge Detection 48

2.3.4 Digital Subtraction 55

2.3.5 Thresholding 59

2.4 Quantitative Measures of Image Recovery 62

CHAPTER 3: THRESHOLD EVALUATION 71

3.1 Introduction 71

3.2 Using a priori Information 74

3.3 Average and Median Thresholding 75

3.4 Valley-seeking Techniques 77

3.5 Distribution Assuming Techniques 83

3.6 Moment Preserving Techniques 90

3.7 Statistical Techniques 94

3.8 Multiple Statistic Thresholding 99

3.9 Other Techniques 101

3.10 Relaxation 106

3.10.1 Introduction 106

3.10.2 Probabilistic Relaxation 109

3.10.3 Other Relaxation Techniques 112

3.11 Calculation of Multiple Thresholds 114

3.12 Conclusion 117

CHAPTER 4: SEGMENTATION AND CLUSTERING 118

4.1 Introduction 118

4.2 Similarity Measures and Distance Criteria 121

4.3 Hierarchical Clustering 123

4.3.1 Introduction 123

4.3.2 Single-linkage 126

4.3.3 Complete-linkage 133

4.3.4 Variance Minimisation 134

4.3.5 Other Techniques 135

4.3.6 Review 136

4.4 Optimisation-partitioning Clustering 140

4.5 Density-seeking Clustering 142

4.6 Clumping 144

4.7 Other Techniques 145

4.8 Contiguous versus Non-contiguous Clusters 147

4.9 Storage Requirements 149

4.10 Conclusion 151

CHAPTER 5: PARTITIONING THROUGH TESSELLATION 153

5.1 Introduction 153

5.2 Delaunay Triangulation 159

5.3 Dirichlet Tessellation 163

5.4 Conclusion 173

CHAPTER 6: CLUSTERING CRITERIA 176

6.1 Introduction 176

6.2 Entropy 178

6.3 Variance 180

6.4 Bumpiness 183

6.5 Conclusion 191

CHAPTER 7: OPTIMISATION OF TESSELLATION 193

7.1 Introduction 193

7.2 Characteristics of the Optimisation Problem 195

7.3 A New Approach 199

7.3.1 Introduction 199

7.3.2 The Method 202

7.4 Some Implementation Details 207

7.5 Conclusion 211

CHAPTER 8: ACHIEVING THRESHOLD SURFACES 213

8.1 Introduction 213

8.2 Calculation of Thresholds for Generating Points 215

8.3 Generalisation to Threshold Surfaces 219

8.3.1 Introduction 219

8.3.2 Interpolation over Tiles 220

8.3.3 Interpolation over Triangles 222

8.3.4 Approximation of Derivatives 226

8.3.5 Conclusion 228

8.4 Calculation of Multiple Threshold Surfaces 230

8.5 Use of Relaxation 233

8.6 Conclusion 236

CHAPTER 9: APPLICATION TO VARIOUS CLASSES OF IMAGES 238

9.1 Introduction 238

9.2 Results 243

9.2.1 Ventriculograms 243

9.2.2 Images of Cells 256

9.2.3 Satellite Images 260

9.2.4 Artificially Generated Images 265

CHAPTER 10: QUANTITATIVE EVALUATION OF SEGMENTATION TECHNIQUES 275

10.1 Discussion of Quantitative Measures 275

10.2 Experimental Results 281

10.3 Conclusion 308

CHAPTER 11: CONCLUSION 311

11.1 Some General Comments 311

11.2 Contribution of this Research 313

11.3 Conclusion 315

APPENDICES

Appendix A 317

Appendix B 326

Appendix C 350

Appendix D 364

REFERENCES 379

TABLE OF FIGURES

2.1	Digitising grid	27
2.2	One dimensional grey level histograms	31
2.3	Two dimensional (grey level, edge value) scatter plots	33
2.4	Poor contrast images	45
2.5	Linearly stretched images	46
2.6	Histogram equalised images	47
2.7	Sobel operator edge-detected images	51
2.8	Laplacian edge-detected images	54
2.9	Digitally subtracted images	57
2.10	Thresholded images	61
3.1	Median thresholding	76
3.2	Bimodal histogram	78
3.3	Non-bimodal histograms	78
3.4	Busyness plots	103
4.1	4- and 8-neighbourhoods	125
4.2	Single-linkage dendograms	127
4.3	Single-linkage's chaining effect	132
4.4	Complete-linkage dendograms	133
4.5	Variance minimisation dendograms	135
5.1	Delaunay Triangulation	161
5.2	Dirichlet Tessellation	163
5.3	Diagram for theorem 5.1	167
5.4	Diagram for case 1 of theorem 5.2	169
5.5	Diagram for case 2 of theorem 5.2	171

6.1 Positioning of charges in a torus 187

6.2 Positioning of charges in a straight line 188

6.3 Positioning of spikes that minimises the bumpiness 189

9.1 Types of images presented 242

9.2 Comparison of clustering criteria and interpolation techniques 250

9.3 Results from segmenting images with high contrast 253

9.4 Results from segmenting images with low contrast 254

9.5 Results from segmenting images which are illuminated unevenly 255

9.6 Segmentation of an image of some cells 258

9.7 Segmentation of unevenly illuminated image of some cells 259

9.8 Satellite image segmented into 3 regions 262

9.9 Unevenly illuminated satellite image segmented into 2 regions 263

9.10 Segmentation of satellite image 264

9.11 Artificially generated image 266

9.12 Comparison of clustering criteria 269

9.13 Effects of approximating derivatives of threshold surface 271

9.14 Comparison of interpolation techniques 274

10.1 8x8 checkerboard 284

10.2 Segmentations of figure 10.1(b) 285

10.3 4 white squares 288

10.4 Segmentations of figure 10.3(b) 289

10.5 Humped ventricle 292

10.6 Segmentations of figure 10.5(b) 293

10.7 Linear ventricle 296

10.8 Segmentations of figure 10.7(b) 297

10.9 Blurred checkerboard 300

10.10 Segmentations of figure 10.9(b) 301

10.11 Noisy squares 304

10.12 Segmentations of figure 10.11(b) 305

TABLE OF TABLES

Table 10.1	286
Table 10.2	290
Table 10.3	294
Table 10.4	298
Table 10.5	302
Table 10.6	306

DEDICATION

This research is dedicated to

my wife, Ingrid,

and to our parents.

ACKNOWLEDGEMENTS

I wish to thank Dr D.M. Monro for supervising this research.

I would also like to thank Professor I. Aleksander for his advice and support, Fr Dr R. Price for the translating work he did for me, and C.S. Kwok, H-K. Chan, Z. El-Hassan and T.Watsuji for their assistance in producing the photographs in this thesis.

My thanks also go to my parents, the Committee of Vice-Chancellors and Principals of the Universities in the United Kingdom, the trustees of the Robert and Cicely Haworth Memorial Scholarship and the South African Council for Scientific and Industrial Research for their financial support.

Most important, my sincere thanks to my wife, Ingrid, who proof read this thesis, corrected my spelling and gave me much needed encouragement and support in a busy three years.

STATEMENT OF ORIGINALITY

The following aspects of the material presented in this thesis are original and constitiute the major part of the contribution made by this research to the body of knowledge in the field of image segmentation:

(i) The suggestion of and justification for dividing an image into a predetermined number of irregular, non-overlapping regions for which threshold values are to be calculated.

(ii) The use of the Dirichlet Tessellation for the initial subdivision of the image as described in (i).

(iii) The investigation of various measures to determine the flatness of histograms and the definition of a new measure, bumpiness.

(iv) The optimisation technique described in chapter 7.

(v) The derivations in chapters 5 and 7 of the properties of the Dirichlet Tessellation, and the discussion showing how these properties can be used to advantage in this application.

The following are also original but constitute lesser contributions to the body of knowledge in the field:

(i) The efficient single-linkage clustering technique and algorithm given in (3) of section 4.3.3 and implemented in Appendix 1.

(ii) The critical discussion and comparative investigation of thresholding techniques in chapter 3.

(iii) The critical discussion and comparative investigation of segmentation and clustering techniques in chapter 4.

(iv) Except where explicitly stated to the contrary, the investigation of the interpolation techniques and the problems encountered when multiple threshold surfaces are to be interpolated through irregularly spaced points.

(v) The Maximum Error Measure defined and used in chapter 10.

CHAPTER 1

INTRODUCTION

1.1 Introduction

This research investigates a particular problem in image processing in which monochrome images containing different objects are divided into "foreground" and "background". The technique is a form of segmentation; a special case of this is thresholding, in which the division is made on the basis of image intensity. If the process succeeds, one category will include only the objects of interest and the other will contain everything else.

Often this process involves two such categories, but the extension to larger numbers is also of interest.

The particular problem considered is to find the "best" segmentation of an unevenly illuminated picture. A general approach to the problem is sought, through image processing by digital computer. A precise definition of the problem is given in section 1.3.

Currently available techniques are explored and evaluated but none are found to work consistently on images with the properties defined in section 1.3. It is concluded that the technique sought should threshold the image using a variable threshold surface, and such a technique is developed, implemented and evaluated.

Techniques for generating multiple threshold surfaces are also investigated and it is shown that the technique developed during

the course of this research has a natural generalisation to generate these threshold surfaces.

Section 1.2 gives a brief introduction to image segmentation, section 1.3 defines the aim of this work and section 1.4 gives an outline of the contents of the chapters in this thesis.

1.2 Image Segmentation

Three of the major areas of research in image processing are:

(i) Image enhacement
(ii) Feature extraction
(iii) Feature recognition

} (1)

Much work has been done in all three of these areas. Chapter 2 describes some of the image enhancement techniques that have been developed, but this research is concerned almost exclusively with feature extraction.

There are many other areas of research in image processing, such as data compression, data capturing techniques and the design of specialised hardware for image processing, but they fall outside the scope of this research.

Feature extraction (often called "image segmentation") is the location of features in a particular image, without necessarily identifying those features (that would be feature recognition). In other words, feature extraction determines where objects are rather than what they are. Sometimes the nature and number of features to be extracted from the image is known a priori and the problem is one of locating where the features are, and at other times, very little is known. Clearly, the more that is known a priori about the features, the more reliable will be the feature extraction.

A related field, and one in much work has been done in other disciplines, is clustering. Clustering is the grouping together of similar objects (in whatever way "similar" may be defined). While segmentation is divisive (images are divided into smaller regions), clustering is agglomerative (pixels are grouped together into larger regions).

More formally, Kendall and Buckland [49] define a cluster as:

> "A group of contiguous elements of a statistical population, e.g. a group of people living in a single house, a run of observations in an ordered series, or a set of adjacent plots in a field."

While clustering and segmentation may appear to be dissimilar, in digital image processing, they amount to being only different ways of achieving the same result. That is, they are implementation details. Clearly, in digital image processing, feature extraction can be viewed either as dividing an image into constituent parts, or as considering the individual pixels in that image and grouping together those that belong to the same object.

For this reason, the terms segmentation and clustering are used interchangably in the literature, and where the distinction between the approaches is necessary, the terms "agglomerative" and "divisive" are used. Strictly speaking, segmentation is top-down and divisive and clustering is bottom-up and agglomerative. In the literature, segmentation is often referred to as "divisive" or "disaggregative" clustering.

Chapter 4 discusses segmentation and clustering, as they are applied in digital image processing, in detail.

1.3 Aim of this Work

It is not possible to define one segmentation technique that will work on all classes of images. One image may have several different segmentations, depending on the application.

Therefore, in each context, it is necessary to define what it is that distinguishes two regions (objects) in a given image, viz. what criteria will be used in determining if two regions of the image are similar or not.

The aim of this work is to develop a segmentation technique that can segment images with the following properties:

(i) The images may have low contrast

(ii) The images may be illuminated unevenly

(iii) The images may contain either large or small objects, or both, of any shape, without these details needing to be known or specified a priori

(iv) The image may contain any number of objects

(v) The regions occupied by the individual objects in the image need not be contiguous.

(1)

The proposed technique does not address the following issues:

(i) Objects may have some areas that are lighter and others that are darker than the surrounding background

(ii) What distinguishes the objects from the surrounding background is not so much the level of grey perceived, as the difference in texture between the object and the surrounding background

(iii) Images in which either the object or the background is highly textured.

(2)

The proposed method should segment the image into "foreground" and "background", based on the brightness of each section of the image relative to its surrounding sections.

In different applications, the "foreground" and "background" will have different physical interpretations, but the technique should be (and we shall see later that it is) application independent, provided the constraints (2) above are satisfied.

It is shown in later sections that the conditions (1) arise and the constraints (2) are satisfied in images used in several

different applications, including:

(i) Remote Sensing
(ii) Medical Imaging
(iii) Biological Sciences

} (3)

Images from all three of the above disciplines are processed and their results are given in chapters 9 and 10.

This research will also investigate methods for segmenting images into more than two regions. For example, satellite photographs may contain clouds, land and sea, and the segmentation of such photographs should reflect this.

1.4 Contents of Thesis

Chapter 2 introduces some digital image processing fundamentals that are relevant to this work. It also introduces the notion of thresholding.

Chapter 3 then discusses threshold evaluation techniques in greater detail. In particular, the usefulness of the various families of thresholding techniques to images with properties (1) of section 1.3 is investigated.

Chapter 4 describes and evalutes the various approachs to image segmentation and concludes that none of the techniques currently available is suitable for the types of images described in (1) and (2) of section 1.3.

Chapters 5 to 8 describe the new technique developed during the course of this research. Chapter 5 suggests that what Everitt [26] calls "optimisation-partitioning" techniques should be investigated more thoroughly and concludes that images should be tessellated, a threshold value should be calculated for the generating point of each tile, and then a threshold surface should be interpolated through these threshold values. It shows the Dirichlet Tessellation is a good tessellation for this purpose.

Chapter 6 introduces several measures of the "goodness" of the various Dirichlet Tessellations of a given image. Chapter 7 then discusses methods for finding the particular Dirichlet Tessellation that optimises these measures for a given image.

Chapter 8 then shows how threshold values are to be calculated for

the generating points of the tiles, and how a threshold surface can be interpolated through these values. It also shows how the method developed during the course of this research can be generalised to threshold images into more than two regions.

Chapter 9 shows the results of applying the technique developed here, and those of several other techniques.

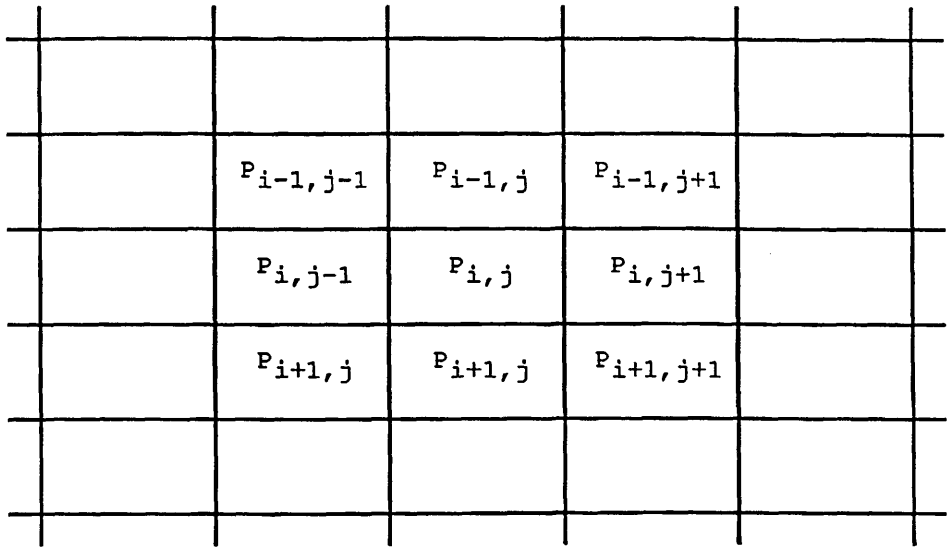
Chapter 10 analyses these results and chapter 11 summarises the conclusions of this research.

CHAPTER 2

SOME DIGITAL IMAGE PROCESSING FUNDAMENTALS

2.1 Introduction

Bi-dimensional images can be considered to be functions of two variables, where the variables specify the region of the image of interest, and the functional value, f(x,y), represents the brightness of the image at that point.



Digitising grid: $P_{i,j}$ denotes the ij^{th} pixel (element of the matrix), representing the average brightness of the image over the corresponding region.

Figure 2.1

However, for images to be processed on digital computers, they need to be digitised in some manner. The most common technique for image digitisation can be viewed, conceptually, as superimposing an mxn grid over the image, averaging the intensity of the light

reflected or emitted over each grid region, and representing these intensities by numeric values. These grid regions are known as "pixels" and the values representing their intensities are known as "pixel values". The image is then represented as an mxn matrix of pixel values, where the ij^{th} element of the matrix represents the average brightness of the image in the corresponding grid region in the image (see figure 2.1).

Typically, the pixels are represented by non-negative integer values, say in the range 0 to (2^L-1), where L is the number of bits required to store each pixel value. Usually, 0 represents black and (2^L-1) white, but sometimes the converse is true. Also, the pixel values are monotonic, in that if 0 represents black, then if $P_i > P_j$, where P_i and P_j are the pixel values for regions i and j respectively, then region i is brighter than region j.

The pixels need not be square and, in fact, they often are not. Similarly, the matrix of pixel values for a given image need not be square, but usually it is. There are some advantages to having m and n being powers of 2, but these fall outside the scope of this research.

There are many terms to describe what pixel values represent. The most common of these are "intensity" and "brightness", and for monochrome images, "grey scale value" or "grey level". Unless stated otherwise, as elsewhere in the literature, these terms shall be used interchangably in this thesis to mean the value of a particular pixel.

More details on the above, and other topics covered in this

chapter, can be found in the references cited in later sections and in the text books by Rosenfeld and Kak [94], Gonzalez and Wintz [34] and Pratt [85].

2.2 Some Image Characteristics

2.2.1 Introduction

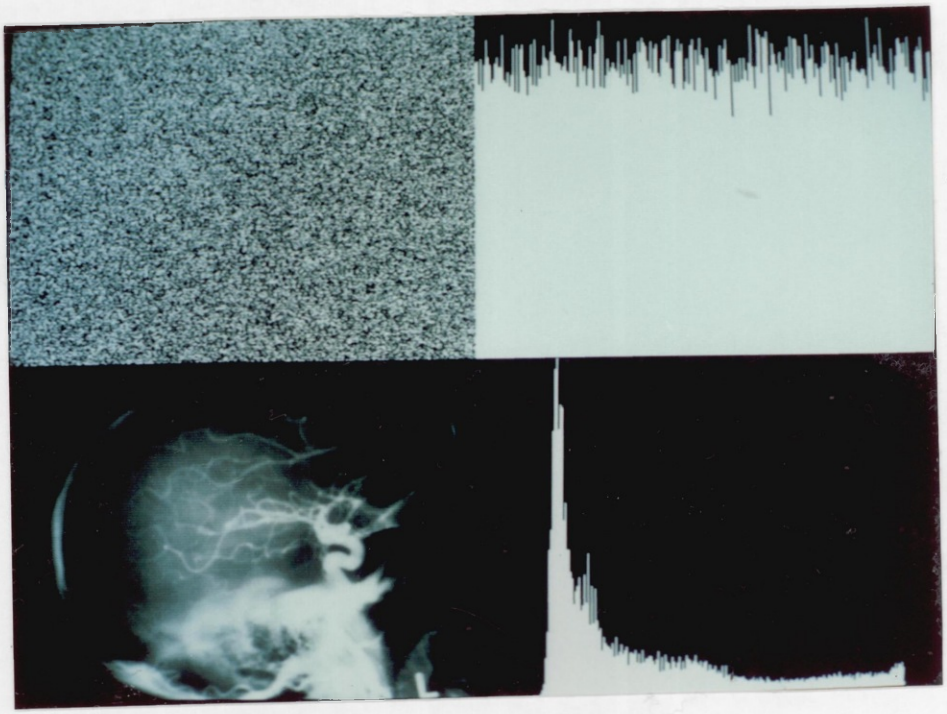
As with any other functions or matrices of numbers, digitised images have certain quantifiable characteristics. These characteristics can be dichotomised into those that describe some property of the image globally and those that describe the properties of the image locally. Global properties include the spread of pixel values (the range of the function describing the image) and various measures thereof, and local properties include the slope of the image function and the average of the values of the pixels neighbouring a particular pixel.

2.2.2 Intensity Distributions

A variety of histograms and probability density functions (PDFs) can be calculated for digitised images.

Histograms show the number of pixels with given characteristics, while PDFs show the probability of a given pixel having those characteristics. Clearly, histograms and probability density functions of the same data have the same shape; only the scale of one of the axes is different. They can be considered to be functions of n variables, where the domains of the n variables specify the characteristics of interest, and the functional value is the number of pixels with those characteristics and the probability of a pixel chosen at random having those characteristics, respectively.

The most commonly used histograms for digitised monochrome images are "grey level histograms". These show the number of pixels in the digitised image with each particular pixel value. These are clearly functions of one variable - pixel value. (See figure 2.2).



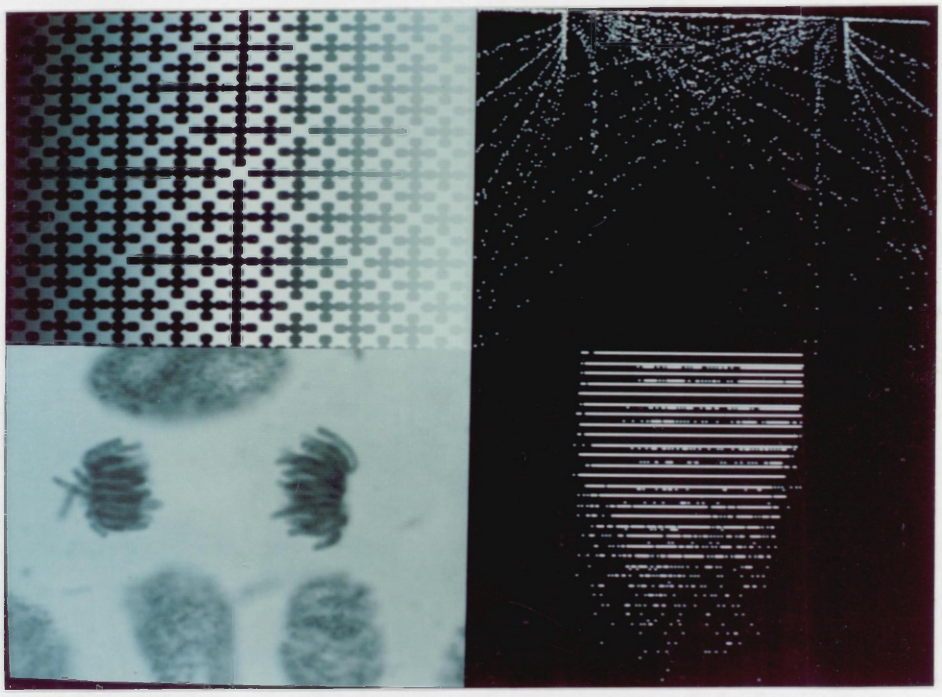
One dimensional grey level histograms: (a) and (c) contain images of pseudo-random noise and an angiogram of a human head, respectively. (b) and (d) contain the histograms (or PDFs) of (a) and (c) respectively. In the histograms, the horizontal axis represents the pixel intensities and the heights of the spikes reflect their respective probabilities of occurrence.

Figure 2.2

However, there are other histograms, for example, Panda and Rosenfeld [80] define a two dimensional (grey level, edge value) histogram. As shown in section 2.3.3, each pixel in an image can be assigned an "edge value". (Grey level, edge value) histograms show the number of pixels with each grey level, edge value

combination (See figure 2.3). Such multi dimensional histograms are sometimes known as "scatter plots".

(a)

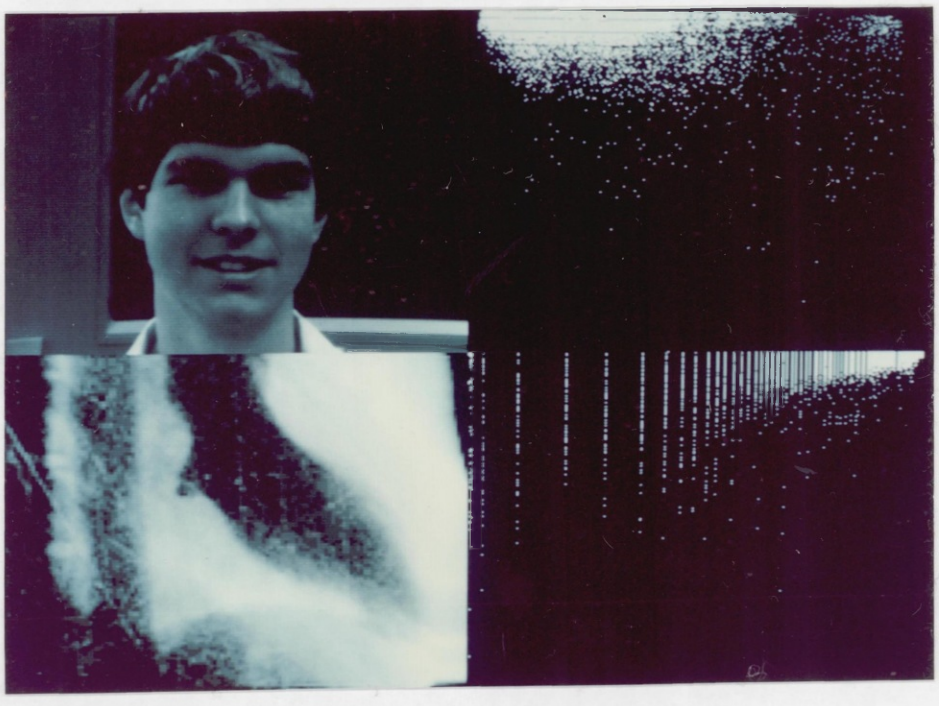


(b)

(c)

(d)

(e) (f)



(g) (h)

Two dimensional (grey level, edge value) scatter plots: (a), (c), (e) and (g) contain images of an unevenly illuminated, filled Sierpinski curve, some cells, a person's face and a ventricle, respectively. (b), (d), (f), and (h) contain the scatter plots of these images respectively. In the scatter plots, the horizontal axis represents pixel intensity (increasing from left to right) and the vertical axis represents the Laplacian edge value of the pixels (increasing from top to bottom).

Figure 2.3

2.2.3 Information Content

Several researchers have tried to quantify the amount of "information" in a given image. That is, they have tried to quantify the average amount of uncertainty removed when an observer is informed of the value of a pixel in the image. For example, in an image containing pixels of only one value, no uncertainty would be removed by being given the value of a pixel

in the image, as the observer knew a priori what the value would be. Therefore, in this case, one would expect the information contained in the image to be zero.

Shannon [99] and Wiener [121] defined a quantitative measure called entropy that they claim measures this gain in information.

Following is a formulation and justification of their entropy measure.

Let a set of n possible events be given by

$$A_n = \{a_1 \ . \ . \ . \ a_n\} \tag{1}$$

and let the probability of event a_i be given by $P(a_i)$.

Clearly

$$\sum_{i=1}^{n} P(a_i) = 1$$

Suppose that for each event a_i, a value $V(a_i)$ is recorded, then the mean, or expected value, of V is given by

$$E(V) = \sum_{i=1}^{n} [V(a_i).P(a_i)] \tag{2}$$

Suppose it is known that some event a_j has occurred to produce output $V(a_j)$, the problem now is to quantify the amount of information, $I(a_j)$, that has been gained.

Entropy is an attempt to measure this "gain in information". It makes two assumptions:

(i) $I(a_j)$ is a continuous function of $P(a_j)$
(ii) No information is duplicated, and therefore redundant. (3)

Assumption (3ii) means that if it is known that events a_i and a_j have occurred, then the information gained is the sum of the information gained from the individual outputs, viz.

$$I(a_i+a_j) = I(a_i) + I(a_j) \tag{4}$$

From (3i), it is clear that $I(a_j)$ is also a continuous function of $\text{Log}(P(a_j))$, for $P(a_j) > 0$. Therefore, a possible formulation of $I(a_j)$ is:

$$I(a_j) = f(\text{Log}(P(a_j))) \tag{5}$$

From (3ii), it is clear that a_i and a_j are independent, therefore:

the probability of events a_i and a_j occurring is $P(a_i).P(a_j)$ (6)

Also, from elementary algebra:

$$\text{Log}(P(a_i).P(a_j)) = \text{Log}(P(a_i)) + \text{Log}(P(a_j)) \tag{7}$$

Defining:

$$x_i = P(a_i) \tag{8}$$

we get

$$f(\text{Log}(x_i) + \text{Log}(x_j)) = f(\text{Log}(x_i.x_j)) \quad \text{by (7)}$$

$$= I(a_i + a_j) \quad \text{by (3ii)}$$

$$= I(a_i) + I(a_j) \quad \text{by (4)}$$

$$= f(\text{Log}(x_i)) + f(\text{Log}(x_j)) \quad \text{by (5)}$$

$\Rightarrow$ f is an additive function of $\text{Log}(x_i)$

$\Rightarrow$ f is linear and homogeneous, viz.

$$f(\text{Log}(P(a_i))) = C.\text{Log}(P(a_i)) \tag{9}$$

Clearly, it is desirable that the information gained be greater than or equal to zero, viz.

$$I(a_i) \geq 0 \tag{10}$$

but $\text{Log}(P(a_i)) \leq 0$, therefore

$$C < 0 \tag{11}$$

but C's value and the base of the Log are mutually dependent. Therefore, we define

$$C = -1 \tag{12}$$

giving

$$I(a(_i)) = - \text{Log}(P(a_i)) \tag{13}$$

where the base of the Log determines the units of information. The two most commonly used units are "nats" (base e is used) and "bits" (base 2 is used).

Equation (13) is called the "self-information" and the entropy,

H(x), is defined as the expected value of the self-information, viz.

$$H(x) = -\sum_{i=1}^{n} [P(a_i).\mathrm{Log}(P(a_i))] \qquad (14)$$

where x is the set of data, in this case, a given image.

Entropy is used widely, and so it was one of the measures used in some of the experiments carried out during the course of this research. The results of these are given in chapters 9 and 10. However, for each application, one needs to determine whether the assumptions (3) above are valid for the data. It shall be seen later that (3ii) is not, in fact, valid in our applications, and so other measures will be developed in later sections.

2.2.4 Variance

The variance of a set of given numerical data is the average square of distance from the mean, of the data. It is often used as a measure of the "spread" of the data. Therefore, the variance of the pixel values of a given image is a measure of the spread of intensities in the digitised image.

The variance of a set of pixel values is defined as:

$$\left.\begin{aligned}\sigma^2 &= \frac{1}{N}\sum_{D}(x_i - x')^2 \\ &= \frac{1}{N}\sum_{D} x_i^2 - x'^2\end{aligned}\right\} \quad (1)$$

where: x_i is the i^{th} pixel value in region D

x' is the mean value of the pixels in region D

D is the region of the image for which the variance is being calculated

N is the number of pixels in the region D.

Another approach to quantifying the spread of pixel values in a given image is to calculate the variance of the pixel value probabilities. That is:

$$\left.\begin{aligned}\sigma^2 &= \frac{1}{N}\sum_{R}(P_i - P)^2 \\ &= \frac{1}{N}\sum_{R}\left(P_i - \frac{1}{N}\right)^2 \\ &= \frac{1}{N}\sum_{R} P_i^2 - \frac{1}{N^2}\end{aligned}\right\} \quad (2)$$

where: P_i is the proportion of the pixels in the region with intensity value i

P is the average of all the P_is

R is the range of permissible pixel values

N is the number of permissible pixel values.

This is a measure of the region's PDF's flatness. Clearly, a large value for σ^2 in (2) would indicate that there is not a very even

spread of pixel values in the region, while a very small value would indicate that there are approximately equal numbers of pixels with each of the pixel values permitted.

In section 6.3, it shall be shown how these two variance measures can be applied to this work and why it may be desirable to do so.

2.2.5 Moments

Moments are a more general form of measure than either average pixel value or the variance of the pixel values in a region. They are also independent of the orientation, size or translation of the image.

The i^{th} moment, M_i, of an image $f(x,y)$ is defined as

$$M_i = \frac{1}{n} \sum_x \sum_y f(x,y)^i, \quad i=1,2,... \tag{1}$$

Clearly, (1) can also be calculated from the histogram or PDF of $f(x,y)$ as follows:

$$M_i = \frac{1}{n} \sum_j (n_j \ (z_j)^i)$$

$$= \sum_j (P_j \ (z_j)^i) \tag{2}$$

where:

- n is the number of pixels in the image
- n_j is the number of pixels in the image with intensity z_j
- P_j is the probability that a pixel chosen at randon is of intensity z_j

- $\{z_j\}$ is the set of permissible pixel values.

In this work, as with most other, $z_j = j$.

Clearly:

$$\left.\begin{array}{ll} \text{(i)} & M_0 = 1 \\ \text{(ii)} & M_1 = E(x) \\ \text{(iii)} & M_2 = \sigma_x^2 + E(x)^2 \end{array}\right\} \quad (3)$$

and hence the mean and variance of the pixel values of a given image are special cases of that image's moments.

Gonzalez and Wintz [34] give more background on the subject, but for the purpose of this work, the above will suffice.

2.3 Some Forms of Image Enhancement

2.3.1 Introduction

There are a number of techniques available for improving the visual quality of images. It should be understood that no image enhancement technique can generate new information; at best, no information is lost during the enhancement, but usually, the information loss is considerable. The object of enhancement is not, therefore, to create new information, but rather to remove unnecessary and confusing information, so that the remaining information is clearer and more apparent.

Techniques exist, for example, to deblur blurred images and to increase the contrast of images with poor contrast.

Following is a description of some of the types of techniques that are relevant to this work.

2.3.2 Histogram Modification

Histogram modification is perhaps the simplest form of image enhancement, and often it is the most effective.

With this class of techniques, the desired shape of the histogram is specified a priori, the histogram of the image is computed, and then the transformation required to convert the latter to the former is computed and applied. The technique determines, from the shape of the image's histogram and the histogram shape desired, a monotonically increasing function that maps the pixel values in the original image to new values in the enhanced image,

such that the histogram of the enhanced image has the desired form.

For example, an image's pixel values may all lie in a small range, in which case the image would have poor contrast (see figure 2.4). One method for improving the contrast would be to determine the linear function that maps the darkest pixels in the image onto black and the lightest onto white, and then map the pixel values accordingly (see figure 2.5).

This function, f(i), which maps the pixel values in the original image to new values in the "enhanced" image, is given by

$$f(i) = (i-a).\frac{y-x}{b-a} + x \qquad (1)$$

where: a is the smallest pixel value in the original image

b is the largest pixel value in the original image

i is a pixel value in the range [a,b]

x is the smallest pixel value desired in the enhanced image

y is the largest pixel value desired in the enhanced image.

Usually, x=0.

The major restriction with this technique is that if the image has most of its pixel values in a very small range, but it has perhaps just one very bright and one very dark pixel, then the technique

will improve the contrast in the image very little, if at all.

A more robust technique that would still improve the contast of the image greatly if the above was true, is histogram equalisation.

This technique seeks to distribute the pixel values as uniformly as possible over the range of permissible values. In other words, it determines the monotonically increasing mapping function, f(i), that maps pixel values in the original image to new values in the the enhanced image, such that the function

$$S_1(i) = \sum_{j=0}^{i} P_j \qquad (2)$$

where: P_j is the probability of a pixel in the enhanced image having value j

is as similar as possible to the function

$$S_0(i) = \frac{i+1}{N} \qquad (3)$$

where: N is the number of permissible pixel values.

An algorithm for determining this mapping function, f(i), subject to the added constraint that

$$S_1(i) \geq S_0(i) \quad , i \in [0,N-1] \qquad (4)$$

follows.

Defining: COUNT - the number of pixels in the original image already assigned values in the enhanced image

N - the number of pixels in the image divided by the number of permissible pixel values

CURRENT - the current pixel value being considered and whose mapping is being sought

NEWVALUE - the next mapped value to be assigned to f(i)

TARGET - the number of pixels in the original image that should have been assigned mapped values in the enhanced image, at any stage

f(i) - the mapping function that will map pixels with value i in the original image to pixels with value f(i) in the enhanced image

n_i - the number of pixels in the original image with intensity i

pseudo code for the algorithm is as follows:

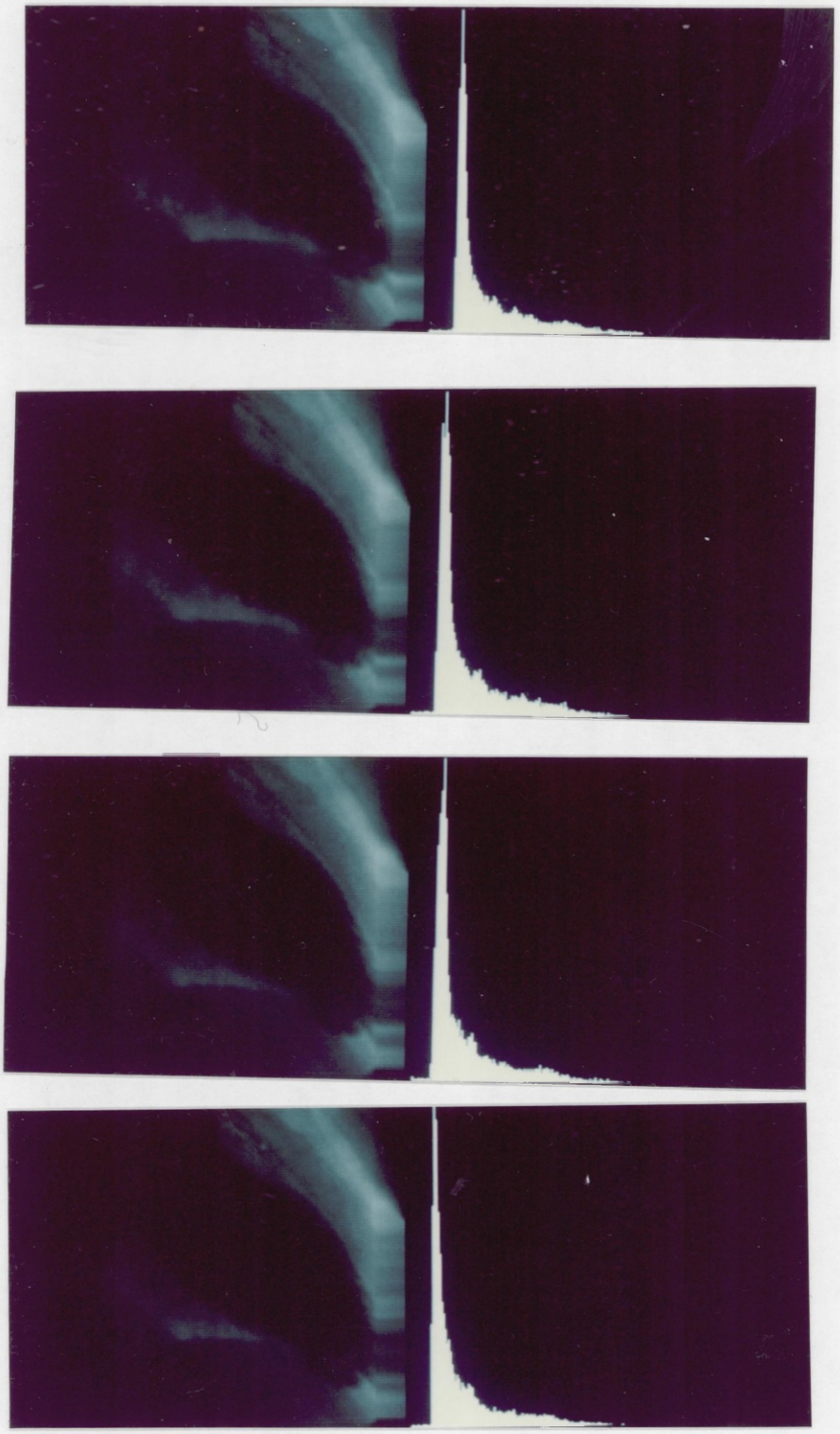
```
COUNT    :=0
CURRENT  :=0
NEWVALUE:=0
TARGET   :=N
while (CURRENT ≤  the largest permissible pixel value) do
   begin
    while (COUNT < TARGET) do
       begin
        f(CURRENT):=NEWVALUE
        COUNT      :=COUNT + n_CURRENT
        CURRENT    :=CURRENT + 1
       end
    NEWVALUE:=NEWVALUE + 1
    TARGET   := TARGET + N
   end
```

(a) (b)

(c) (d)

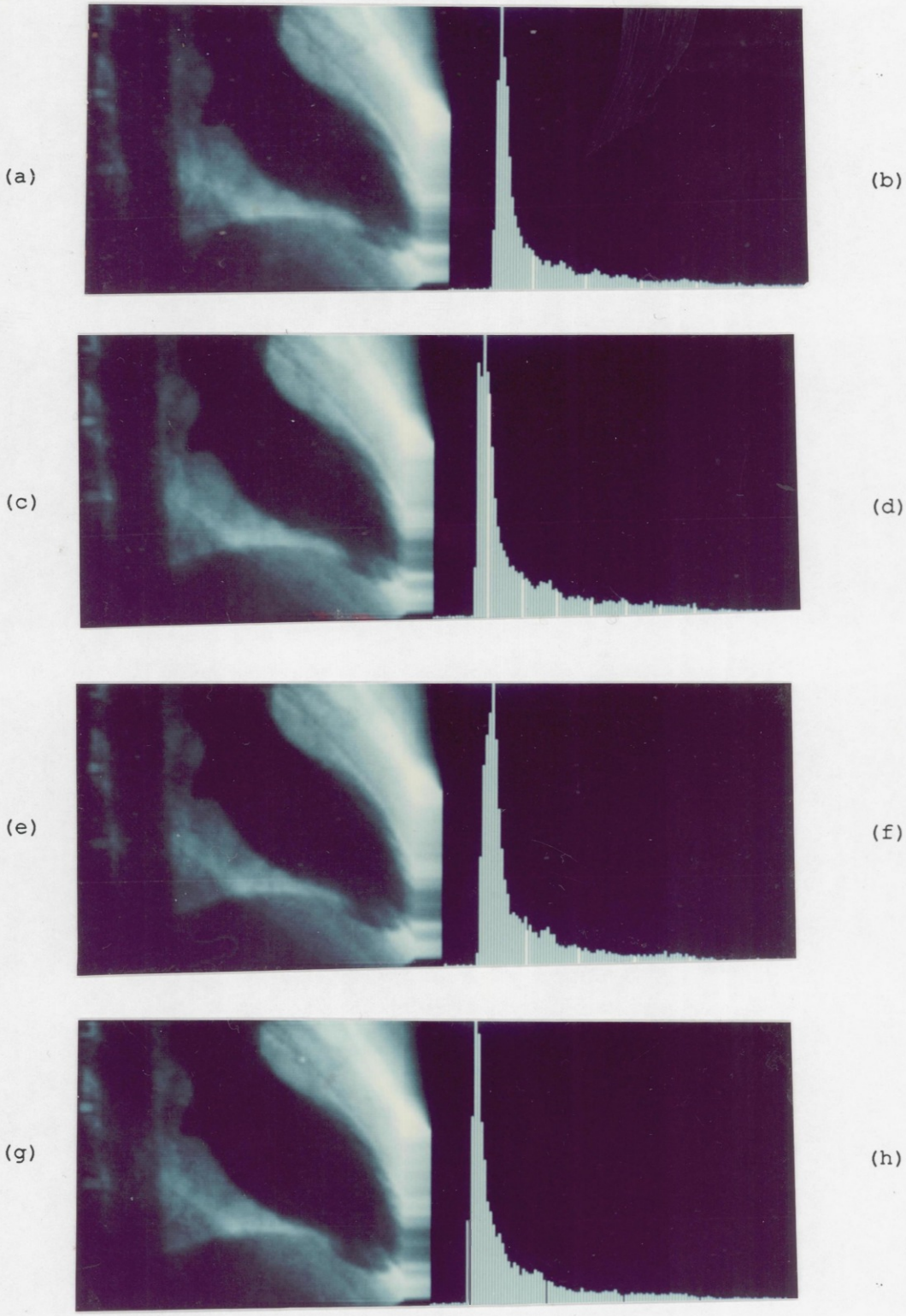
(e) (f)

(g) (h)



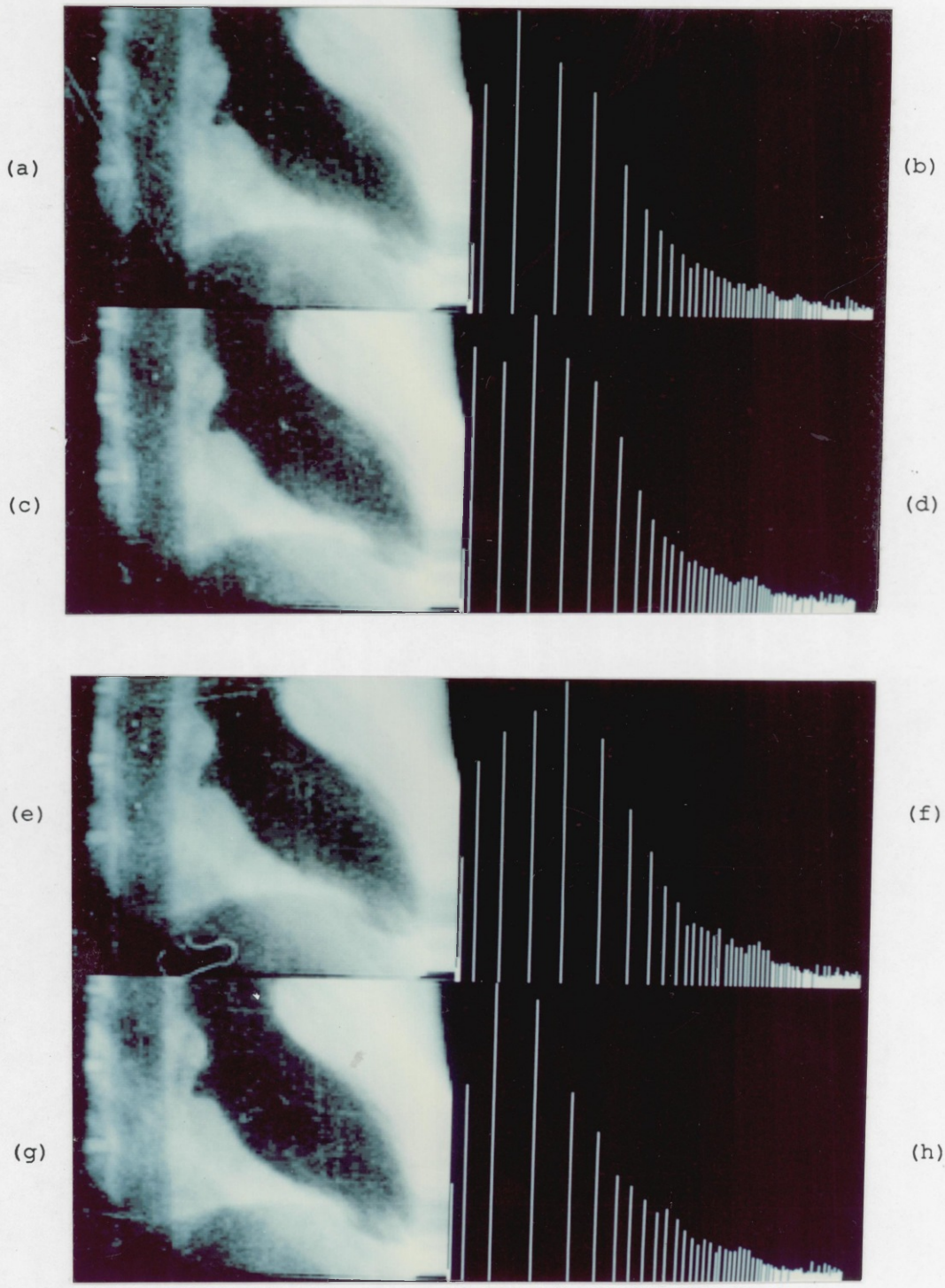
Poor contrast images: (a), (c), (e) and (g) contain low contrast images of ventricles. (b), (d), (f) and (h) contain the histograms of the these images respectively.

Figure 2.4



Linearly stretched images: (a), (c), (e) and (g) show the results of stretching the range of pixel values linearly in the corresponding images in figure 2.4. (b), (d), (f) and (h) contain the histograms for the images in (a), (c), (e) and (g) respectively.

Figure 2.5



Histogram equalised images: (a), (c), (e) and (g) show the results of histogram equalising the range of pixel values in the corresponding images in figure 2.4. (b), (d), (f) and (h) contain the histograms of the images in (a), (c), (e) and (g) respectively.

Figure 2.6

2.3.3 Edge Detection

What sometimes distinguishes objects from their backgrounds is a sudden change in pixel value across the boundary between them. This amounts to a discontinuity or steep slope in the underlying function describing the image, or a great difference in the values of adjacent pixels in the digitised image. Such boundaries can be located and identified by measuring the difference between adjacent pixel values or by approximating the derivatives (slope) of the underlying function.

Several techniques for approximating this slope (the derivative of the underlying function) from the digitised image have been proposed.

One such technique is to apply the Sobel Operator to the original image, $f(x,y)$, to obtain two new "edge value images", $f_x(x,y)$ and $f_y(x,y)$, corresponding to the partial derivatives with respect to x and y respectively. This operator assigns to each non-boundary pixel in the edge value images values given by:

$$f_x(x,y) = \{(f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1) - f(x-1,y-1) - 2f(x-1,y) - f(x-1,y+1)) / 4\} \quad (1a)$$

and

$$f_y(x,y) = \{(f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1) - f(x-1,y-1) - 2f(x,y-1) + f(x+1,y-1)) / 4\} \quad (1b)$$

These operators are sometimes represented, respectively, as two dimensional windows to be applied to the digitised image:

$$f_x(x,y) = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \} \quad (1c)$$

and

$$f_y(x,y) = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \} \quad (1d)$$

Several techniques have been suggested for obtaining edge values for pixels on the boundary on the image f(x,y), amongst them:

(i) Use smaller windows near the edges of the image so that no part of the window ever lies outside the image space

(ii) Assume wrap around of the image's pixel values at the edges

(iii) Assume reflection of the image's pixel values at the edges

(iv) Assume that outside the image space, all the values are equal to some predetermined constant

(v) Assume that outside the image space, all the values are equal to the value of the pixel in the image space that is closest to that point.

} (2)

Fernando [28] discusses these in great detail.

The partial derivatives of the image can be approximated by using the technique given in (1) above. Then, at any point (x,y),

$$f_x(x,y) = S.\mathrm{Cos}(\Theta) \qquad (3a)$$

and

$$\left.\begin{aligned} f_y(x,y) &= S.\mathrm{Cos}(\Phi) \\ &= S.\mathrm{Sin}(\Theta) \end{aligned}\right\} \qquad (3b)$$

where: S is the slope in the direction of steepest ascent/descent at point (x,y)

Θ is the angle between the direction of steepest ascent/descent at point (x,y) and the x-axis

Φ is the angle between the direction of steepest ascent/descent at point (x,y) and the y-axis.

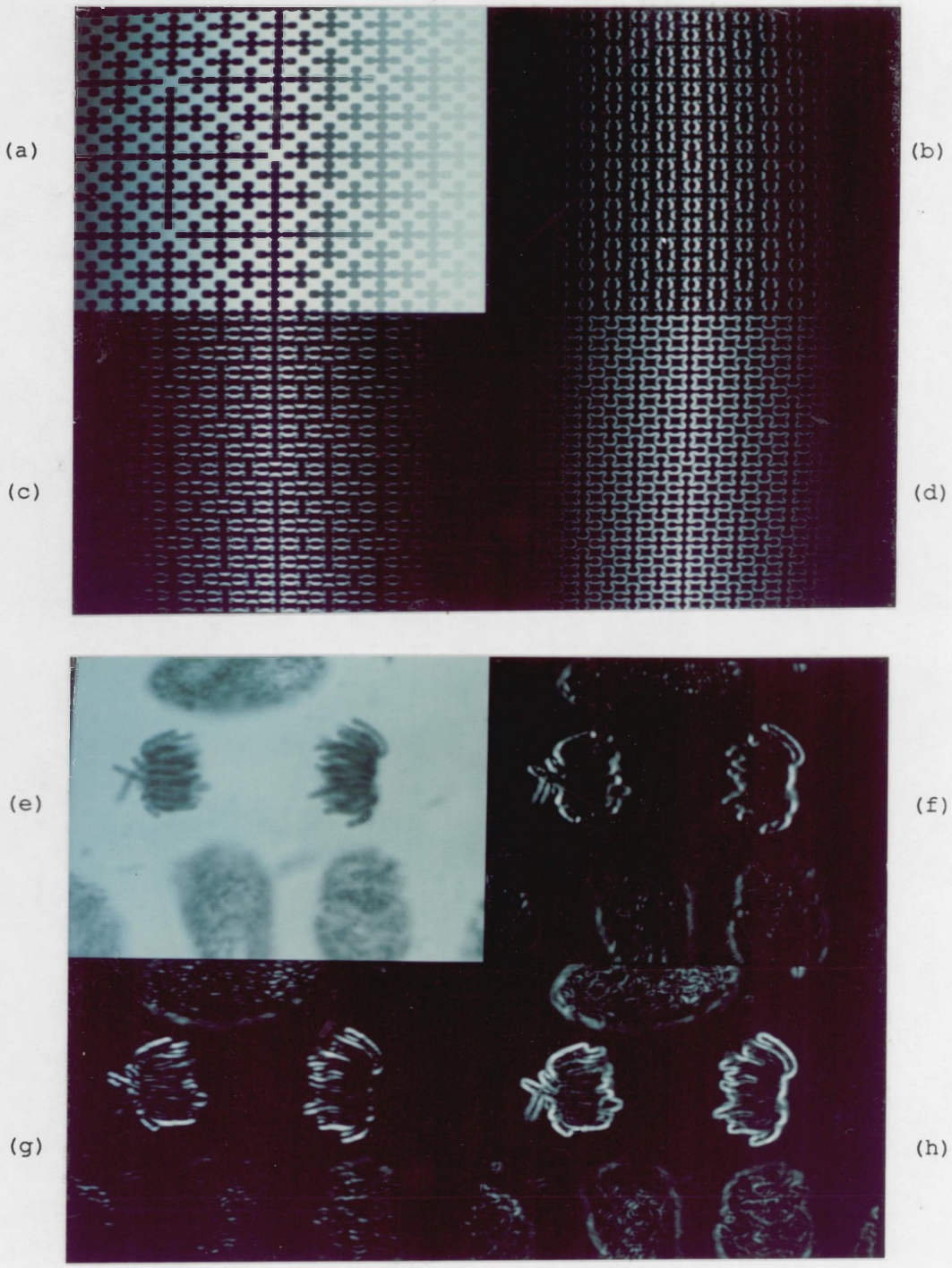
From (3):

$$\left.\Theta = \mathrm{ArcTan}\left(\frac{f_y(x,y)}{f_x(x,y)}\right)\right\} \qquad (4a)$$

and

$$S = f_x(x,y).\mathrm{ArcCos}(\Theta) \qquad (4b)$$

Figure 2.7 shows the results of applying the Sobel operator to some images.



Sobel operator edge-detected images: (b) and (f) show the partial derivatives with respect to x of images (a) and (e), (c) and (g) show the partial derivatives with respect to y, and (d) and (h) show the total derivatives. (b), (c), (d), (f), (g) and (h) have had their pixel values stretched linearly.

Figure 2.7

Similar edge detection operators are those of Prewitt and Roberts, which are given by (5) and (6) below, respectively.

Prewitt Operator:

$$P_x(x,y) = \{ (f(x+1,y-1) + f(x+1,y) + f(x+1,y+1) - f(x-1,y-1) - f(x-1,y) - f(x-1,y+1)) / 3 \} \quad (5a)$$

and

$$P_y(x,y) = \{ (f(x-1,y+1) + f(x,y+1) + f(x+1,y+1) - f(x-1,y-1) - f(x,y-1) + f(x+1,y-1)) / 3 \} \quad (5b)$$

or

$$P_x(x,y) = \{ \frac{1}{3} \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \} \quad (5c)$$

and

$$P_y(x,y) = \{ \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \} \quad (5d)$$

Roberts Operator:

$$R_1(x,y) = f(x,y+1) - f(x+1,y) \quad (6a)$$

and

$$R_2(x,y) = f(x,y) - f(x+1,y+1) \quad (6b)$$

or

$$R_1(x,y) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad (6c)$$

and

$$R_2(x,y) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad (6d)$$

The above operators assume that the objects and background each contain pixels of only a single value (or whose values lie in a very small range). This is often not true. A more general approach is to assume that the pixel values in the objects and background lie in planes that are not necessarily parallel to the xy-plane. In this case, the first order partial derivatives of the image would be constant for pixels interior to the objects and background, but the second derivative would be zero over the objects and background and non-zero at boundaries between the two.

The discrete equivalent of the Laplacian operator proves to be a good edge detector when edge points are defined to be those at which the second derivative is large in magnitude. It is given by:

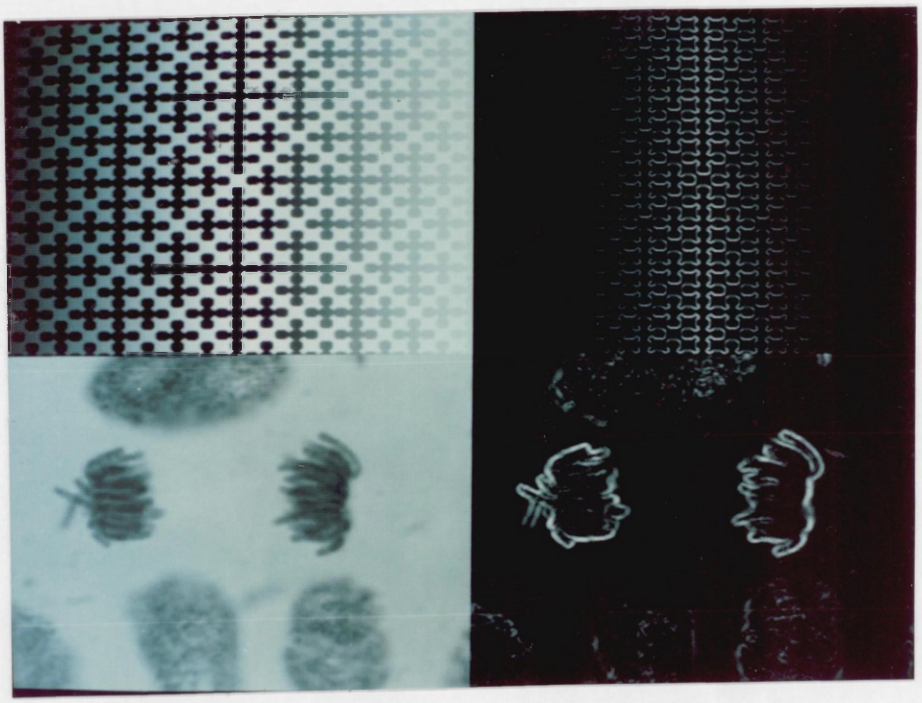
$$L(x,y) = 4f(x,y) - (f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1)) \qquad (7a)$$

or

$$\left.\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}\right\} \qquad (7b)$$

Results of applying the Laplacian operator are given in figure 2.8.

(a) (b)



(c) (d)

Laplacian edge-detected images: (b) and (d) show the Laplacian edge values of the pixels in images (a) and (c) respectively. Images (b) and (d) have had their pixel values stretched linearly.

Figure 2.8

The preceding discussion illustrates the principles of edge detection as well as showing how the better known techniques work. This, in fact, is sufficient for the purpose of this research, but for completeness, following is a brief description of some other approaches that have been suggested, together with references to where more details on the techniques can be found.

Haralick [40], for example, tries to approximate the underlying continuous function for parts of the image. He then classifies those regions that have high first or second derivatives in the underlying function, as being edges.

De Souza [21] investigates, amongst others, the cases where the values of the pixels interior to each of the background and foreground obey a bi-quadratic rule (rather than a bi-linear rule as with the Laplacian operator), and where the distinction between the foreground and background pixels is made on texture, rather than average pixel value.

Hueckel [41,42] proposed a technique whereby circular regions of the image are examined to see if edges or lines run through them. Kindelan and de Lezo [50] claim to have used his technique successfully in tracking arteries in angiograms.

Granlund, Knutsson and Wilson [37,38,52-54,124] generalised the idea of edges and introduce the concept of "image orientation", which they are able to determine and use in enhancing images.

Nakagawa and Rosenfeld [74] suggest a method for reducing the level of noise in edge value images. They suggest using some edge detector, like those described above, to generate a preliminary "edge image", then threshold the image, using one of the thresholding techniques discussed in chapter 3, and then join up those edges detected that coincide with borders in the thresholded image, and ignore the rest.

2.3.4 Digital Subtraction

In the mid 1970s and early 1980s, much research was done in the field of digital image subtraction, particularly as applied to medical imaging [16,25,67,71,78,98,106,107].

Radiologists routinely failed to detect approximately 30% of abnormalities in angiograms [129], and so some technique was sought to improve this situation. At the same time, radiologists wished to decrease the amount of radio-opaque dye that needed to be administered to patients so that the arteries, blood vessels and heart chambers would show up in the angiograms. An new field, known as "digital subtraction angiography", emerged.

In angiography, traditionally, radiologists would inject a radio-opaque dye into the artery or heart chamber of interest and then take an X-ray image of that part of the patient. The arteries and chambers of interest would show up in these X-rays, but so would bones and some tissue structures. These often obscured what the radiologists were interested in, viz. the arteries and heart chambers. A technique was sought that would remove the unnecessary and unwanted structures in the images.

The solution was as follows:

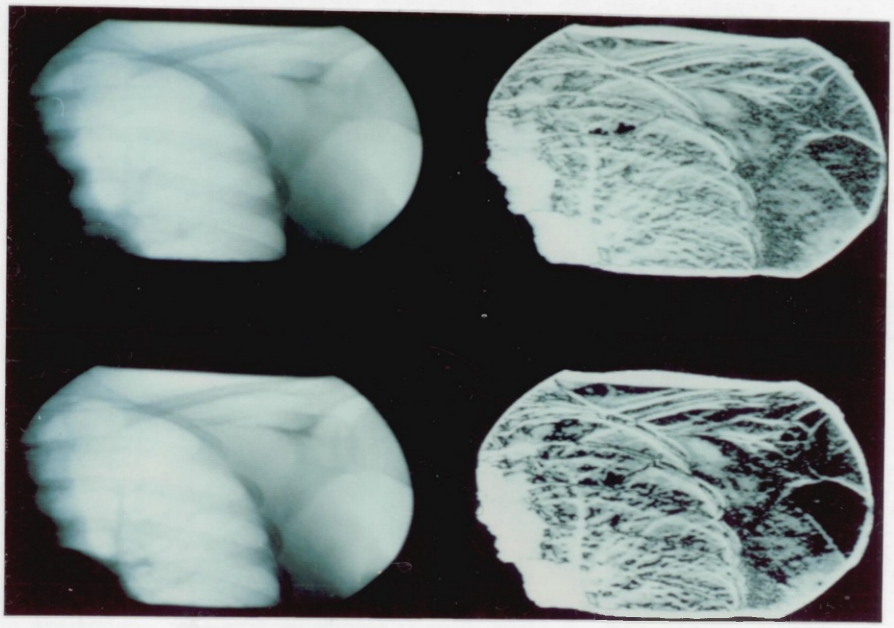
(i) Take an X-ray of the patient.
(ii) Inject the radio-opaque dye into the patient.
(iii) Take another X-ray of the patient.
(iv) Subtract the first image from the second, and display the result.

(1)

The theory was that the background structures would appear in both X-ray images and so would not appear in the subtracted image. Only the structures of interest (viz. those containing the dye) would be shown in the subtracted image. This approach enjoyed much

success initially, but one serious limitation emerged: the two images needed to be aligned perfectly, otherwise motion artifacts will appear in the subtracted image. Several researchers have attempted to find ways of "undoing" motion artifacts caused by patient motion [8,12,55,68,114], but the techniques are still quite primitive, and patient motion remains a serious problem.

Figure 2.9 shows some results of using digital subtraction.



Digitally subtracted images: (a) and (c) show x-ray photographs before and after the injection of radio-opaque dye. (b) and (d) show the results of subtracting one of the images from the other linearly and logarithmically, respectively. The absolute values of the pixel differences were recorded, and images (b) and (d) have had their pixel values histogram equalised.

Figure 2.9

Some authors have suggested that logarithmic, instead of linear, subtraction be used in step (1iv) above [5,10,56,89]. That is:

$$f_s(x,y) \triangleq \text{Log}(f_2(x,y) + 1) - \text{Log}(f_1(x,y) + 1) \qquad (2a)$$

instead of:

$$f_s(x,y) \triangleq f_2(x,y) - f_1(x,y) \qquad (2b)$$

where: $f_1(x,y)$ is the pre-injection image taken in step (1i)

$f_2(x,y)$ is the post-injection image taken in step (1iii)

$f_s(x,y)$ is the subtracted image obtained in step (1iv).

This means that, because of the exponential nature of the amount of energy absorbed by the tissue, as a function of its thickness, the intensity of a pixel in the resultant subtracted image would be a linear function of the amount of radio-opaque dye passed through.

Digital subtraction can also be used in data compression. During an angiogram lasting 10 seconds, as many as 250 images may be captured. Storing these images poses a serious problem: there is too much data to store many frames digitally, and the quality of currently available analogue video storage devices is poor, giving a poor signal to noise ratio when the images are retrieved.

Some authors [11,90] have proposed the following solution:

(i) Store the first frame digitally (accurately).

(ii) Digitise all subsequent frames and subtract the first frame from each of them.

(iii) The images obtained in (ii) should have a much smaller range of pixel values, and so the image's dynamic range can be stretched, and then be converted into analogue form and be stored on a video tape. (3)

(iv) When the sequence of frames is to be reconstructed, the video frames are redigitised, the image's dynamic range is compressed to what it was, and the result is added to the first frame, referred to in (i).

This technique has the advantage that the errors introduced by the video recording equipment are reduced by the amount indicated by the scaling factor in step (3iii) above.

2.3.5 Thresholding

When an analogue image is digitised, a potentially infinite number of intensities is reduced to a relatively small finite number, which is limited in magnitude by the number of bits used to represent each pixel's value. This means that certain "cut-off"

intensities need to be specified and all regions in the analogue image whose intensities lie between the same two adjacent cut-off values will be assigned the same value in the digitised image. This process is a special case of thresholding in which the cut-off values are known as thresholds or threshold values.

More generally, thresholding is the determining of N ($\geq$ 1) distinct threshold values, T_i (i=1..N), such that these values divide the range of permissible pixel values, [0,L], into (N+1) subclasses and

$$\left.\begin{array}{ll} \text{(i)} & T_i \leq T_{i+1} \\ \text{(ii)} & 0 \leq T_1 \\ \text{(iii)} & T_N \leq L \end{array}\right\} \quad (1)$$

Figure 2.10 shows some results of thresholding images.

The N threshold values can either be constant for the whole image (this is known as "global thresholding"), or they may be given by a set of N non-overlapping functions (known as "variable thresholding").

The concept of thresholding is not dificult, but what is difficult is determining whether to use global or variable thresholding, and what the threshold values or functions should be. Chapter 3 contains a detailed discussion of how threshold values are calculated.

(a)



(b)

(c)

(d)

Thresholded images: Images (a), (b) and (c) show the results of thresholding the image in (d) into 4, 8, and 16 permissible pixel values repectively, using global thresholds.

Figure 2.10

2.4 Quantitative Measures of Image Recovery

Several quantitative measures of the degree of image recovery resulting from the application of image recovery techniques, have been developed by previous workers. However, for the degree of recovery to be measured accurately, the original image needs to be known.

These techniques are often used, for example, to measure the degree of image recovery when known images are compressed (i.e. stored in a more compact form) and then restored, as in these cases, both the original and the reconstructed images are known.

Following is a description of some of the degree of image recovery measures described in the literature, together with a discussion of how applicable the methods are when comparing binary images, the problem addressed in chapter 10 of this thesis.

Let the original image be given by $g(x,y)$ and the restored (reconstructed) image be given by $f(x,y)$, then the difference between the two (i.e. the error, $e(x,y)$, in the reconstruction) can be determined from:

$$f(x,y) = g(x,y) + e(x,y) \tag{1}$$

Clearly, for binary images, we can assume that:

$$\left.\begin{array}{l} \text{(i) } f(x,y) \text{ and } g(x,y) \text{ can take values of 0 or 1} \\ \text{(ii) } e(x,y) \text{ can take values of -1, 0 or 1} \end{array}\right\} \tag{2}$$

For an NxN pixel image, the mean square error, E_{ms}, is given by [7,34,109]:

$$E_{ms} = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^2(x,y) \qquad (3)$$

and the root mean square error, E_{rms}, is given by [34]:

$$E_{rms} = (E_{ms})^{\frac{1}{2}} \qquad (4)$$

For binary images, E_{ms} is the fraction of the pixels in the original image that are misclassified in the reconstructed image. This measure is often used to evaluate restored binary images [1], and therefore it will also be used as a measure in the evaluation in chapter 10. E_{rms} is derived from E_{ms}, but is not usually used to compare binary images.

Similarly, the mean square signal to noise ratio, SNR_{ms}, is given by [34]:

$$(SNR)_{ms} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g^2(x,y) \; / \; \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^2(x,y) \qquad (5)$$

and the root mean square signal to noise ratio, SNR_{rms}, is given by [34]:

$$(SNR)_{rms} = (SNR)_{ms}^{\frac{1}{2}} \qquad (6)$$

For binary images, $(SNR)_{ms}$ gives the ratio of foreground pixels in the original image to the number of misclassified pixels in the restored image. The higher this value, the more successful the

restoration is presumed to be. This measure will tend to ∞ as the number of misclassified pixels tends to 0, but it considers misclassifying a pixel in an image dominated by the background to be more serious than misclassifying a pixel in an image that is dominated by the foreground. It is not clear that this assumption is valid in our application and so the error measure is not used in this work.

Gonzalez and Wintz also give an alternative signal to noise ratio measure, SNR_p, which is given by [34]:

$$(SNR)_p = \{[\text{peak value of } g(x,y)]^2 \; / \; E_{rms}\}^{\frac{1}{2}} \qquad (7)$$

For binary images containing at least one foreground pixel, $(SNR)_p$ gives the reciprocal of the fourth root of the fraction of the pixels in the restored image that have been misclassified. While this measure does give an indication of the error in a binary image, no reason was found to suggest that it is a better measure than just the fraction of the pixels that have been misclassified in the restored image. Therefore, the measure is not used in the evaluation given in chapter 10.

Some researchers "normalise" the mean square error by dividing it by the variance of the original image. They define the normalised mean square error, E_{nms}, as [27,32,96]:

$$E_{nms} = E_{ms} \; / \; (\text{variance of original image}) \qquad (8)$$

Binary images containing mostly foreground or background pixels will have a lower variance than binary images with approximately

equal numbers of foreground and background pixels. Therefore, misclassifying a pixel in a binary image which is dominated by one of the regions is considered more serious than misclassifying a pixel in a binary image with a more even distribution of pixel values. In chapter 10, it is explained why this is considered to be a desirable property of the measure and therefore it will be used in the evaluation given in that chapter.

Farrelle and Jain, measuring the signal to noise ratio in dB, define it as [27]:

$$SNR = 10.\log_{10}\left(\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} g^2(x,y) \Big/ \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} e^2(x,y) \right) \quad (9)$$

This measure is similar to $(SNR)_{ms}$, as given in (5), and the objections to $(SNR)_{ms}$ apply to SNR as well. Therefore, SNR as defined above, is not used as a measure in this work.

Biemond et al describe the use of two other measures, namely [7]:

(i) the square error between the auto-correlations of the original image and that of the reconstructed image

(ii) the Cramér-Rao bound, which is a lower bound for the E_{ms}

(10)

The Cramér-Rao bound is not used in this application as the E_{ms}, which is used, can be computed directly and simply. The square-error between the auto-correlations has merit as a measure,

in that a high value would indicate that the images were quite different, but it is computationally expensive and it is not clear that this expense is justified in our application when more easily computed measures yield meaningful results.

Loo et al describe five measures of similarity, namely [65]:

(i) Amplitude SNR, which is given by:

$$SNR_A^2 = \frac{(\text{Density of the peak signal})^2}{\text{(variance of the noise)}} \qquad (11)$$

(ii) Quadratic SNR, which is given by:

$$SNR_Q^2 = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y)^2 \; / \; \text{(variance of the noise)} \qquad (12)$$

(iii) Statistical decision theory SNR, which is given by:

$$SNR_S^2 = \left(\sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y) \right)^2 / \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y).e(x,y) \qquad (13)$$

(iv) Matched filter SNR, which is given by:

$$SNR_M^2 = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \begin{cases} 0 & \text{if } e(x,y) = 0 \\ (g(x,y)^2 \; / \; e(x,y)) & \text{otherwise} \end{cases} \qquad (14)$$

(v) Information capacity SNR, which is given by:

$$SNR_C = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \begin{cases} 0 & \text{if } e(x,y) = 0 \\ \log_2(1+g(x,y)^2)/e(x,y) & \text{otherwise} \end{cases} \qquad (15)$$

For binary images containing at least one foreground pixel, SNR_A^2 is the reciprocal of the variance of e(x,y). While this measure is

not without merit, in that a low value would indicate a high degree of error, it does not reflect, for example, the number of misclassified pixels. An image containing exactly one foreground pixel but which is classified as having only foreground pixels in the restored image will have the same SNR_A^2 value as if the restored image had only one misclassified pixel. Therefore, this measure is not used in this work. SNR_Q^2 has the same property and so it is not used either.

SNR_S^2 gives:

$$\frac{-\left\{\begin{array}{l}\text{(the number of foreground pixels in } g(x,y))^2\end{array}\right\}}{\left\{\begin{array}{l}\text{(the number of foreground pixels misclassified as being background pixels in the restored image)}\end{array}\right\}} \qquad (16)$$

As this measure ignores the number of background pixels classified as being foreground pixels in the restored image, it is also not used in this study.

SNR_M^2 gives:

$$-\left\{\begin{array}{l}\text{(the number of foreground pixels classified as background pixels in the restored image)}\end{array}\right\} \qquad (17)$$

as does SNR_C. As neither of these measures considers the number of background pixels classified as foreground in the restored image, neither of them are used in this study.

Venot et al, in describing techniques for aligning images, define two measures of image similarity for alignment purposes [113]:

(i) when comparing two images, $f_1(x,y)$ and $f_2(x,y)$, with a relatively high level of noise, which has zero mean and a symmetric distribution, they define:

$$D(x,y) = f_1(x,y) - f_2(x,y) \qquad (18)$$

Then the stochastic similarity criterion (SSC) is defined as the number of sign changes in $D(x,y)$, scanning along the rows or columns. If the images align, then the SSC measure should be high; if not it should be lower.

(ii) when comparing two images, $f_1(x,y)$ and $f_2(x,y)$, with negligible noise, they first define a small number q, and then define:

$$\left.\begin{aligned} f_3(x,y) &= f_2(x,y) + q \quad \text{if } x+y \text{ is even} \\ f_3(x,y) &= f_2(x,y) - q \quad \text{if } x-y \text{ is odd} \end{aligned}\right\} \qquad (19)$$

Then $D(x,y)$ is defined as:

$$D(x,y) = f_1(x,y) - f_3(x,y) \qquad (20)$$

The deterministic sign change (DSC) criterion is then defined as the number of sign changes in $D(x,y)$, scanning along the rows or columns. If the images align perfectly, then DSC = N-1, where N is the number of pixels in each of the images, otherwise it will be some smaller value.

As the images segmented in this study are aligned, these measures will yield values that can be used to give rough estimates of the number of misclassified pixels in a restored image. As the number of misclassified pixels can be computed easily and precisely, these measures are not used.

Other authors have suggested that the number of boundary pixels in [118] or the first N moments of [112] segmented images may be significant. These measures have merit in that if the first N moments or the number of boundary pixels is very different for two images, that would indicate that the images are different. However, if these values are similar for two images, then the two images are not necessarily similar. While there are objections to these measures, they will be used in chapter 10, for comparative purposes.

The use of so many different, although often quite similar, measures suggests that there is no one accepted measure of the quality of an image or any reconstruction or restoration thereof. The problem is compounded further when the images are to be viewed by people. This is because the criteria used by people are largely subjective and are as yet little understood. This is supported by Gonzalez and Wintz [34]:

> "When the output images are to be viewed by people, ... it is more appropriate to use a subjective fidelity criterion corresponding to how good the images look to human observers. The human visual system has peculiar characteristics so that two pictures having the same amount of rms error may appear to have drastically different visual qualities."

An example of a large scale subjective assessment of the quality of images is given by Fredendall and Behrend [29]. After a long investigation, the Radio Corporation of America (RCA) were unable to find a satisfactory objective measure of the quality of television pictures. They resorted to using a subjective measure determined by a panel of viewers.

In chapter 9, the results of applying the segmentation technique developed during the course of this research will be presented, without any attempt being made to quantify the effectiveness of the technique. The conclusions drawn are based on a subjective analysis of the results.

In chapter 10, known binary images will be degraded and then restored, using the technique developed during the course of this research, and these results will be evaluated using quantitative measures similar to those described above.

CHAPTER 3

THRESHOLD EVALUATION

3.1 Introduction

The image segmentation technique developed during the course of this research has threshold evaluation as one of its stages. This chapter reviews several different approaches to threshold evaluation, and assesses the suitability of each of the families of techniques for various classes of image.

As stated in section 2.3.5, thresholds in image processing are the one or more values which divide the pixels' brightness range (possible pixel values) into regions. The pixels whose values lie within each subrange could be assigned a particular value; if so, the image has been thresholded.

A variety of different techniques for calculating thresholds have been proposed, and the choice of which one to use in a particular application will depend on what is known, suspected or assumed a priori about the image to be thresholded. In addition, it is possible that some trial and error may be involved in finding the appropriate method.

There are two different types of thresholding:

Global thresholding: in which the threshold values are constant for the whole image.

Variable thresholding: in which the threshold values are given as non-overlapping functions.

Each pixel has its own set of threshold values.

Clearly, global thresholding is a special case of variable thresholding in which the threshold surfaces are flat and parallel to the image plane.

Within this classification, variable thresholding techniques can be subdivided further, as follows:

Moving window techniques: with these techniques, the calculation of each pixel's threshold values is calculated from the values of the pixels within a window generally of fixed size and shape around that pixel.

Other techniques: this category includes all variable thresholding techniques that are not moving window techniques. They usually involve the calculation of threshold values for a small number of pixels, and then interpolating threshold surfaces through these values thus defining threshold values for all the pixels in the image.

An overview of early thresholding techniques can be found in Weszka [116].

Sections 3.3 to 3.9 describe some of the known classes of thresholding techniques, with examples of the techniques in each class, and a discussion of the conditions under which each

approach is most appropriate. Section 3.10 discusses the application of relaxation techniques to threshold evaluatuion, section 3.11 discusses techniques for generating multiple threshold surfaces and associated problems, and section 3.12 assesses the suitability of the techniques described in sections 3.3 to 3.9 to this work.

3.2 Using a priori Information

Clearly, the more information available a priori about an image and the objects of interest therein, the greater the possibility of finding those objects, if the a priori knowledge is used in the segmentation process. For example, one might know that there is exactly one object of interest in the image, that the object is brighter than its background and the proportion of the image that it occupies. In that case, the object of interest can be extracted quite easily by selecting the appropriate number of brightest pixels in the image. However, this amount of information is rarely available a priori.

Most threshold evaluation techniques make assumptions about the images for which the thresholds are to be calculated, and often there is a trade off between using a powerful technique that requires much a priori information (that will need to be estimated or assumed), or a less powerful technique that requires less a priori information.

The amount of a priori information required for each of the techniques described in this chapter will be given when the techniques are discussed.

3.3 Average and Median Thresholding

Average and median thresholding are perhaps the simplest (and most primitive) forms of thresholding.

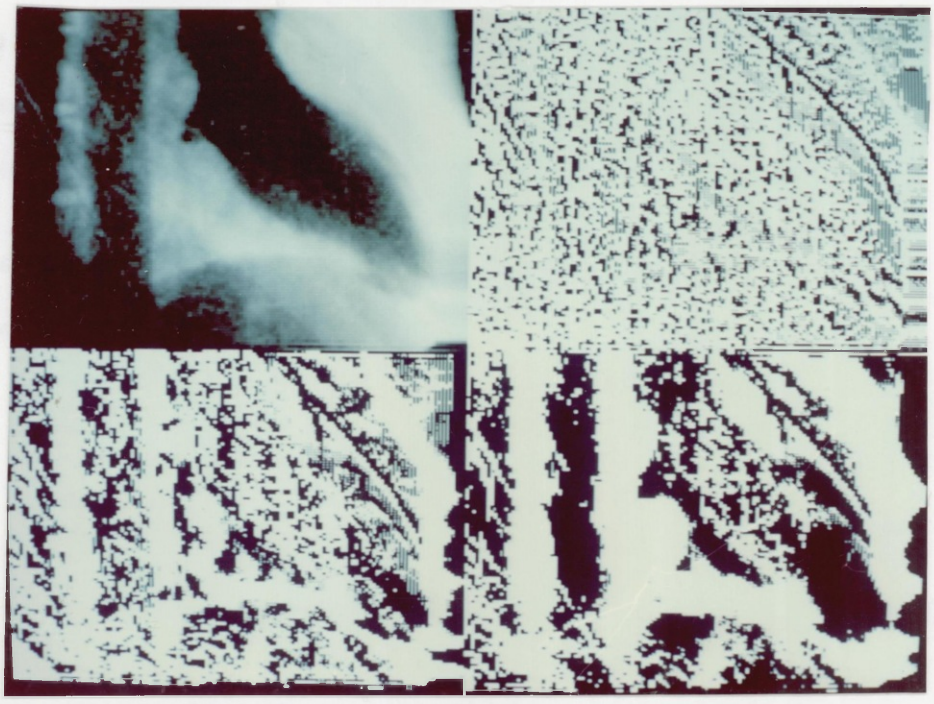
With these techniques, the average and median of the pixel values in the region of interest are used as thresholds, respectively. The assumption is made that half of the pixels in the region belong to the object (or objects) of interest, and the other half belong to the background.

Average and median values are used very rarely as global thresholds. They are usually used in moving window, variable thresholding techniques.

Fernando [28] discusses these techniques at great length.

Figure 3.1 shows some results of applying moving window median thresholding. Chapter 9 contains further examples.

(a)



(b)

(c)

(d)

Median thresholding: Images (b), (c) and (d) show the results of median thresholding image (a) with square windows of radii 4, 16 and 32 pixels respectively.

Figure 3.1

3.4 Valley-seeking Techniques

With these techniques, a natural division of the histogram is sought. It is assumed that the histogram of the image has two (or more) distinct modes as in figure 3.2, corresponding to the object and background pixels, separated by a valley. The bottom of this valley is sought, which is then taken as the threshold value.

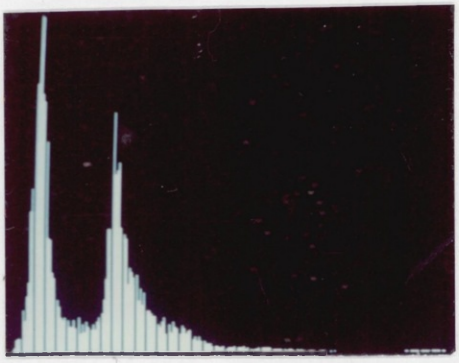
The simplest approach would be to find the lowest point on the histogram. Unfortunately, the following problems arise:

(i) This value may not lie between the modes

(ii) The only way of being certain that it does lie between the modes is by determining the means of the modes first, a task almost as difficult as finding the valley itself. (1)

Furthermore, unless the foreground and background contain approximately equal numbers of pixels and their modes are well separated, the histogram could appear to contain only a single mode with a shoulder on one side. In this case, there is no valley to find.

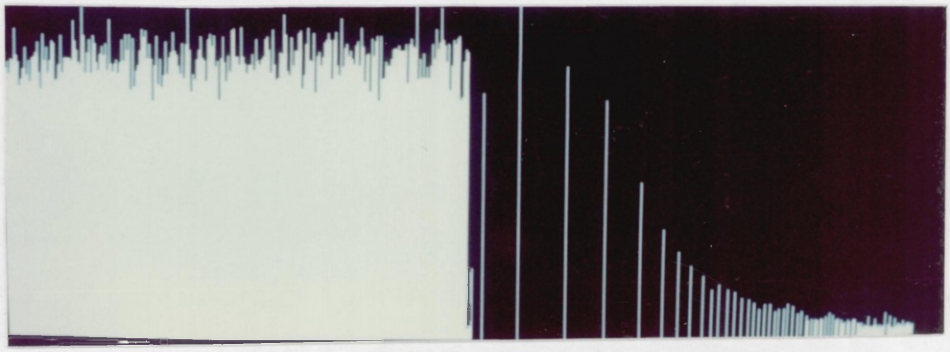
Even if the histogram is bimodal, if the lowest point between the means of the two modes is sought, the wrong valley bottom could still be found, as the histogram may not be smooth (see figure 3.3).



Bimodal histogram: Histogram showing two modes separated by a clear, deep, distinguishable valley.

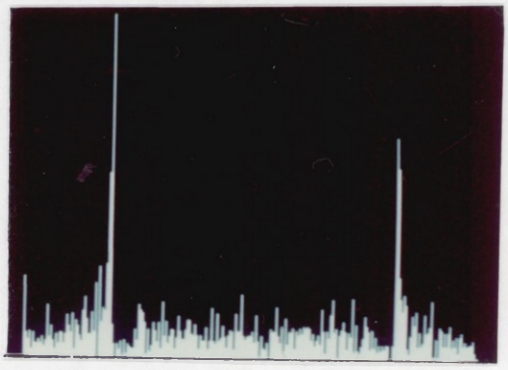
Figure 3.2

(a)



(b)

(c)



Non-bimodal histograms: Typical non-bimodal histograms from which threshold values sometimes need to be computed.

Figure 3.3

The above approach relies on the histogram being smooth and bimodal with the modes being separated by a clearly distinguishable valley. If the histogram is not smooth, it may be necessary to low-pass filter (smooth) it. However, this is difficult, as if the modes are not distinct, they may be filtered away, while if the histogram is very "jagged", it may not be smoothed enough.

Wu, Hong and Rosenfeld [126] suggest a method for smoothing the histogram using quadtrees. Their technique seeks to find regions of the image with nearly constant brightness, smooth them, and then calculate the threshold value from the histogram of the smoothed image, using a valley-seeking technique. This value is then used to threshold the original image.

Their technique is as follows:

(i) Calculate the variance of the pixel values in the image.

(ii) Divide the image into four quadrants, and calculate the variance of the pixel values in each quadrant.

(iii) For a particular quadrant, if the variance of the pixel values is less than or equal to one third of that figure for the whole image, then assign the quadrant's average pixel value to all pixels in the quadrant, otherwise, divide

the quadrant into four further subquadrants and repeat step (iii) using each of the sub quadrants in turn. } (2)

(iv) Compute the histogram for the "smoothed" image generated in step (iii).

(v) Calculate a threshold value, using a valley-seeking technique, from the histogram computed in step (iv).

(vi) Threshold the original image using the threshold value calculated in step (v).

Weszka, Nagel and Rosenfeld [117] propose thresholding an image with a threshold calculated, using a valley-seeking technique, from the histogram of those pixels in the image with high Laplacian edge value. The justification for the technique, is that if the image contains only one small object, then the histogram will not contain two modes of approximately the same size. However, the Laplacian operator, being a second order edge detector, should yield high values for background pixels that are adjacent to object pixels, and object pixels that are adjacent to background pixels, and for no other pixels. As there are approximately equal numbers of each, the histogram should be bimodal with two approximately equally sized modes.

Weszka and Rosenfeld [119] propose using weighted projections of (grey level, edge value) scatter plots, onto the grey level axis, for valley deepening. They evaluate several different weightings for the projections and methods for calculating edge values. They also compare their results with those of other methods. The justification for this approach is almost totally contrary to that given above. It is that near background/foreground boundaries, there will be blurring, and so there will be a gradual change in pixel value at the boundary, rather than the step one would hope for. Therefore, they claim that the pixels close to the boundary (high edge value) should carry a lower weighting in the histogram than pixels not near the boundary, where blurring is not so noticeable.

Ahuja and Rosenfeld [2] propose a similar technique, thresholding an image using a threshold calculated from the histogram of only those pixels corresponding to points close to or far from the main diagonal of the (grey level, neighbour's grey level) scatter plot. The former should yield histograms with deepened valleys, while the latter should yield histograms with spikes corresponding to the valleys in the histogram of the original image.

Rosenfeld [91] proposed a technique of histogram peak sharpening and valley deepening by forming a new, modified histogram in which the values would be functions of the corresponding values in the original histogram, and the three values on either side of it.

That is:

$$H_i' = f(H_{i-3}, H_{i-2}, H_{i-1}, H_i, H_{i+1}, H_{i+2}, H_{i+3}) \quad (3)$$

where: H_i is the i^{th} value in the original histogram

H_i' is the i^{th} value in the modified histogram.

3.5 Distribution Assuming Techniques

With these techniques, the values of the pixels belonging to each of the classes to be extracted by the segmentation (thresholding), are assumed to have a given distribution.

For simplicity, the rest of this section will deal with the case of bilevel thresholding, where one threshold value (or function) is used to threshold the image into "foreground" and "background". The generalisation to more than one threshold follows easily.

These techniques can be classified as follows:

(1)

(i) Those in which a certain fraction of the image's pixels are assumed to belong to the foreground and the rest are assumed to belong to the background.

(ii) Those in which the values of the pixels in the foreground and background, respectively, are assumed to have particular distributions.

The techniques falling into (1i) above are variants of the median and average thresholding techniques described in section 3.3 and are rarely applied globally to images. They are usually implemented as moving window techniques.

The techniques falling into (1ii) are usually more demanding computationally than those in (1i), and they are usually used to calculate either global thresholds or threshold values for each of

a small number of regions in the image. The rest of this section describes this family of techniques.

The approach is as follows:

(i) Calculate the image's (or region's) histogram.

(ii) From the histogram, calculate the Probability Density Function (PDF) for the pixels' intensities.

(iii) The PDF is the weighted sum of the distributions of the intensities of the pixels in the foreground and background, respectively. The statistics for these two distributions and the proportion of background and foreground pixels in the image (region) need to be calculated so that the weighted sum of the distributions best models the calculated PDF.

(iv) Having calculated these statistics, the threshold value that would minimise the probability of misclassifying a pixel, is calculated.

(2)

Section 2.2.2 discussed (2i) and (2ii). Following is description of how steps (2iii) and (2iv) are performed.

Let the following notation be adopted:

	Background	Foreground	
Mean	μ_b	μ_f	}
Standard Deviation	σ_b	σ_f	}
Density Distribution Function	$f_b(z)$	$f_f(z)$	} (3)
Fraction of Image	P_b	P_f	}

Without loss of generality, we may assume that

$$\mu_f > \mu_b \tag{4}$$

By elementry statistics, the image has the following characteristics:

$$\left.\begin{array}{ll} \text{PDF} & : P_f.f_f(z) + P_b.f_b(z) \\ \text{Mean} & : P_f.\mu_f + P_b.\mu_b \\ \text{Variance} & : P_f.\sigma_f^2 + P_b.\sigma_b^2 + P_f.P_b.(\mu_f-\mu_b)^2 \end{array}\right\} \tag{5}$$

and

$$P_b + P_f = 1 \tag{6}$$

The image $f(x,y)$ is to be thresholded at t, giving a new, thresholded image $f_t(x,y)$ such that:

$$f_t(x,y) = \begin{cases} \text{foreground, if } f(x,y) > t \\ \\ \text{background, if } f(x,y) \leq t \end{cases} \qquad (7)$$

The probability of classifying a background pixel as being an object pixel is given by:

$$Q(t) = \int_t^{\infty} f_b(z).dz \qquad (8a)$$

and the probability of classifying an object pixel as being a background pixel is given by:

$$1-P(t) = \int_{-\infty}^{t} f_f(z).dz \qquad (8b)$$

From (8a) and (8b), it is seen that the probability of misclassifying a pixel is given by:

$$M(t) = P_f.(1-P(t)) + P_b.Q(t) \qquad (9)$$

To find the threshold that minimises (9), and thus minimises the probability of misclassifying a pixel, we need to differentiate $M(t)$, with respect to t, set the result equal to zero, and solve. Before solving, we get:

$$P_b.f_b(t) = P_f.f_f(t) \qquad (10)$$

The solution of (10) depends on P_b, P_f, $f_b(t)$ and $f_f(t)$. To

illustrate the technique more fully, as is done most often when no a priori information to the contrary is available, the values of the foreground and background pixels are assumed to be distributed Normally, viz.

$$\left.\begin{aligned} f_f(t) &= \frac{1}{\sigma_f.\sqrt{(2.\pi)}} \cdot \exp\left(\frac{-(t-\mu_f)^2}{2.\sigma_f^2}\right) \\ f_b(t) &= \frac{1}{\sigma_b.\sqrt{(2.\pi)}} \cdot \exp\left(\frac{-(t-\mu_b)^2}{2.\sigma_b^2}\right) \end{aligned}\right\} \quad (11)$$

Substituting (11) in (10):

$$\frac{P_b}{\sigma_b.\sqrt{(2.\pi)}} \exp\left(\frac{-(t-\mu_b)^2}{2.\sigma_b^2}\right) = \frac{P_f}{\sigma_f.\sqrt{(2.\pi)}} \exp\left(\frac{-(t-\mu_f)^2}{2.\sigma_f^2}\right)$$

$$\Rightarrow \ln(P_b) - \ln(\sigma_b) + \left(\frac{-(t-\mu_b)^2}{2.\sigma_b^2}\right) = \ln(P_f) - \ln(\sigma_f) + \left(\frac{-(t-\mu_f)^2}{2.\sigma_f^2}\right)$$

$$\Rightarrow \frac{(t-\mu_f)^2}{2.\sigma_f^2} - \frac{(t-\mu_b)^2}{2.\sigma_b^2} = \ln(\sigma_b) + \ln(P_f) - \ln(\sigma_f) - \ln(P_b)$$

$$\Rightarrow \sigma_b^2.(t-\mu_f)^2 - \sigma_f^2.(t-\mu_b)^2 = 2.\sigma_f^2.\sigma_b^2.\ln\left(\frac{\sigma_b.P_f}{\sigma_f.P_b}\right)$$

$$\Rightarrow (\sigma_b^2-\sigma_f^2).t^2 + 2.(\sigma_f^2.\mu_b-\sigma_b^2.\mu_f).t +$$

$$\left.\sigma_b^2.\mu_f^2 - \sigma_f^2.\mu_b^2 - 2.\sigma_f^2.\sigma_b^2.\ln\left(\frac{\sigma_b.P_f}{\sigma_f.P_b}\right) = 0\right\} \quad (12)$$

The above (12) is a simple quadratic that has at most two distinct solutions for t, exactly one of which will lie between the

respective means of the values of the foreground and background pixels, viz. $\mu_b < t < \mu_f$.

However, before this can be done, values need to be defined for μ_b, μ_f, σ_b, σ_f and P_b. The value of P_f can be derived from P_b. As these will not normally be known a priori, some method for determinining, or at least estimating, them is needed. There are two approaches to the problem:

(i) Make an initial estimate of the five values above, calculate the "difference" between the image's histogram and the "ideal" histogram, viz. the PDF, as defined in (5) and (11), using the estimated values for μ_b, μ_f, σ_b, σ_f and P_b and then use an optimisation technique to determine the values for μ_b, μ_f, σ_b, σ_f and P_b that will minimise this difference.

(ii) Use some other technique to calculate an initial estimate of the threshold, t, assume that all the values less than t correspond to pixels in the background and all the values greater than t correspond to pixels in the foreground, compute the values of μ_b, μ_f, σ_b, σ_f and P_b based on this assumption, calculate the new estimate of the threshold, using (12), and then repeat this process until the method converges to a solution, or until a specified number of iterations have been performed. (13)

The "difference" in (i) above is usually taken to be either the area between the two curves or the sum of the squares of the differences between the histogram values for each pixel value, viz.

$$D = \sum_{i=0}^{L} |H_i - P_i| \qquad (14a)$$

or

$$D = \sum_{i=0}^{L} (H_i - P_i)^2 \qquad (14b)$$

where: D is the distance measure

L is the largest permissible pixel value

H_i is the fraction of the pixels in the image with value i

P_i is the fraction of the pixels in the image that would have value i if the values of the pixels in the background and foreground were distributed Normally.

Chow and Kaneko [15] and Nakagawa and Rosenfeld [75] have experimented with (13i) and present results in their publications.

3.6 Moment Preserving Techniques

Tsai [112] presents a method of thresholding an image into N regions. The user need only supply this number N, and the technique will not only determine the threshold values, but will also determine the intensities that the individual segments should have in the thresholded image.

The algorithm needs to determine N intensities for the segments in the thresholded image and (N-1) threshold values. This means that there are (2N-1) variables whose values are as yet unknown. Tsai computes values for them by imposing the (2N-1) constraints that the moments M_i, i=1..(2N-1), should be the same in the thresholded image as they were in the original image. By noting that the first moment, M_0 is always equal to one, we see that the first (2N) moments are preserved in the thresholded image.

The method, correcting an error that Tsai made in step (6) below, is as follows.

Let the i^{th} moment, M_i, of the image f(x,y), be defined, as it was in section 2.3.4, as:

$$M_i = \frac{1}{n} \sum_{j=0}^{L} (n_j . j^i)$$

$$= \sum_{j=0}^{L} (P_j . j^i) \qquad (1)$$

where: L is the largest permissible pixel value

n is the total number of pixels in the image

n_j is the number of pixels in the image with intensity j

P_j is the fraction of the pixels in the original image that have intensity j.

Clearly, given any image f(x,y), we can calculate M_i for any $i \geq 0$. Equally clearly:

$$M_0 = 1 \tag{2}$$

Bilevel thresholding will be considered first, that is, the case where the image is to be thresholded into two regions. The thresholded image is to contain P_0' of its pixels with intensity z_0, and P_1' with intensity z_1, where P_0' and P_1' are fractions of the thresholded image. Without loss of generality, it may be assumed that $z_0 < z_1$. Then, clearly, in the original image, f(x,y), P_0' of the pixels are below the threshold value, t, and P_1' are above it.

From (1), it can be seen that the first four moments, M_i', of the thresholded image, f'(x,y), are defined as:

$$M_i' = \sum_{j=0}^{1} (P_j' . z_j^i) \qquad ,i=0,1,2,3 \tag{3}$$

As stated earlier, the first four moments are to be preserved in the thresholded image, so

$$M_i' = M_i \qquad ,i = 0,1,2,3 \tag{4}$$

The equations in (4) are equivalent to

$$
\left.
\begin{aligned}
P_0'.z_0^{\,0} + P_1'.z_1^{\,0} &= M_0 \\
P_0'.z_0^{\,1} + P_1'.z_1^{\,1} &= M_1 \\
P_0'.z_0^{\,2} + P_1'.z_1^{\,2} &= M_2 \\
P_0'.z_0^{\,3} + P_1'.z_1^{\,3} &= M_3
\end{aligned}
\right\} \qquad (5)
$$

From (1), the values of M_i, i=0,1,2,3, in (5) are known, therefore, we are left with four unknowns and four equations.

In general, an image f(x,y) will be thresholded into N pixel classes, viz. (N-1) threshold values, t_i, i=0,..,(N-2), will need to be determined. In doing so, the first 2N moments of f(x,y) will be preserved in the thresholded image.

Let z_i denote the intensity of the i^{th} pixel class. Then, as in (5), we get:

$$
\left.
\begin{aligned}
P_0'.z_0^{\,0} + P_1'.z_1^{\,0} + .. + P_{N-1}'.z_{N-1}^{\,0} &= M_0 \\
P_0'.z_0^{\,1} + P_1'.z_1^{\,1} + .. + P_{N-1}'.z_{N-1}^{\,1} &= M_1 \\
\cdot \qquad\quad \cdot \qquad\quad \cdot \qquad\quad \cdot & \\
\cdot \qquad\quad \cdot \qquad\quad \cdot \qquad\quad \cdot & \\
P_0'.z_0^{\,2N-1} + P_1'.z_1^{\,2N-1} + .. + P_{N-1}'.z_{N-1}^{\,2N-1} &= M_{2N-1}
\end{aligned}
\right\} \qquad (6)
$$

This set of simultaneous equations can be solved to get values for

all the P_is and z_is. However, the solution is non-trivial as the equations are non-linear, and so Tsai suggests the following algorithm:

(i) Solve the following linear equations to obtain values for C_i, i=0,1,...,(N-1) :

$$\left.\begin{array}{l} C_0.M_0 + C_1.M_1 + \ldots + C_{N-1}.M_{N-1} = -M_N \\ C_0.M_1 + C_1.M_2 + \ldots + C_{N-1}.M_N = -M_{N+1} \\ C_0.M_2 + C_1.M_3 + \ldots + C_{N-1}.M_{N+1} = -M_{N+2} \\ \quad . \qquad\quad . \qquad\qquad . \qquad\quad . \\ \quad . \qquad\quad . \qquad\qquad . \qquad\quad . \\ \quad . \qquad\quad . \qquad\qquad . \qquad\quad . \\ C_0.M_{N-1} + C_1.M_N + \ldots + C_{N-1}.M_{2N-2} = -M_{2N-1} \end{array}\right\} \quad (7)$$

(ii) The Z_is can now be calculated as being the roots of the equation

$$Z^N + C_{N-1}.Z^{N-1} + C_{N-2}.Z^{N-2} + \ldots + C_1.Z + C_0 = 0 \qquad (8)$$

(iii) Substitute the N values of Z obtained in (8) (call them z_i, i=0,...N-1) in the first N equations of (6), to obtain values for P_i.

(iv) From these P_is, calculate the values of t_i.

The above will define values for t_i, i=0..(N-2), in step (iv) and z_i, i=0..(N-1), in step (ii).

3.7 Statistical Techniques

Several statistical approaches to image segmentation have been proposed, many of which are ad hoc. One technique that is not ad hoc is the one proposed by Otsu [79].

The technique defines a segmentation of an image into two regions such that the values of the pixels in each region are as "similar" as possible and the values of pixels in different regions are as "dissimilar" as possible. Three measures of "similarity" are given and it is shown that the use of any of them will give the same segmentation of a given image.

Otsu's technique is as follows for bilevel thresholding.

Let a given image contain N pixels that are represented by (L+1) pixel intensities, [0..L].

Define:

$$\left.\begin{array}{ll} \text{(i)} & n_i = \text{the number of pixels with intensity } i \\ \text{(ii)} & P_i = \text{the proportion of the pixels in the image with intensity } i \end{array}\right\} \quad (1)$$

Clearly:

$$\left.\begin{array}{ll} \text{(i)} & N = \sum_{i=0}^{L} n_i \\ \text{(ii)} & P_i = \dfrac{n_i}{N} \end{array}\right\} \quad (2)$$

$$\left.\begin{array}{l}\\ \text{(iii)}\ P_i \geq 0 \\ \\ \text{(iv)}\ \sum_{i=0}^{L} P_i = 1\end{array}\right\}$$

Suppose the range of pixel values is divided into two classes, C_b (background) and C_f (foreground), by a threshold at level t, then the probabilities of class occurrence are:

$$\left.\begin{array}{l} Pr_b = Pr(C_b) = \sum_{i=0}^{t} P_i = Pr_t \\ \\ Pr_f = Pr(C_f) = \sum_{i=t+1}^{L} P_i = 1 - Pr_t \end{array}\right\} \quad (3)$$

and the class means are:

$$\left.\begin{array}{l} \mu_b = \sum_{i=1}^{t} i.Pr(i|C_b) = \sum_{i=0}^{t} i.P_i \,/\, Pr_b = \dfrac{\mu_t}{Pr_t} \\ \\ \mu_f = \sum_{i=t+1}^{L} i.Pr(i|C_f) = \sum_{i=t+1}^{L} i.P_i \,/\, Pr_f = \dfrac{\mu_T - \mu_t}{1 - Pr_t} \end{array}\right\} \quad (4)$$

where:

$$\left.\begin{array}{l} \mu_t = \sum_{i=0}^{t} i.P_i \\ \\ \mu_T = \sum_{i=0}^{L} i.P_i \end{array}\right\} \quad (5)$$

Clearly:

$$\left.\begin{aligned} Pr_b.\mu_b + Pr_f.\mu_f &= \mu_T \\ Pr_b + Pr_f &= 1 \end{aligned}\right\} \quad (6)$$

The variances of the background and foreground pixel values, respectively, are defined by:

$$\left.\begin{aligned} \sigma_b^2 &= \sum_{i=0}^{t} (i-\mu_b)^2.Pr(i|C_b) = \sum_{i=0}^{t} (i-\mu_b)^2.P_i/Pr_b \\ \sigma_f^2 &= \sum_{i=t+1}^{L} (i-\mu_f)^2.Pr(i|C_f) = \sum_{i=t+1}^{L} (i-\mu_f)^2.P_i/Pr_f \end{aligned}\right\} \quad (7)$$

In the discriminant analysis, three measures of class separability are used commonly, namely:

$$\left.\begin{aligned} \lambda &= \frac{\sigma_B^2}{\sigma_W^2} \\ \kappa &= \frac{\sigma_T^2}{\sigma_W^2} \\ \eta &= \frac{\sigma_B^2}{\sigma_T^2} \end{aligned}\right\} \quad (8)$$

where for two classes:

$$\left.\begin{aligned} \sigma_W^2 &= Pr_b.\sigma_b^2 + Pr_f.\sigma_f^2 \\ \sigma_B^2 &= Pr_b.(\mu_b-\mu_T)^2 + Pr_f.(\mu_f-\mu_T)^2 \\ \sigma_T^2 &= \sum_{i=1}^{L} P_i.(i-\mu_T)^2 \end{aligned}\right\} \quad (9)$$

The meanings of λ, κ, η, σ_W^2, σ_B^2 and σ_T^2 are as follows:

σ_W^2 is the "within class variance". This is a measure of the variance of the values of the pixels within each of the background and foreground. It reflects the "similarity" of the values of the pixels within the classes.

σ_B^2 is the "between class variance". This is a measure of the difference between the values of pixels from different classes.

σ_T^2 is the "total variance". This is the variance of the values of all the pixels in the image.

λ is the between class variance divided by the within class variance.

κ is the total variance divided by the within class variance.

η is the between class variance divided by the total variance.

Clearly, λ, κ and η are all functions of the threshold value, t, which should be chosen to maximise one of these functions. The task now is to determine which of the measures given in (8) is the most suitable.

From (9):

$$\sigma_B^2 + \sigma_W^2 = \sigma_T^2 \qquad (10)$$

and from (8) and (10):

$$\left.\begin{aligned} \kappa &= \lambda + 1 \\ \eta &= \lambda \,/\, (\lambda+1) \end{aligned}\right\} \quad (11)$$

Therefore, while λ, κ and η are not identical functions, the value(s) of t that maximise them are the same for all three functions. Therefore the choice of which function to maximise should be based on ease of implementation and speed of execution rather than on analytical grounds.

By noting that:

$$\left.\begin{aligned} &\text{(i) } \sigma_T \text{ is constant for a given image} \\ &\text{(ii) in (10), } \sigma_B^2 = Pr_b.Pr_f.(\mu_f-\mu_b)^2 \end{aligned}\right\} \quad (12)$$

it is seen that the value for t that maximise σ_B^2, as defined in (12ii), will be the same value that maximises λ, κ and η. As σ_B^2 is computationally inexpensive to implement, the threshold value t is defined to be the threshold value that maximises σ_B^2 as defined in (12ii).

Otsu's paper [79] discusses bilevel, global thresholding. In section 3.11, it will be shown that his technique has a natural generalisation to N-level thresholding, and in sections 8.2 and 8.4, it will be shown how it can be used to generate variable threshold surfaces.

3.8 Multiple Statistic Thresholding

Sometimes a good threshold is not obvious from an image's grey-level histogram. For example, the histogram could be almost flat or broadly unimodal. Several researchers have proposed that, in such cases, the threshold value be calculated from a multi-dimensional histogram, rather than from the one-dimensional grey-level histogram. This section discusses some techniques that do this.

Panda and Rosenfeld [80] suggest a technique whereby the threshold value is calculated from the (grey level, edge value) scatter plot. The technique assumes that the object is brighter than the background, and that the scatter plot will contain a concentration of points representing pixels with low intensity and low edge value, corresponding to pixels in the background, a concentration of points representing pixels with high intensity and low edge value, corresponding to pixels in the foreground, and some points representing pixels with intermediate intensities and higher edge value, corresponding to pixels near the boundary between the object and the background. Therefore, the scatter plot should be "U" shaped, and the pixel value corresponding to the bottom of the U is selected as the threshold value.

Several variations of this approach have been proposed. For example:

(i) Kirby and Rosenfeld [51] propose using the local average pixel value, rather than edge value

(ii) Weszka and Rosenfeld [119] use (grey level, edge value) scatter plots as an intermediate stage in modifying histograms to make valley seeking easier

(iii) Ahuja and Rosenfeld [2] propose using (grey level, neighbours' grey level) scatter plots to either deepen valleys in histograms or turn them into spikes (see section 3.4).

3.9 Other Techniques

Many techniques that do not fall into any of the classes discussed in previous sections, have been proposed. A description of some of these follows.

Weszka and Rosenfeld [118] propose a technique whereby the thresholded image's "busyness" should be minimised, within certain constraints.

They work on the assumption that a well-thresholded image should contain a few relatively large objects, whereas a badly-thresholded image would contain many smaller, disjoint pieces of supposed objects. They define the "busyness" of a thresholded, binary image to be the sum of the lengths of the boundaries of the objects in the image. The threshold value is chosen to minimise this measure of busyness in the thresholded image. The authors also give an efficient algorithm for calculating the busyness of the thresholded image for all possible threshold values.

However:

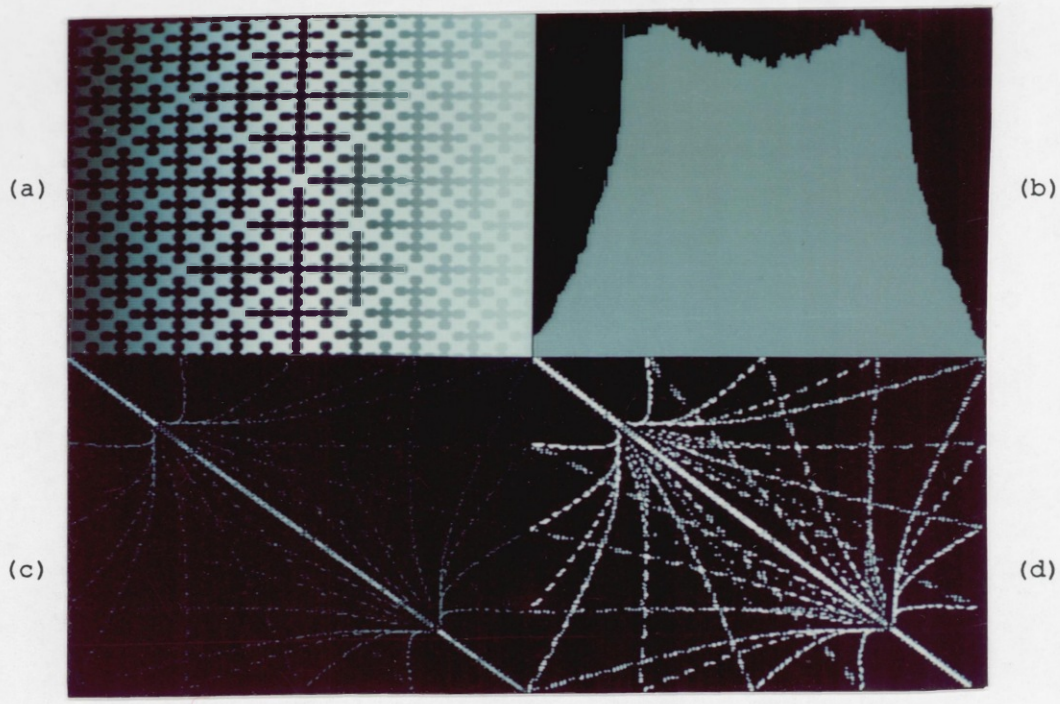
(i) The busyness of a thresholded, binary image is always non-negative.

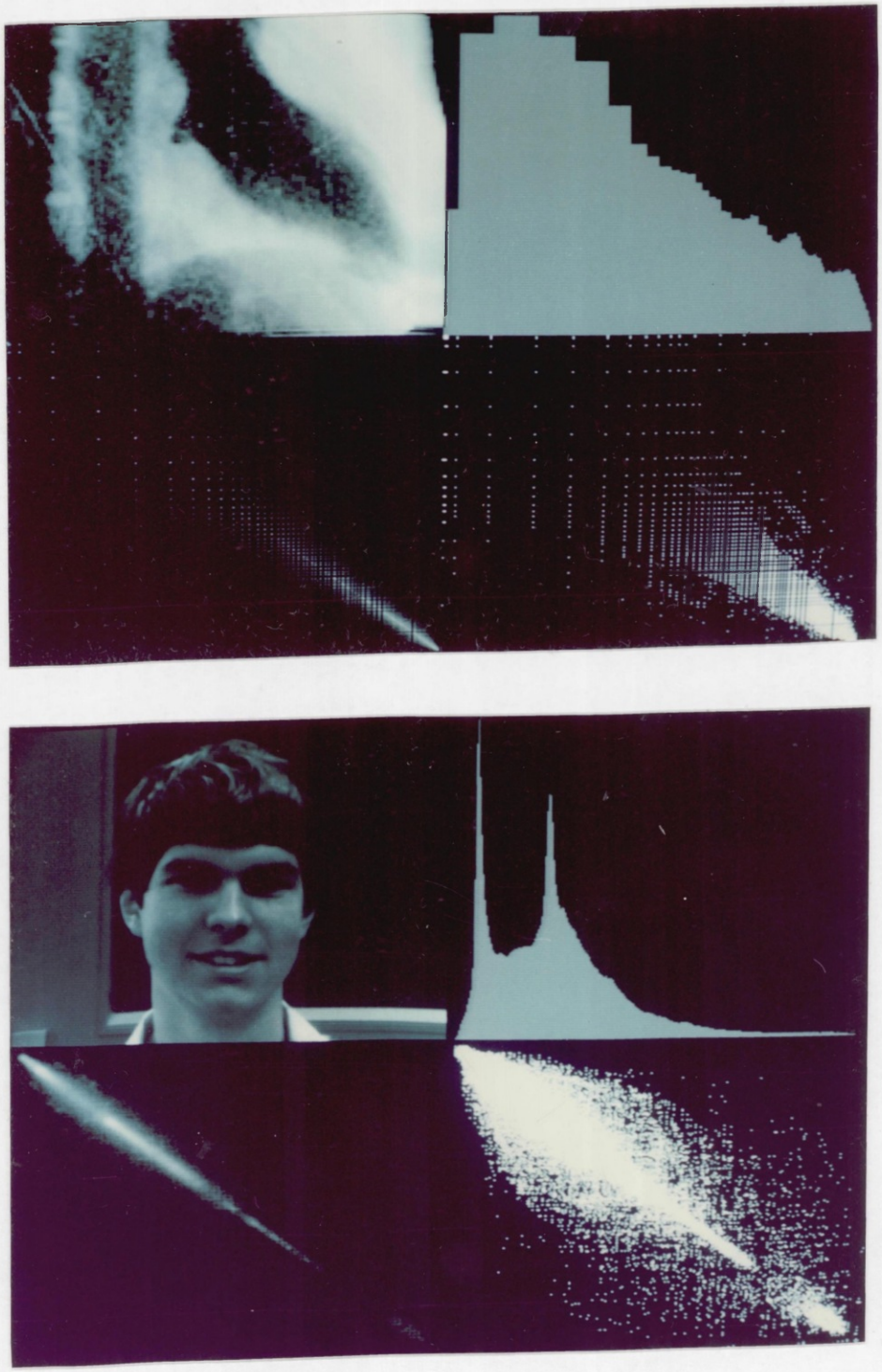
(ii) The busyness of thresholded images in which all pixels are classified as belonging to the foreground or background is zero. (1)

(iii) Any image containing both foreground and background pixels will have busyness greater than zero.

From (1), it is seen that the threshold values that minimise the busyness in the thresholded images are those which classify all the pixels as belonging to the foreground or all as belonging to the background. Clearly, these are not the solutions sought.

To overcome this problem, the authors suggest that the means of the foreground and background pixels be estimated, and the threshold value between these two means that minimises the busyness of the thresholded image be chosen as the threshold value. Figure 3.4 shows the busyness of the thresholded image for various threshold values, for a given image.





Busyness plots: (a), (e) and (i) contain images, (b), (f) and (j) show their Weszka busyness for the various possible threshold values, (c), (g) and (k) show the logarithms of the (pixel value, neighbour's pixel value) scatter plots (where the pixel value increases from left to right and the neighbour's pixel value increases from top to bottom), and (d), (h) and (l) are the scatter plots where all non-zero values appear white.

Figure 3.4

Davis, Rosenfeld and Weszka [18] suggest locally averaging images before they are thresholded. This will remove some noise that may be present in the image, and will also enable the separation of two textured regions which have different average grey levels.

They also propose an extension to this by suggesting that if the property that distinguishes the foreground from the background is not average grey level, but some other property, then the image may be preprocessed before it is smoothed. For example, if a highly textured region was to be extracted from a smooth background, but the foreground and background had the same average grey level, then the image should be edge detected, and the result averaged and then thresholded. This would mean that the highly textured region would have a high average edge value, and so the thresholding would not be on grey level, but on the edge value of the region.

Pun [86,87] and Kapur, Sahoo and Wong [48] propose techniques whereby the threshold value is derived from the entropy of the histogram.

Ridler and Calvard [88] suggest an iterative scheme whereby it is first assumed that a small square of pixels in each corner of the image, form the background, and the rest of the image forms the foreground (or object of interest). Then, at each iteration, the average values of the pixels in the foreground the background, respectively, are calculated and the average of these is then taken as the threshold value and is used to threshold the image into foreground and background for the next iteration. The process

is repeated until the thresholded image does not change between iterations.

3.10 Relaxation

3.10.1 Introduction

Traditional segmentation techniques can be classified as being either inherently sequential or inherently parallel. Parallel techniques have the advantage that they tend to be relatively simple to implement and they can be done in parallel on parallel architecture computers, whereas sequential techniques have the advantage that they can use knowledge gained at a previous stage of the classification process.

Relaxation techniques combine the best features of both these traditional approaches. Several iterations are applied, so previously acquired knowledge can be used at each stage, but each iteration requires many calculations that can be done in parallel.

Consider the following two traditional approaches to the problem of trying to find the borders of objects in an image:

(i) Use the Laplacian edge detector

(ii) Use a boundary tracking algorithm, such as one of those described in section 10.4 of [94].

(1)

Approach (1i) has the advantage of being very easy to implement and being inherently parallel, but it does not use any knowledge about whether a pixel's neighbours lie on the boundary of an object, or not, when trying to ascertain whether the current pixel lies on a boundary, or not. Approach (1ii), on the other hand, is

relatively intelligent, in that it uses previously acquired knowledge, but it cannot be done in parallel. Both have the disadvantage that once a decision has been made, it cannot be reversed. When using (1i), the results obtained from the Laplacian operator will normally be thresholded (into edge or not edge), and so an irrevocable decision is made, even for those pixels whose values are very close to the threshold value.

With relaxation techniques, the approach is as follows.

Given a set of m objects, A_i, i=1...m, that need to be classified into n categories, C_j, j=1..n, an initial weighting or "likelihood measure", P^0_{ij}, that each object A_i is in category C_j is calculated. This can be done in parallel. These measures are then updated, based on the likelihood measures for each pixel concerned and that of its neighbours, in some neighbourhood $N(A_i)$. This can also be done in parallel. The whole process is then repeated until some terminating condition is satisfied, for example, until the answer converges to a value or a specified number of iterations have been performed.

The approach is as follows:

(i) For each object, A_i, (in this case, pixel), assign an initial weighting, P^0_{ij}, indicating the likelihood that the object is in category C_j

(ii) At each iteration, assign to $P^{(r+1)}_{ij}$ a value that is a function of $P^{(r)}_{hk}$, where A_h lies in

the neighbourhood of A_i, $N(A_i)$, and for all classes C_k } (2)

(iii) If $p^{(r+1)}_{ij}=p^{(r)}_{ij}$ for all i and j, or a specified number of iterations have been performed, then stop, otherwise increment r and return to step (ii).

This technique has the following advantages:

(i) Each iteration can be done in parallel

(ii) The decision which assigns pixels to regions is based on information gleaned from neighbouring pixels and previous processing } (3)

(iii) Dubious decisions can be reversed and "mis-classifications" corrected.

Correct decisions can also be reversed, but this would suggest that there is an error in the assumptions made, the algorithm, the implementation of the algorithm or in one of the factors, rather than in the principle.

Rosenfeld [92] gives an excellent introduction to the use of relaxation techniques in a variety of applications.

3.10.2 Probabilistic Relaxation

With these techniques, we are given a set of n pixels, A_i, i=1..n, which are to be classified into m classes, C_j, j=1..m.

As with other relaxation schemes, the classification of a pixel depends, in part, on the classification of its neighbours. Therefore, a "compatibility measure" or weighting, W(i,j;h,k), is needed, which measures the compatibility of pixel A_i being in class C_j and pixel A_h being in class C_k. More specifically, W(i,j;h,k) is a measure of how much more (or less) likely it is that pixel A_i is in class C_j, given that pixel A_h is in class C_k.

Following is an illustration of how probabilistic relaxation is implemented.

Without loss of generality, it may be assumed that:

$$W(i,j;h,k) \in [-1,1] \qquad (1)$$

where:

$$\left.\begin{array}{l} W(i,j;h,k) = -1 \text{ indicates complete incompatibility} \\ \qquad\qquad \text{of } A_i \in C_j \text{ and } A_h \in C_k \\ \\ W(i,j;h,k) = 0 \text{ indicates total indifference} \\ \qquad\qquad \text{between } A_i \in C_j \text{ and } A_h \in C_k \\ \\ W(i,j;h,k) = 1 \text{ indicates total compatibility} \\ \qquad\qquad \text{of } A_i \in C_j \text{ and } A_h \in C_k \end{array}\right\} \qquad (2)$$

Note that $W(i,j;h,k)$ is not necessarily equal to $W(h,k;i,j)$, although often it will be.

Let $P_{ij}^{(0)}$ be the initial probability that pixel A_i is in class C_j, and let $P_{ij}^{(r)}$, $r \in N$, be the "revised" probability that pixel A_i is in class C_j, after the r^{th} iteration of the relaxation process.

Clearly:

$$\left.\begin{aligned} & P_{ij}^{(r)} \in [0,1] \\ & \\ & \sum_{j=1}^{m} P_{ij}^{(r)} = 1 \end{aligned}\right\} \quad (3)$$

Clearly, it is desirable that:

$$\left.\begin{aligned} & \text{(i)}\ P_{ij}^{(r)} = 0 \Rightarrow P_{ij}^{(r+1)} = 0 \\ & \\ & \text{(ii)}\ P_{ij}^{(r)} = 1 \Rightarrow P_{ij}^{(r+1)} = 1 \\ & \\ & \text{(iii)}\ P_{hk}^{(r)} = 1 \text{ and } W(i,j;h,k) = 1 \\ & \qquad \Rightarrow P_{ij}^{(r+1)} = 1 \end{aligned}\right\} \quad (4)$$

(iv) $P_{hk}^{(r)} = 1$ and $W(i,j;h,k) = -1$ }

}

$\Rightarrow$ $P_{ij}^{(r+1)} = 0$ }

Also, if $P_{hk}^{(r)}$ or $|W(i,j;h,k)|$ are small, then $P_{ij}^{(r+1)}$ should not be changed very much, whereas if $P_{ij}^{(r)}$ and $|W(i,j;h,k)|$ are large, this is expected to have a significant impact on $P_{ij}^{(r+1)}$.

Clearly, from their definitions, the four rules in (4) are compatible and should never conflict. Should there ever be a conflict, the error would be with the estimation of the initial probabilities, $P_{ij}^{(0)}$, or the weighting rules, $W(i,j;h,k)$, rather than with the principles described above.

Several updating rules have been proposed. For example, Rosenfeld, Hummel and Zucker [93] propose:

$$P_{ij}^{(r+1)} = \frac{P_{ij}^{(r)} \cdot (1+q_{ij}^{(r)})}{\sum_{j=1}^{m} P_{ij}^{(r)} \cdot (1+q_{ij}^{(r)})} \qquad (5)$$

where $q_{ij}^{(r)}$ is defined as the average, over all neighbours, A_h, of pixel A_i, of

$$\sum_{k=1}^{m} W(i,j;h,k) \cdot P_{hk}^{(r)} \qquad (7)$$

and Peleg [81] proposes an updating rule whereby $P_{ij}^{(r+1)}$ is equal to the average, over all neighbouring pixels A_h of pixel A_i, of

$$\frac{P_{ij}^{(r)} \sum_{k=1}^{m} W(i,j;h,k) . P_{hk}^{(r)}}{\sum_{j=1}^{m} \sum_{k=1}^{m} W(i,j;h,k) . P_{ij}^{(r)} . P_{hk}^{(r)}} \qquad (6)$$

Rosenfeld and Smith [95] show several results using both the updating rules proposed by Rosenfeld, Hummel and Zucker [93] and Peleg [81] (as described above), obtaining similar results for each method.

3.10.3 Other Relaxation Techniques

Probabilistic relaxation is a special case of a more general class of relaxation methods.

In the more general case, the $P_{ij}^{(r)}$ of the previous section are not probabilities in the sense of (3) in section 3.10.2, but rather are "weights".

Typically, $P_{ij}^{(r)} > 0$ means that it is more than 50% likely that object A_i belongs to class C_j, $P_{ij} = 0$ means that it is 50% likely that the object belongs to class C_j (and 50% likely that it does not), and $P_{ij} < 0$ means that it is less than 50% likely that object A_i belongs to class C_j.

The advantage of this more general approach is that for each ordered pair (A_i,C_j), P_{ij} is a measure of absolute compatibility. That is, it is an absolute measure of how compatible object A_i is with class C_j. This means that when object A_i is eventually assigned to a class, there will be an absolute measure of the

"goodness" of the classification rather than just a measure of goodness relative to how good the other possible classifications would have been.

Besides the constraints (2), (3) and (4) of section 3.10.2 not necessarily applying, and the $P^{(r)}_{ij}$ not needing to be scaled accordingly, the updating rules are very similar to those discussed in the previous section, and require no further discussion here. However, more details can be obtained from the book by Rosenfeld and Kak [94] and the papers by Rosenfeld and others [24,43].

3.11 Calculation of Multiple Thresholds

Several of the threshold evaluation techniques described in sections 3.3 to 3.9 can be generalised to yield multiple threshold values or surfaces.

The average thresholding technique described in section 3.3 has no natural generalisation. Median thresholding, however, does. If n threshold values, t_i (i=1..n), are to be calculated for an image with N pixels, then t_i is chosen so that (N.i/(n+1)) of the image's pixels have intensity value less than or equal to t_i. This, however, assumes that each of the classes of pixels in the thresholded image contains the same number of pixels.

Problems could be encountered using valley-seeking techniques, as when the bottoms of several valleys are sought, each bottom corresponding to a threshold value, exactly one threshold value should be found between each pair of adjacent modes in the histogram. However, it is likely that only very local minima will be found and that several values will be found in some valleys while none are found in others. The techniques do not work when one threshold value is to be calculated, and there is no reason to assume that they will work in the more general case.

It is easy enough to calculate multiple thresholds using one of the distribution assuming techniques described in section 3.5, but this approach can be unstable. If, for example, it is assumed that a histogram is the sum of three Gaussians, the procedure would be to calculate the "optimal" parameters for the three distributions. However, often at least one of the Gaussians will not correspond

to a hump in the histogram, but rather to a very weak shoulder. Furthermore, if the distributions of the values of the pixels in each of the regions to be extracted is not known and is guessed or assumed, the technique has very little chance of working.

Tsai's method, presented in section 3.6, is defined for the general case and does not suffer from the problems described above.

Otsu's technique, presented in section 3.6, has a natural generalisation to thresholding into N regions. Instead of defining Pr_b and Pr_f in (3) of section 3.7, Pr_i (i=1..N) are defined to be the proportion of the pixels in the original image assigned to each of the regions, i, in the thresholded image. Define μ_i and σ_i^2 similarly. Then define:

$$\left.\begin{aligned} \sigma_W^2 &= \sum_{i=1}^{N} Pr_i.\sigma_i^2 \\ \sigma_B^2 &= \sum_{i=1}^{N} Pr_i.(\mu_i-\mu_T)^2 \end{aligned}\right\} \quad (1)$$

The definitions of λ, κ and η remain unchanged, as do the equations (11) of section 3.7. Therefore, the threshold values, t_i (i=1..(N-1)), that maximise σ_B^2 are chosen as the "optimal" threshold values. Results from using this technique will be presented in chapter 9.

Clearly, there is no reason why multiple statistic multiple thresholding should yield results worse than those obtained using single statistic multiple thresholding techniques. However, most

of the work done on multiple statistic thresholding so far has been on simple thresholding, and none of the techniques described in section 3.9 have a natural, implementable generalisation to the calculation of multiple thresholds.

Takiyama [108] adopts a different approach, defining a learning "multiple threshold perceptron". He defines rules by which the perceptron learns and tries to detect linear separability in the input data, i.e. the histogram.

3.12 Conclusion

Most of the bilevel thresholding techniques that have been analysed during the course of this research are ad hoc and make assumptions about the properties of images that are often not valid. If the assumptions made by a particular technique are known a priori to be valid for a given image, then the technique should work well. However, this a priori knowledge is usually not available.

Exceptions to this are the techniques proposed by Tsai [112] and Otsu [79], which are discussed in sections 3.6 and 3.7 respectively. Both techniques are also shown to have natural generalisations to multi-level thresholding.

It is asserted that maximising the separability of the classes detected by the thresholding is more desirable than preserving the first 2N moments in the thresholded image. Therefore, it is concluded that of the thresholding techniques evaluated during the course of this research, Otsu's technique, with the generalisation given in section 3.11, is the most appropriate for images with the properties described in (1) and (2) of section 1.3.

Section 3.11 discussed some of the problems encountered when multiple threshold values are to be calculated. Section 8.3 will discuss the problems encountered when interpolating multiple threshold surfaces through a small number of possibly irregularly spaced points for which multiple thresholds have been calculated.

CHAPTER 4

SEGMENTATION AND CLUSTERING

4.1 Introduction

Clustering is the grouping together of "similar" objects. Cluster analysis is the study of how this grouping is done, and what criteria are used to determine whether objects, or groups of objects, are similar or not.

Jain [44] defines cluster analysis as:

> ". . . a process of partitioning a collection of patterns into groups such that patterns belonging to the same group are more similar to each other than are patterns belonging to different groups."

The principles and objectives of cluster analysis are understood easily; what is not understood so easily is what constitutes "similar" and "dissimilar". Section 4.2 of this thesis discusses this, while sections 4.3 to 4.7 explore various clustering techniques.

Segmentation, on the other hand, is the partitioning of objects or data into groups, such that objects in the same same group are similar, while objects in different groups are dissimilar.

Clearly, the objectives of clustering and segmentation are the same and for this reason, the terms are often used interchangeably. However the approach is different. The former is bottom-up and agglomerative and the latter is top-down and disaggregative or divisive.

While cluster analysis is relatively new in image processing, it has been studied in many different fields for a long time. For example, in 1737, Linnaeus described the importance of clustering in zoology [64]:

> "Every discernment that we can truly make depends on a clear method by which we distinguish the similar from the dissimilar. The more comprehensive the natural distinctions made by this method, the clearer the mental construction of reality that arises in our minds. The greater the number of objects treated in our conception, the harder it is to work out a method and yet the more necessary does it turn out to be ...
>
> We must not on this account unite under the same genus the horse and the pig, even though both species are single-hooved, nor separate in genus the roebuck, the reindeer and the elk, even though they differ in their horns. Therefore, we need to investigate the boundaries of the genera by careful and diligent observation, since they would be hard to determine a priori, even though this task is laborious, for 'if the genera were confused, all would necessarily be confusion' (Cesalpino)."

Anderberg [3] groups the disciplines in which significant work on cluster analysis has been done, into the following six major areas of research:

(i) Life Sciences

(ii) Medical Sciences

(iii) Behavioral and Social Sciences

(iv) Earth Sciences

(v) Engineering Sciences

(vi) Information, Policy and Decision Sciences

While this research is concerned solely with a particular application of clustering in image processing, it is important to see how this work relates to other work that has been done in the field, as many of the techniques used were developed for a different application.

Because cluster analysis developed in several different fields simultaneously and independantly, it is also known as Q-analysis, typology, grouping, clumping, classification, numerical taxonomy, morphometrics, botryology, nostology, systematics and unsupervised pattern recognition.

Many different approaches to cluster analysis have been proposed. Because there are so many clustering techniques, if any study of them is to be made, similar approaches need to be grouped together. This is an interesting clustering problem in itself. Everitt [26] proposed the following classification:

(i) Hierarchical techniques

(ii) Optimisation-partitioning techniques

(iii) Density or mode-seeking techniques

(iv) Other techniques

There have been several very good review works on segmentation and clustering, particularly those by Sklansky [104] (review of early work), Fu and Mui [30], Anderberg [3], Everitt [26] and Young and Fu [128].

4.2 Similarity Measures and Distance Criteria

Perhaps the most important - and difficult - step in any clustering process is defining what is meant by "similar". This section describes some of the problems associated with similarity measures.

Usually, clustering together similar objects means that the pixel values in a particular cluster are as similar as possible, while the values of pixels in different clusters are as different (dissimilar) as possible. However, even this definition is ambiguous, as it could equally well mean that the most similar pixels in each cluster are as similar as possible as it could mean that the most dissimilar pixels in each cluster are as similar as possible.

Similarity measures based on pixel values alone are discussed in detail in sections 4.3 to 4.7 of this thesis.

Many other measures of similarity have been proposed. For example, Mitchell and Carlton [69] define similarity in terms of a texture measure they develop, Jain, Smith and Backer [45] use the shape of the boundaries of the clusters, as do Prager [84] and Perkins [82], Thompson [110] bases his clustering on several frames of a scene and clusters on a combination of pixel value and motion estimation, and Lee and Fu [63] define a technique in which each pixel is classified as being either part of a plain, slope, ridge, valley, shoulder or foot, and then contiguous groups of pixels of the same type are clustered together.

Anderberg [3] discusses similarity measures in great detail. He also investigates thoroughly the various scaling problems involved when multiple statistic similarity measures are used.

Section 2.4 discussed measures of similarity between images, and chapter 10 will apply some of these measures in this reasearch.

4.3 Hierarchical Clustering

4.3.1 Introduction

Hierarchical clustering techniques are those iterative techniques in which each pixel is initially assigned to a class, and then at each iteration, either two classes are merged, or one is split into two, until some terminating criterion is satisfied.

Hierarchical clustering techniques can be classified as being either agglomerative or disaggregative (sometimes called "divisive").

Agglomerative techniques are those in which each object is assigned, initially, to its own class, and at each iteration, the two (or more) classes that are most similar are merged. The fewer classes that are desired, the more iterations that become necessary. This merging of classes is often represented graphically by a dendogram (see figures 4.2, 4.4 and 4.5).

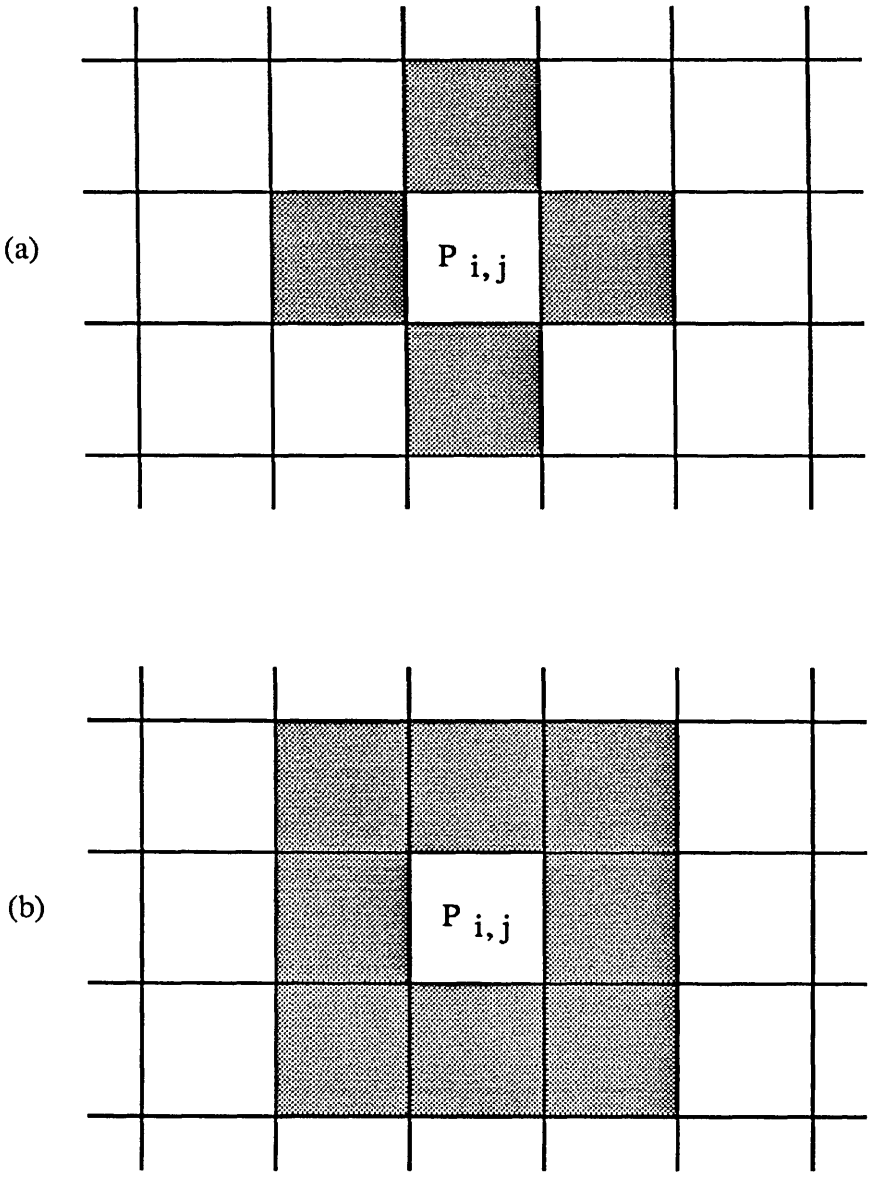
Disaggregative techniques are those in which all the objects are assigned, initially, to the same cluster, and then at each iteration, one cluster is split into two, such that the splitting will maximise the increase in similarity of objects in the same cluster and dissimilarity of objects in different clusters. Clearly, this splitting of clusters can also be represented by a dendogram.

Both techniques are irreversible, in that with agglomerative techniques, clusters are never split and with disaggregative techniques, clusters are never merged.

Strictly speaking, all clustering is agglomerative and what is referred to as disaggregative clustering is actually segmentation. However, as most hierarchical clustering techniques can be implemented either agglomeratively or disaggregatively, with both implementations yielding the same result, the terms clustering and segmentation are used interchangably in the literature.

Usually, in image processing applications of agglomerative, hierarchical clustering, the two clusters that are merged at each stage must be adjacent to each other. By adjacent, it is meant that there exists a pixel in each of the two regions such that the pixels lie in each other's 4- or 8-neighbourhoods.

A pixel's 4-neighbourhood is the set of at most four pixels that lie immediately above, below, to the left of or to the right of the pixel in question, while the 8-neighbourhood is the set of at most eight pixels that contains the 4-neighbourhood in addition to the pixels immediately above and to the left, above and to the right, below and to the left, and below and to the right of the pixel in question (see figure 4.1).



4- and 8-neighbourhoods: (a) shows the pixels in the 4-neighbourhood of pixel $P_{i,j}$ and (b) shows the pixels in the 8-neighbourhood of pixel $P_{i,j}$. The pixels in the neighbourhoods are those that are shaded.

Figure 4.1

4.3.2 Single-linkage

Single-linkage, agglomerative, hierarchical clustering is that technique in which each pixel is assigned, initially, to its own cluster, and at each iteration, the two clusters that satisfy the the following conditions, are merged:

$$
\left.\begin{array}{ll}
\text{(i)} & P_i \text{ and } P_j \text{ are neighbours} \\
& \text{(either 4- or 8-neighbours)} \\
\text{(ii)} & P_i \in C_i \text{ and } P_j \in C_j \\
\text{(iii)} & |P_i - P_j| \leq |P_a - P_b| \quad \forall \; P_a \in C_a \text{ and } P_b \in C_b \\
& \text{where Pa and Pb are neighbouring pixels} \\
\text{(iv)} & i \neq j, \quad a \neq b
\end{array}\right\} \quad (1)
$$

where: P_i, P_j, P_a and P_b are pixels

C_i, C_j, C_a and C_b are clusters.

Clearly, i and j are not necessarily unique (viz. the unordered pairs (P_i,P_j) and (C_i,C_j) are not necessarily unique). In this case, usually, the pair of clusters to be merged is chosen arbitarily from the set of clusters satisfying (1) above.

This arbitary choice does not, in fact, alter the final result, as long as, when the iterative merging process terminates, the last two clusters that were merged were more similar than the two most similar clusters in the final clustering. Figure 4.2 shows the dendograms for clustering some simple data both ways possible. As can be seen, both courses give the same final clustering.



Single-linkage dendograms: (a) and (b) show dendograms for the two possible ways of clustering the given data using single-linkage. As can be seen, the final clusterings are the same for both dendograms.

Figure 4.2

In section 4.3.3, it is shown that the otherwise very similar "complete-linkage" does not have this property, and in section 4.3.6, it is shown that single-linkage is the only hierarchical clustering technique that does have this property.

In image processing, the objects being clustered are usually pixels, of which there are many, even in a single image. Therefore, if this approach is to have any credibility, an efficient algorithm must be found to implement it.

In the course of this research, an efficient O(N.LogN) algorithm, where N is the number of pixels to be clustered, based on that of Kruskal [57], has been developed and implemented to perform single-linkage.

As stated earlier, with hierarchical techniques, the two most common interpretations of "neighbouring regions" are that the regions each have a pixel such that they lie in each other's four or eight neighbourhoods. For the purpose of this discussion, the eight neighbourhood case will be considered.

In general terms, the algorithm is as follows:

(i) Defining the vertices of an undirected graph to be the pixels in the image to be segmented, and the length (weight) of each edge in the graph to be the absolute difference in the values of the pixels the edge runs between, form a list of all the edges in the graph in which each pixel (vertex) is connected to each of the pixels in its 8-neighbourhood.

(ii) Assign each of the image's pixels to its own region.

(iii) Sort the list formed in (i), into increasing edge length.

(iv) At each iteration, find the edge of shortest length in the list sorted in (iii) that runs between two pixels that lie in different regions.

(v) Merge the regions that these two pixels lie in, thus decreasing the number of regions by one.

(vi) Perform steps (iv) and (v) until the terminating criterion is satisfied.

(2)

More specifically, for an mxn image, the implementation of the algorithm is as follows:

(i) Assign to each pixel a unique integral value between 0 and (mn-1). Henceforth, each pixel will be identified by this number.

(ii) Form a list, EDGES, 4mn-3m-3n+2 elements long, as described in (2i), above. It will contain m(n-1) horizontal, n(m-1) vertical and 2(m-1)(n-1) diagonal edges. Each edge is represented in EDGES by the numbers of the pixels between which the edge lies, in no particular order, and the length of the edge.

(iii) Use a quick sort to sort the edges, in EDGES, in to order of increasing length.

(iv) Form an integer array, TREE, (mn) elements long, with subscripts being the same as the numbers of the pixels, as defined in (i) above. If a pixel is the smallest numbered pixel in the region to which it belongs (which is the case initially for all the pixels), then the corresponding value in TREE is -1, otherwise, it contains the number of a smaller numbered pixel in the region. Clearly, at any stage, for

any pixel, the smallest numbered pixel in the original pixel's region can be determined from TREE. } (3)

(v) The clustering process can now begin. Each element of EDGES is considered in turn, in order of increasing length of the edge, until some terminationg criterion is satisfied (such as the required number of regions is obtained).

At each iteration, it is determined if the two end pixels of the edge under consideration lie in the same region. For each edge, this can be done in O(LogN) time, by comparing the smallest numbered pixels in each of the end pixels' regions. If they are the same, then the end pixels are already in the same region and the edge is redundant. If not, the two regions are to be merged.

This is done by finding the smallest numbered pixels in each of the two regions (say the pixels are numbered P_i and P_j), and

if ($P_i < P_j$)

then set TREE(P_j)=P_i,

otherwise set TREE(P_i)=P_j.

Clearly, steps (i) and (ii) and the initialisation described in

step (iv) contain O(N) operations, each iteration described in step (v) is of O(LogN), there are O(N) of these iterations and step (iii) is of O(N.LogN). Hence, the algorithm is of O(N.LogN).

Appendix A contains the listing of a Fortran program that implements the above algorithm. While it is written in standard Fortran-77 and should run on any 64-bit, 4 Megabyte, ASCII computer with a Fortran-77 compiler, it is written to run efficiently on a Cray-1S computer running CFT 1.14 Fortran.

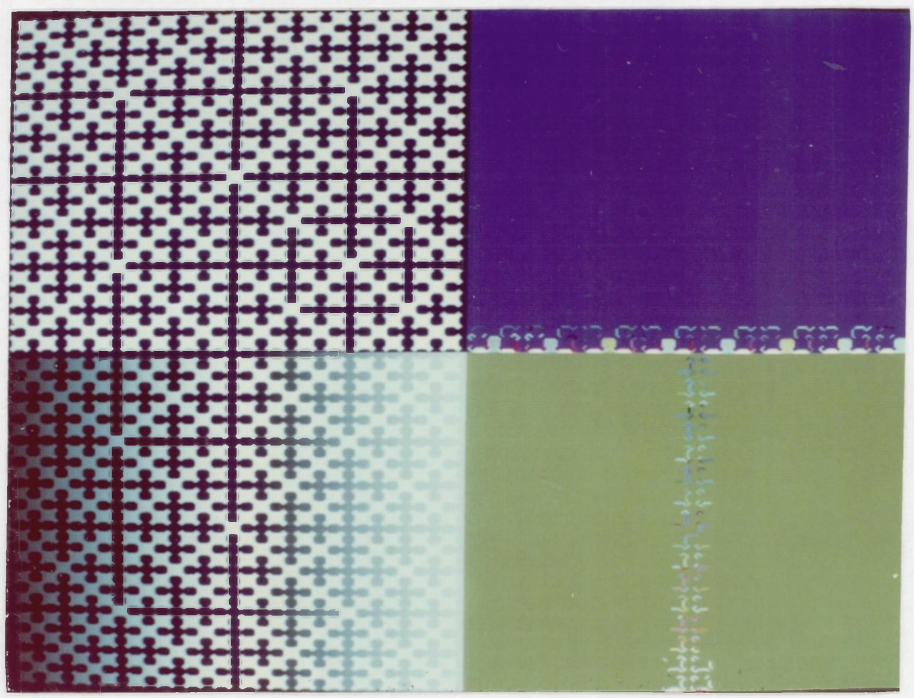
Narendra and Goldberg [76] did some interesting work in this field, which is described in their paper. Unfortunately, because of their arbitary resolution of ties (page 187 of [76]), they are incorrect in stating that "it is easy to see that, at the conclusion of the procedure, all points on a unimodal plateau will belong to the same directed tree with one point on the plateau that will be called a root", as this is not necessarily the case at all.

Single-linkage has the advantages of being intuitively appealing and easy to implement. However, it suffers from what is known as "chaining". This is the effect caused when two very dissimilar, possibly adjacent, pixels are assigned to the same cluster because a path exists between them in which the difference between the values of any two consecutive pixels is relatively small. This path is known as a "chain".

Figure 4.3 shows the effect of chaining clearly. One would expect the white and the black regions of the image in figure 4.3(a) to

be assigned to different clusters (shown in figure 4.3(b)), as these are the most different pixels in the original image, but because the original image is blurred slightly, the border between the white and the black region contains a whole range of intermediate values, through which the white and the black regions are chained erroneously.

(a) (b)



(c) (d)

Single-linkage's chaining effect: (b) and (d) show the results of clustering the pixels in images (a) and (c), respectively, into 250 regions using single-linkage. The colours in (b) and (d) have no significance other than to enable one to distinguish between the regions.

Figure 4.3

4.3.3 Complete-linkage

One method proposed to overcome this chaining effect is complete-linkage. It is similar to single-linkage, except that at each iteration, the two clusters whose most dissimilar elements are most similar, are merged.

This has the effect of generating more "compact" clusters, and clearly overcomes the problem of chaining.

However, it has the disadvantage that the order in which ties are resolved affects the final clustering. For example, in the dendograms in figure 4.4, two different clusterings result from different decisions when a tie was encountered in the first iteration.



Complete-linkage dendograms: (a) and (b) show dendograms for the two possible ways of clustering the given data using complete-linkage. As can be seen, the final clusterings are different for the two dendograms.

Figure 4.4

As shall be shown in section 4.3.6, this property is a major flaw in the technique and suggests that the technique is mathematically unsound.

Much effort has been put into researching this technique, but that

does not make it good. While being intuitively appealing, the property described above makes it unpredictable, and therefore unreliable.

4.3.4 Variance Minimisation

With this approach, at each iteration, the two adjacent clusters whose merging would cause the smallest increase in the combined variance, V_N , are merged, where V_N is defined as:

$$\left.\begin{array}{l} \\ V_N = \sum_{i=1}^{N} P_i . \mathrm{Var}(C_i) \\ \\ \end{array}\right\} \qquad (1)$$

where: N is the number of clusters

C_i is the i^{th} cluster

P_i is the fraction of the image occupied by pixels in cluster C_i

$\mathrm{Var}(C_i)$ is the variance of the values of the pixels in cluster C_i

Clearly, at each iteration, the number of clusters, N, is decreased by one.

However, as Jain [44] points out, this technique has the undesirable property that when the process has iterated enough times to yield N clusters, the clusters obtained are not necessarily the N clusters that actually minimise the weighted variance, V_N, as defined by (1) above. Also, as with complete-linkage, the resolution of ties affects the final clustering. This is demonstrated in the dendogram in figure 4.5 in

which two different final clusterings are obtained, depending on how the tie in the first iteration is resolved.



Variance minimisation dendograms: (a) and (b) show dendograms for the two possible ways of clustering the given data using the variance minimisation technique described above. As can be seen, the final clusterings are different for the two dendograms.

Figure 4.5

4.3.5 Other Techniques

Several other techniques have been proposed, particularly variations of single-linkage.

As stated in section 4.3.2, single-linkage suffers from the chaining effect. In order to try to overcome this, several authors have suggested "K^{th} nearest neighbour clustering" [4,23,111,125]. With this technique, the two clusters whose K^{th} shortest link is the shortest, are merged. Clearly, single-linkage is a special case of this, with K=1.

Gowda and Krishna introduce the notion of "mutual neighbourhood value" and propose both disaggregative [35] and agglomerative [36] clustering techniques using this measure. At each iteration, the cluster C_j that is closest to cluster C_i is given a neighbourhood value of 1 with respect to C_i, the second closest cluster is given

a neighbourhood value of 2, and so on. Each unordered pair of clusters, (C_a,C_b), is assigned a "mutual neighbourhood value" which is the sum of C_a's neighbourhood value with respect to C_b, and C_b's neighbourhood value with respect to C_a. At each iteration, the two clusters C_a and C_b, $a \neq b$, whose mutual neighbourhood value is the smallest, are merged.

Delattre and Hansen [19] propose a similar technique that is a hybrid of single and complete-linkage.

In section 4.3.6, it will be shown that the only hierarchical technique that may have any mathematical justification is single-linkage. Therefore, detailed descriptions of other hierarchical techniques here would only serve to detract from the aim of this section, which is to give an overview of the types of approaches that have been proposed. The reader is referred to the publications cited for more detailed descriptions of the techniques.

4.3.6 Review

Perhaps the most hotly debated issue in cluster analysis is which of the hierarchical techniques has any mathematical, or any other, justification besides the rather unconvincing and uninspiring argument of "well, it seems to work quite well quite often".

In the late sixties, two schools of thought emerged, one in Australia [60-62,122,123] which criticized single-linkage and proposed alternative approaches, and one at Cambridge [46,47,100-102] that claimed that single-linkage was the only

hierarchical technique that had any known mathematical basis or justification.

Lance and Williams [60] condemned "nearest neighbour sorting", calling it "obsolete".

Jardine and Sibson [46] later stated that all hierarchical clustering techniques should satisfy seven conditions. Considering the clustering to be a transformation, the conditions are as follows:

(i) The result obtained from given data should be unique

(ii) Small changes in the data should produce small changes in the dendogram

(iii) If the dissimilarity coefficient is ultrametric, then it should still be so after the transformation

(iv) In some sense, the result obtained should impose the minimum distortion upon the dissimilarity coefficient, subject to conditions (i)-(iii) and (v)-(vii) (1)

(v) The transformation should be independent of scale

(vi) The transformation should be independent of any preliminary labelling of the objects to be classified

(vii) If a cluster is excised and the transformation is applied to the restriction to it of the dissimilarity coefficient, the resultant dendogram should be the restriction to that cluster of the original dendogram.

They go on to add that "the single-link (nearest-neighbour) method is the only method that [they] have seen that satisfies conditions (i)-(vii)" and that "the defects of the single-link method should be regarded as defects of hierarchic classification itself."

Williams et al [123] criticise all except (iv) and (v) of the conditions in (1) on "pragmatic" grounds, claiming that the highly non-linear nature of their strategies corresponds rather closely with human intuitive mental processes. They then proposed the following three criteria that they believe may over-ride the mathematical criteria of the Cambridge school:

(i) The grouping must be more intense than that implied by the original dissimilarity measures

(ii) The grouping must be relatively insensitive to outlying values, due either to aberrant individuals or to errors or inaccuracies in the (2)

data }
}
(iii) The ultrametric transformation should not be }
necessarily, or even usually, invariant over }
the entire population. }

Sibson [100] replied by stating that, while some of the conditions in (1) may be "open to discussion", (i), (vi) and (vii) were "unexceptionable", and that techniques such as complete-linkage did not satisfy even these three conditions. Jardine and Sibson [47] went on to show that conditions (2), above, proposed by the Australian school were either equivalent to some of to their criteria, given in (1) above, or were irrelevant. Once again, they state that single-link clustering is the only technique that satisfies their criteria.

Much as it would be convenient if Jardine and Sibson were wrong and the conditions given in (1) were not valid, it is believed that they are valid and necessary. Therefore, it is believed that single-linkage is the only hierarchical clustering technique that may be justifiable. However, it is not clear that the conditions (1) given by Jardine and Sibson are sufficient or complete. Therefore, single-linkage may not be justifiable either. In any case, the effect of chaining is a serious problem with single-linkage and so other non-hierarchic approaches must be investigated.

4.4 Optimisation-partitioning Clustering

Optimisation-partitioning clustering techniques are those techniques in which some equation is solved or function is optimised, the solution of which enables the data to be clustered or segmented.

Clearly, both global and variable thresholding are forms of clustering as "similar" data (viz. pixels with similar values) are clustered together. More specifically, they are forms of optimisation-partitioning clustering, in that a threshold value needs to be calculated (this involves either solving an equation or optimising a function), and the threshold value calculated is then used to segment (partition) the image.

However, thresholding is not the only form of optimisation-partitioning clustering. For example, the clustering of an image's pixels into a specified number, N, of regions such that the function (1) in section 4.3.4 was minimised, would also be classified as optimisation-partitioning clustering.

These techniques have the advantage over hierarchical techniques in that they are usually based on much better understood mathematical principles than the latter, which, by and large, are borrowed from the botanists and zoologists who developed the techniques.

Too many ad hoc techniques have been developed that work at times and do not at other times, without the user being able to determine a priori whether the technique will or will not work in

a given application. More alarmingly, many techniques developed and that are claimed to work (such as those described in [60]) have not only not been based on any sound mathematical theory, but have been shown to be mathematically unsound [100].

Optimisation-partitioning techniques, on the other hand, are not ad-hoc, but rather are based on sound mathematical principles. These techniques also tend to be derived mathematically before they are implemented, rather than being implemented first and then seeing if they can be justified mathematically. However, this mathematics need not be complicated. For example, thresholding an image using a global threshold that is the average of all the pixels' values is an optimisation-partitioning technique based on sound, albeit very simple, mathematics.

The complexity of the mathematics involved will depend on the complexity of the underlying assumed mathematical model of the data that the clustering technique is designed to work on.

4.5 Density-seeking Clustering

Density-seeking clustering (sometimes known as "mode-seeking clustering") refers to those techniques in which high concentrations of data with a particular quality are searched for. Sometimes, the number of clusters, N, is specified a priori and the N most dense sub-populations are located, and at other times, the minimum degree of density of interest will be specifed a priori.

An example of the latter is given by Goldberg [33]. He suggests a density-seeking technique for segmenting multispectral images.

In his paper, in which he deals with the segmentation of four band LANDSAT images, he suggests the following approach:

(1)

(i) Reduce the resolution of each of the bands from 256 to 64 intensity values

(ii) Compute the four dimensional histogram of the pixel values

(iii) Identify those cells in the histogram that contain more pixels than average, and then merge together neighbouring cells that have been identified, to give a set of non-touching "cluster cores"

(iv) The remaining cells are then joined to the nearest cluster core.

Mizoguchi and Shimura [70] explain their rather similar technique, very well. Unfortunately, they made a rather obvious error in their "definition 5".

4.6 Clumping

Clumping techniques are those segmentation techniques in which the segments are allowed to overlap.

These techniques are not of much use in this research because:

(i) The images that are to be processed are to be segmented into "foreground" and "background", or "light regions" and "dark regions", and a pixel cannot lie in the foreground and the background simultaneously, and neither can it lie in both the light and dark regions of the image simultaneously. (1)

(ii) Even in the case where a datum (in our case, a pixel) can lie in more than one of the N regions simultaneously, clumping can be simulated by segmenting the image into (2^N-1) regions, corresponding to all non-trivial combinations of the original N regions.

This family of techniques is only mentioned here for the sake of completeness.

4.7 Other Techniques

Many clustering techniques have been suggested, and the aim of this section is not to give an exhaustive list of them, nor is it the aim to describe any of the techniques in great detail, but rather, the aim is to give an overview of the types of approaches that have been suggested, with references to the literature, where further details of individual techniques can be found, when appropriate.

Following is a brief description of some of the other clustering techniques that have been proposed, that do not fall into one of the categories described in sections 4.3 to 4.6.

Lai and Ehrich [59] present a region growing technique, in which those pixels that clearly should be clustered together are, and then the remaining pixels are absorbed by the nearest cluster already formed. In many ways, this approach is similar to the density-seeking techniques discussed in section 4.5.

Pong, Shapiro and Haralick [83] present an interesting technique in which the clustering is based solely on pixel values, but the clustering does not necessarily cluster together pixels with similar values; rather, it clusters together pixels whose values lie in the same facet (sub-plane). Mitchell and Carlton [69] proposed clustering on a texture measure that they defined, while Lee and Fu [63] suggest clustering on a "grey level geographical structure" (GLGS) that they defined.

Thompson [110] suggests that instead of basing the clustering on information gleaned from only one image, several frames should be used and the segmentation should be based on pixel values as well as on the motion of objects detected between frames.

Several authors [45,82,84] have suggested using an edge detector to assign an edge-value to each pixel, and then basing the segmentation on these values, rather than on the original pixel values.

4.8 Contiguous versus Non-contiguous Clusters

Clustering is the grouping together of similar objects or data. In order to do this, one needs to define what criteria are used in deciding whether objects are similar or not, and to define a measure of this similarity. Clearly then, all the objects in a particular cluster are contiguous in the clustering criteria domain.

For example, if an image is thresholded using a global threshold, then the pixels in each of the thresholded segments are contiguous in the pixel intensity domain.

However, the data in a particular cluster are not always necessarily contiguous spatially, if indeed a spatial domain is defined.

Clearly, in image processing, a spatial domain is always defined. The question then arises as to whether the pixels in a particular cluster are contiguous (for the rest of this section, unless stated otherwise, "contiguous" means "spatially contiguous") or not - and whether spatial contiguity is desirable or not.

The hierarchical techniques described in section 4.3 all yield a known number of contiguous clusters, the optimisation-partitioning techniques of section 4.4 may or may not yield contiguous clusters, depending on what the partitioning technique is, and the density seeking techniques yield a variable number of contiguous clusters.

If the structures being sought are contiguous and the number of structures is known, then a technique that ensures contiguous clusters may be more appropriate than one that does not. This should tend to reduce the amount of noise in the clustered (segmented) image. However, if one cluster was influenced by noise, a structure may go undetected in the clustering.

However, it would be an advantage if the number of objects being sought did not need to be known a priori and if the presence of noise did not increase greatly the probability of missing a valid cluster altogether. It is better to have clusters that are a little noisy than to have clusters that could either be noise-free or could just detect noise and miss the structures of interest altogether.

Property (1v) of section 1.3 states that the objects to be extracted by the technique developed during the course of this research need not be contiguous.

4.9 Storage Requirements

In image processing applications, large amounts of data have to be clustered Therefore, storage requirements are a major consideration when selecting an appropriate clustering technique for a given problem. Following is an indication of the amount of storage that some classes of techniques typically need.

Hierarchical techniques do not require the whole of the image being segmented to be in memory at the same time. When determining the difference in value between each pixel and each of its neighbours, only two rows of the image need be in memory at any time. With the algorithm presented in section 4.3.2 to perform single-linkage, the O(N) edges stored in EDGES (defined in (3i) of 4.3.2) should be stored in memory when they are sorted into order of increasing size. However, some other disc-based sort could be used if there was not enough memory to do so. Furthermore, the O(N) values of TREE, which record which cluster each pixel is in at any stage of the process, should also be kept in memory.

With optimisation-partitioning techniques, the amount of memory required is determined by the function that is to be optimised. For example, if a simple global threshold is to be calculated, only the histogram needs to be stored in memory, whereas with the technique that has been developed in this work and which is described in chapter 5, the whole image needs to be stored in memory for the function optimisation.

With density-seeking techniques, usually some multi-dimensional histogram needs to be stored in memory. However, the scales of the dimensions tend to be course.

4.10 Conclusion

As stated in (1iv) and (1v) of section 1.3, the technique developed during the course of this research should segment images successfully, in which the number of objects of interest in the images is not known a priori and the objects need not be contiguous. However, the number of regions into which the image is to be segmented is specified a priori. For example, in an image of some cells, it may be specified that the image is to be segmented into three regions (corresponding to the cell nuclei, cytoplasm and the extra-cellular fluid). However, it is not known, for example, how many nuclei are present, where they are or if they overlap or not. We seek to develop a technique that will detect all the nuclei, regardless of how many there are, or where they are.

Clearly, the more a priori information used in a clustering method, the better chance the technique has of succeeding in finding the "correct" clusters. However, in our applications, the only a priori information available is

(i) The number of, possibly non-contiguous, structures that are of interest

(ii) The quality that distinguishes the structures is pixel intensity.

} (1)

Clearly, hierarchical techniques are inappropriate, as they assume that the features to be extracted are contiguous. Point (1ii) suggests thresholding, while (1i) shows that the number of

threshold values that would need to be calculated for each pixel (i.e. the number of threshold surfaces to be calculated) is known a priori. Moreover, thresholding does not presuppose that the structures are contiguous.

Global thresholding is inappropriate because, from (1ii) of section 1.3, it is seen that the images may be illuminated unevenly. Also, moving window thresholding which uses a window of fixed size is not adequate because the size and location of the structures of interest are not known.

Chapter 5 gives a justification for the thresholding technique developed during the course of this research, and chapters 6 to 8 describe the technique and its implementation. Chapters 9 to 11 show the results of applying the technique and compare them to those of other techniques. Appendix B contains the listing of a Fortran program used to implement the technique developed during the course of this research.

CHAPTER 5

PARTITIONING THROUGH TESSELLATION

5.1 Introduction

As stated in chapter 1, the aim of this work is to develop an image segmentation technique which can segment images with the following properties:

(i) The images to be segmented may have low contrast

(ii) The images may be illuminated unevenly

(iii) The images may contain either large or small objects, or both, of any shape, without the method needing to know any of these details a priori

(iv) The image may contain any number of objects

(v) The images of the individual objects in the image need not be contiguous.

(1)

The technique may assume that:

(i) Objects may be either lighter or darker than their backgrounds, but not both

(ii) The attribute that distinguishes objects from their backgrounds is the difference in pixel intensity, rather than some other attribute such as texture } (2)

(iii) Neither objects nor their backgrounds are highly textured. }

The assumptions made in (2), above, suggest the use of thresholding on pixel value. One of the more sophisticated threshold evaluation techniques described in chapter 3 should be able to segment images with properties (1i), (1iv) and (1v) successfully. However, properties (1ii) and (1iii) are more difficult to deal with and images with these properties will not necessarily be segmented successfully by any of the thresholding techniques encountered during the course of this research, except the one that was developed.

A moving window threshold, such as the moving window median thresholding technique described in section 3.3, can be successful in thresholding unevenly illuminated images. However it is only successful if the size and shape of the feature to be extracted is known a priori, the feature occupies about half of the image and it is either in the centre of the image or, at least, its shape is such that it does not only occupy one region towards the edge of the image.

Clearly, this does not cover all images with properties (1), and so these techniques are not adequate for our purposes.

Assuming that a suitable threshold evaluation technique can be found, the problem of determining the regions of the image for which the histograms will be calculated in preparation for the threshold evaluation, remains unresolved.

If the region for which a threshold is to be calculated does not contain any of the foreground's pixels - or only very few, relative to the number of pixels in the region - then the background will be thresholded into "foreground" and "background", erroneously. Similarly, if the region only contains very few background pixels, then the foreground will be thresholded into "foreground" and "background", erroneously.

On this basis, the standard fixed size and shape moving window thresholding techniques are not adequate. What is needed is for the regions for which the thresholds are to be calculated, to contain both foreground and background pixels, whose numbers are approximately in the same ratio to each other as the ratio of foreground to background pixels in the whole image. As the size and location of the objects of interest in an image are fixed, the windows will need to vary their size and shape to ensure that they contain both foreground and background pixels.

There are two approaches to this:

(3)

(i) Use a moving window (of variable size and shape) to calculate a threshold value for each pixel in the image

(ii) Divide the image into regions (either overlapping or non-overlapping), calculate a threshold value for some point (or points) that characterizes each region, and then interpolate a threshold surface that will be used to threshold the image.

With approach (3i), the threshold evaluation technique must be chosen with caution so as to ensure that the threshold surface generated is continuous. For this and computational reasons, option (3ii) has been adopted in this work.

A technique is sought that will divide the image into regions, such that each region contains some foreground and some background pixels. Also, by (3ii), each region must have an identifiable "characteristic point", however that is defined.

If a threshold value is to be calculated for the characteristic point of each region from that region's histogram, it is highly desirable (if not essential) that the point lie within the region.

A definition of "characteristic point" is sought such that:

(i) Each region has a unique characteristic point

(ii) The characteristic points have some sort of mathematical significance

(iii) Each region's characteristic point is certain to lie within the region

(iv) No two regions may share a characteristic point.

(4)

Condition (4i) can be satisfied easily, but conditions (4ii) and (4iii) can conflict. For example, if a region is torus or U shaped, (4ii) would suggest that the region's characteristic point may lie outside the region itself. Also, if regions are allowed to overlap, their characteristic points may be shared.

The tiles (regions) obtained from the Dirichlet Tessellation [20] of a space satisfy all the criteria in (4), in that they have the following properties:

(i) The tiles are non-overlapping

(ii) The tiles span the space (image)

(iii) Each tile has a unique generating point that can be defined to be its characteristic point

(5)

}

(iv) Each tile's generating point is certain to lie }
within the tile. }

Properties (5iii) and (5iv) satisfy requirements (4i) and (4iii) and because of (5iv), property (5i) satisfies (4iv). Also, the tiles' generating points are defined mathematically, thus satisfying (4ii).

A further discussion of these and other properties of the Dirichlet Tessellation is given in section 5.3. Before that, section 5.2 discusses the Delaunay Triangulation, which is shown in section 5.3 to be the dual of the Dirichlet Tessellation.

Ahuja, An and Schachter [1] describe the use of the Dirichlet Tessellation in describing the texture of images (however, their approach is entirely different to that adopted here), while Schachter, Rosenfeld and Davis [97] describe several different tessellation techniques.

5.2 Delaunay Triangulation

In many applications, such as interpolating a surface through irregularly spaced points, a given set of points is triangulated. This means that as many triangles as possible are drawn, with the points as the vertices of the triangles, such that the triangles do not overlap.

Elementary geometry shows that the boundary of the triangulated region will be a convex hull passing through the outermost points, and with all the other points lying within it. Equally elementary graph theory shows that:

$$N_T = 2N - B - 2 \qquad (1)$$

and

$$N_E = 3N - B - 3 \qquad (2)$$

where: N_T is the number of triangles in the triangulation

N_E is the number of edges in the triangulation

N is the number of points triangulated

B is the number of points lying on the convex hull that is the boundary of the triangulation.

Clearly, except for a few trivial cases, there are many different triangulations through the same points.

The Delaunay Triangulation of a set of points is the triangulation that satisfies the following condition [103]:

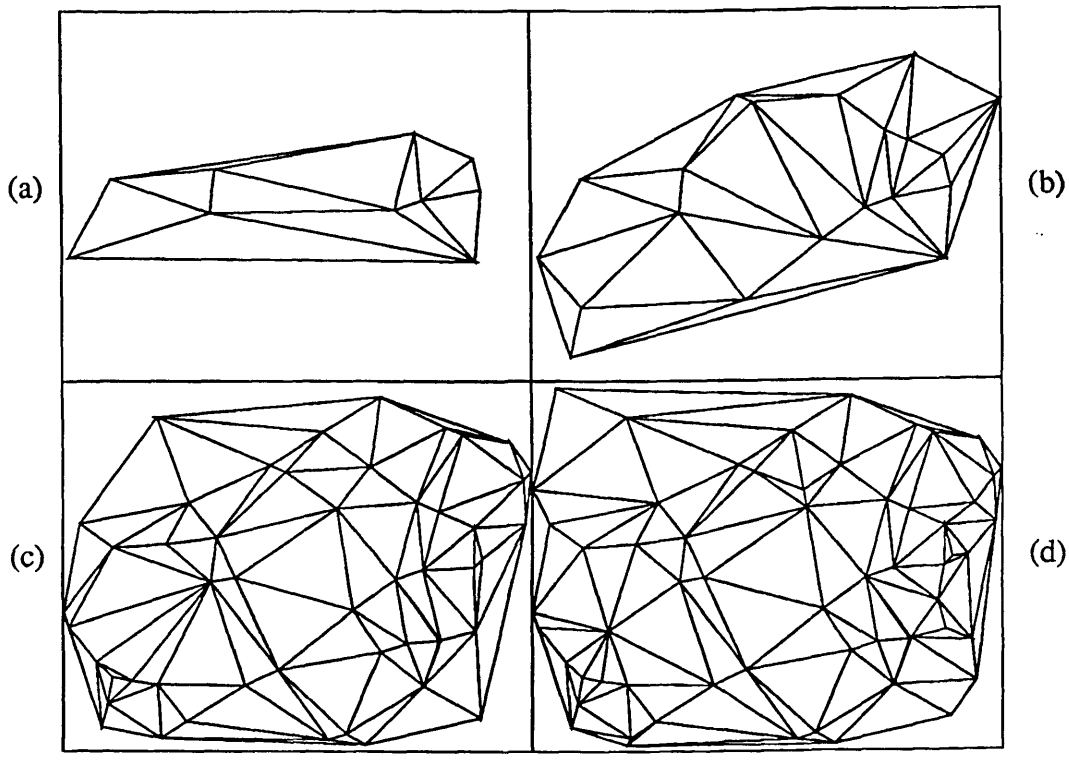
> If two triangles in the triangulation share a common edge, they define a quadrilateral with that common edge as diagonal. If that quadrilateral is strictly convex (that is, each vertex is an external point of it) then replacement of the chosen diagonal by the alternative one must not increase the minimum of the six angles in the two triangles making up the quadrilateral, and this must hold for all such strictly convex quadrilaterals. (3)

Sibson [103] showed that this is equivalent to the following condition:

> None of the triangulated points may lie within the circumcircle of any of the triangles in the triangulation. (4)

Figure 5.1 gives examples of the Delaunay Triangulations of sets of points.

There are, unfortunately, degenerate cases in which the Delaunay Triangulation is not defined uniquely, although the triangulations are usually all equivalent. These degenerate cases arise when more than three vertices (note that the generating points of the tiles are the vertices of the triangles) lie on the same circumcircle that does not enclose any other vertices. An example of this situation is when a regular grid of vertices is triangulated.



Delaunay Triangulation: (a), (b), (c) and (d) show the Delaunay Triangulations of 10, 20, 50 and 60 pseudo-randomly positioned points, respectively. The points being triangulated form the vertices of the triangles.

Figure 5.1

Several algorithms for performing this triangulation - and handling the degeneracies - have been proposed, and the one used in this work is based on that of Monro [72,73] as adapted by Young [127].

As with all triangulations, the technique has one major objection:

The vertices may not all lie in a straight line. (5)

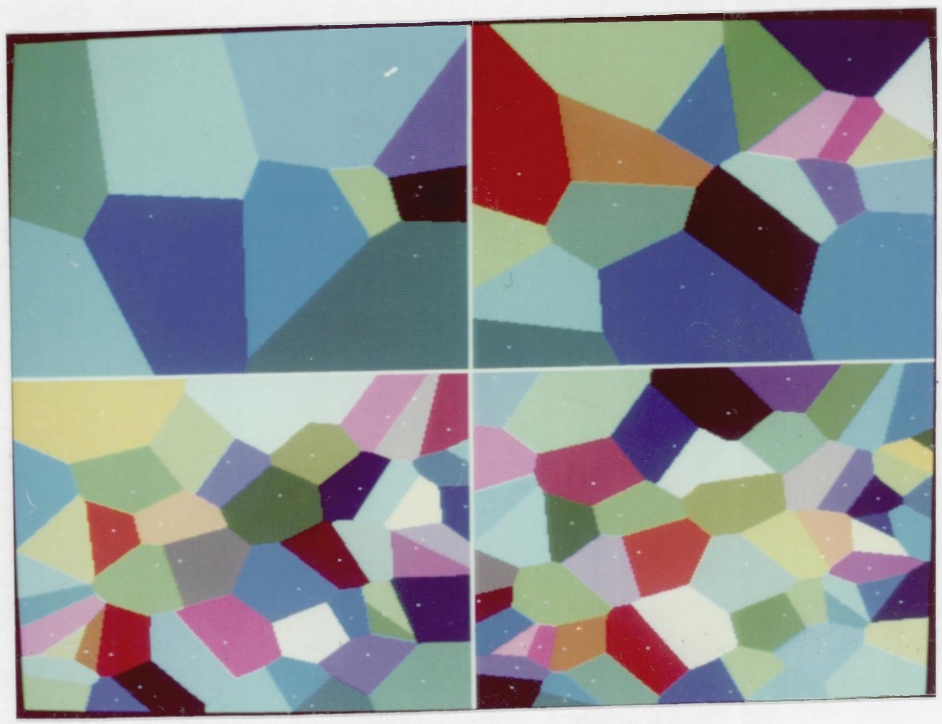
The Delaunay Triangulation is of interest to us in this research when calculating the Dirichlet Tessellations of images and for interpolating the threshold surfaces used for thresholding the images processed. Section 5.3 discusses the use of the Dirichlet Tessellation and chapter 8 describes how the threshold surfaces are interpolated.

5.3 Dirichlet Tessellation

For a given set of points (known as generating points) in a plane, the Dirichlet Tessellation of that plane is defined as:

> that tessellation that associates with each of the generating points a unique tile that comprises the set of points in the plane that are closer to that generating point than to any other generating point. (1)

(a)



(b)

(c)

(d)

Dirichlet Tessellation: (a), (b), (c) and (d) show the Dirichlet Tessellations of 10, 20, 50 and 60 pseudo-randomly positioned points, respectively. The points being tessellated (the generating points of the tiles) are in the same positions as those used figure 5.1 and are shown in the images as white dots. The colours of the tiles have no significance other than to enable one to distinguish between the tiles easily, visually.

Figure 5.2

It has been shown that the Delaunay Triangulation and the Dirichlet Tessellation are duals of each other [9,39,103,115], meaning that each can be computed from the other, uniquely. More specifically, for any generating point, C, the Dirichlet tile is given by those points enclosed by the perpendicular bisectors of the edges in the triangulation, emanating from C (see theorems 5.1 and 5.2 below).

For the purposes of this work, let the following terms be defined as follows:

Generating Point - a point in the plane that is one of those that is either to be triangulated or tessellated, or both

Surround - those other generating points that are connected to a given generating point, by edges in the Delaunay Triangulation of all the generating points

Minimal Surround - those other generating points that are connected to a given generating point, C, by edges in the Delaunay Triangulation of all the generating points, and where degeneracies exist amongst the generating points, the ambiguities in the (2)

triangulation are resolved so as to minimise the number of points in C's surround

Maximal Surround - those other generating points that are connected to a given generating point, C, by edges in the Delaunay Triangulation of all the generating points, and where degeneracies exist amongst the generating points, the ambiguities in the triangulation are resolved so as to maximise the number of points in C's surround.

There are two different approaches to calculating the Dirichlet Tessellation of an image for a particular set of generating points:

(3)
(i) Considering each generating point in turn, determine its tile's boundary analytically

(ii) Considering each pixel in the image in turn, determine in which tile it lies.

While approach (3i) is appealing in analytical applications, it is not suited to automatic, digital applications. While (3ii) is easy to implement digitally, since all that is needed is to calculate

the distance from the pixel in question to each generating point, and then allocate it to the appropriate tile, this is very expensive computationally.

Theorems 5.1 and 5.2, below, form the basis of a theory that is developed here and in chapter 7, that enables us to tessellate images efficiently.

Theorem 5.1:

If a point, P, is closer to a given generating point, C, than to any of the generating points in the minimal surround of C, S(C), then P lies in T(C), the tile of generating point C.

Proof:

Let P and C be arbitary but fixed (see figure 5.3).

From the definition given in (1), we see that it must be shown that point P is closer to generating point C than to any other generating point.

The proof is by contradiction.

Suppose that P is closer to some other generating point, D, than it is to C. By the definition of P, $D \notin S(C)$.

As the boundary of the triangulated area forms a convex hull, clearly, D lies in the wedge formed by the rays CX and CY produced (including the rays themselves), where X and Y are adjacent generating points in S(C). (A)

As the line separating the points that are closer to D than to C is straight (viz. the perpendicular bisector of the line-segment CD), by (A), if P is closer to D than it is to C, then so is point Q, the point of intersection of the perpendicular bisectors of the line-segments CX and CY.

This implies that D lies in the circle circumscribed through C, X and Y, and centred at Q. However, this contradicts (4) of section 5.2.

Therefore, if point P is closer to C than it is to any of the generating points in S(C), then it is closer to C than to any other generating point, and thus, $P \in T(C)$. □

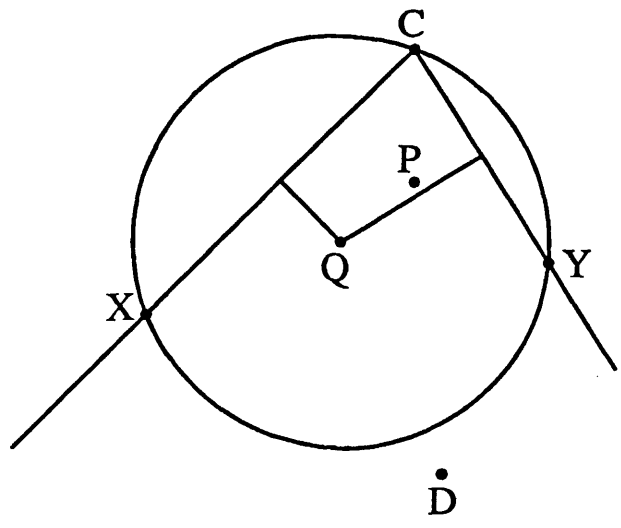


Figure 5.3

Theorem 5.2:

The tile of generating point C, $T'(C)$, in the tessellation of the plane about the set of generating points given by $\{C\} + S(C) - \{Y\}$, where $S(C)$ is the set of generating points in the minimal surround of C and $Y \in S(C)$, is necessarily different to the tile of C, $T(C)$, in the tessellation of the plane about the set of generating points given by $\{C\} + S(C)$.

Proof:

Case 1: C and Y are both members of the convex hull formed by the triangulation of all the generating points (see figure 5.4).

Let X be the generating point adjacent to Y in $S(C)$, and let M_X and M_Y be the midpoints of the line segments CX and CY, respectively. As CX and CY are not parallel, neither are their perpendicular bisectors, which must therefore intersect, say at point P.

Clearly, the wedge formed by the rays PA and PM_Y, where A is a point on M_XP produced, contains the non-empty set of points that are in $T'(C)$ but not in $T(C)$.

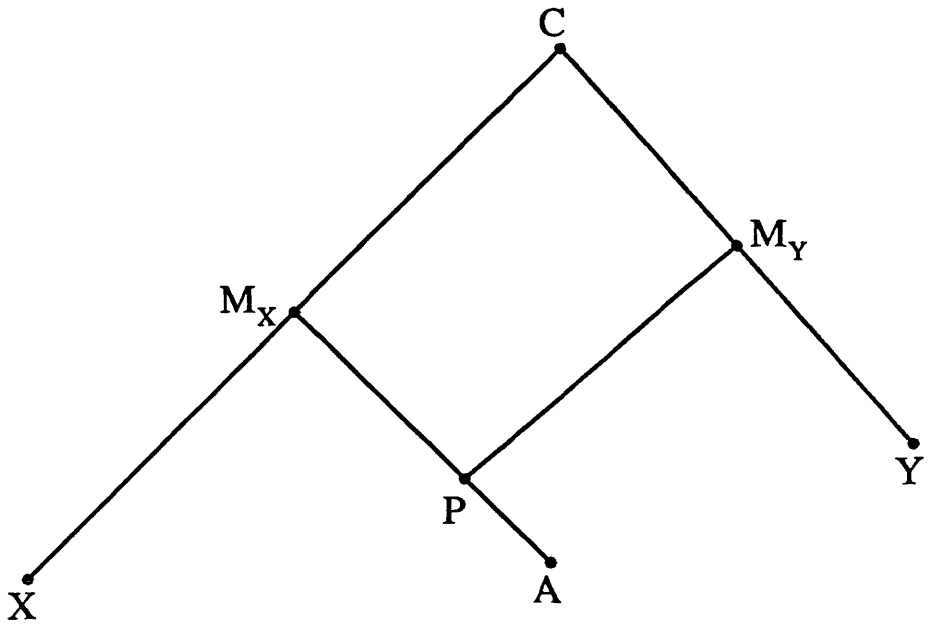


Figure 5.4

Case 2 : C and Y are not both members of the convex hull formed by the triangulation of all the generating points (see figure 5.5).

Clearly then, Y has two other generating points that are adjacent to it in S(C), say X and Z.

As the line segments CX and CZ are not parallel, neither are their perpendicular bisectors, which must therefore intersect, say at point P. Clearly P is the centre of the the circle circumscribed through C, X and Z. (A)

By (4) of section 5.2, none of the generating points in S(C) - {X,Y,Z} lie in this circle. Therefore, P is closer to C, X and Z (to which it is equidistant) than to any of the generating points in S(C) - {X,Y,Z}, which implies that there exist points at distance less the δ (a finitely small number) from P that are closer to C than to any of the generating points in S(C) - {Y}, and which therefore lie in tile T'(C).

However, from (4) of section 5.2, we see that Y must necessarily lie in the circle described in (A), otherwise X would lie in the circle circumscribed through C, Y and Z, and so P and the points of distance less than δ from it, are closer to Y than to C, and therefore do not lie in T(C).

Therefore, the tiles T'(C) and T(C) are always necessarily different. □

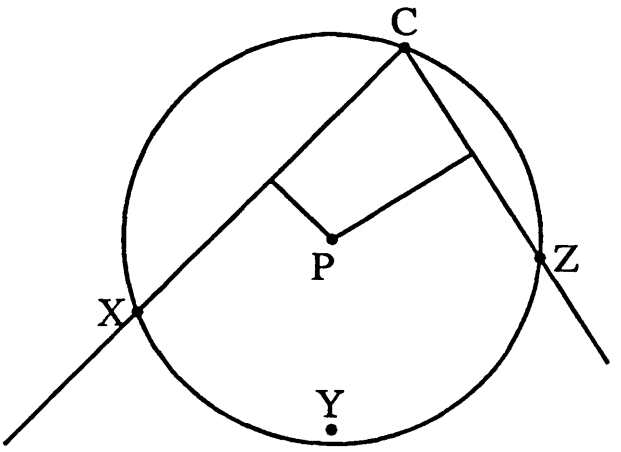


Figure 5.5

These theorems show that, when calculating the locus of points in a generating point's tile, one must, and need only, consider the generating point in question and those in its minimal surround. Alternatively, when determining whether a point lies in a particular generating point's tile or not, one must and need only calculate the distance of the point from the generating point concerned and from the generating points in its minimal surround.

In chapter 7, it will be shown how the results of these theorems can be used to reduce the amount of computation required to tessellate an image.

Furthermore, theorem 5.3 below shows that the average number of points in a generating point's minimal surround is strictly less

than 6.

Theorem 5.3:

The average number of points in a generating point's minimal surround is strictly less than 6.

Proof:

From (1) in section 5.2, and using the same notation as there, we have that:

$$N_T = 2N - B - 2 \qquad (A)$$

Clearly, each triangle will appear in the triangulation of 3 generating points and their surrounds, so that the sum of number of points in each each generating point's surround, T_S, is given by:

$$T_S = 3.N_T + B \qquad (B)$$

From (B), we have that the average number of points in each generating point's surround, A_S, is given by:

$$\begin{aligned} A_S &= (3.(2N - B - 2) + B) \; / \; N \\ &= 6 - (2B + 6) \; / \; N \end{aligned} \qquad (C)$$

Which is strictly less than 6. Hence, the average number of points in a generating point's minimal surround will also be less than 6. □

5.4 Conclusion

It is proposed that the Dirichlet Tessellation is a good method for dividing an image into regions, each of whose generating points will be given a threshold value calculated from the histogram for that region. The reasons for this assertion are as follows:

(i) The tiles span the image space. This ensures that all the pixels are considered when generating the threshold surface

(ii) The tiles are non-overlapping

(iii) Each tile has a unique generating point that always lies in that tile. This makes it reasonable to assign the threshold value calculated from a tile's histogram to that tile's generating point

(1)

(iv) No point can be the generating point of more than one tile. This is deduced from (ii) and (iii)

(v) Each set of generating points, defines a unique tessellation. This makes a given tessellation easy and efficient to describe

(vi) By varying the positions of the generating

points, the tiles can be made to span the objects and their backgrounds.

The Dirichlet Tessellation also has the following desirable properties:

(i) The tessellation can be calculated quite efficiently

(ii) All the points in a particular generating point's tile are closer to that generating point than to any other generating point. Besides being intuitively appealing, it shall be shown, in chapter 8, that this is a desirable property when the threshold surfaces are to be interpolated. (2)

In chapter 3, a number of thresholding techniques were described and in this chapter, it has been shown how a set of generating points can be used to segment the image space into a unique tessellation that has the useful properties described in (1) and (2) above. It now remains to define a method to define the location of the generating points such that the resultant tiles each contain both foreground and background pixels in approximately the same ratio as that of the number of pixels in the foreground and background in the whole image.

For a given set of generating points, a unique tessellation has been defined. The tessellation can be changed by varying the positions of the generating points.

A "goodness of tessellation" measure is needed to evaluate whether a particular tessellation has the desired properties or not, and a technique needs to be defined to locate the set of generating points that maximises this measure for the resultant tessellation.

Chapter 6 deals with the former issue, while chapter 7 deals with the latter. Chapter 8 then shows how the discrete threshold values calculated for the possibly irregularly spaced generating points can be used to generate a continuous threshold surface, with which the image can be thresholded.

Chapter 9 shows some results from applying the method developed during the course of this research, as well as those from applying other techniques, and chapters 10 and 11 will evaluate the results obtained.

CHAPTER 6

CLUSTERING CRITERIA

6.1 Introduction

In section 5.4, it was proposed that the Dirichlet Tessellation is a good technique for dividing an image into regions, for which threshold values can be calculated.

In this investigation, the feature distinguishing the foreground from the background pixels is pixel brightness. Therefore, if a threshold value is to be calculated for the generating point of each tile from the histogram of that tile, the histogram should contain pixels from both the foreground and the background, viz. both light and dark pixels. It is primarily the lack of this property that makes the otherwise promising technique of using moving windows of fixed size and shape inappropriate in many applications. Furthermore, it is desirable that the ratio of foreground to background pixels in each of the tiles be approximately the same. Also, when calculating threshold values from n-modal histograms, it is easier to calculate good threshold values if the modes are spaced far apart rather than close together, as the modes are then clearly distinguishable.

Given this, a measure of how good a particular tessellation of a given image is, in terms of the criteria given above, is needed.

It is seen then that the measure required should:

(i) Measure the "flatness" of a given histogram, or at least measure, in some way, the "spread of pixel values", whatever that may mean } (1)

(ii) Distinguish between histograms with spikes (modes) far apart and those with spikes (modes) closer together. } (1)

The remaining sections of this chapter introduce and examine several measures of statistics relating to histogram flatness and distance apart of spikes in histograms.

Section 6.2 discusses the use of entropy (described in section 2.2.3) and examines its appropriateness as a "flatness of histogram" measure. Section 6.3 discusses the use of variance as a measure, and in section 6.4, a new measure developed during the course of this research, is described and evaluated.

Section 6.5 shows how the measures described in sections 6.2 to 6.4 can be generalised to measure the goodness of a given tessellation rather than the goodness of a particular tile within that tessellation.

6.2 Entropy

As stated in section 2.2.3, entropy is a measure of information content; the higher the entropy, the more information conveyed by each piece of data.

In image processing, the entropy, H(x), of a region can be calculated from the probability density function of that region as follows:

$$H(x) = - \sum_{\substack{i=0 \\ P_i \neq 0}}^{n} P_i . \mathrm{Log}(P_i) \qquad (1)$$

where: P_i is the fraction of the pixels in the region with value i

n is the largest permissible pixel value.

The base of the logarithm in (1) serves only as a positive scaling factor and in this research, base e was used. H(x) is smallest when all the pixels have the same value, in which case H(x) = 0, and is largest when there are equal numbers of pixels of each of the permissible pixel values, in which case H(x) = Log(n+1).

This measure can distinguish between histograms which are reasonably flat and those in which the P_i vary greatly, thus satisfying criterion (1i) of section 6.1. For this reason, it was used by Fernando [28] as a "flatness of histogram" measure in his research.

However, it is not immediately apparent that the two assumptions made in the justification of equation (1), given in (3) of section

2.2.3, are justified in this application. Furthermore, the measure does not distinguish between histograms with spikes that are close together and those with spikes the same size, but which are further apart. Therefore, it does not satisfy criterion (1iii) of section 6.1, and a better measure should be sought.

However, entropy is a commonly used measure in image processing and Fernando [28] did use it successfully. Therefore it was used in some of the experiments carried out during the course of this research and the results are given in chapters 9 and 10, as well as those from using the measures described in sections 6.3 and 6.4.

6.3 Variance

As stated in section 2.2.4, the variance of the values of the pixels in a region is given by:

$$\sigma^2 = \frac{1}{N}\sum_{D} x_i^2 - x'^2 \qquad (1)$$

where: x_i is the value of the i^{th} pixel in region D

x' is the mean value of the pixels in region D

D is the region of the image for which the variance is being calculated

N is the number of pixels in the region D.

The variance of the pixel values in a region measures the "spread of pixel values" in that region in that it is the mean square deviation of the pixel values from the mean. To this extent, it does give some indication of the shape of the histogram.

However, in ascertaining whether variance, as defined in (1) above, satisfies the criteria given in (1) if section 6.1, it is clear that:

(i) It does not measure the "flatness" of a given histogram

(ii) It does measure the "spread of pixel values", where this is taken to mean the "average of the deviation from the mean, squared"

(iii) The variance of the values of the pixels in a (2)

region whose histogram contains two spikes close together will be less than that of a histogram with spikes further apart, if the spikes are on opposite sides of the mean pixel value, in the histogram. However, if the spikes lie on the same side of the mean value, the opposite will be true.

Because of (2ii), (2i) may be acceptable. However, (2iii) poses a serious objection to the use of variance, as defined in (1) above, as a measure in our applications. Clearly, it not only does not satisfy criterion (1ii) of section 6.1, but it is inherently unstable and unpredictable as a measure in our applications. For this reason, this measure was not used in the course of this research.

Another approach to using variance as a measure, is the more direct one of using the variance of the probability density function, which from (2) of section 2.2.4, we have as:

$$\sigma^2 = \frac{1}{N} \cdot \frac{\Sigma}{R} \left(P_i - \frac{1}{N} \right)^2 \qquad (3)$$

where: P_i is the fraction of the pixels in the region with intensity i

R is the range of permissible pixel values

N is the number of permissible pixel values.

This measures the spread of the values in the probability density function.

Clearly, the measure is minimised when all pixel values are equally probable (i.e. the histogram is flat and $P_i=(1/N)$ $\forall$ i) and is maximised when all the pixels have the same value. However, as with entropy, this measure can not distinguish between histograms with spikes far apart and those with spikes close together.

We see then that:

(i) Variance of the PDF, as defined in (3) above, satisfies criterion (1i) of section 6.1, in that it does measure "flatness of histogram"

(ii) The measure does not take into account the distance apart of spikes in the histogram. (4)

Property (4ii) means that this measure does not have all the properties required of the measure as described in (1) of section 6.1. However, it is a most direct and intuitively appealing approach to satisfying criterion (1i) of section 6.1, and it does not have the disadvantages of entropy described in section 6.2. Therefore it was used as a benchmark in this research, and results from using it as a measure are presented in chapters 9 and 10.

6.4 Bumpiness

In sections 6.2 and 6.3, it was shown that neither entropy nor variance satisfy the criteria (1) of section 6.1. Therefore, during the course of this research, a new measure, "bumpiness", was defined.

This section introduces and justifies this measure, leading to a formal definition of the measure, which is finally generalised.

Bumpiness measures the flatness of a given histogram, but considers histograms with spikes far apart to be flatter than histograms with spikes the same size, closer together.

The criteria (1) of section 6.1 refer to distance apart of spikes in histograms. However, no mention is made of valleys in histograms. Therefore, at least to start with, valleys in histograms should be ignored.

For the purposes of this research, the following terms are defined for histograms:

Spike:- Those values that are greater than the mean value, viz. histogram values reflecting a more than average probability of a pixel chosen at random having the corresponding pixel value.

Valley:- Those values that are less than the mean value, viz. histogram values reflecting a less than average probability of a pixel chosen at random having the corresponding pixel value.

Height of spike:- The height of a spike at position i of a given histogram is defined as $P_i - P'$, where P_i is the fraction of the pixels in the region (i.e. tile) having value i, and P' is defined as the number of pixels in the region divided by the number of permissible pixel values.

Depth of valley:- The depth of a valley at position i of a given histogram is defined as $P' - P_i$, where P_i is the fraction of the pixels in the region (i.e. tile) having value i, and P' is defined as the number of pixels in the region divided by the number of permissible pixel values.

As the primary interest, at least at this stage, is with spikes, we define a "height of spike" measure, K_i, as:

$$K_i = \begin{cases} 0 & \text{if } P_i < \frac{1}{N} \\ P_i - \frac{1}{N} & \text{otherwise} \end{cases} \qquad (1)$$

where: N is the number of permissible pixel values

P_i is the fraction of the pixels in the region with value i.

The measure of histogram bumpiness is obtained by analogy with

electrical charges.

From elementary electrostatics, it is known that the electrostatic potential energy, E_p, in a system containing N like charges is given by:

$$E_p = K. \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} \frac{q_i . q_j}{r_{ij}} \qquad (2)$$

where: K is a constant

q_i is the magnitude of charge i

r_{ij} is the distance between charge i and charge j.

The charges will tend to move to the position that minimises the electrostatic potential energy.

Treating the spikes as point charges with magnitude K_i, the "potential energy" of a histogram, F, can be defined as:

$$F = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} \frac{K_i . K_j}{j-i} \qquad (3)$$

Clearly, F is minimised when the histogram is flat (all the K_is are equal to 0), and it will yield a smaller value when spikes in the histogram are far apart than when they are closer together. However, F is also minimised when all the pixels have the same value. It would be better for our purposes if the measure was maximised in this case, instead of minimised.

Therefore, the bumpiness, B, of a given histogram is defined as:

$$B = \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} \frac{K_i . K_j}{j-i+1} \qquad (4)$$

The case where all the pixels have the same value is a "worst case" using B as the measure, rather than a "best case" when using F. In all other respects, B and F are very similar.

This measurement of bumpiness, B, satisfies the criteria (1) of section 6.1, in that:

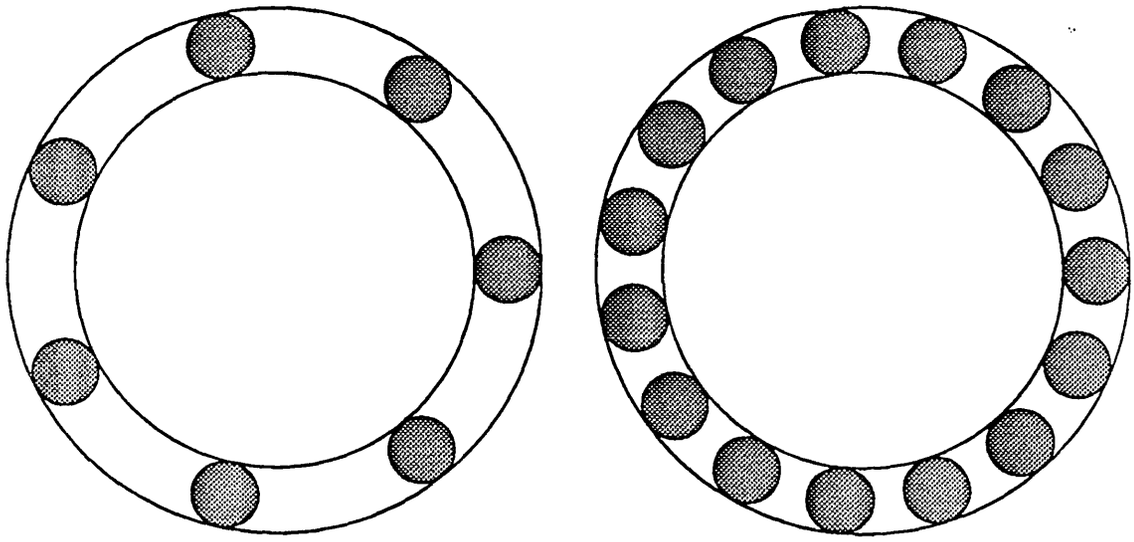
(i) The measure does measure the flatness of a given histogram, in that, the smaller the histogram's spikes (viz. the smaller the K_is), the smaller the bumpiness, B

(ii) The measure does take into account the distance apart of spikes, in that the further apart the spikes are, the smaller the bumpiness, B. (5)

In evaluating the bumpiness measure, it is important to consider where the spikes should be in a histogram containing a fixed number of spikes, all the same height, so as to minimise the bumpiness B.

The solution is understood most clearly by returning to the analogy of the charged particles.

Given a set of point charges (the magnitude and sign of all the charges is the same) in a circular tube (torus), in a system in which the repulsive force between the charges acts only within the tube, clearly, the particles would tend to space themselves out evenly around the tube (see figure 6.1).

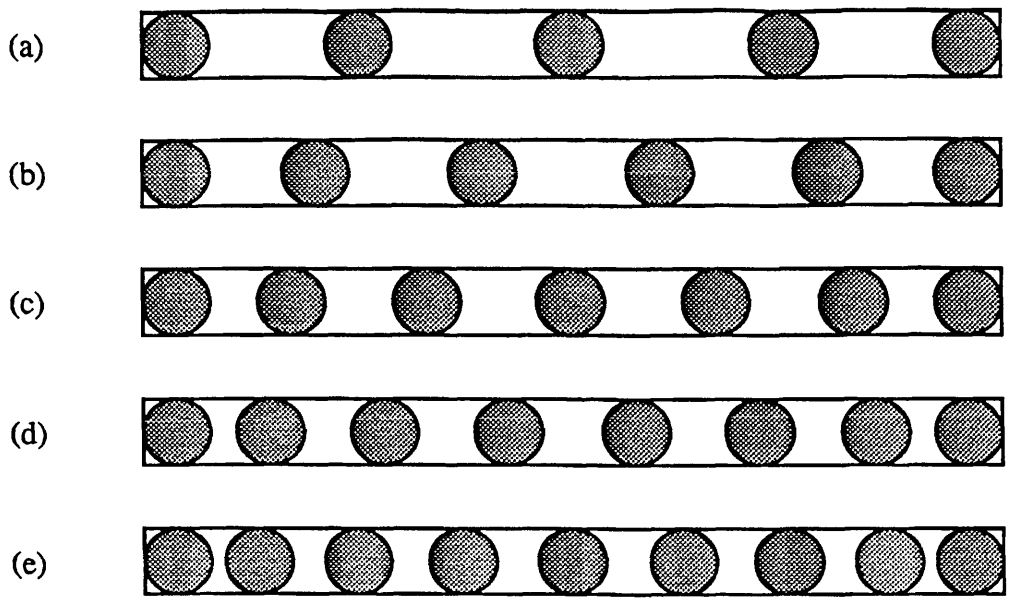


Positioning of charges in a torus: 'Optimal' positioning of 7 and 15 charged particles, respectively, in a torus.

Figure 6.1

If we now consider the same set of charges, obeying the same laws, but this time placed in a straight tube (see figure 6.2), clearly, the charges would repel each other with the same force, along the tube, but the wrap-around effect of the circle would be removed. That is to say that the points would not be repelled from the ends of the tube as much as they would be if there was wrap-around.

Clearly then, the charges would tend to bunch slightly towards the ends of the tube.



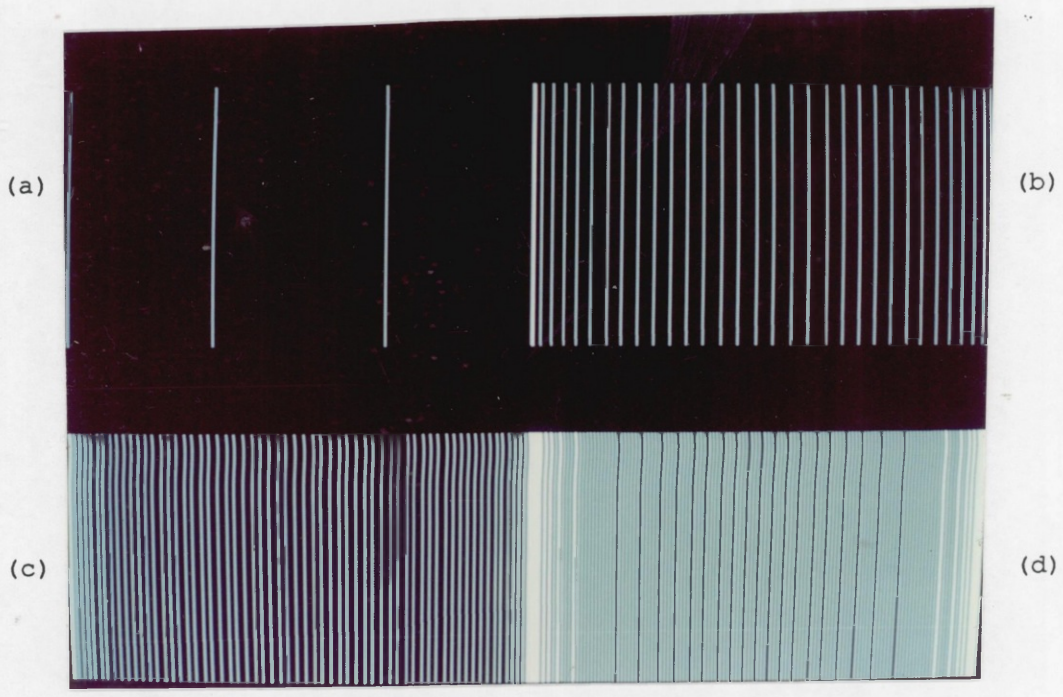
Positioning of charges in a straight line:
(a), (b), (c), (d) and (e) show the positioning of 5, 6, 7, 8 and 9 charged particles, respectively, on a line segment that minimises the electrostatic potential energy.

Figure 6.2

Similarly, with a histogram containing a fixed number of spikes, all the same size, the bumpiness would be minimised if the spikes were fairly evenly spaced, but were slightly bunched towards the ends of the histogram. This has been verified by finding the positions of the N spikes, in a histogram containing exactly N spikes of equal height, that minimise the bumpiness measure, B. Some of the results of these experiments are given in figure 6.3.

Clearly, this is a desirable property for a histogram from which a threshold value is to be calculated, to have.

We see then that bumpiness satisfies the criteria (1) of section 6.1, and therefore advocate it as a good "flatness of histogram" measure.



Positioning of spikes that minimises the bumpiness: (a), (b), (c) and (d) show the positions that 4, 32, 64 and 128 spikes, respectively, would need to occupy in a histogram, so as to minimise the bumpiness of that histogram.

Figure 6.3

The measure can be generalised to deal with valleys as well as spikes. For example, it may be desirable to try to avoid wide, deep valleys in histograms, but specify that wide, deep valleys are still more desirable than wide, high hills.

An explanation of how the bumpiness measure is generalised follows.

Define $K_{\alpha,i}$ as follows:

$$K_{\alpha,i} = \text{Max}(T_i \ , \ -\alpha.T_i) \tag{6}$$

where: $T_i = P_i - (1/N)$

α is a constant in [0,1]

N is the number of permissible pixel values

P_i is the fraction of the pixels in the region with value i.

The generalised bumpiness measure, B_α, is then defined as:

$$B_\alpha = \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} \frac{K_{\alpha,i}.K_{\alpha,j}}{j-i+1} \tag{7}$$

When α=0, valleys are ignored completely, and when α=1, valleys are treated as if they were spikes of height equal to the depth of the valley concerned.

Clearly:

$$B = B_0 \tag{8}$$

In chapters 9 and 10, results are presented from using bumpiness measure B_α, with α=0 and α=0.5.

6.5 Conclusion

Sections 6.2 to 6.4 discussed several measures which could be used to evaluate the desirability of a particular tile, in our applications. However, any particular tessellation could contain both desirable and undesirable tiles. What is needed is a method of generalising these measures to give a "goodness of tessellation" measure.

For the purposes of this research, this goodness of tessellation measure, G, was taken to be a weighted sum of the goodness of tile measures for each of the tiles in the tessellation, viz.

$$G = \sum_{i=1}^{t} W_i . M_i \qquad (1)$$

where: W_i is the weighting factor for tile i

M_i is the goodness of tile measure for tile i

t is the number of tiles in the tessellation (viz. the number of generating points).

and

$$\sum_{i=1}^{t} W_i = 1 \qquad (2)$$

During the course of this research, it was found that setting

$$W_i = 1 / t \qquad (3)$$

gave good results for all the measures described in sections 6.2 to 6.4, except measure F given in (3) of section 6.4. With this

measure, some tiles ended up being very small and only containing pixels of a single value (the "best case"), while other tiles were very large. However, with all the other measures, in which tiles containing pixels of only one value are a "worst case", the tessellations were reasonable.

Therefore, all the results presented in chapters 9 and 10 used the value for W_i given in (3).

CHAPTER 7

OPTIMISATION OF TESSELLATION

7.1 Introduction

In chapter 5, it was proposed that, for a given set of generating points, the Dirichlet Tessellation of an image is a good preliminary segmentation of the image into regions for which threshold values could be calculated.

Clearly, each set of generating points defines a unique Dirichlet Tessellation. However, these tessellations are not all equally good. Chapter 6 introduced and evaluated several "goodness of tessellation" measures which give an indication of how desirable a particular tessellation is, in the sense of (1) of section 6.1.

Clearly, for a given image, each of these measures can be considered to be a function of the positions of the generating points.

The problem then is determining where the generating points should be positioned so that the tessellation is optimal in terms of one of the measures described in chapter 6, for any given image. This is equivalent to finding the positioning of the generating points that optimises the goodness of tessellation measure used. This problem is investigated in this chapter.

Section 7.2 investigates the characteristics of the optimisation problem, identifying some of the features that make it difficult using currently available techniques.

Section 7.3 shows that none of the commonly used optmisation techniques is suited to the problem, and suggests an approach to the problem, which is a form of simulated annealing.

Section 7.4 shows how the optimisation technique proposed in section 7.3 can be implemented efficiently, thus making the problem solvable in a "reasonable time" using currently available computing power.

Section 7.5 summarises the conclusions of this chapter.

7.2 Characteristics of the Optimisation Problem

As stated in section 7.1, the goodness of a particular tessellation depends on the tessellation, and the Dirichlet Tessellation of an image is a function of the positions of the generating points. Therefore, the goodness of tessellation (for a particular goodness of tessellation measure) for a particular image is a function of the positions of the generating points.

Clearly, for the case of N generating points, this function can either be considered a function of 2N integers (the co-ordinates of the generating points), or of N complex values. For the purposes of this discussion, the latter approach shall be adopted.

Define:

$$\left.\begin{array}{ll} \mathrm{Re}(X) & \text{to be the real part of the complex value } X \\ \mathrm{Im}(X) & \text{to be the imaginary part of the complex value } X \\ f(\underline{x}) & \text{to be the goodness of tessellation measure for a particular image, for the set of generating points } \underline{x}, \text{ where } \underline{x} = (x_1, x_2, \ldots, x_N) \text{ and } x_i \in \mathbb{C}. \end{array}\right\} \quad (1)$$

Clearly,

(i) $0 \leq Re(x_i) \leq L-1$, where L is the number of rows of pixels in the image

(ii) $0 \leq Im(x_i) \leq M-1$, where M is the number of pixels in one row of the image

(iii) $x_i \neq x_j$ for $i \neq j$

(iv) $f(x_1,..,x_{i-1},x_i,x_{i+1},..,x_{j-1},x_j,x_{j+1},..,x_N) = f(x_1,..,x_{i-1},x_j,x_{i+1},..,x_{j-1},x_i,x_{j+1},..,x_N)$ (2)

(v) If the x_is all lie in a straight line, while the Dirichlet Tessellation will be defined, the Delaunay Triangulation will not be. Therefore, the efficient technique, given in chapter 5, for calculating the Dirichlet Tessellation for a given set of generating points, will not work.

For the purposes of this work, L and M above were always taken to be equal to each other, and unless stated otherwise, they were equal to 256.

For 256x256 images, N was taken to be 16, and for 512x512 images, N was taken to be 64.

$f(\underline{x})$ is not a well behaved function. It has many unusual and/or undesirable properties, among them:

(i) It is only defined over a small domain (from (2i) and (2ii) above)

(ii) It has several values for which it is undefined (from (2iii) above)

(iii) It has many symmetries (from (2iv) above)

(iv) It is a function of many variables

(v) There is no analytical method for calculating its derivatives, of any order, and the function is such that it would be difficult to approximate them accurately and efficiently.

(vi) The function is computationally very expensive to evaluate

(vii) The function could have very many local extrema, and (iii) above suggests that it probably does.

(3)

Clearly, properties (3) above make $f(\underline{x})$ a difficult and computationally very expensive function to optimise.

Because of (3v) and (3vii), no commonly used optimisation technique was found that was suitable to optimise $f(\underline{x})$. Section 7.3 describes a new and more suitable optimisation technique, and section 7.4 shows how some of the features of $f(\underline{x})$ can be used so that this technique can be implemented efficiently.

7.3 A New Approach

7.3.1 Introduction

A number of standard texts describe the commonly used deterministic optimisation methods [6,13,17,22,31,58,77]. However, none of the techniques described in these works was found to be suitable. Most of the techniques need to know the function's partial derivatives, which are not available.

The simplex method (Caceci and Cacheris [14] present the listing of a program that implements the simplex method efficiently) does not require derivatives, but it does not perform well in the presence of many local extrema.

Slump and Hoenders [105] describe a technique for finding the global maximum of a function in the presence of several local extrema, but their technique requires derivatives and uses deflation. In this problem, the derivatives are unavailable and deflation would be very expensive computationally.

It is usually very difficult to visualise a function in many variables, where the global extrema are and how techniques like the simplex method operate in many dimensions. However, it is relatively easy to visualise what is happening with $f(\underline{x})$, the function that is to be optimised in this case, in that it can be seen in terms of the generating points and tiles in the sub-plane.

A technique is needed that can use the features of the optimisation problem given in section 7.2 to its advantage rather than considering them limitations. Furthermore, there is nothing

analytically or mathematically significant about the optimum value of the function (unlike finding the roots of an equation, for example), other than that it is better than any other answer. That is to say that all sets of positions of the generating points have an associated "goodness of tessellation" value and the optimum positioning of the generating points is that one which has the greatest goodness of tessellation value.

Let $\underline{x}$ be the optimal positioning of the generating points (i.e. it is the optimum value), and let $\underline{y}$ be some other positioning of the generating points such that $\underline{x}$ and $\underline{y}$ are far apart but $|f(\underline{x})-f(\underline{y})|$ is relatively small, then $\underline{y}$ is almost as good a solution as $\underline{x}$, even though the values are far apart. This may not be the case if the roots of an equation were being sought, for example.

The optimisation technique proposed here is a Monte Carlo simulated annealing type approach. It is understood more easily by considering the following analogous problem.

In a twisted piece of metal, there may be many internal stresses. It is known that a good method for relieving these stresses is to heat up the piece of metal until it is very hot (the molecules get "excited" and move around more freely, trying to find a position or orientation in the piece of metal that will relieve the stresses on them) and then let it cool down slowly. This is known as annealing.

Our problem is analogous to this in that the generating points can be considered to have a stress on them proportional to the amount

by which their functional value (goodness of tessellation measure) deviates from the optimum functional value. Also, the positions of the generating points are limited to a finite, bounded region of a plane, and two generating points cannot occupy the same position in that region.

We can simulate the annealing process described above by randomly moving the generating points over a wide range of co-ordinates at first, but later by less and less, until we converge towards a solution that is stable. With the piece of metal, all the internal stresses are not necessarily relieved, and the amount of relief given to them will depend on how much the metal was heated up and how slowly it was allowed to cool down. But in any case, there will be a considerable reduction in the internal stress of the metal object. Similarly, we are not certain to have the optimal solution. The goodness of the solution will depend on how much we let the generating points move around and for how long. Clearly, we have a trade-off between speed and accuracy.

Section 7.3.2 describes how this technique is implemented.

7.3.2 The Method

From section 7.3.1, we see that the optimisation technique has the following steps:

(i) Specify an initial configuration for the generating points

(ii) Let the generating points vary their positions, by large distances at first, and then by lesser amounts, until we converge towards a stable solution.

(1)

Because of the sequential nature of the computer architecture used for this research, it is not practical to let the generating points all vary their positions simultaneously. Therefore a Monte Carlo technique is used whereby a generating point is chosen at random and is allowed to move to another position within a square region about its original position. If this causes an improvement in the goodness of tessellation measure, the generating point is allowed to remain in its new position and the size of the square is increased (or at least allowed to remain the same). If there is no improvement, the generating point is restored to its previous position and the size of the square is reduced. The process is repeated for another generating point, and the whole process is repeated until the square is of unit size.

For the purposes of this research, the random number generator used was that of Wichmann and Hill [120].

Following is a description of the algorithm for M generating points on an NxN image.

Defining the following variables:

STEP - the maximum amount by which $Re(x_i)$ and $Im(x_i)$ may be changed at any iteration

BEST - the goodness of tessellation measure associated with the most optimal set of generating points found so far, at any iteration

ATTEMPT - the goodness of the tessellation currently under consideration

the algorithm is as follows:

(i) Set STEP = N.

(ii) Arrange the generating points into an initial (e.g. a square grid) formation in the image space.

(iii) Calculate and store each generating point's minimal surround.

(iv) Compute each generating point's tile from its minimal surround, and then calculate each tile's goodness of tile measure, and store it.

(v) Set BEST equal to the goodness of tessellation measure, viz. the weighted sum of the individual generating point's tile's goodness of tile measures.

(vi) Choose a generating point, C, at random.

(vii) Generate two random integer values, Δ_x and Δ_y, each in the interval [-STEP,STEP].

(viii) If $(Re(C) + \Delta_x) < 0$, then
set $\Delta_x = (\Delta_x * Re(C) / STEP)$,
else if $(Re(C) + \Delta_x) > (N-1)$, then
set $\Delta_x = (\Delta_x * (N - Re(C) - 1) / STEP)$.

(ix) If $(Im(C) + \Delta_y) < 0$, then
set $\Delta_y = (\Delta_y * Im(C) / STEP)$,
else if $(Im(C) + \Delta_y) > (N-1)$, then
set $\Delta_y = (\Delta_y * (N - Im(C) - 1) / STEP)$. (2)

(x) Set the real and imaginary parts of C equal to $Re(C) + \Delta_x$ and $Im(C) + \Delta_y$ respectively. If another generating point already has this value, then restore C's value to what it was in step (ix) and go back to step (vii).

(xi) Recalculate the minimal surround, tiles and goodness of tile measures for all the generating points that were in C's minimal surround

(xii) Calculate C's new minimal surround, tile and goodness of tile measure

(xiii) Calculate the minimal surround, tiles and goodness of tile measures for all the generating points in C's minimal surround

(xiv) Calculate the new goodness of tessellation measure, ATTEMPT

(xv) If ATTEMPT is better than BEST, then allow C to remain where it has been moved to, set BEST = ATTEMPT, and set STEP = Min(STEP+20,N), otherwise, restore the generating point data as it was before step (x) above, and decrement STEP.

(xvi) If STEP > 0, then go to step (vi) above, otherwise stop.

Variations of this technique were tried, that accumulated statistics concerning which types of movements of generating points appeared to be more acceptable than others (for example,

generating points may tend to move away from the top of the image to the bottom right hand corner) and then used these statistics to bias the movement of generating points in the future. However, it was found that the results were no better, the number of iterations required was essentially the same, and the storage and computation required at each iteration were increased.

Section 7.2 described features of the function that made it necessary to develop a new optimisation technique. Section 7.4 will show how other features of the function can be used to implement the optimisation technique developed here, efficiently.

7.4 Some Implementation Details

The Dirichlet Tessellation has two properties that make the optimisation of the goodness of tessellation measure, $f(\underline{x})$, easier using the technique described in section 7.3, namely:

(i) Each generating point's tile can be specified completely with reference only to the generating point concerned and the generating points in its least surround

(ii) Each tile is convex.

(1)

Property (1i) means that when a particular generating point is moved from one position to another, the only tiles that need to be recomputed (and then have their goodness of tile measures recalculated) are those for the generating point concerned, the generating points that were in that generating point's minimal surround and those that will now be in that generating point's minimal surround.

This is done most efficiently by storing, for each generating point, the numbers of the generating points in that generating point's minimal surround, and the goodness of tile measure for the generating point's tile. When a generating point is moved and the relevant tiles and their goodness of tile measures have been recomputed, the goodness of tile measures for all the tiles needs to be combined to give the new goodness of tessellation measure.

Theorem 5.1 showed that a particular generating point's tile could

be computed from that generating point and the generating points in its minimal surround. Theorem 5.2 showed this to be the mimimal subset of generating points that is necessary to compute the tile, and theorem 5.3 showed that the average number of generating points in a particular generating point's minimal surround is strictly less than six. Above, we have specified the minimum number of tiles that need to be recomputed when a particular generating point is moved. The only remaining question is, given that a particular generating point's tile must be computed from that generating point and the generating points in its minimal surround, how this is done most efficiently.

In (1ii), we see that each tile is convex. From this and (1iii) of section 5.4, we see that:

(i) Each generating point lies in its own tile

(ii) If two points (pixels) lie in a particular generating point's tile, so do all the points on the straight line between them

(iii) Each tile consists of exactly one connected region.

(2)

From (2), we see that in computing a particular generating point's tile, it is not necessary to consider each pixel in the image, or even each pixel enclosed by the convex hull about the generating point's least surround. If the pixels on the boundary of the tile

can be determined, then for each row containing two or more distinct boundary points, from (2ii) we know that all the points between the two extreme boundary points also lie within the tile. Therefore, we need only compute the extremal boundary points for each tile, and the other points in the tile will follow trivially from the result.

The algorithm implemented during the course of this research to compute the tile for a given generating point, and the justification thereof, is as follows:

(3)

(i) We know that the the generating point lies within the tile. Therefore, a binary search is used to determine the left and right extremities of the tile for the row containing the generating point.

(ii) If we are at the top of the image, then go to step (iv), otherwise consider the row immediately above the last one considered.

(iii) Determine the left and right extremities of the tile in the current row, if there are any (this can be done quite easily as we know what the extremities were for the row immediately below). If there are extremities in the current row, then record that they and all the points between them are in the tile and go back to step (ii), otherwise proceed to step (iv).

}

(iv) Make the row containing the generating point } the current row. }

}

(v) If we are at the bottom of the image, then } stop, otherwise consider the row immediately } below the last one considered. }

}

(vi) Determine the left and right extremities of the } tile in the current row, if there are any (this } can be done quite easily as we know what the } extremities were for the row immediately above).} If there are extremities in the current row, } then record that they and all the points } between them are in the tile and go back to } step (v), otherwise stop. }

The algorithm above computes a given generating point's tile very efficiently.

7.5 Conclusion

It is not claimed that the optimisation technique developed during the course of this research is optimal, nor that it is suited to a large family of problems. It is not even central to this research, except in the pragmatic sense that without it, the segmentation technique developed here would have been more demanding computationally to implement.

During the course of this research, it became clear that none of the commonly used optimisation techniques was suited to the optimisation problem encountered, because of its properties given in (3) of section 7.2. Therefore a new technique was sought that could overcome these difficulties and also take advantage of the function characteristics given in (2) of section 7.2 and (1) and (2) of section 7.4. To this extent, the technique is useful and its application was highly successful.

As stated in section 7.3.1, the problem was seen to be analogous to that of the internal stresses in a twisted piece of metal, and so it was suggested that the annealing process used to relieve these stresses should be simulated in an attempt to optimise the function. The optimisation technique developed is described in section 7.3.2.

Section 7.4 then showed how, by drawing on results from chapter 5, the optimisation technique could be implemented efficiently.

Using this technique, it was possible to "optimise" the tessellation (16 generating points) of a 256x256 image in between

40 and 75 seconds (depending on the goodness of tessellation measure used) on a Cray-1S. The time required did vary slightly from image to image, but it was found that the clustering criterion used influenced the time necessary to a much greater degree.

The optimisation technique presented here has proved most interesting and could itself be the basis of further research in its own right.

CHAPTER 8

ACHIEVING THRESHOLD SURFACES

8.1 Introduction

Section 5.1 stated that the technique developed during the course of this research should:

(i) Divide a given image into regions

(ii) Compute a threshold value for a characteristic point of each region

(iii) Interpolate a threshold surface through the characteristic points' threshold values.

(1)

Chapter 5 suggested using the tiles obtained from the Dirichlet Tessellation of a particular (yet to be determined) set of generating points as the regions referred to in (1i) above and using the tiles' generating points as the characteristic points. Chapter 6 defined several measures of how good a particular tessellation was, and chapter 7 described a new optimisation technique to compute the optimum positions of the tiles' generating points.

Given a particular set of generating points and the corresponding Dirichlet Tessellation, we are now in a position to compute the threshold values for the generating points and then interpolate a threshold surface through these values.

Section 8.2 draws on results from chapter 3 and describes how the threshold values are to be computed for the generating points. Section 8.3 describes and evaluates various techniques for generalising these values to generate a threshold surface and section 8.4 shows how the techniques of sections 8.2 and 8.3 can be generalised to give multiple threshold surfaces, thus allowing thresholding into more than two levels.

Section 8.5 explores the use of relaxation techniques in the generation of threshold surfaces, and section 8.6 summarises the conclusions of this chapter.

8.2 Calculation of Thresholds for Generating Points

Having found the "optimal" tessellation in chapter 6, this section describes the technique used to compute threshold values for the tiles' generating points.

Clearly, if the distributions of the foreground and background pixels are known a priori, then the technique used to evaluate the generating points' threshold values should be modelled on this knowledge. Therefore, if the foreground and background pixel values are known to be, say, distributed Normally, then the threshold evaluation technique used should be the one given in section 3.5. However, these are special cases and usually no such a priori information is available.

This section deals with the computation of threshold values for the generating points when the only a priori information available is that given in (1) and (2) of section 1.3.

Chapter 3 describes many techniques for computing threshold values from given histograms. Many of these are ad hoc, but some are not.

In reviewing the types of techniques described in chapter 3, with a view to selecting a technique for our application, the following should be noted:

(i) It is not known what fraction of the pixels in each tile are foreground and what fraction are background. Therefore there is no justification for using median or average thresholding techniques to compute the threshold for a given

generating point. Fernando [28] used the median value as the threshold in a similar application, but this is not desirable in the light of (1iii) of section 1.3

(ii) The distributions of the foreground and background pixels are not known a priori, and so there is no justification for using one of the distribution assuming techniques described in section 3.5 (1)

(iii) The direct valley-seeking techniques are ad hoc and have been shown not to work, so they should not be used

(iv) During the course of this research, no evidence was found to suggest that moment preservation is necessarily desirable in our application.

What is required is a technique that assumes that the pixels belong to two classes and which tries to find the dichotomy that makes all the pixels in each of the classes as "similar" as possible and all the pixels in different classes as "dissimilar" as possible.

Otsu's technique (described in section 3.7) is such a technique. Furthermore, experimental results obtained during the course of this research show that the technique does perform well on real

data. This technique is both mathematically appealing (in that it assumes no a priori information other than that which is available and given) and functionally useful.

It also has the added advantage that it does not simply specify which is the "optimum" threshold value, but rather, it assigns a "goodness of threshold" value to each of the possible threshold values, and the "optimum" threshold value is simply that one which has the highest goodness of threshold value. This would enable us to calculate a "confidence of threshold" value (although this has not been done in this research) and also permits the use of relaxation techniques to refine preliminary threshold value estimates. Section 8.5 discusses this latter point in more detail.

Experimental results yielded similar threshold values using Otsu [79], Tsai [112] and Nakagawa [75] 's methods. However because of (1), the reasons given in the previous two paragraphs and the fact that Otsu's method is computationally less expensive, it was concluded that Otsu's method was the most appropriate for this application and so it is the one that has been used in this research.

Given a particular optimal tessellation, the technique for computing each generating point's threshold value is as follows:

(i) Compute the generating point's minimal surround.}

}

(ii) Compute the generating point's tile and its histogram using a technique very similar to (3) of section 7.4. } (2)

(iii) Compute the generating point's threshold value as being the value that maximises σ_B^2 as given in (12ii) of section 3.7.

8.3 Generalisation to Threshold Surfaces

8.3.1 Introduction

Section 8.2 described a technique for computing a threshold value for the generating point of each tile. This section investigates techniques for generalising these threshold values to threshold surfaces so that each pixel in the image has its own threshold value.

If the points were spaced evenly (eg. they formed a regular, rectangular grid), the problem of interpolation would be much simpler. However, they are not and the more general case needs to be investigated.

Several techniques for interpolating surfaces through irregularly spaced points have been developed, many of which make use of the Delaunay Triangulation or Dirichlet Tessellation of the points. Results from this research have shown these techniques to be adequate, and since the segmentation technique developed here already relies so heavily on these (i.e the Delaunay Triangulation and Dirichlet Tessellation), for the sake of completeness and consistency, they are used.

Section 8.3.2 describes two techniques for interpolating over the tiles, and sections 8.3.3 and 8.3.4 describe techniques for interpolating over the triangles.

Section 8.3.5 summarises the conclusions of this section.

8.3.2 Interpolation over Tiles

Since each generating point's threshold value is computed from that generating point's tile's histogram and each pixel in the tile contributes equally much to the histogram, an obvious (and easy) method for interpolating the threshold surface is to use the generating point's threshold value as the threshold value for all the pixels in the tile. This technique has the following advantages:

(i) It is easy to implement

(ii) There is some rationale behind the technique

(iii) It is often difficult to extrapolate the part of the threshold surface that lies outside the convex hull enclosing the generating points. This technique has no problems with these regions of the image. (1)

(iv) Each generating point's threshold value in the final, interpolated threshold surface, is the value that was computed for it originally

(v) All the values in the interpolated threshold surface are certain to lie within the range of permissible pixel values.

However, this technique has the serious fault that it is very discontinuous. The threshold values computed for adjacent tiles may be very different, thus causing a large "jump" in the threshold surface, which would almost certainly be reflected in the final thresholded image.

For this reason, it seems reasonable to insist that the threshold surface be at least C_0 continuous.

Fernando [28] encountered this problem and resolved it by smoothing the discontinuous threshold surface by taking the cosine transform of it, low-pass filtering, and then taking the inverse cosine transform of the result. This approach has the advantage that the threshold surface obtained is smooth, while still maintaining property (1iii) above. However, it does have the following disadvantages that the technique described earlier does not have:

(2)

(i) It is no longer so easy (or inexpensive) to implement

(ii) The justification for the technique is not so strong (apart from the argument that the threshold surface is now C_0 continuous)

(iii) The threshold surface no longer necessarily passes through the threshold values computed for the generating points

(iv) The interpolated threshold values do not necessarily lie within the range of permissible pixel values.

As both the techniques described above have serious objections, they were rejected and other techniques for interpolating the threshold surfaces were investigated.

8.3.3 Interpolation over Triangles

Several authors, such as McLain [66] and Monro [72] have proposed techniques for interpolating surfaces through irregularly spaced data, using the Delaunay Triangulation.

The simplest technique for doing this is to fit a flat surface through each of the triangles. Each triangle has values defined for each of its vertices and clearly, for any triangle, these values define a unique, flat surface. This method for interpolating the threshold surfaces has, amongst others, the following advantages:

(i) It is easy to implement

(ii) There is some rationale behind the technique

(iii) Each generating point's threshold value in the final, interpolated threshold surface, is the value that was computed for it originally

(iv) Pixels closer to a particular generating point have threshold values closer to the threshold value of that generating point than do pixels that are further away, in the same triangle } (1)

(v) The interpolated threshold surface is C_0 continuous }

(vi) All the values in the interpolated threshold surface are certain to lie within the range of permissible pixel values. }

The major disadvantage of the technique is that the threshold surface is only defined for the portion of the image enclosed by the convex hull around the generating points. However, this objection is overcome quite easily by specifying that there must be one generating point located at each corner of the image. This research did not reveal any serious objections to this, but it did discover the following advantages:

(i) The threshold surface is defined in a consistent manner for the whole image }

(ii) As four of the generating points have their positions fixed, the optimisation of the goodness of tessellation function, described in chapter 7, is easier and less demanding computationally, in that four of the functions }

variables have been removed } (2)

(iii) Objection (5) of section 5.2 no longer holds, and so the Delaunay Triangulation of the generating points is always defined

(iv) The situation described in (2v) of section 7.2 will not arise, and so neither will the associated problem.

During the course of this research, no reason was found for insisting that the threshold surface be C_1 continuous, besides the fact that C_1 surfaces are intuitivly more appealing mathematically than those that are only C_0 continuous. Furthermore, in experimental results obtained during the course of this research, the lack of smoothness (C_1 continuity) of the threshold surface did not appear to produce any erroneous edges in the segmented image. However, for completeness, techniques were explored that generated C_1 continuous threshold surfaces.

The technique used during the course of this research was a variant Young's implementation [127] of Monro's technique [72], which illustrates the issues involved well. The technique:

(i) Approximates the partial derivatives of the threshold surface at the generating points (this is discussed in section 8.3.4)

(ii) Then it fits a bicubic patch over each of the triangles. } (3)

This technique has the advantage that the threshold surface is C_1 continuous, but it has the following disadvantages:

(i) It is not as easy or inexpensive to implement as fitting flat surfaces through the triangles }

(ii) There is no real reason for imposing the added condition that the threshold surfaces should be C_1 continuous }

(iii) Pixels closer to a particular generating point, lying in the same triangle, do not necessarily have threshold values closer to that of that generating point than do pixels that are further away, in the same triangle } (4)

(iv) All the values in the interpolated threshold surface do not necessarily lie within the range of permissible pixel values. }

Objectons (4iii) and (4iv) can, however, be overcome.

In one dimension, if a cubic is to be interpolated between the points (a,f(a)) and (b,f(b)) and it is specified that $f'(a)=f'(b)=0$, then either f(a) or f(b) will be the functional

maximum and the other will be the minimum in the interval [a,b], and the function will be monotonic in that interval. Therefore, in the two dimensional case, if we specify that the partial derivatives of the threshold surface at the generating points is zero, then the maximum and minimum values along the the circumference of the triangle will occur at vertices of the triangle (viz. generating points). From the way that the threshold values are computed for the generating points, clearly, all values along the circumference of the triangle will be within the range of permissible pixel values. Therefore, if the values interpolated within each triangle are, for example, weighted sums of values along the edge of the triangle, then all the values interpolated over the triangles will be within the range of permissible pixel values and the highest and lowest values that the function will take over the triangle will occur at the triangle's vertices.

This overcomes objections (4iii) and (4iv) above.

Results from using both planar and bicubic interpolation are given in chapters 9 and 10.

8.3.4 Approximation of Derivatives

When using cubic interpolation over the triangles, the problem of how to approximate the partial derivatives of the threshold surface at the generating points arises.

Young [127] solved the the problem as follows. For each generating point:

(i) Compute and triangulate the generating point's surround.

(ii) Calculate the parameters for the planes fitted through each of the triangles in the surround.

(iii) The approximations of the partial derivatives of the threshold surface at the generating point are defined to be the weighted sums of those for the planes calculated in (ii), where the weighting is based on the relative areas of the triangles. (1)

In experiments during the course of this research, this technique was found to be good, except:

(i) When the surround is not defined uniquely, it is ambiguous which one is meant, or whether they all yield equivalent results.

(ii) When approximating a particular generating point's derivatives, long, thin triangles that make a small angle at the generating point are given the same weight as shorter, wider triangles that make a bigger angle at the generating point. (2)

In experiments during the course of this research, the above problems were resolved as follows:

(i) When approximating a particular generating point's partial derivatives, the generating point's maximal surround was always used.

(ii) Instead of weighting the values obtained in (1iii) on the relative areas of the triangles, they were weighted on the angles the triangles made at the generating point.

(3)

However, the objections given in (4) of section 8.3.3 are still valid.

8.3.5 Conclusion

While it is recognised that C_1 continuous surfaces may be more appropriate than C_0 continuous surfaces in some applications, no evidence was found to suggest that they may be more appropriate in this application. On the contrary, many reasons were found why fitting flat surfaces through the triangles is a better method of interpolation in this application than fitting bicubic surfaces through the triangles.

The most serious objection to using nonlinear interpolation was that the interpolated values may not necessarily be valid pixel values. Section 8.3.3 proposed a method for overcoming this

problem, but the surfaces generated, while being C_1 continuous, can be very "wavy", unnecessarily. For example, if the threshold values for the generating points all lie in the same plane, but the values are not all the same, then the interpolated threshold surface will be very wavy, whereas it should be flat.

The conclusion of this investigation is that the most justifiable of the forms of interpolation investigated, for our purposes, is planar interpolation through the triangles. Furthermore, experimental results obtained during the course of this research show the technique to work well and so it was not found necessary to investigate other interpolation techniques.

8.4 Calculation of Multiple Threshold Surfaces

In many applications, it is necessary to threshold a given image into more than two levels (viz. more than one threshold value needs to be computed for each pixel). This section investigates the possibility of generalising the technique developed during the course of this research to deal with such cases and compares the suitability of the technique with variants thereof.

Clearly, the arguments of chapters 5, 6 and 7 are independant of the number of threshold values that need to be computed for each generating point. Therefore, what remains to be done is to evaluate multiple threshold evalulation techniques and techniques for interpolating multiple threshold surfaces. These issues are addressed in this section.

Computing multiple threshold values for the generating points is much more complicated and inherently less stable than computing only one threshold value. For example, the generalisation of median thresholding would assume that (1/n) of the pixels in each tile belonged to each of the pixel classes in the thresholded image, which is even less reasonable an assumption than that in normal bilevel median thresholding. Similarly, with distribution assuming techniques, there is the same objection as in the bilevel case, in that the pixels may not have the distribution that they are assumed to have, but even if they do, often the technique will only recognise two large modes and the other modes will appear as shoulders of the larger modes. Clearly, this is undesirable.

However, all the reasons advanced for using Otsu's technique (in section 3.7) are equally valid for the case when multiple threshold values are to be computed. Experimental results obtained during the course of this research also support Otsu's method. Therefore, it is concluded that Otsu's technique is a good technique for computing either single or multiple threshold values for the generating points of the tiles in this application.

Interpolating the threshold surfaces also becomes much more complex when multiple threshold surfaces are required. Not only should each of the threshold surfaces:

(i) be C_0 continuous

(ii) pass through the threshold values computed for the generating points of the tiles (1)

(iii) contain only values that are within the range of permissible pixel values

as is the case with a single threshold surface, but also:

the threshold surfaces should not intersect. (2)

Planar interpolation satisfies all the conditions in (1) and (2). It was shown earlier that bicubic interpolation did not always satisfy condition (1iii) for a single threshold surface, so clearly, it will not necessarily satisfy this condition when multiple threshold surfaces are interpolated. Also, the threshold

surfaces are not definitely not going to intersect, unless restrictions similar to those described below (4) in section 8.3.3 are imposed, in which case the threshold surfaces will not intersect.

In section 8.3.5, it was concluded that planar (bilinear) interpolation was appropriate in this application when one threshold surface was to be interpolated. The other techniques investigated were all found to be inappropriate. In this section, it has been shown that planar interpolation is still appropriate when multiple threshold surfaces are to be computed, while all the objections to the other techniques in the single threshold case still apply, as well as the added objection to many of them, given in (2) above. Therefore, it is concluded that an appropriate interpolation technique for interpolating multiple threshold values is that in which the generating points are triangulated and planes are fitted through the triangles for each of the surfaces to be interpolated.

8.5 Use of Relaxation

Section 8.2 described techniques for computing threshold values for the tiles' generating points, and it was concluded that Otsu's technique was appropriate for our applications. However, it was pointed out that the technique did not produce only one absolutely correct answer, but rather, it assigned a goodness of threshold value to each permissible threshold value, and the threshold value selected finally would be the one with the largest goodness of threshold value. However, what appears to be the second best threshold value may, in fact, be a better choice when the threshold values computed for the generating point's surrounding generating points are considered.

Section 3.10 discusses relaxation techniques which address this problem.

Using Otsu's thresholding technique to compute a single threshold value for each generating point, it could be desirable, for example, to apply probabilistic relaxation to the results obtained. In this case, it is suggested that when updating a particular generating point's probability vector, only probability vectors for the generating points in that generating point's maximal surround should be considered.

Using the same notation as in section 3.10.2, clearly W(i,j;h,k) should be a function of:

(i) $|j-k|$ } (1)

(ii) the distance apart of generating points i and h. }

More specifically:

(i) The $P_{ij}^{(0)}$s are computed from the λ values given in (8) of section 3.7, taking j as the threshold value for generating point i

(ii) $$W(i,j;h,k) = \frac{1}{(|j-k|+1).D_{ih}} \quad (2)$$

where D_{ih} is the Euclidean distance between generating points i and h.

While Otsu's method for computing threshold values lends itself well to relaxation, during the course of this research, the threshold values computed were believed to be good enough and so it was not found necessary to apply these techniques to the threshold values calculated.

The problem becomes much more complex for multilevel thresholding, in that for each generating point, a probability vector with a probability value for each valid combination of threshold values is required. This means that for images in which pixels may take values between 0 and 255, say, each probability vector would need 32 896 values for computing 2 threshold values, and 2 829 056

values for computing 3 threshold values for each generating point. The computational demands would increase similarly.

We see, therefore, that relaxation techniques can be used in conjunction with Otsu's technique when computing one threshold value for each generating point (although the results of this research suggest that this is not necessary), but relaxation would be very expensive with currently available computer hardware when more than one threshold value is to be calculated for each generating point.

8.6 Conclusion

Section 8.2 compared and assessed the threshold evaluation techniques described in chapter 3 and concluded that given the information (or lack thereof) available a priori and the assumptions made in chapter 1, Otsu's technique was the most appropriate of the techniques investigated, in our applications. Experimental results obtained during the course of this research, some of which are presented in chapters 9 and 10, support this conclusion and show it to work well.

Section 8.3 evaluated techniques for interpolating a threshold surface through the threshold values computed for the generating points. It was suggested that the threshold surface should:

(i) be C_0 continuous

(ii) pass through the threshold values computed for the generating points of the tiles

(iii) contain only values that are within the range of permissible pixel values.

(1)

It was shown that fitting flat surfaces through the triangles satisfied these conditions and was easy to implement, while the other techniques tried raised further problems. For this reason, planar interpolation was advocated as being the most appropriate of the techniques investigated, for this application.

Section 8.4 investigated methods for generalising the techniques discussed in sections 8.2 and 8.3 so that multiple threshold values could be computed for each generating point, and then multiple threshold surfaces could be interpolated over the image space.

It was stated that when multiple threshold surfaces are interpolated, not only should each satisfy conditions (1) above, but also:

the threshold surfaces should not intersect. (2)

It was shown that Otsu's threshold evaluation technique had a natural generalisation for the compution of multiple threshold values, and experimental results obtained during the course of this research have shown it to work well. It was also suggested that planar interpolation was appropriate for interpolating multiple threshold surfaces.

Section 8.5 showed how relaxation techniques could be used to refine the threshold values obtained from Otsu's technique when one threshold value was computed for each generating point, although it was suggested that it was not necessary to apply these techniques in this application. However, it was also shown that using relaxation techniques to refine multiple threshold values imposes unreasonable computational demands at present.

CHAPTER 9

APPLICATION TO VARIOUS CLASSES OF IMAGES

9.1 Introduction

Chapters 5 to 8 developed and justified a new image segmentation technique based on variable thresholding.

This chapter shows the results of applying this technique to various classes of images and compares the results with those of other techniques. It also shows the results of applying variants of the technique, such as using different clustering criteria and interpolation techniques.

Figure 9.1 shows the formats that will be used to present the results in section 9.2.

The explanation of the images in figure 9.1 is as follows:

(a) A 256x256 pixel, 8-bit resolution image. Pixel value 0 is represented by black and 255 by white.

(b) Grey-level histogram of the image shown in (a). The horizontal axis corresponds to the range of permissible pixel values (with 0 being on the left and 255 on the right) and the vertical axis corresponds to the relative proportion of the pixels in the image with the corresponding pixel value.

(c) Image of (a) after it has been segmented into two regions, represented by black and white respectively.

(d) The boundaries of the regions of the image shown in (c).

(e) The boundaries shown in (d) superimposed on the original image, shown in (a).

(f) The threshold surface used to threshold the image shown in (a) to produce the segmented image shown in (c). Threshold surfaces can be viewed as images; if a pixel in the original image is lighter (has greater pixel value) than the corresponding pixel in the threshold surface image, then it is thresholded into white, otherwise it is thresholded into black.

(g) "Wire frame" drawing of the image shown in (a). As stated in section 2.1, images can be considered to be functions of two variables, x and y, where the values of the variables specify the point of interest and the functional value (pixel value) denotes the brightness of the image at that point (pixel). The wire frame drawings of images shown in this chapter consider images to be functions as described above, with the origin being at the bottom left of the image and with x increasing from left to right, y increasing from bottom to top, and with the (x,y) co-ordinates of the top right of the image being (255,255). In this chapter, when wire frames are drawn, the images are always viewed from three-space co-ordinate (128,-256,512). The drawings show perspective.

(h) Wire frame drawing of the image of the threshold surface shown in (f).

(i) The outlines of the tiles (shown in magenta) and triangles (shown in yellow) used in the segmentation of the image shown in (a).

(j) The outlines of the tiles and triangles shown in (i) superimposed on the image shown in (a).

(k) The image shown in (a), with one row of pixels highlighted.

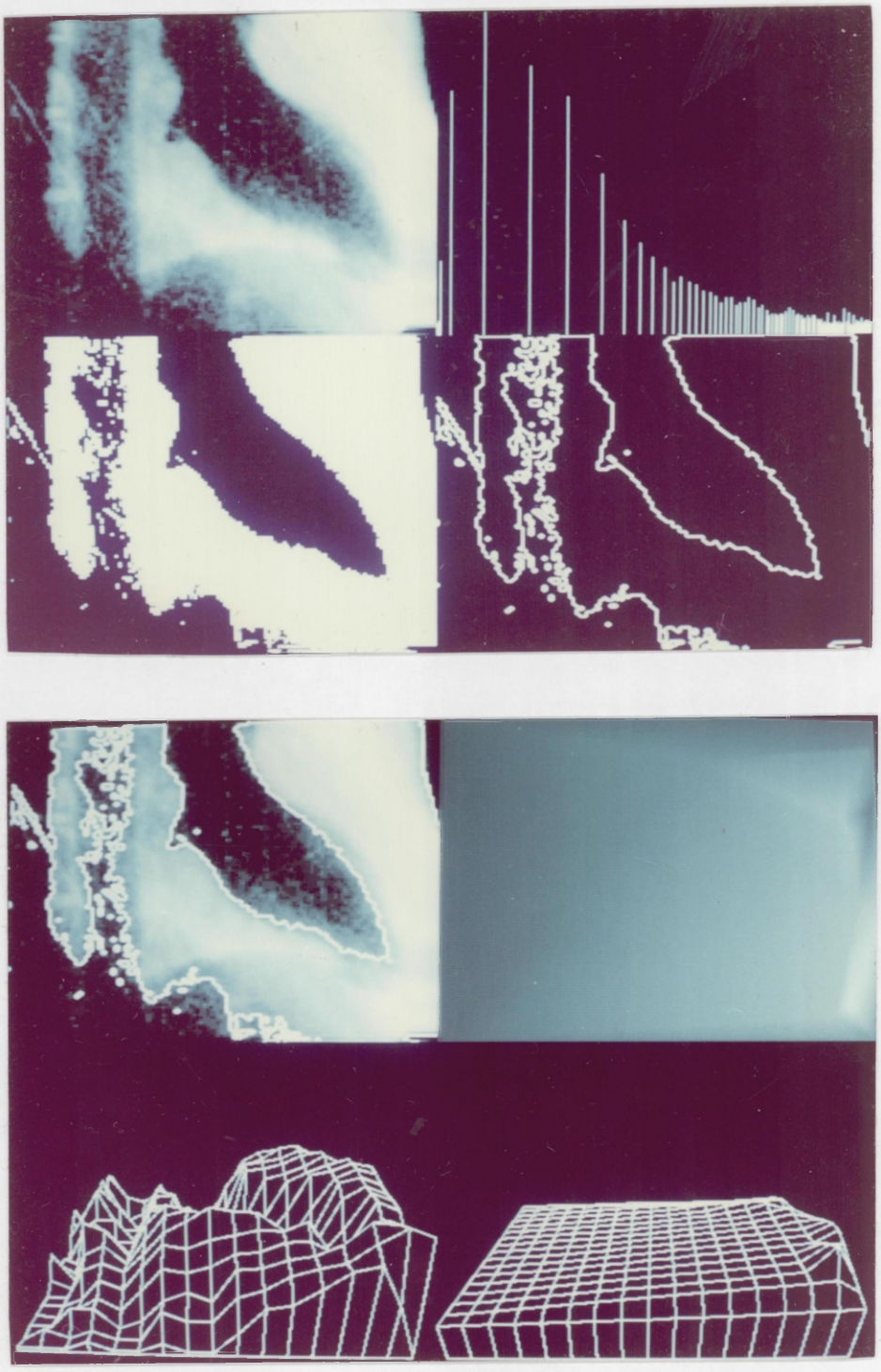
(l) Cross-sectional plot of the row of pixels highlighted in (k). The height of the plot in a particular column corresponds to the magnitude of the pixel value in the corresponding column of the row of pixels highlighted in (k).

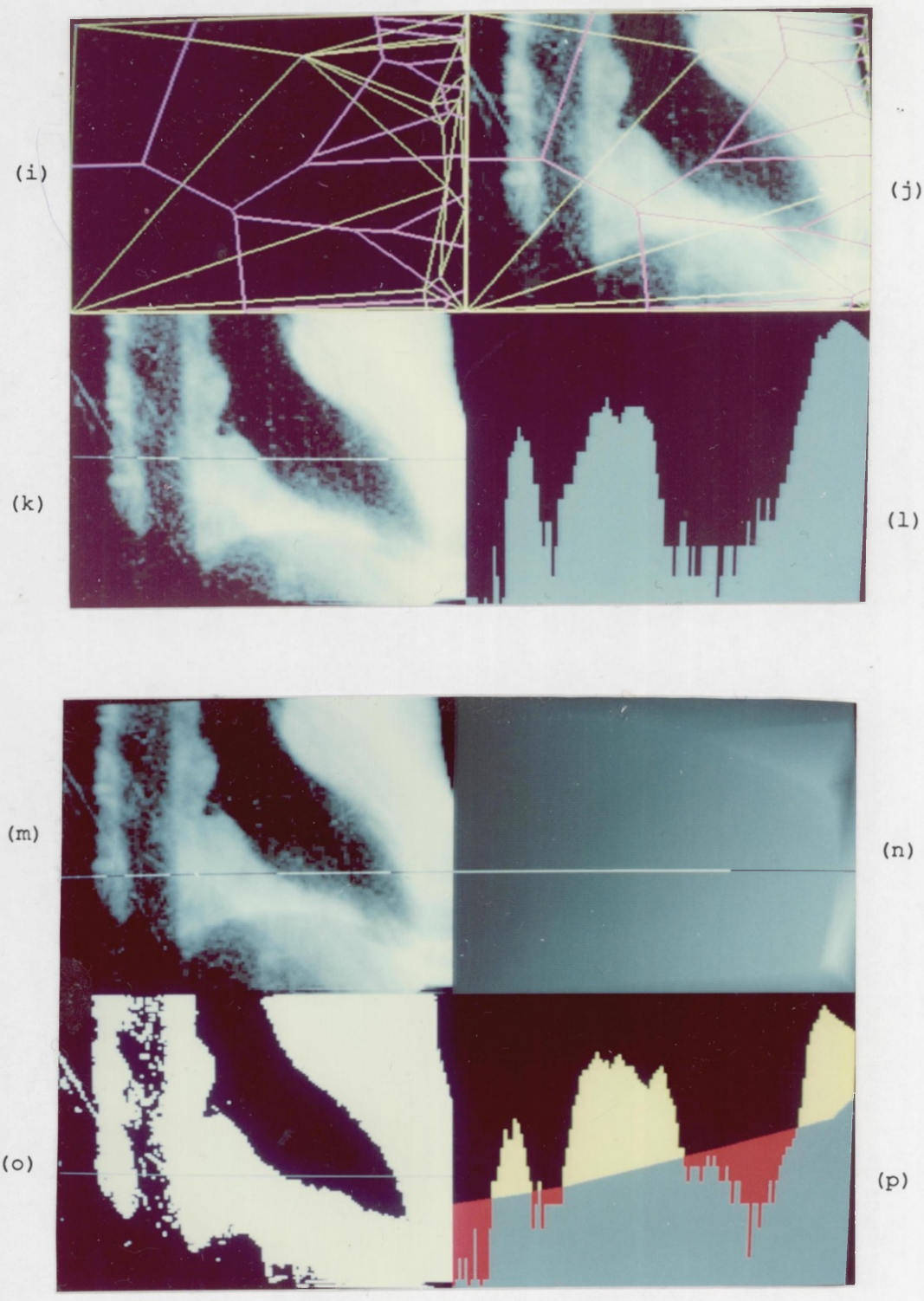
(m) The image shown in (a), with one row of pixels highlighted.

(n) The threshold surface shown in (f), with one row of pixels highlighted. The row that is highlighted is the same one that is highlighted in (m).

(o) The thresholded image shown in (c), with one row of pixels highlighted. The row that is highlighted is the same one that is highlighted in (m) and (n).

(p) Cross-sectional plots of the rows of pixels that are highlighted in (m) (shown in yellow) and (n) (shown in red). Regions that should be both red and yellow are shown as mid-grey.


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)



Types of images presented: Examples of the types of images presented in this chapter.

Figure 9.1

9.2 Results

9.2.1 Ventriculograms

Figure 9.2 shows the segmentation of a high-contrast image of a human left ventricle, using variants of the segmentation technique developed during the course of this research. Each photograph (containing four images) shows the image to be segmented with the outlines of the "optimal" tiles and triangles superimposed on it (top left), the threshold surface computed (top right), a wire frame drawing of the threshold surface (bottom left), and the segmented image (bottom right).

Its purpose is to compare 4 clustering criteria (bumpiness B_0, bumpiness $B_{0.5}$, variance of the histogram, and entropy) and 3 threshold interpolation techniques (bi-linear, bi-cubic with the partial derivatives approximated, and bi-cubic without the partial derivatives approximated).

In figure 9.2, (a)-(d) used bi-cubic interpolation with the derivatives computed (as described in sections 8.3.3 and 8.3.4), (e)-(h) used bi-cubic interpolation with the partial derivatives of the threshold surface at the vertices of the triangles set to 0, and (i)-(l) used bi-linear (planar) interpolation, to interpolate the threshold surfaces. In all cases, Otsu's technique (as described in section 3.7) was used to compute threshold values for the vertices of the triangles (generating points of the tiles).

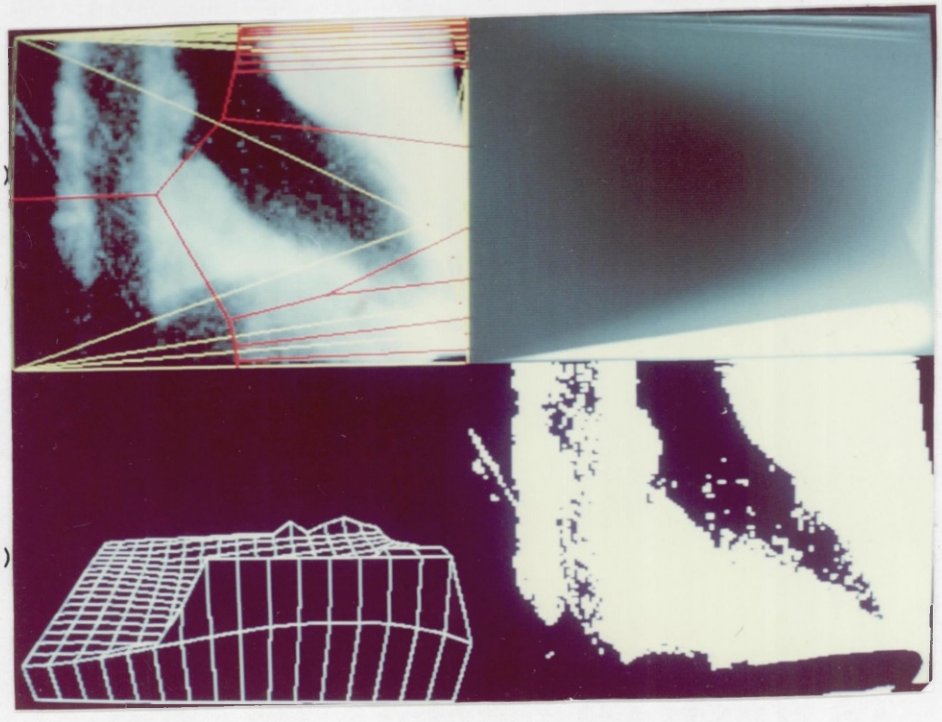
In (a), (e) and (i), the clustering criterion was bumpiness

measure B_0, in (b), (f) and (j), the clustering criterion was bumpiness measure $B_{0.5}$, in (c), (g) and (k) the clustering criterion was the entropy of the histogram (as described in section 2.2.3) and in (d), (h) and (l) the clustering criterion was the variance of the histogram (as described in section 2.2.4).

In figure 9.2, the results obtained using bumpiness measure $B_{0.5}$ are consistently better than those obtained using the other clustering criteria. Similarly, the results obtained using bi-linear interpolation are consistently better than those obtained using bi-cubic interpolation. The criterion for judging how good a particular segmentation is is how similar the result is to the "correct" segmentation, as defined by a panel of experienced radiologists, given in [28].

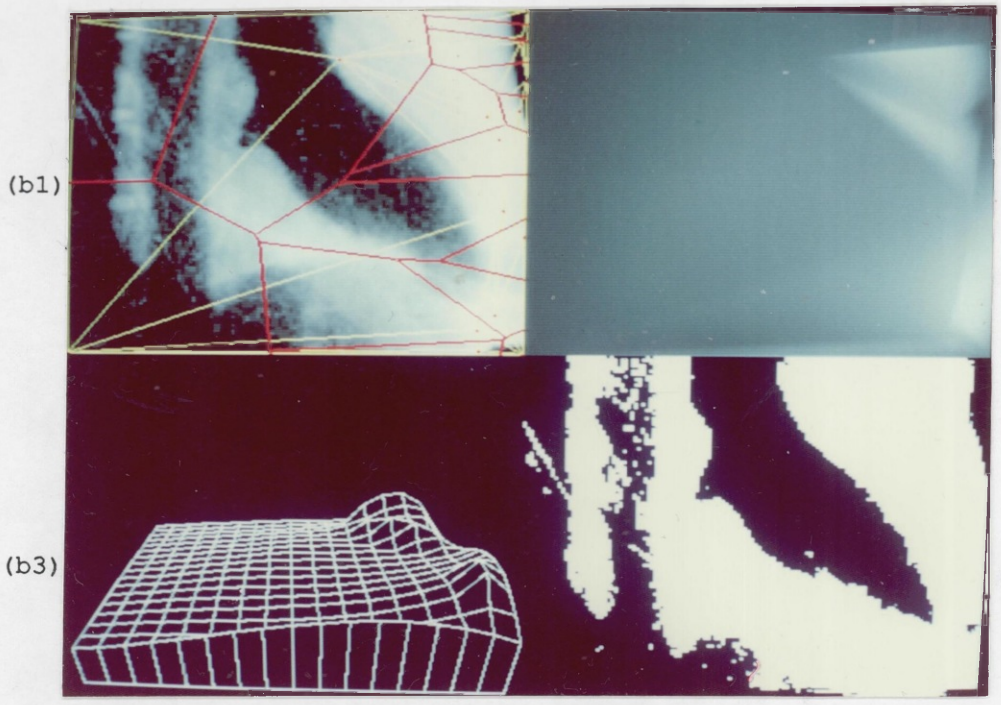
This is in keeping with other results obtained during the course of this research.

(a1) (a2)

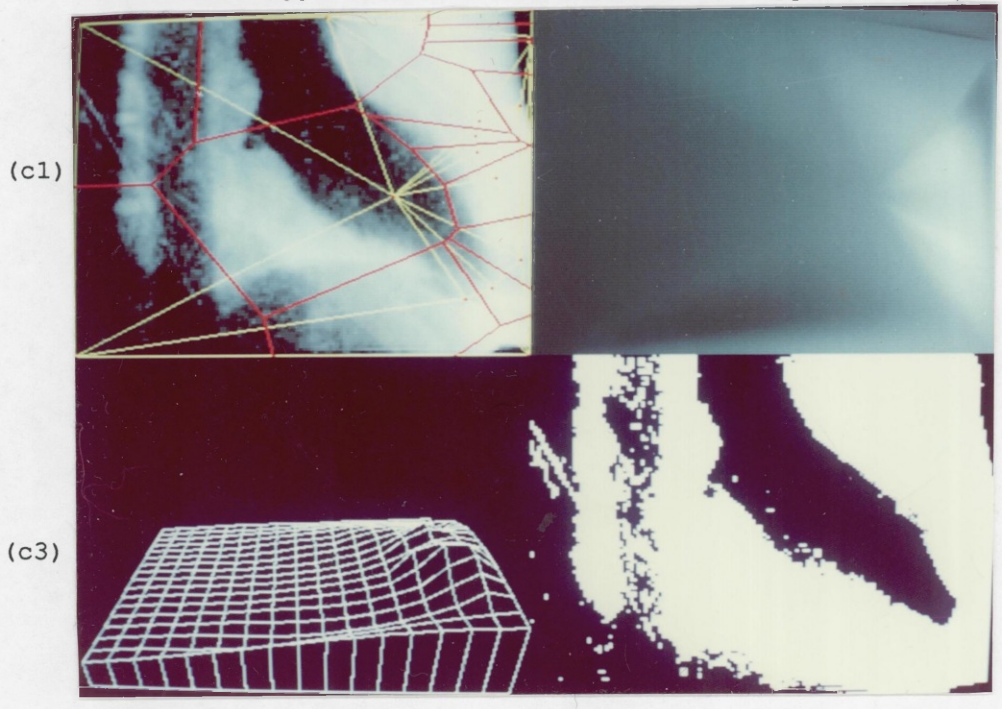


(a3) (a4)

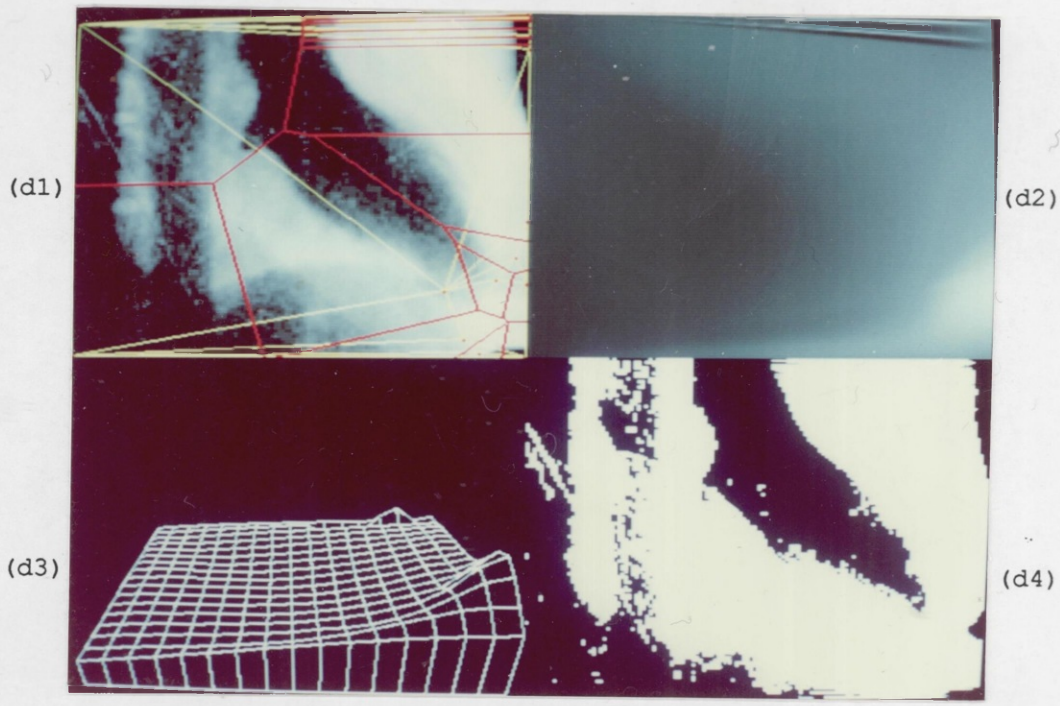
(Bumpiness B_0, bi-cubic with derivatives approximated)



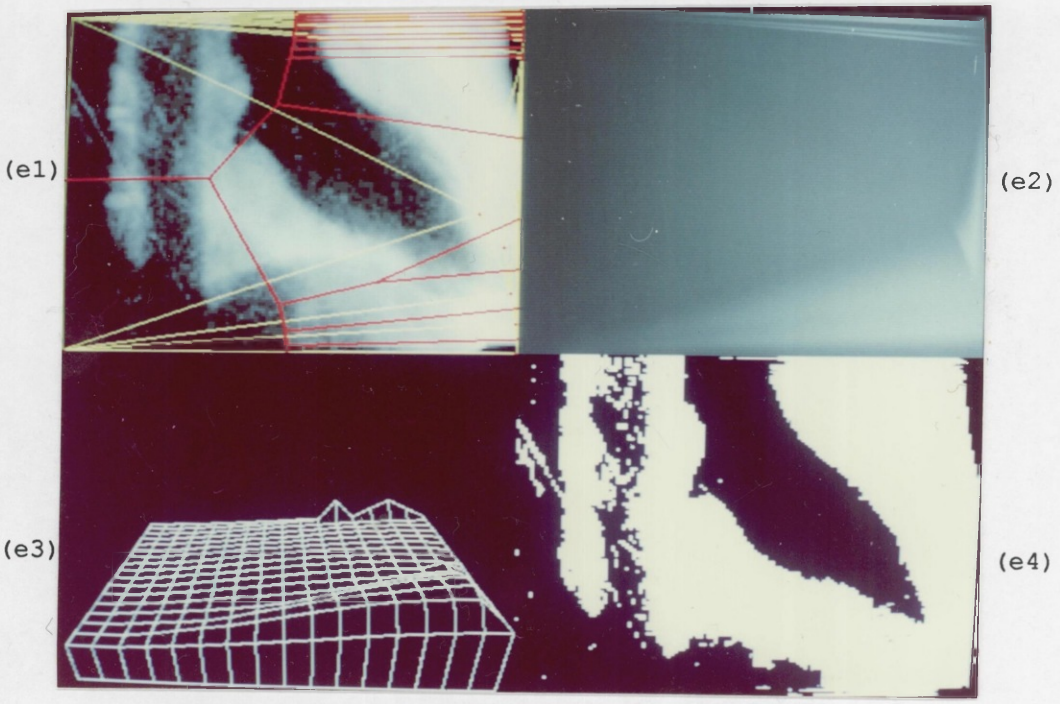
(Bumpiness $B_{0.5}$, bi-cubic with derivatives approximated)



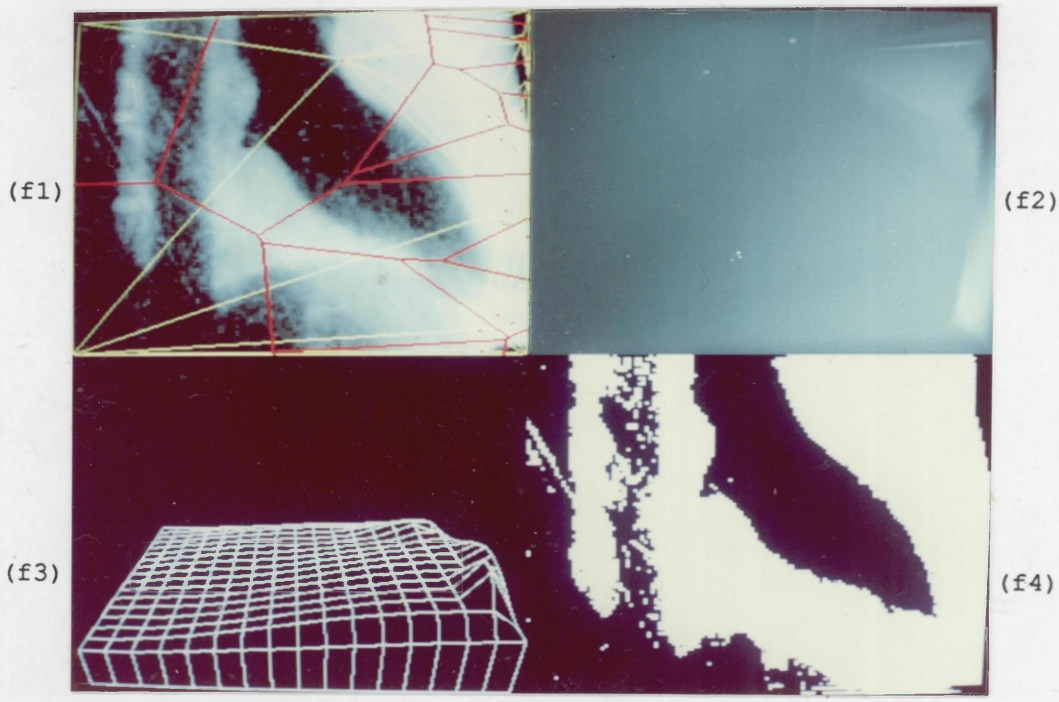
(Entropy, bi-cubic with derivatives approximated)



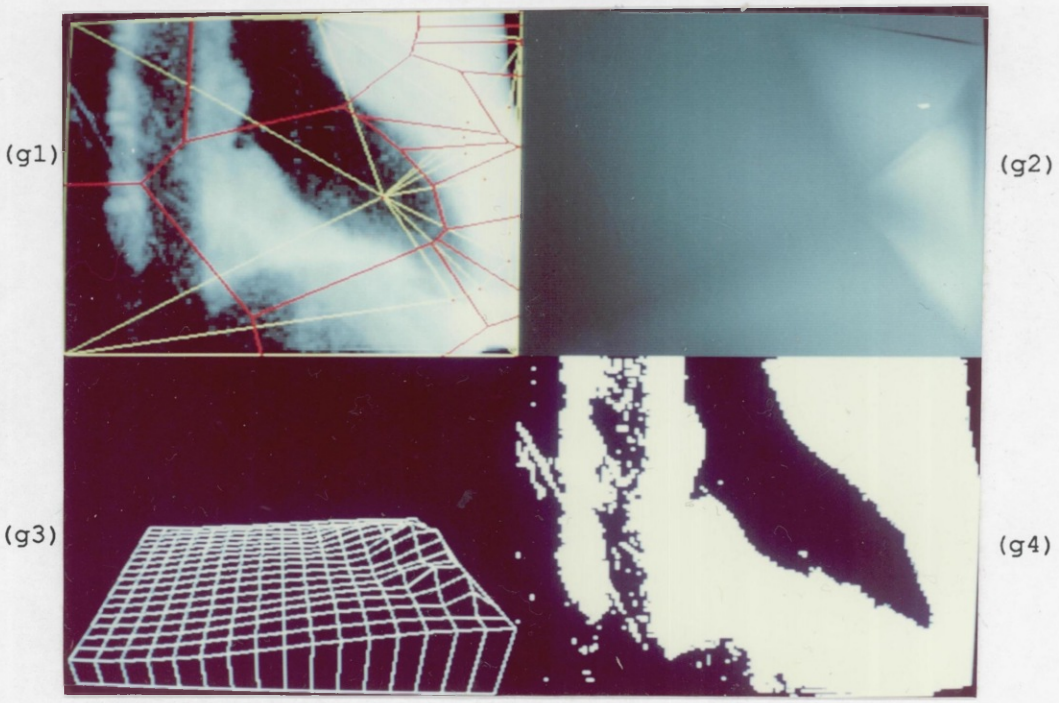
(Variance, bi-cubic with derivatives approximated)



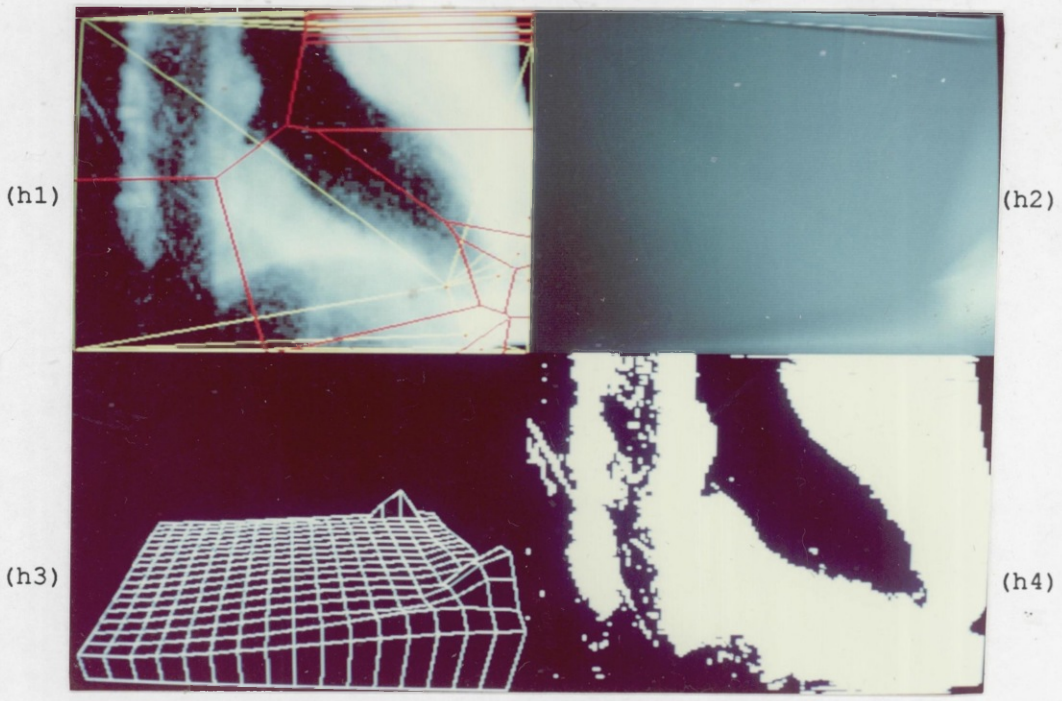
(Bumpiness B_0, bi-cubic with derivatives not approximated)



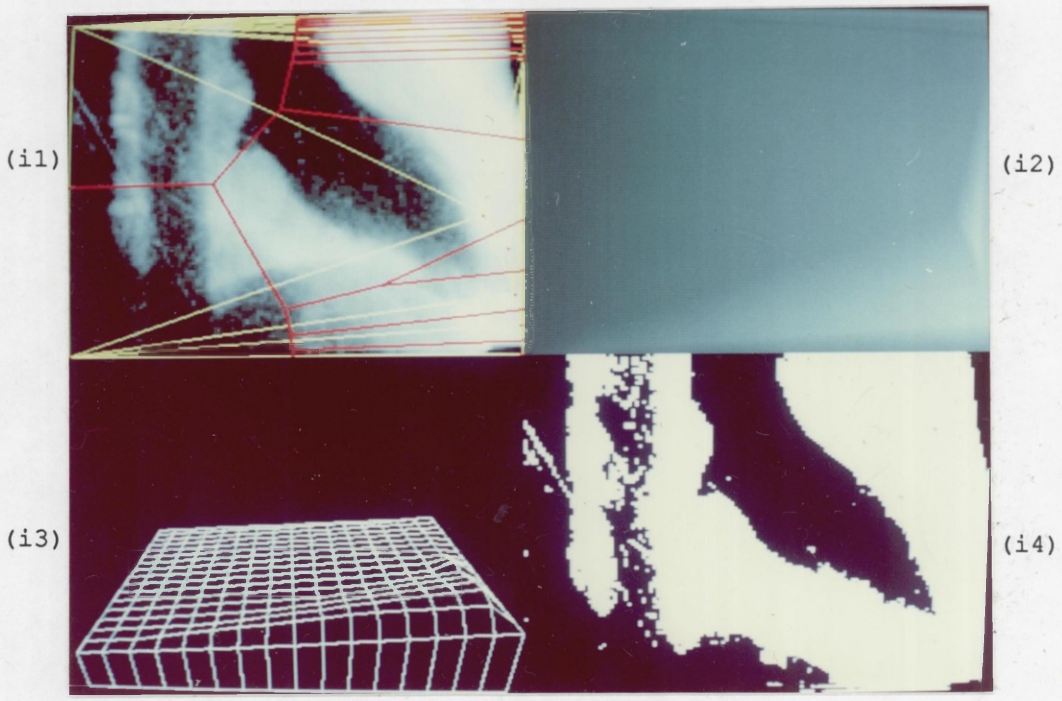
(Bumpiness $B_{0.5}$, bi-cubic with derivatives not approximated)



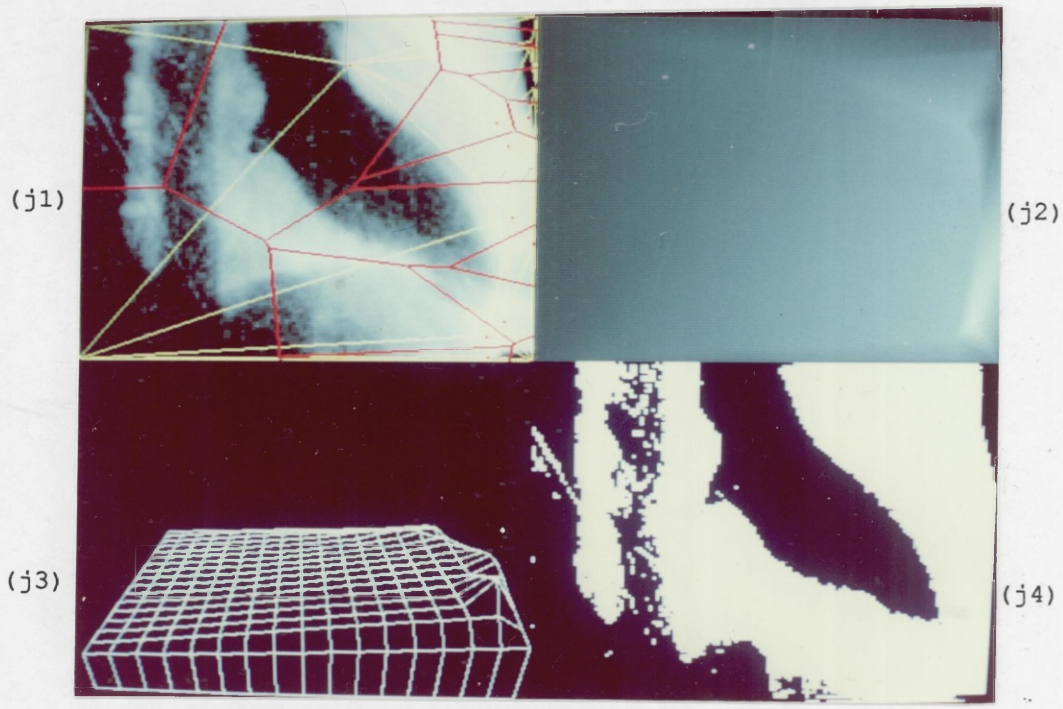
(Entropy, bi-cubic with derivatives not approximated)



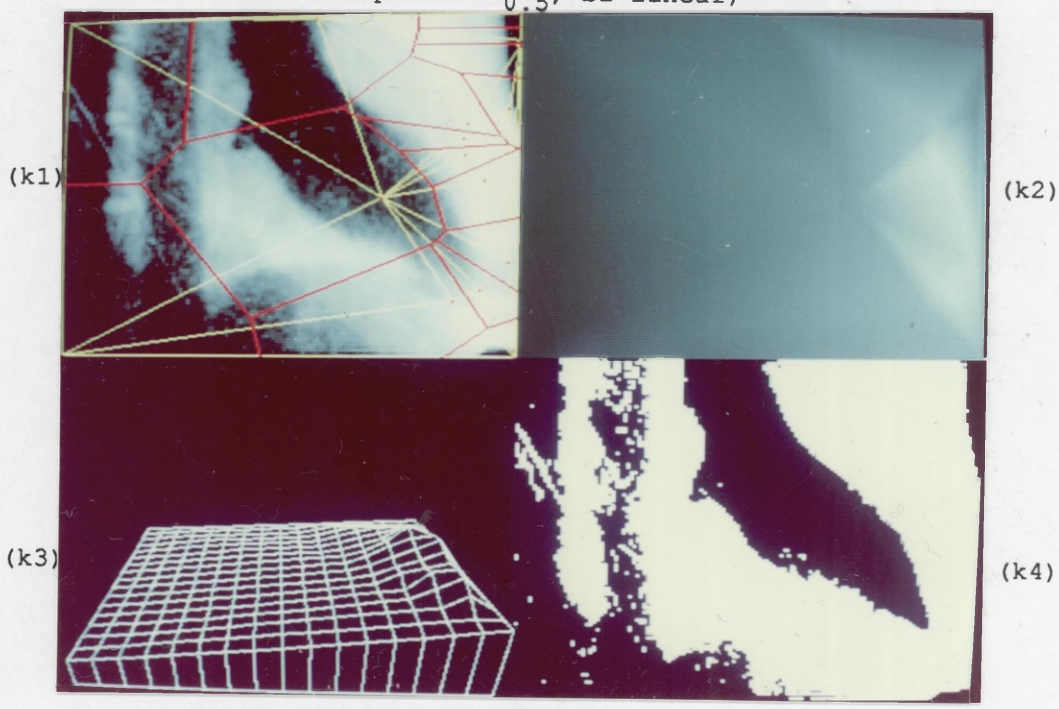
(Variance, bi-cubic with derivatives not approximated)



(Bumpiness B_0, bi-linear)

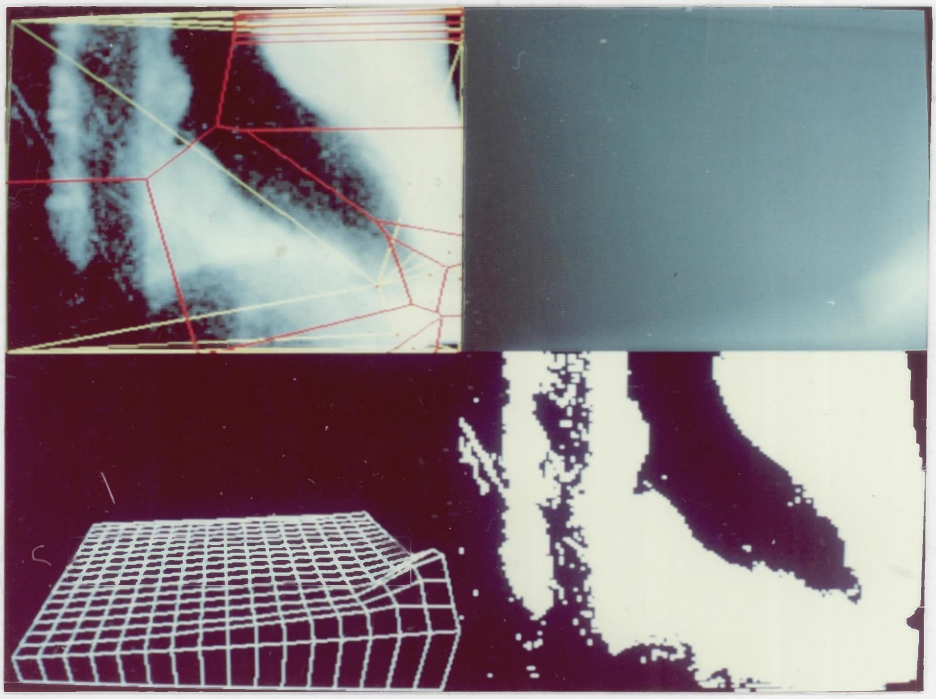


(Bumpiness $B_{0.5}$, bi-linear)



(Entropy, bi-linear)

(11)



(12)

(13)

(14)

(Variance, bi-linear)

Comparison of clustering criteria and interpolation techniques: The above images show the effects of using different clustering criteria and interpolation techniques, but the same method of threshold evaluation, in segmenting (thresholding) a given image.

Figure 9.2

Figure 9.3 compares the results obtained using the segmentation technique developed during the course of this research with those obtained from using a moving window median threshold, on an image with relatively high contrast.

The explanation of the images shown in figure 9.3 is as follows:

(a) - The image to be segmented.

(b) - The grey-level histogram of the pixel values in the image shown in (a).

(c) - A wire frame drawing of the threshold surface computed using the segmentation technique developed here with the clustering criterion being bumpiness $B_{0.5}$ and the threshold surface being interpolated using bi-linear interpolation.

(d) - The image shown in (a) segmented using the segmentation technique developed during the course of this research.

(e) - A wire frame drawing of the threshold surface computed using moving window median thresholding with window radius 32 pixels, making the size of the moving window the same as the average size of the tiles in the Dirichlet Tessellation used to generate the threshold surface used to generate (d).

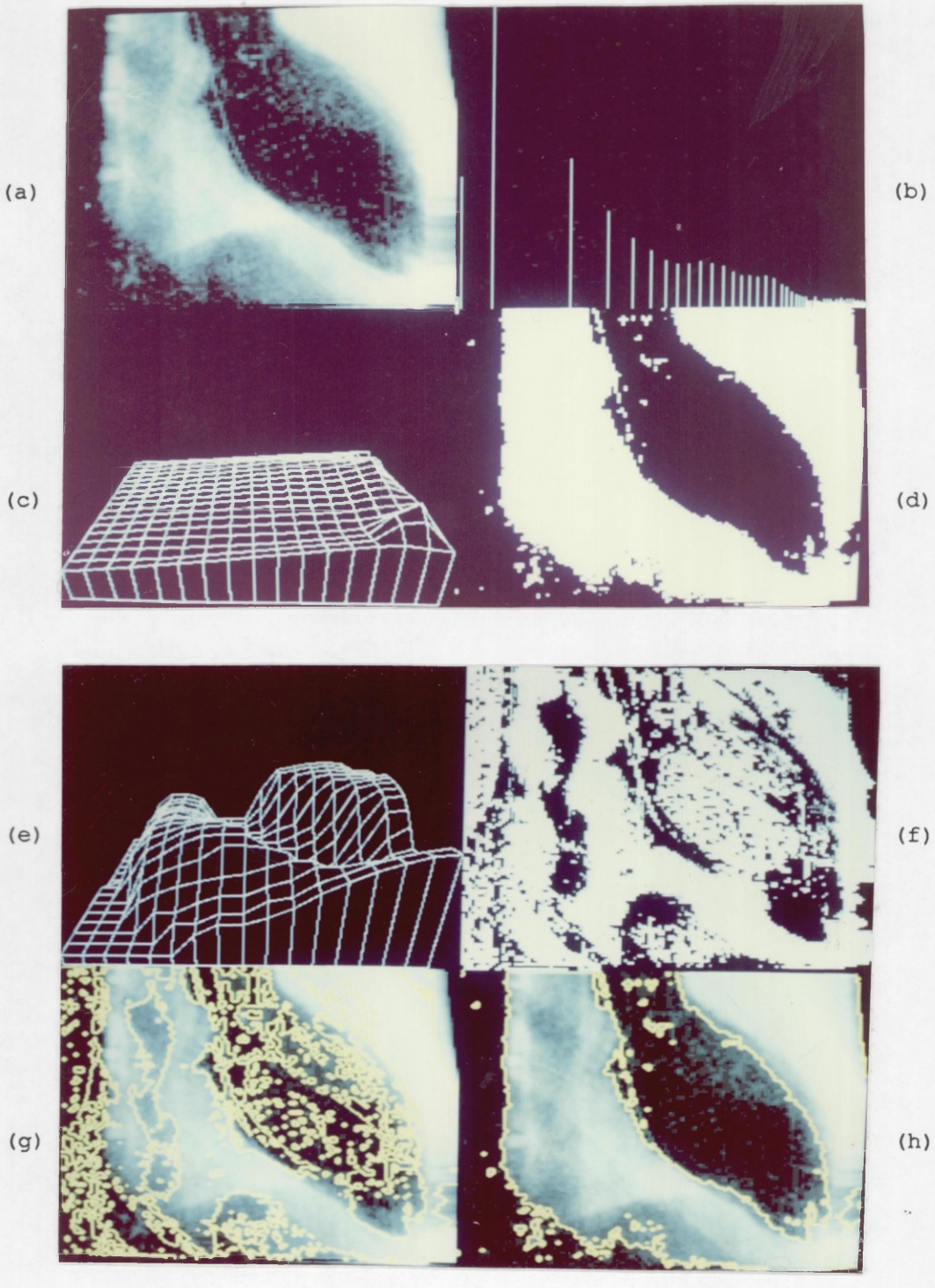
(f) - The image shown in (a) segmented using the threshold surface shown in (e).

(g) - The edges of the segmented image shown in (f) superimposed on the image shown in (a).

(h) - The edges of the segmented image shown in (d) superimposed on the image shown in (a).

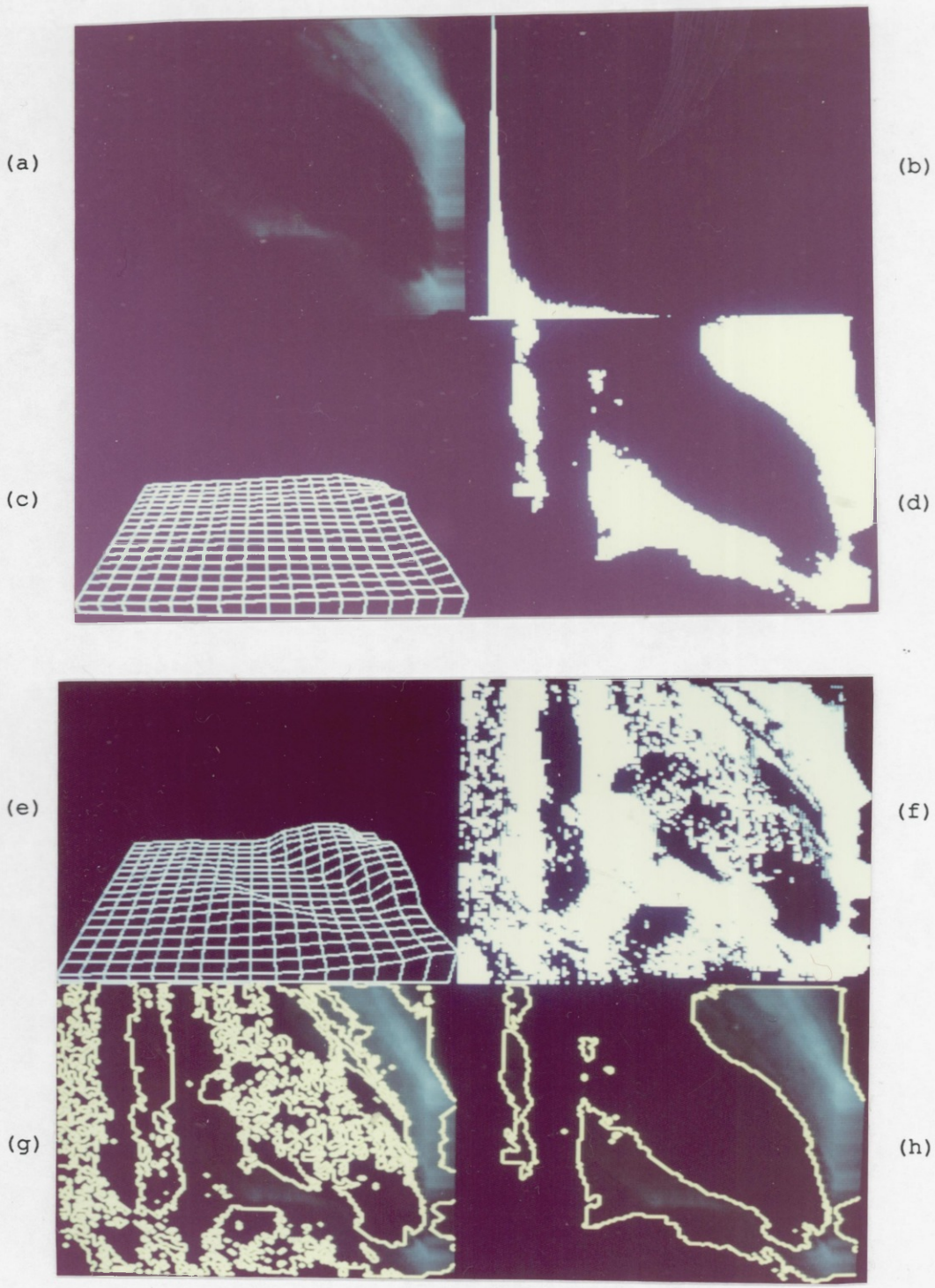
Figures 9.4 and 9.5 are similar to figure 9.3, except that they show the results of applying the techniques to low contrast and unevenly illuminated images, respectively.

In figures 9.3 to 9.5, the segmentation technique developed during the course of this research is shown to work better than moving window median thresholding, consistently. This is in keeping with other results obtained during the course of this research. Furthermore, in figure 9.5(c), it is shown how the threshold surface computed using the technique developed during the course of this research will compensate for the uneven illumination of the image to be segmented.



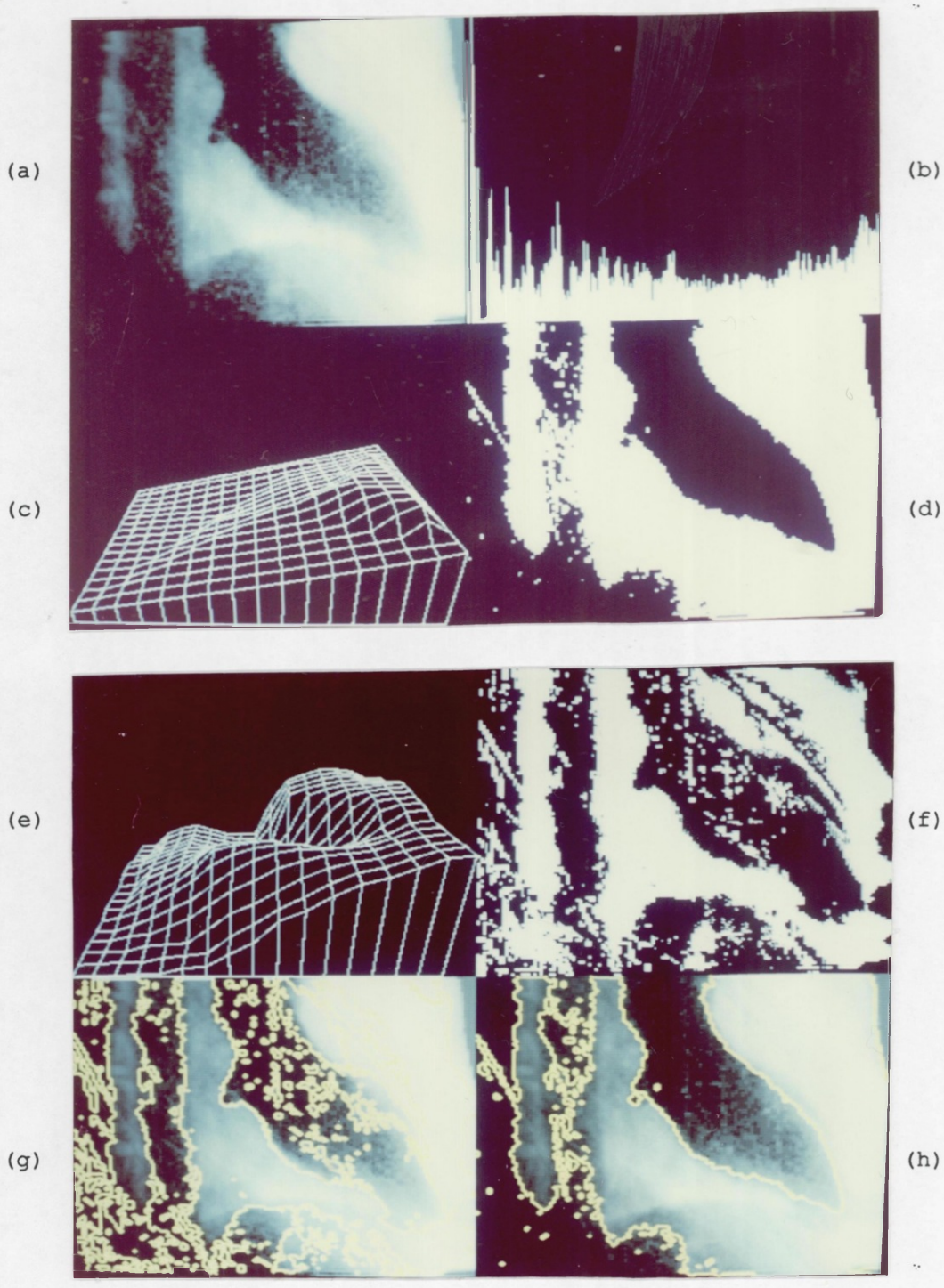
Results from segmenting images with high contrast: Comparison of results from using the technique developed during the course of this research with those of using moving window median thresholding, on an image with relatively high contrast.

Figure 9.3



Results from segmenting images with low contrast: Comparison of results from using the technique developed during the course of this research with those of using moving window median thresholding, on an image with relatively low contrast.

Figure 9.4



Results from segmenting images which are illuminated unevenly: Comparison of results from using the technique developed during the course of this research with those of using moving window median thresholding, on an image with is illuminated unevenly.

Figure 9.5

9.2.2 Images of Cells

Figures 9.6 and 9.7 show the results of applying the technique developed during the course of this research, to two images of some cells. In these applications, the technique was used to locate the cell nuclei in the images.

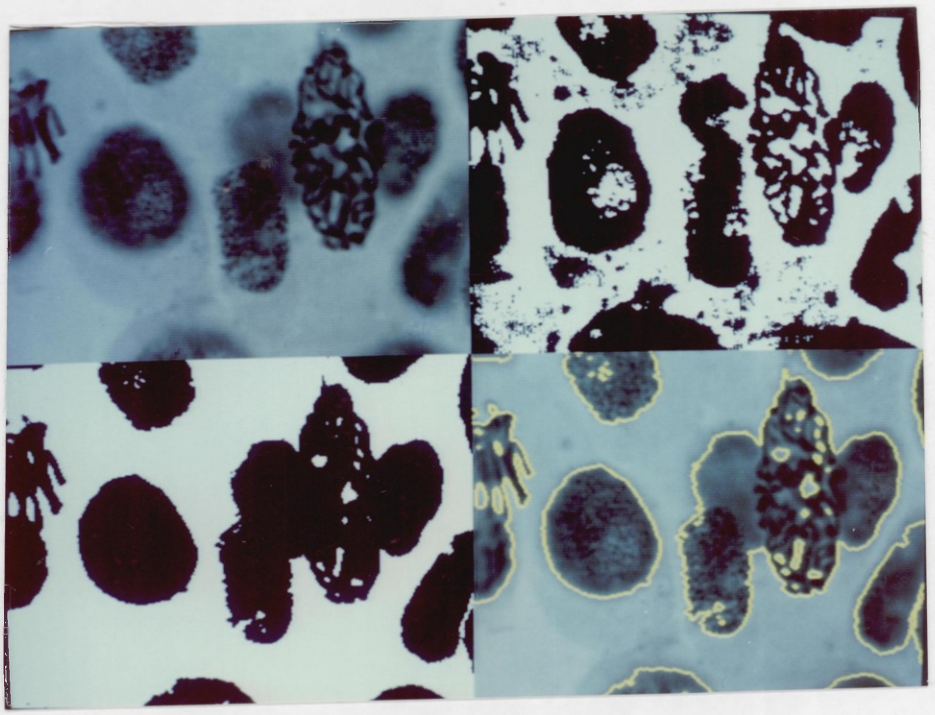
In figure 9.6, (a) shows an image of some cells, (b) shows the result of thresholding the image using a 65x65 pixel moving window median threshold, (c) shows the result of applying the segmentation techniques developed during the course of this research (using planar interpolation of the threshold surface and clustering criterion bumpiness $B_{0.5}$), and (d) shows the boundaries of the regions in the segmented image (in yellow) superimposed on the original image, shown in (a).

In figure 9.7, (a) shows an artificially unevenly illuminated image of the cells shown in (a) of figure 9.6, (b) shows the histogram of the image shown in (a), (c) shows the result of segmenting the image using the techniques developed during the course of this research (the clustering criterion and interpolation technique used were the same as for figure 9.6), and (d) shows the boundaries of the regions in the segmented image (in yellow) superimposed on the original image, shown in (a).

The result of using the technique developed during the course of this research, shown in figure 9.6, is better than that of using moving window median thresholding. The results shown in (c) and (d) of figure 9.7 are not as good as those shown in (c) and (d) of figure 9.6. This is to be expected as the image shown in figure

9.7(a) is considerably more difficult to segment than the one shown in figure 9.6(a). However, the segmentation shown in figure 9.7(c) is still better at isolating the nuclei than that shown in figure 9.6(b), showing the technique developed during the course of this research to perform considerably better than moving window median thresholding.

(a)



(b)

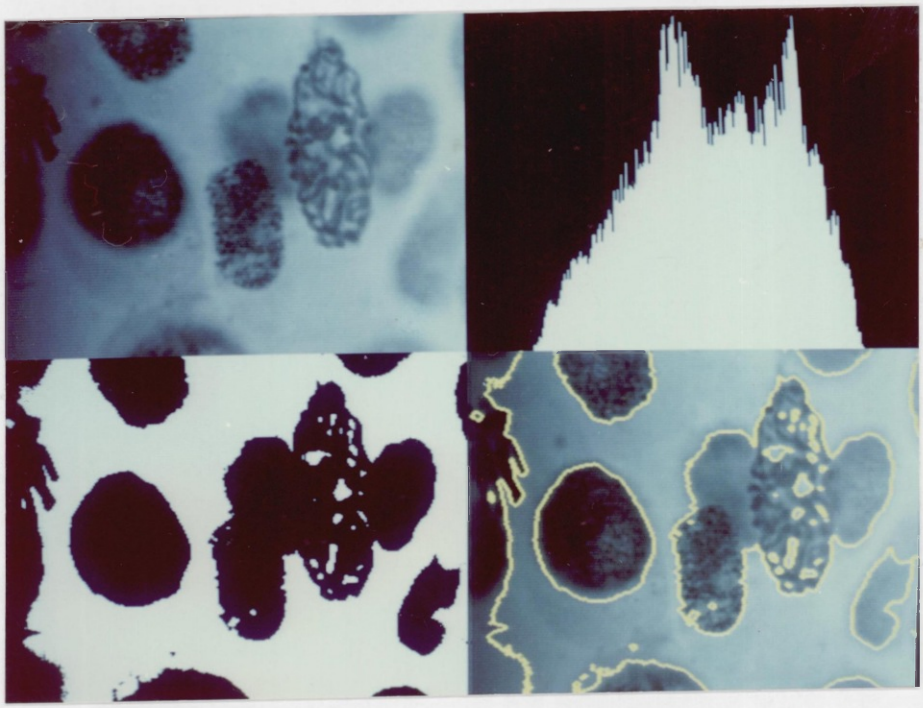
(c)

(d)

Segmentation of an image of some cells: (b) shows the result of thresholding (a) using a moving window median threshold, (c) shows the results of applying the technique developed during the course of this research to (a), and (d) shows the edges of the segmented image (c) superimposed on the original image, (a).

Figure 9.6

(a)



(b)

(c)

(d)

Segmentation of unevenly illuminated image of some cells: (a) is an unevenly illuminated image of some cells, (b) shows the grey-level histogram of the image shown in (a), (c) shows the result of segmenting the image shown in (a) using the segmentation technique developed during the course of this research, and (d) shows the edges of the segmented image shown in (c), superimposed on the original image, shown in (a).

Figure 9.7

9.2.3 Satellite Images

Figures 9.8 to 9.10 show the results of segmenting some satellite images into 2 or 3 regions, using the segmentation technique developed during the course of this research. In all cases, the clustering criterion used was bumpiness $B_{0.5}$ and the threshold surfaces were interpolated bi-linearly.

In this study, when segmenting a satellite image into 2 regions, it is intended that one region will contain all the land and clouds and the other will contain all the sea, while when segmenting an image into 3 regions, it is intended that the regions should correspond to land, sea and areas covered by cloud, respectively, in the original image.

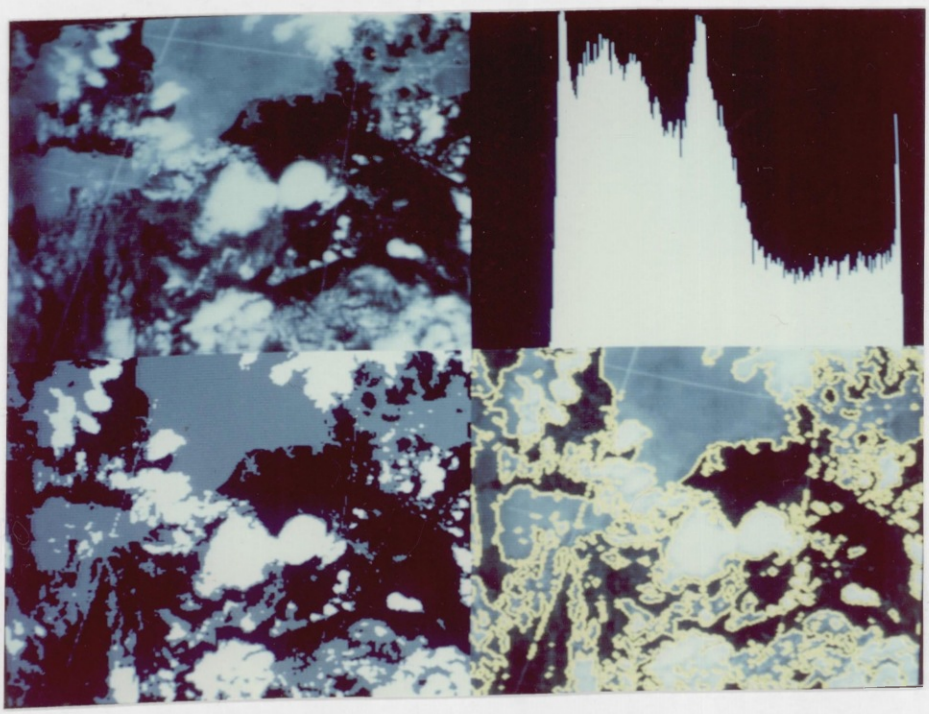
In figure 9.8, (a) shows a satellite image, (b) shows the grey-level histogram for the image shown in (a), (c) shows the segmentation into 3 regions of the image shown in (a), and (d) shows the boundaries of the regions of the image shown in (c), superimposed on the original image, shown in (a). The technique is shown to work well on the image shown in (a).

In figure 9.9, (a) contains an artificially unevenly illuminated satellite image, (b) shows the grey-level histogram of the image shown in (a), (c) shows the segmentation of the image shown in (a), into 2 regions, and (d) shows the edges of the segmented image shown in (c), superimposed on the original image, shown in (a). Once again, the technique is shown to work well.

In figure 9.10, (a) contains a satellite image, (b) shows the

boundaries (in yellow) of the regions detected when the image shown in (a) is segmented into 2 regions, superimposed on the original image, (c) shows the three regions detected (shown as black, mid-grey and white, respectively) when the image is segmented into three regions, and (d) shows the edges of the regions shown in (c), superimposed on the original image, shown in (a). In (d), the boundary around the region shown as white in (c), is drawn in red, and the border between the mid-grey and black regions of (c) are drawn in yellow. In this figure, the the technique is shown to segment the image shown in (a) into 2 regions successfully. It is not clear that the segmentation into 3 regions is as successful, but then it is difficult even for a person to distinguish the areas covered by cloud from the land, in (a).

(a)



(b)

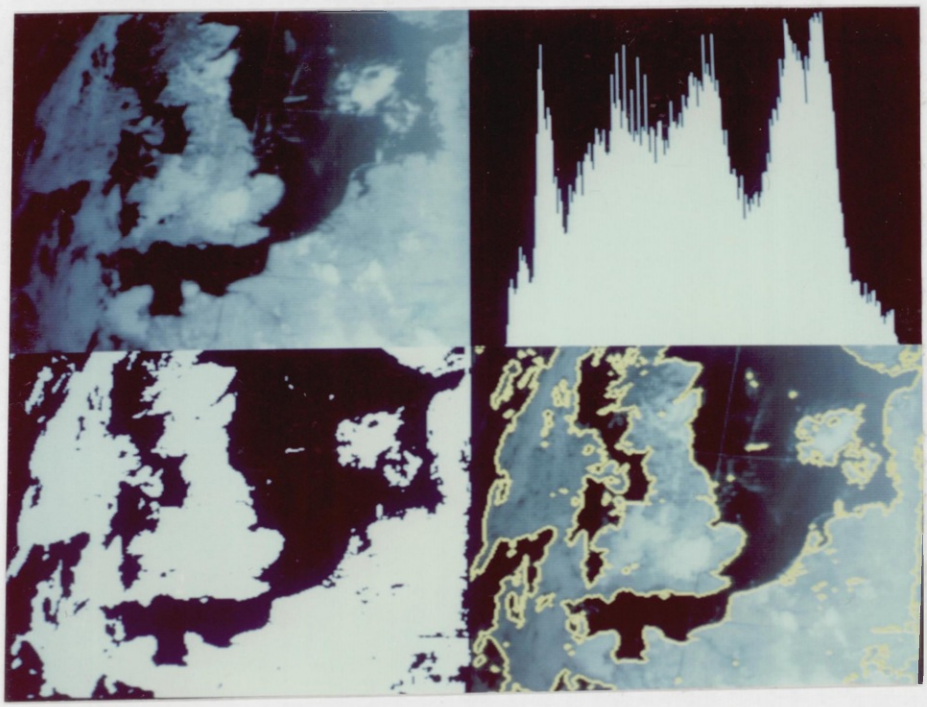
(c)

(d)

Satellite image segmented into 3 regions: (a) shows a satellite image, (b) shows the grey-level histogram for the image shown in (a), (c) shows the segmentation of the image into 3 regions, and (d) shows the boundaries of the regions detected superimposed on the original image.

Figure 9.8

(a)



(b)

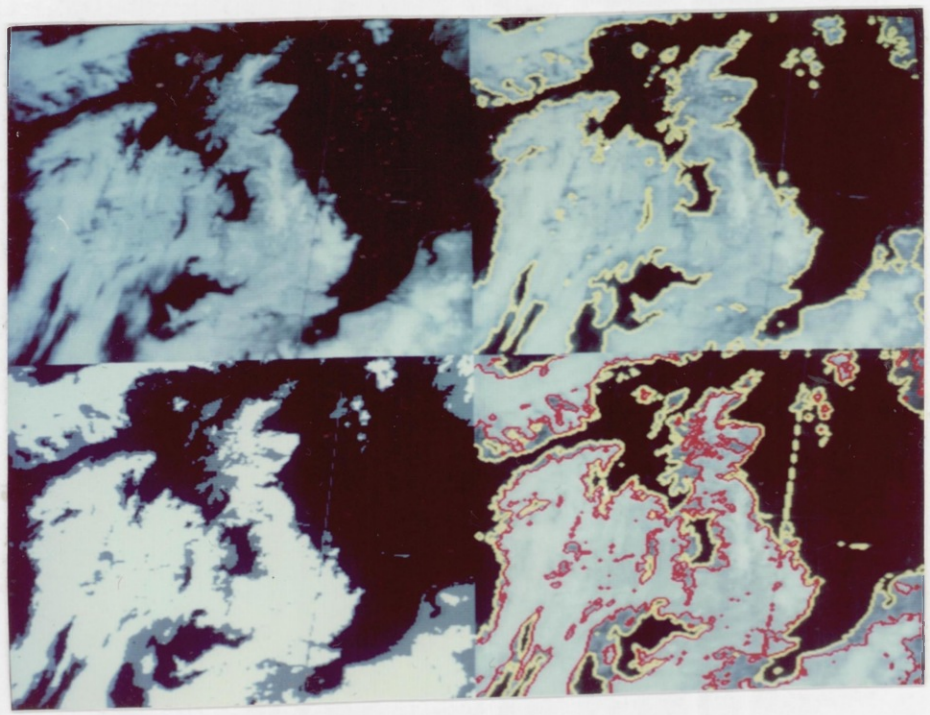
(c)

(d)

Unevenly illuminated satellite image segmented into 2 regions: (a) contains an unevenly illuminated satellite image, (b) shows the grey-level histogram of the image shown in (a), (c) shows the segmentation of the image into 2 regions, and (d) shows the edges of the regions detected superimposed on the original image, shown in (a).

Figure 9.9

(a)



(b)

(c)

(d)

Segmentation of satellite image: (a) contains a satellite image, (b) shows the boundaries of the regions detected when the image is segmented into 2 regions, superimposed on the original image, (c) shows the regions detected when the image is segmented into 3 regions, and (d) shows the boundaries of the regions shown in (c), superimposed on the original image, shown in (a).

Figure 9.10

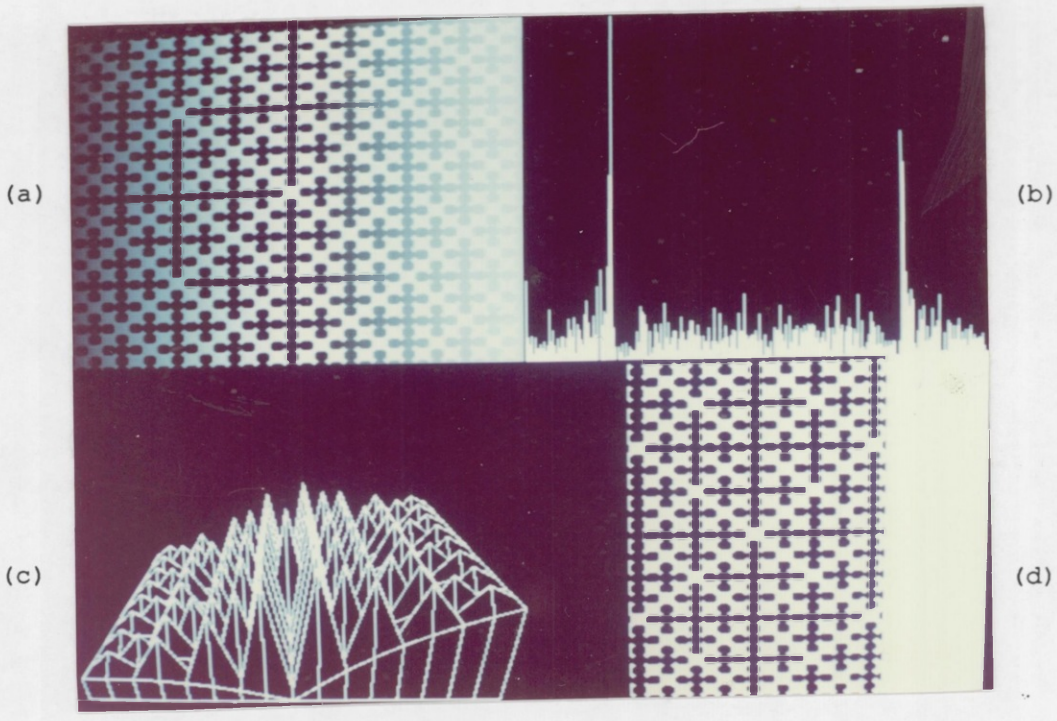
9.2.4 Artificially Generated Images

As a final test, the segmentation technique developed during the course of this research was tested on an artificially generated image. The objective of the excercise was to see how the technique performed in a controlled environment and when the "correct" segmentation is known a priori.

This section shows the results of applying the technique developed during the course of this research to an artificially generated image.

The image used is a filled Sierpinski curve that has been relit unevenly. This means that, while the foreground (filled region) is lighter than the surrounding background locally, the background on the right of the image is lighter than the foreground on the left of the image. This is shown in figure 9.11.

In figure 9.11, (a) shows the image that is segmented in this section, (b) shows the image's grey-level histogram, (c) shows a wire frame drawing of the image, and (d) shows the result of thresholding the image at pixel value 128, globally. (b) and (c) show that achieving the correct segmentation could be difficult.



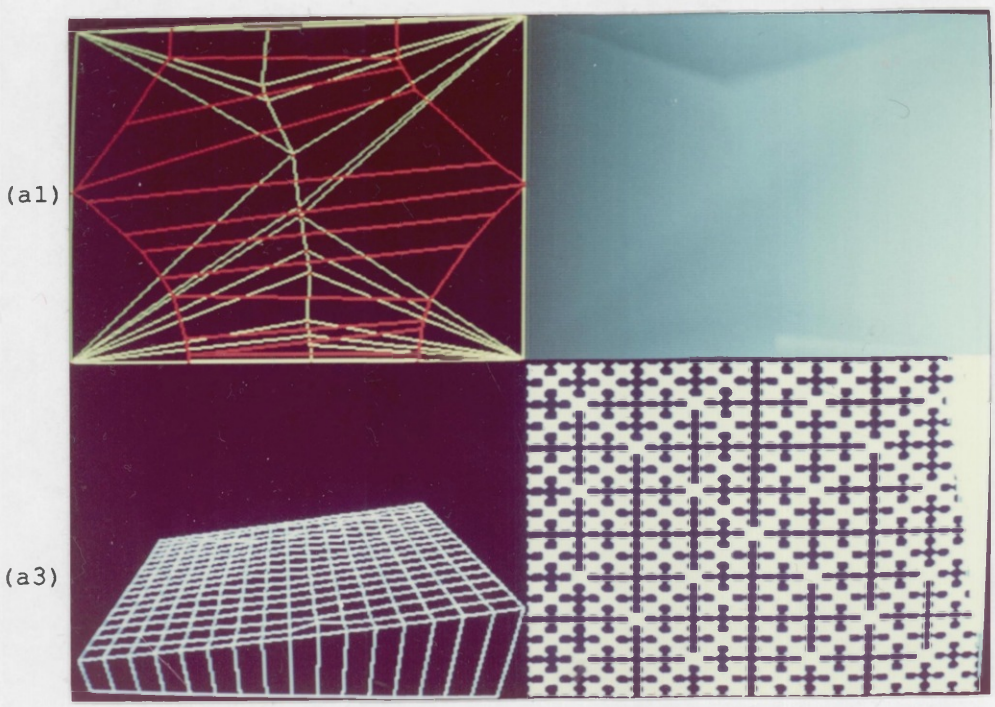
Artificially generated image: (a) shows the artificially generated image that is processed (segmented) in this section, (b) shows the image's grey-level histogram, (c) contains a wire frame drawing of the image, and (d) shows the result of thresholding the image at pixel value 128, globally.

Figure 9.11

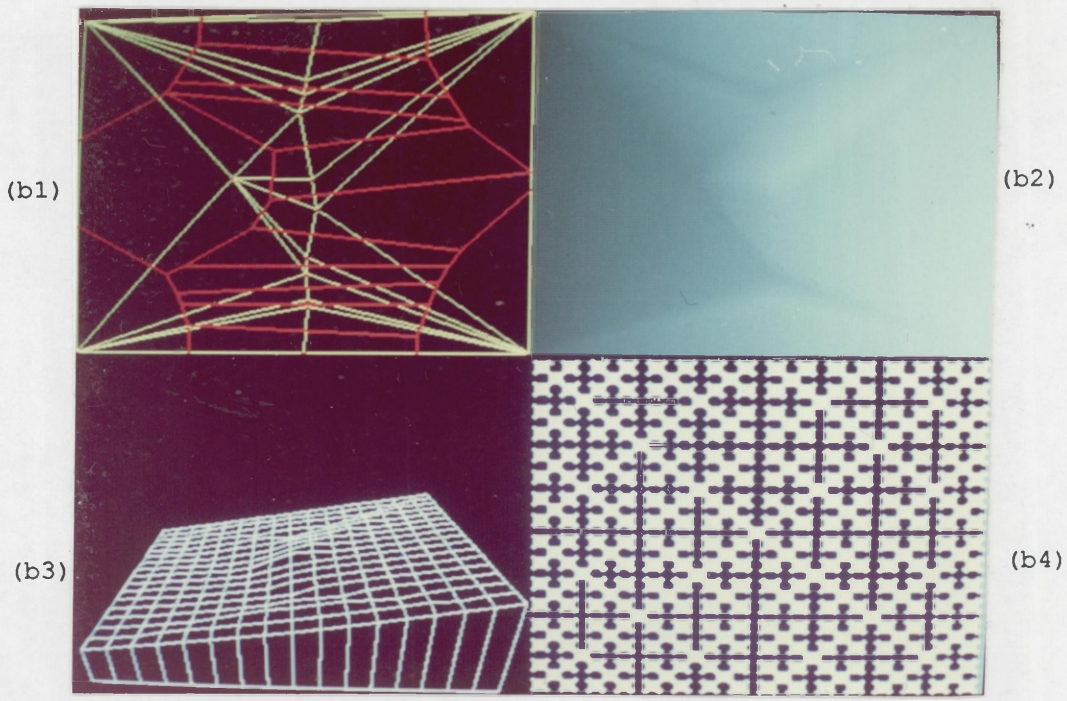
Figure 9.12 shows the results of using different clustering criteria in finding the optimal Dirichlet Tessellation of the image shown in (a) of figure 9.11. Picture (a) shows the results of using bumpiness measure B_0, picture (b) used bumpiness measure $B_{0.5}$, picture (c) used entropy and picture (d) used variance of the histogram. In all of these pictures, image (1) shows the boundaries of the tiles in the optimal Dirichlet Tessellation (in red) and the Delaunay Triangualtion (in yellow), (2) shows the threshold surface computed (using bi-linear interpolation over the triangles), (3) contains a wire frame drawing of the threshold

surface shown in (2), and (4) shows the result of thresholding the image using the threshold surface shown in (2).

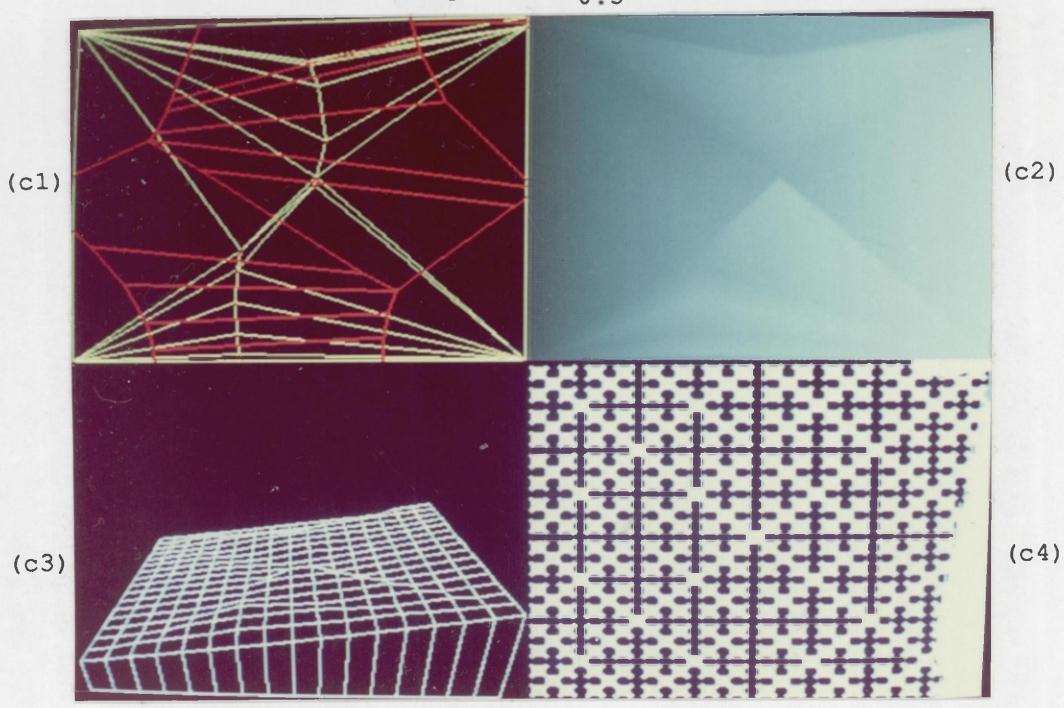
In this figure, both bumpiness B_0 and $B_{0.5}$ are shown to work better than either entropy or the variance of the histogram, as clustering criteria. Furthermore, bumpiness $B_{0.5}$ is shown to perform the best of the 4 measures, in this application.



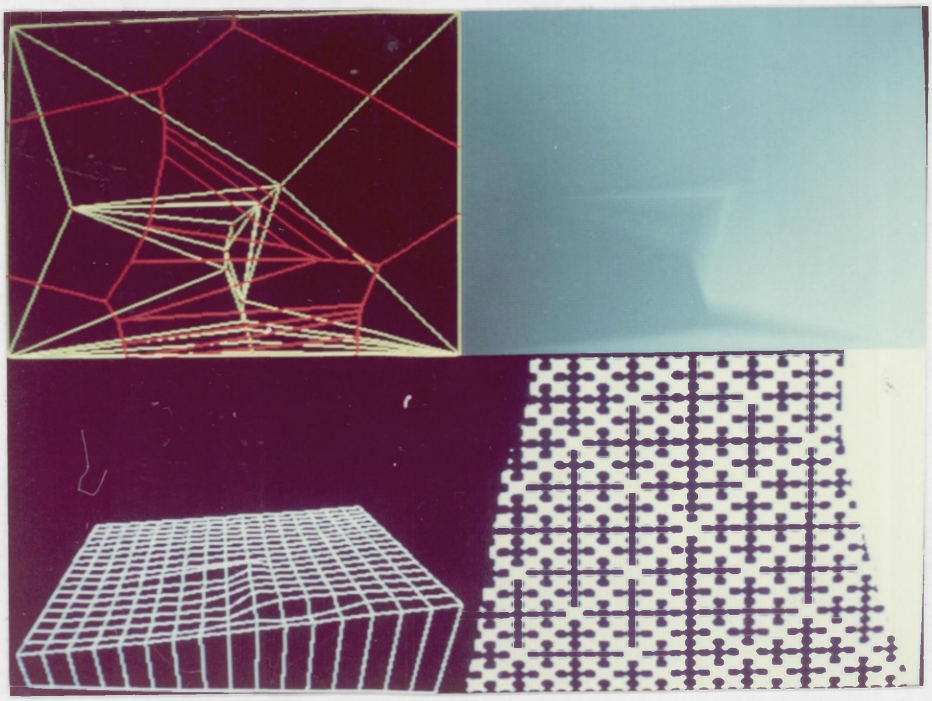
(Bumpiness B_0)



(Bumpiness $B_{0.5}$)



(Entropy)



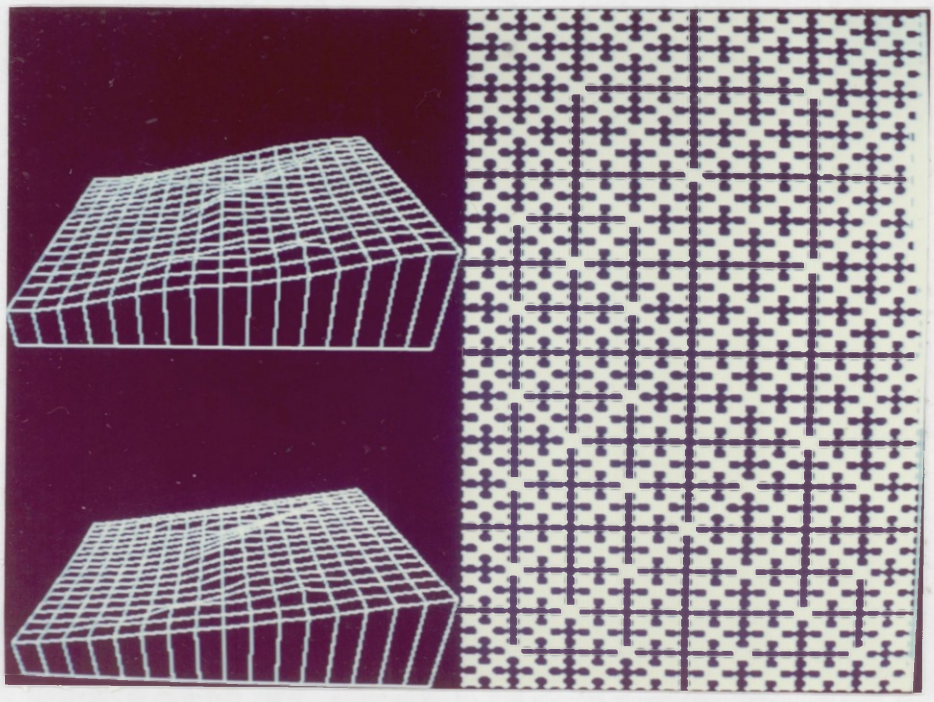
(Variance)

Comparison of clustering criteria: Picture (a) shows the results of using bumpiness measure B_0 as the clustering criterion, (b) used bumpiness measure $B_{0.5}$, (c) used entropy, and (d) used the variance of the histogram. In each picture, (1) shows the optimal Delaunay Triangulation and Dirichlet Tessellation, (2) shows the threshold surface computed, (3) contains a wire frame drawing of the threshold surface, and (4) shows the result of thresholding the image using the threshold surface shown in (2).

Figure 9.12

Figure 9.13 shows the threshold surfaces generated using bumpiness measure $B_{0.5}$ and bi-cubic interpolation of the threshold surface, with the partial derivatives of the threshold surface set to 0 and with them approximated, at the vertices of the triangles, respectively. (a) shows a wire frame drawing of the threshold surface when the partial derivatives are set to 0, and (c) for when they are approximated, using the technique described in section 8.3.4. (b) and (d) show the results of thresholding the image using the threshold surfaces shown in (a) and (c), respectively. In this example, approximating the partial derivatives produces a better result than when they are set to 0.

(a)



(b)

(c)

(d)

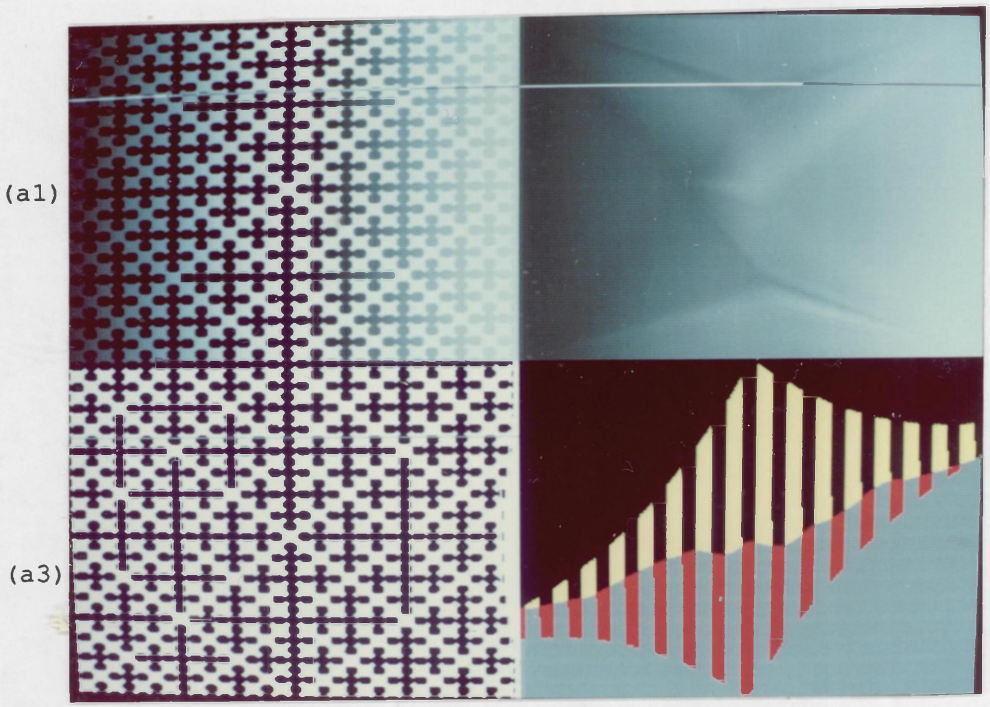
Effects of approximating derivatives of threshold surface: (a) contains a wire frame drawing of the threshold surface computed when the partial derivatives of the threshold surface are set to 0, and (c) for when the partial derivatives are approximated. In both cases, the threshold surfaces were interpolated bi-cubically. (b) and (d) show the results of thresholding the image using the threshold surfaces shown in (a) and (c), respectively.

Figure 9.13

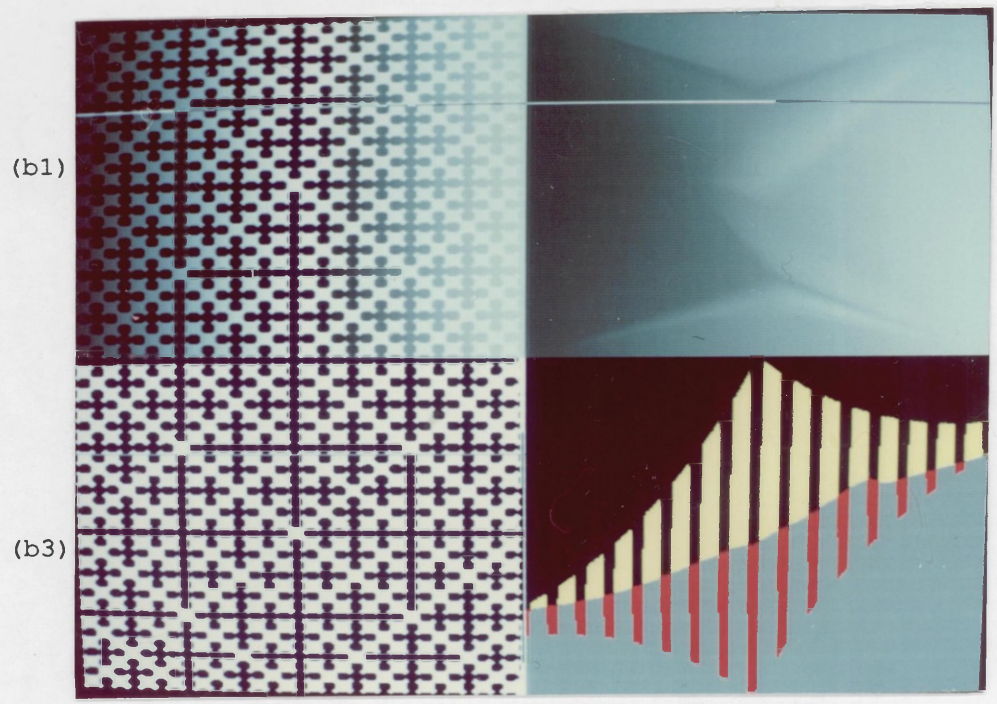
Figure 9.14 shows the results of using different techniques to interpolate the threshold surface used to segment the image shown in (a) of figure 9.11. Picture (a) shows the results of using bi-cubic interpolation, setting the partial derivaties of the threshold surface at the triangles' vertices to zero, (b) uses bi-cubic interpolation, approximating the partial derivatives of the threshold surface at the triangles' vertices, and (c) uses bi-linear interpolation. The Delaunay Triangulation and Dirichlet Tessellation for all three cases is the same and the clustering criterion used to obtain them was bumpiness measure $B_{0.5}$.

In all three pictures, image (1) shows the image to be segmented, with one row of pixels highlighted, (2) shows the threshold surface computed, with the same row of pixels highlighted, (3) shows the result of the segmentation using the threshold surface shown in (2) (also with the row of pixels highlighted), and (4) contains cross-sectional plots of the rows of pixels highlighted in (1) (shown in yellow) and (2) (shown in red).

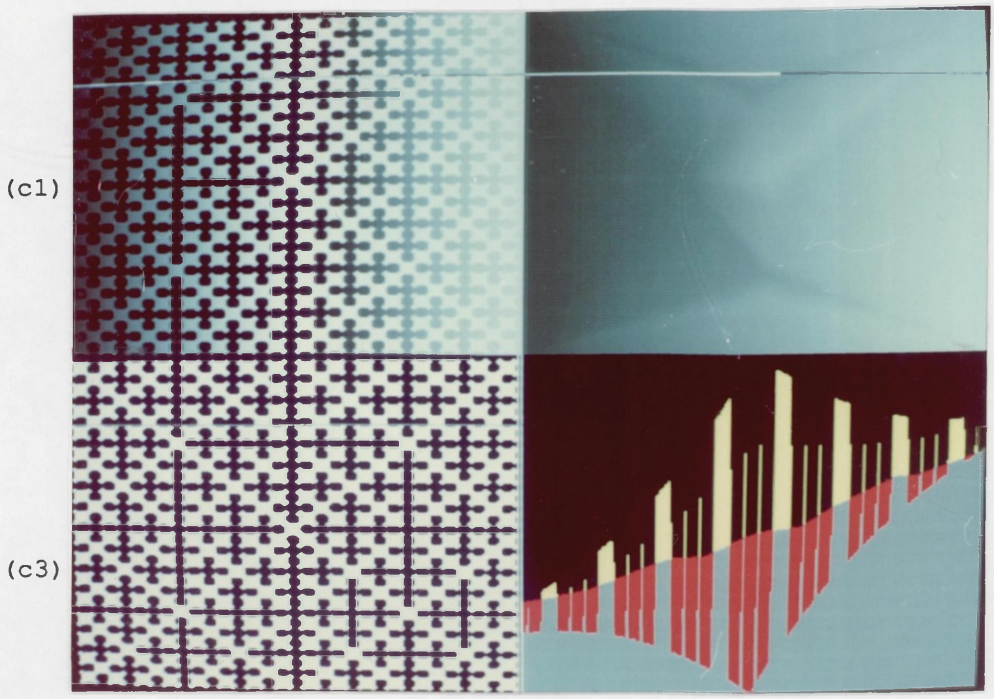
In this example, bi-linear interpolation is shown to work better than the other two methods tried.



(Bi-cubic interpolation, derivatives set to 0)



(Bi-cubic interpolation, derivatives approximated)



(Bi-linear interpolation)

Comparison of interpolation techniques: (a) shows the effects of using bi-cubic interpolation, setting the partial derivatives of the threshold surface at the triangles' vertices to 0, (b) shows the same, but this time approximating the partial derivatives of the threshold surface at the triangles' vertices, and (c) shows results from using bi-linear interpolation.

Figure 9.14

CHAPTER 10

QUANTITATIVE EVALUATION OF SEGMENTATION TECHNIQUES

10.1 Discussion of Quantitative Measures

Section 2.4 described several quantitative measures of the degree of image restoration achieved by the application of image restoration techniques. This section discusses the properties of the images processed and the suitability of various techniques, in this application.

Section 10.2 shows some experimental results from degrading known binary images, restoring them using various segmentation techniques and then using several measures to determine how effective the restoration has been. Section 10.3 then discusses the effectiveness of the segmentation techniques and evaluates the suitability of the measures used, in this application.

In image segmentation, the correct result (segmentation) is not usually available; in that sense, it is not a restoration technique. However, to quantify the quality of a segmentation, the segmented image needs to be compared to a known correct answer. Therefore, in this evaluation, known binary images are degraded and then various segmentation techniques are used to try to restore the degraded images. This way, the effectiveness of the segmentation techniques can be quantified objectively as the "correct" segmentations are known.

For the purpose of evaluation, binary images are used which are assumed to contain only pixels with values 0 (black) and 1 (white), corresponding to background and foreground pixels respectively. This means that in (1) of section 2.4:

- the values of f(x,y) and g(x,y) can be 0 or 1 }
- the values of e(x,y) can be -1, 0 or 1. } (1)

The images will be square (256x256 pixels) and each contain 65536 pixels. This simplifies the formulation of many of the measures given in section 2.4. For example:

(i) the mean of a binary image is given by:

(the number of foreground pixels) / 65536 (2a)

(ii) the variance of these images is given by:

(the mean of the image) *
(1 - (the mean of the image)) (2b)

(iii) the square error is given by:

the number of misclassified pixels. (2c)

In comparing two aligned images, the most commonly used measure of similarity is mean square error or root mean square error. In the predominantly statistical paper by Therrien [109], the measure used was mean square error. Sometimes the mean square error is normalised by the variance of the original image [27,32,96] or that of the noise [7]. This is because images with a wide spread

of pixel values (large variance) are expected to misclassify more pixels in the restored image than images with a narrower spread of pixel values.

When comparing binary images, the usual measure of similarity is the number of misclassified pixels (square error) [1]. This could be broken down into the number of foreground pixels that have been classified as being background and the number of background pixels classified as being foreground, but both misclassifications affect both regions. In the tables in section 10.2, both the number of foreground pixels classified as belonging to the background and the the number of background pixels classified as belonging to the foreground are given, as well as the total number of misclassified pixels. If these measures are to be compared for images of different sizes, then they should be scaled by the numbers of pixels in the images (to yield the mean square error). As all the segmentations evaluated in section 10.2 are the same size, this scaling is not really necessary, but as MSE (mean square error) is used so widely, the measure is used.

An analysis of results obtained during the course of this research suggests that the "goodness of segmentation" measure used should be a function of both the number of pixels misclassified and the sizes of the foreground and background in the original image. For example, misclassifying 20 pixels is considered more serious in an image that should contain only 5 foreground pixels than it is in an image containing 32000 foreground pixels. Also, misclassifying foreground pixels as background and background pixels as

foreground, are considered to be equally undesirable.

It is suggested that the error measure used should:

(i) tend to 0 as the reconstructed image tends to the original image

(ii) only tend to 0 as the reconstructed image tends to the original image

(iii) take into account the sizes of the object and the background in the original image.

(3)

Clearly, one dimensional grey level histogram based measures, such as comparisons between the moments of the original and the reconstructed images, do not satisfy (3ii) above. Neither does the number of foreground/background boundary pixels satisfy this condition. This does not mean that the measures are without merit; clearly, if two images have very different means, the images are quite different. However, two images with the same mean are not necessarily similar.

While the differences between the means, variances and number of boundary pixels in the original and reconstructed images can not be relied upon to detect even significant differences between the images, they will be used in the evaluation given in the next section, as large differences in these values for the respective images would suggest differences between the images.

A new measure, the Maximum Error Measure (MEM), which is independant of image size, is suggested and is defined for binary images as:

$$\text{MEM} \triangleq \text{NMP} \;/\; \text{Min}(\text{Number of foreground pixels}, \text{Number of background pixels}) \qquad (4)$$

where: - NMP is the number of misclassified pixels

- the number of foreground and background pixels are those numbers for the original image.

This measure could rather be given by two values, MEM_0 and MEM_1, where:

$$\text{MEM}_i \triangleq \text{NMP} \;/\; (\text{the number of pixels in region i}) \qquad (5)$$

which would give measures of how successful the segmentation was at locating each of the foreground and the background. However, it is suggested that in this application, the quality of a segmentation is the poorest of the qualities of the segmentations of the individual regions in the image. For this reason, the Maximum Error Measure as given in (4) above, is used in the evaluation given in section 10.2.

For images permitting N distinct pixel values, the generalised Maximum Error Measure is given as:

$$\text{MEM}_N \triangleq \max_i \{\text{NMP}_i \;/\; \text{Number of pixels in region i}\} \qquad (6)$$

where: - NMP_i is the sum of the number of pixels which should have been given a value of i in the restored image but were not and the number of pixels that were given a value of

i in the restored image but should not have been.

However, the measure is best suited to the binary case and for values of N much above 3, the measure is much too severe.

In the binary case the measure is similar to (but more severe than) the mean square error normalised by the variance of the original image. Both measures will record a higher error when a given number of pixels, N, are misclassified in an image most of whose pixels belong to the same region than when the same number of pixels are misclassified in an image containing approximately equal numbers of foreground and background pixels. Both measures satisfy all the conditions given in (3).

10.2 Experimental Results

This section evaluates the results of segmenting images for which the "correct" segmentations are known. This is done by giving several quantitative measures of properties of the segmented images and how good the segmentations are.

Six known binary images are degraded and then the degraded images are segmented (restored). Nine different segmentation techniques are used in this section, but only six are used to segment any one degraded image. These techniques will be called "Method 1" to "Method 9" for the rest of this chapter.

Following is an explanation of each of these methods:

Method 1: Global threshold at intensity 128.

Method 2: Moving window median threshold with window size of 65x65 pixels. When part of the window is outside the image space, the pixel values are assumed to be reflected at the image's edge.

Method 3: Segmentation technique developed during the course of this research, using clustering criterion Bumpiness $B_{0.5}$ and planar interpolation.

Method 4: Segmentation technique developed during the course of this research, using clustering criterion Bumpiness $B_{0.5}$ and cubic interpolation with the partial derivatives of the threshold surface approximated at the vertices of the triangles.

Method 5: Segmentation technique developed during the course of this research, using clustering criterion Bumpiness B_0 and planar interpolation.

Method 6: Segmentation technique developed during the course of this research, using variance of the histogram as the clustering criterion, and planar interpolation.

Method 7: Segmentation technique developed during the course of this research, using the entropy of the histogram as the clustering criterion, and planar interpolation.

Method 8: Segmentation technique developed during the course of this research, using variance of the histogram as the clustering criterion, and cubic interpolation with the partial derivatives of the threshold surface approximated at the vertices of the triangles.

Method 9: Segmentation technique developed during the course of this research, using the entropy of the histogram as the clustering criterion, and cubic interpolation with the partial derivatives of the threshold surface approximated at the vertices of the triangles.

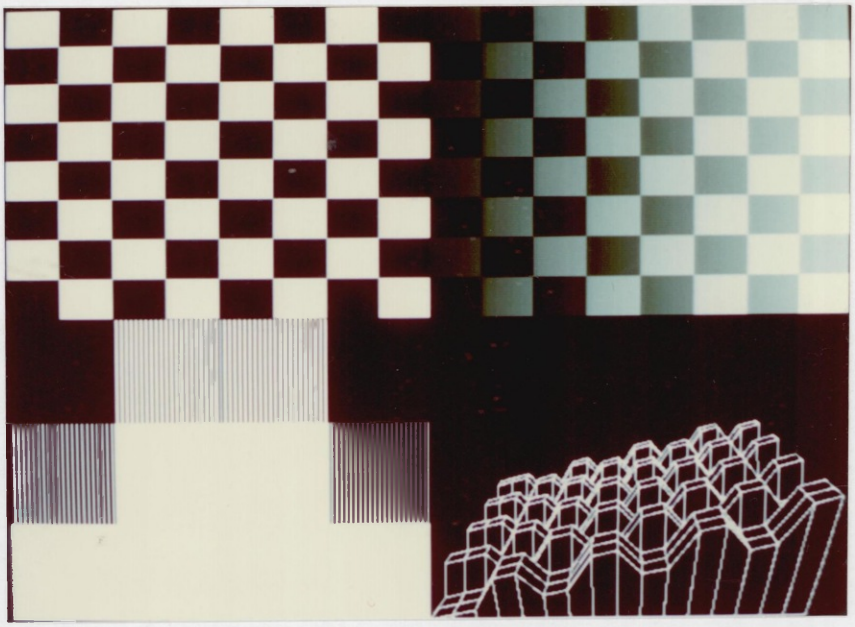
For each set of experiments, the original binary image, the degraded version thereof, the histogram of the degraded image, and

a "wire frame" plot of the degraded image, will be shown. These will be followed by pictures of the the reconstructed images and tables giving the following statistics for both the original (when applicable) and the restored (segmented) images:

- the number of foreground pixels in the image
- the number of background pixels in the image
- the mean pixel value in the image
- the variance of the pixel values in the image
- the number of background/foreground boundary pixels in the image
- the number of pixels that are classified as belonging to the background in the original image but which are classified as foreground pixels in the restored image
- the number of pixels that are classified as belonging to the foreground in the original image but which are classified as background pixels in the restored image
- the number of misclassified pixels
- the mean square error (MSE)
- the MSE / (the variance of the values of the pixels in the original image)
- the MSE / (the variance of the noise induced by the degradation and restoration process)
- the Maximum Error Measure, as defined in (4) of section 10.1.

Section 10.3 then discusses the results presented in this section.

(a)



(b)

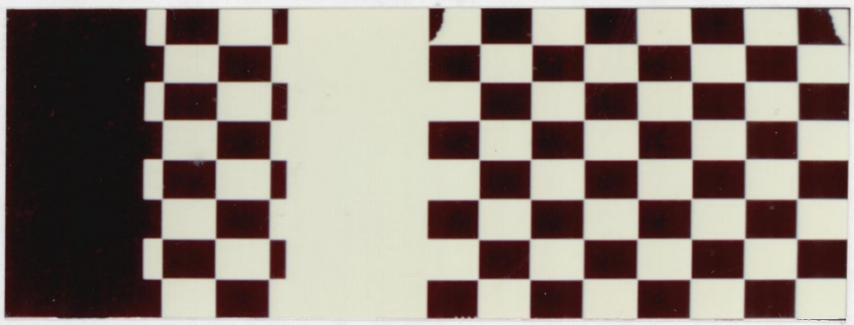
(c)

(d)

8x8 checkerboard: (a) shows an 8x8 checkerboard, (b) shows the image in (a) degraded by illuminating the image unevenly, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

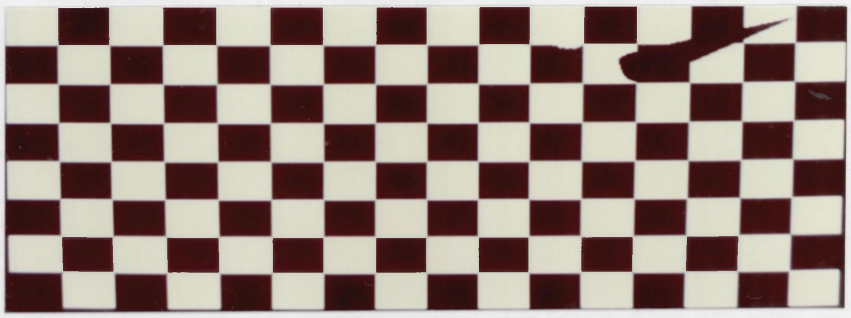
Figure 10.1

(a)



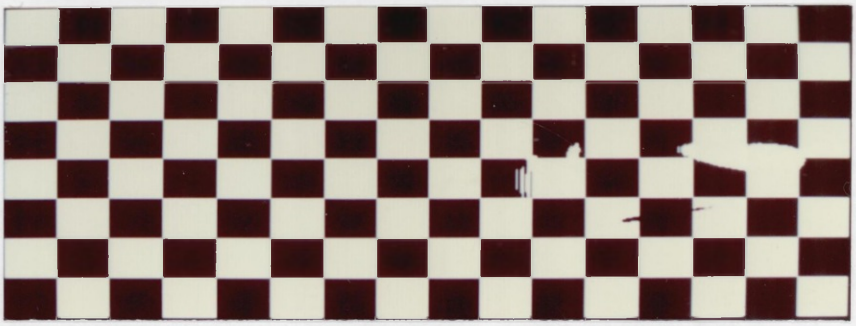
(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.1(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.1(b) using Methods 1, 2, 3, 4, 7 and 9 respectively.

Figure 10.2

	Original	Method 1	Method 2	Method 3	Method 4	Method 7	Method 9
Number of foreground pixels	32768	32896	32816	32768	32204	32802	33552
Number of background pixels	32768	32640	32720	32768	33332	32734	31984
Mean pixel value	0.5000	0.5020	0.5007	0.5000	0.4914	0.5005	0.5120
Variance of pixel values	0.2500	0.2500	0.2500	0.2500	0.2499	0.2500	0.2499
Number of boundary pixels	10528	4751	10785	10528	10714	10626	11084
Number of background pixels classified as foreground	-	11008	268	0	36	34	844
Number of foreground pixels classified as background	-	10880	220	0	600	0	60
Number of misclassified pixels	-	21888	488	0	636	34	904
Mean Square Error (MSE)	-	0.3340	0.0074	0	0.0097	0.0005	0.0138
MSE / Variance(original)	-	1.3359	0.0298	0	0.0388	0.0021	0.0552
MSE / Variance(noise)	-	1.0000	1.0001	-	1.0077	1.0005	1.0105
Maximum Error Measure	-	0.6680	0.0149	0	0.0194	0.0010	0.0276

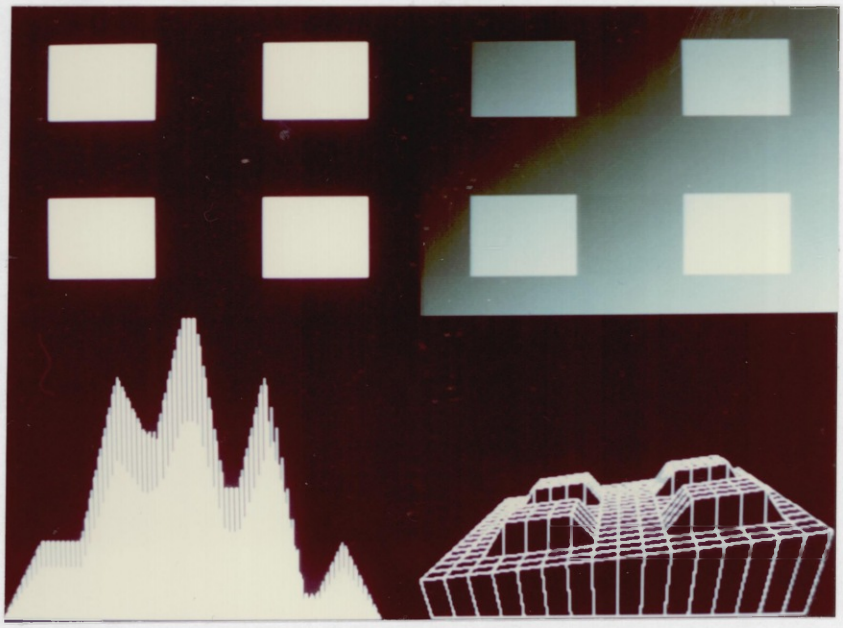
Table 10.1

The figures in table 10.1 clearly show Method 3 ($B_{0.5}$, planar) to be the only technique to achieve perfect restoration of the image. Using MEM as the error measure, Method 7 (entropy, planar) is the next most successful and Method 2 (moving window) is the third most, but being much less successful than Method 7. Method 1 (global threshold) is the worst by far, with a Maximum Error of 66.8%.

Using the difference in mean pixel value as the error measure, Methods 3, 2 and 7 are still the most successful, but Method 1 appears to be more successful than Methods 4 ($B_{0.5}$, cubic) and 9 (entropy, cubic), which might seem to contradict the result of using MEM. This is because Method 1 misclassifies approximately equal numbers of foreground and background pixels, thus producing an image with approximately the same number of foreground and background pixels as the original, and therefore having approximately the same moments as well.

The measure (MSE/Variance(noise)) also considers Method 1 to be the most successful, but this is because the noise has a high variance rather than because the method has a low MSE. This would suggest that (MSE/Variance(noise)) is a poor measure rather than that Method 1 is a good segmentation technique. This can be verified by considering figure 10.2(a).

(a)



(b)

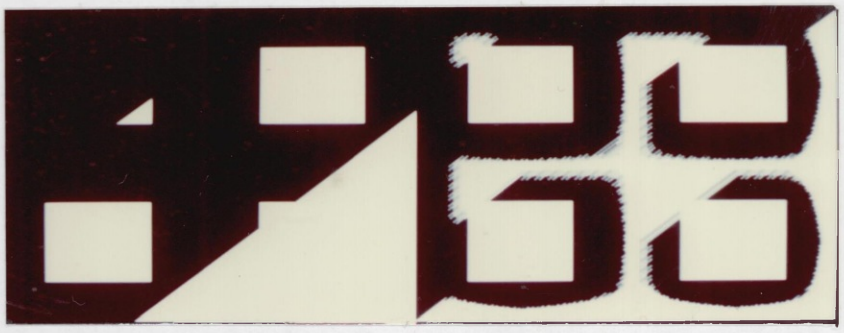
(c)

(d)

4 white squares: (a) shows an image of 4 white squares on a black background, (b) shows the image in (a) degraded by illuminating the image unevenly, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

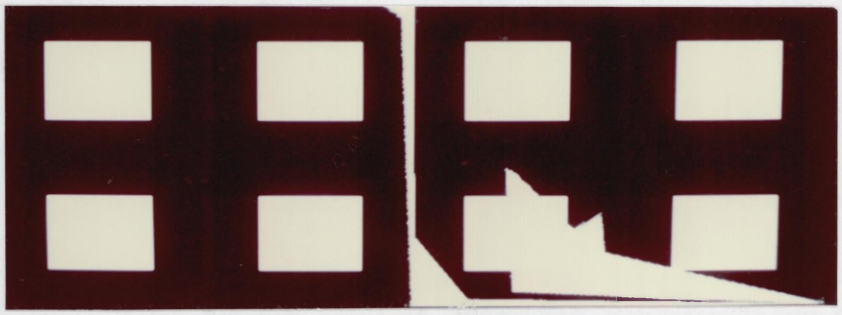
Figure 10.3

(a)



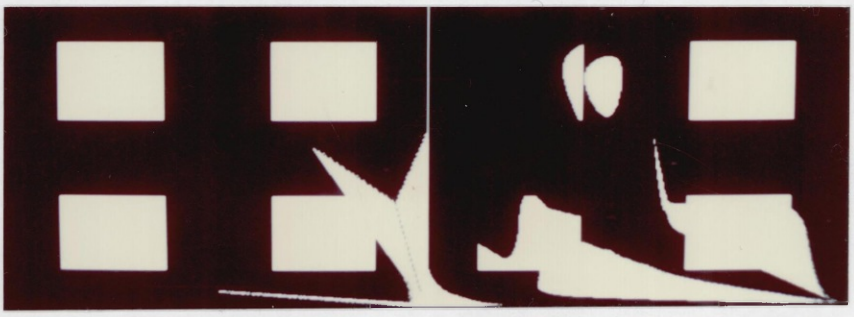
(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.3(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.3(b) using Methods 1, 2, 3, 6, 7 and 8 respectively.

Figure 10.4

	Original	Method 1	Method 2	Method 3	Method 6	Method 7	Method 8
Number of foreground pixels	16384	23189	31971	17644	23249	19703	17649
Number of background pixels	49152	42347	33565	47892	42287	45833	47887
Mean pixel value	0.2500	0.3538	0.4878	0.2692	0.3548	0.3006	0.2693
Variance of pixel values	0.1875	0.2286	0.2499	0.1967	0.2289	0.2103	0.1968
Number of boundary pixels	3056	2457	14253	4274	5272	4997	5594
Number of background pixels classified as foreground	-	10670	15587	1260	6865	3319	6437
Number of foreground pixels classified as background	-	3865	0	0	0	0	5172
Number of misclassified pixels	-	14535	15587	1260	6865	3319	11609
Mean Square Error (MSE)	-	0.2218	0.2378	0.0192	0.1048	0.0506	0.1771
MSE / Variance(original)	-	1.1829	1.2685	0.1025	0.5587	0.2701	0.9447
MSE / Variance(noise)	-	1.0511	1.3121	1.0196	1.1170	1.0533	1.0021
Maximum Error Measure	-	0.8871	0.9514	0.0769	0.4190	0.2026	0.7086

Table 10.2

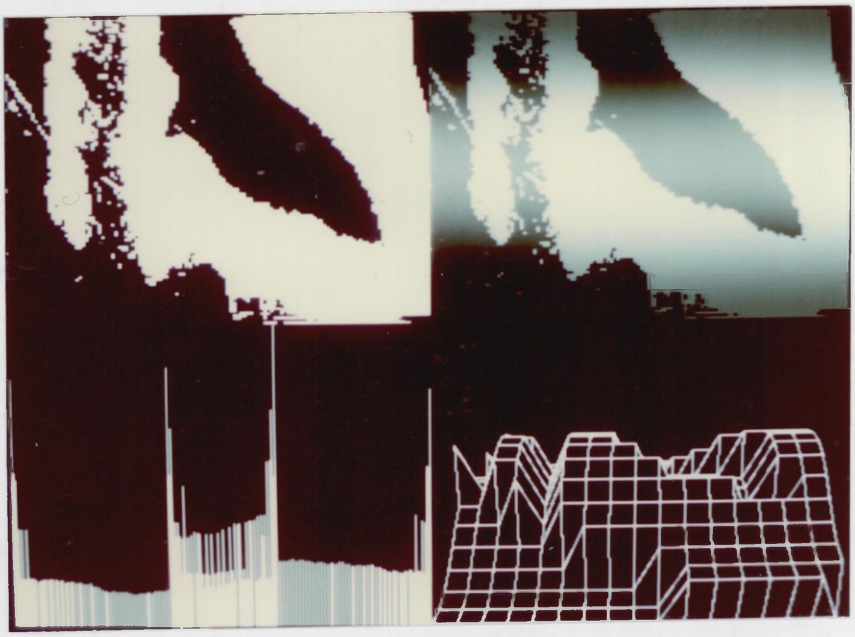
The figures in table 10.2 clearly show Method 3 ($B_{0.5}$, planar) to work the best for the image processed, except for the measures (MSE/Variance(noise)), which shows Method 8 (variance, cubic) to perform better than Method 3, and the differences in the number of boundary pixels between the original and the restored images, which shows Method 1 (global threshold) to perform the best.

The reason for the first conflict between the measures is that Method 8 misclassified many pixels (which is why it performs badly when using MEM as the error measure), but misclassifies approximately equal numbers of foreground and background pixels, therefore giving the noise induced a high variance (therefore, (MSE/Variance(noise)) is small).

Method 7 (entropy, planar) performs quite well again, but Method 2 (moving window) performs the worst for all the error measures used, with a Maximum Error of over 95%. This is because moving window median thresholding assumes that the image being processed has approximately equal numbers of foreground and background pixels, an assumption that is not valid here.

(a)

(b)



(c)

(d)

Humped ventricle: (a) contains the binary image of a ventricle, (b) shows the image in (a) with "humped" illumination, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

Figure 10.5

(a)



(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.5(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.5(b) using Methods 1, 2, 3, 4, 5 and 6 respectively.

Figure 10.6

	Original	Method 1	Method 2	Method 3	Method 4	Method 5	Method 6
Number of foreground pixels	36887	35688	40178	30505	23619	29403	32772
Number of background pixels	28649	29848	25358	35031	41917	36133	32764
Mean pixel value	0.5629	0.5446	0.6131	0.4655	0.3604	0.4487	0.5001
Variance of pixel values	0.2460	0.2480	0.2372	0.2488	0.2305	0.2474	0.2500
Number of boundary pixels	8932	5321	10449	6289	6514	7492	5220
Number of background pixels classified as foreground	-	9026	6003	5479	4314	5385	8057
Number of foreground pixels classified as background	-	10225	2712	11861	17582	12869	12172
Number of misclassified pixels	-	19251	8715	17340	21896	18254	20229
Mean Square Error (MSE)	-	0.2937	0.1330	0.2646	0.3341	0.2785	0.3087
MSE / Variance(original)	-	1.1939	0.5405	1.0753	1.3579	1.1320	1.2545
MSE / Variance(noise)	-	1.0011	1.0193	1.0372	1.1398	1.0491	1.0129
Maximum Error Measure	-	0.6720	0.3042	0.6053	0.7643	0.6372	0.7061

Table 10.3

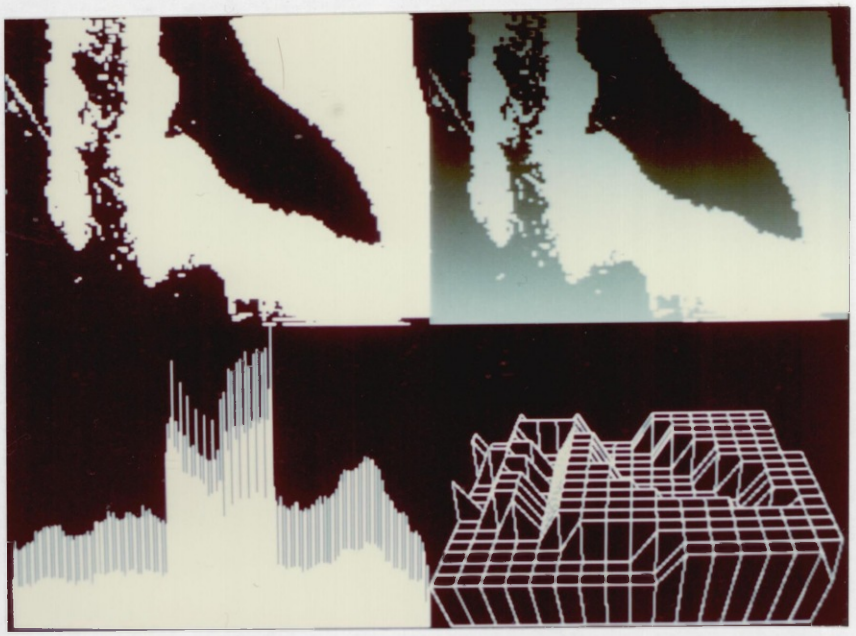
The figures in table 10.3 do not agree on the most successful method for processing the image given in figure 10.5(b). The measures mean pixel value and (MSE/Variance(noise)) consider Method 1 (global threshold) to be the most successful, the differences between the number of boundary pixels in the original and restored images and MEM consider Method 2 (moving window) to be the most successful, and the differences between the variance of the pixel values in the original and reconstructed images, considers Method 5 (B_0, planar) to be the most successful.

This is because Method 2 misclassified relatively few pixels while Method 1, in misclassifying relatively many pixels, misclassified approximately equal numbers of foreground and background pixels.

Once again, we must conclude that the histogram based measures, the differences in the number of boundary pixels, and (MSE/Variance(noise)), are not good measures for our application.

Using MEM as the error measure, we see that Method 3 ($B_{0.5}$, planar), while performing badly with just over a 60% Maximum Error, performed the second best of the methods used.

(a)



(b)

(c)

(d)

Linear ventricle: (a) contains the binary image of a ventricle, (b) shows the image in (a) illuminated unevenly, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

Figure 10.7

(a)



(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.7(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.7(b) using Methods 1, 2, 3, 4, 5 and 6 respectively.

Figure 10.8

	Original	Method 1	Method 2	Method 3	Method 4	Method 5	Method 6
Number of foreground pixels	36887	35856	39290	35884	28305	35576	36536
Number of background pixels	28649	29680	26246	29652	37231	29960	29000
Mean pixel value	0.5629	0.5471	0.5995	0.5475	0.4319	0.5428	0.5575
Variance of pixel values	0.2460	0.2478	0.2401	0.2477	0.2454	0.2482	0.2467
Number of boundary pixels	8932	6213	12109	10591	10454	10658	10406
Number of background pixels classified as foreground	-	6976	3847	1895	1086	1994	1778
Number of foreground pixels classified as background	-	8007	1444	2898	9668	3305	2129
Number of misclassified pixels	-	14983	5291	4793	10754	5299	3907
Mean Square Error (MSE)	-	0.2286	0.0807	0.0731	0.1641	0.0809	0.0596
MSE / Variance(original)	-	0.9292	0.3281	0.2972	0.6669	0.3286	0.2423
MSE / Variance(noise)	-	1.0011	1.0169	1.0032	1.1167	1.0050	1.0005
Maximum Error Measure	-	0.5230	0.1847	0.1673	0.3754	0.1850	0.1364

Table 10.4

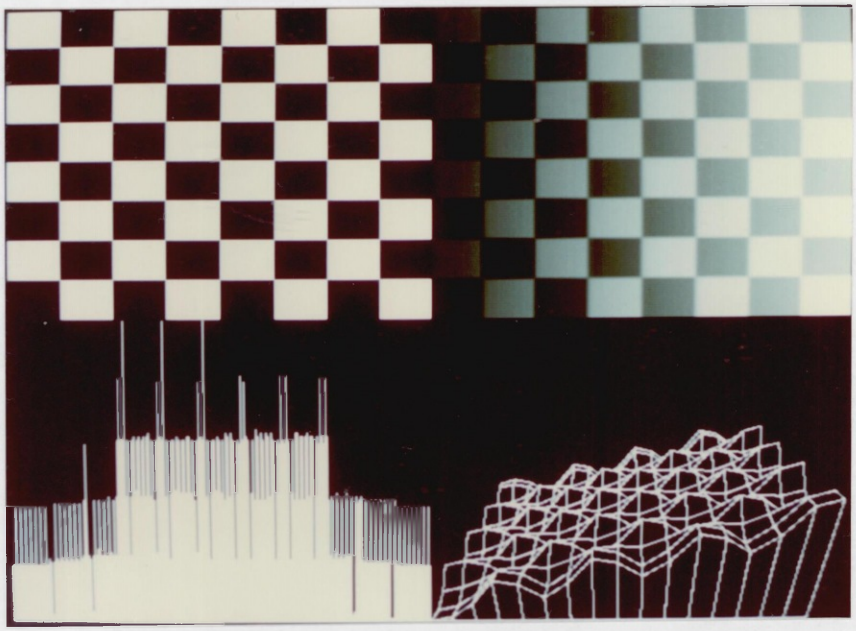
The figures in table 10.4 suggest that Method 6 (variance, planar) is the technique most successful at segmenting this image, although the variance of the pixel values suggests that it is only the second most successful technique. Using MEM as the error measure, Method 3 ($B_{0.5}$, planar) performs the second best again, with a Maximum Error of 17%, and Method 1 (global threshold) performs relatively poorly, with a Maximum Error of 52%.

However, once again, Method 1 misclassifies approximately equal numbers of foreground and background pixels, and so appears to perform quite well when (MSE/Variance(noise)) is used as the error measure.

Method 2 (moving window) is seen to perform well with only an 18.5% Maximum Error.

(a)

(b)



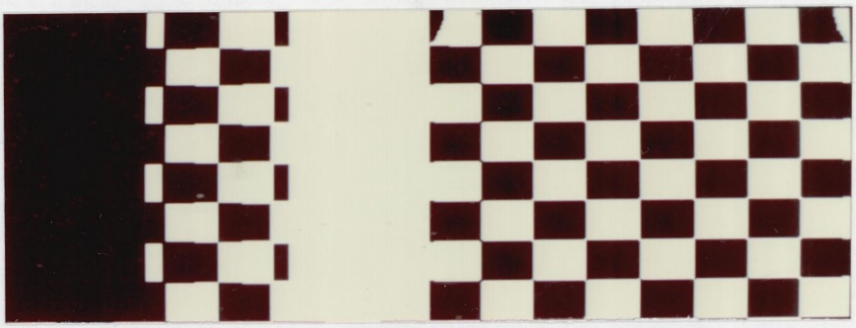
(c)

(d)

Blurred checkerboard: (a) shows an 8x8 checkerboard, (b) shows the image in (a) degraded by illuminating the image unevenly and then locally averaging the image, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

Figure 10.9

(a)



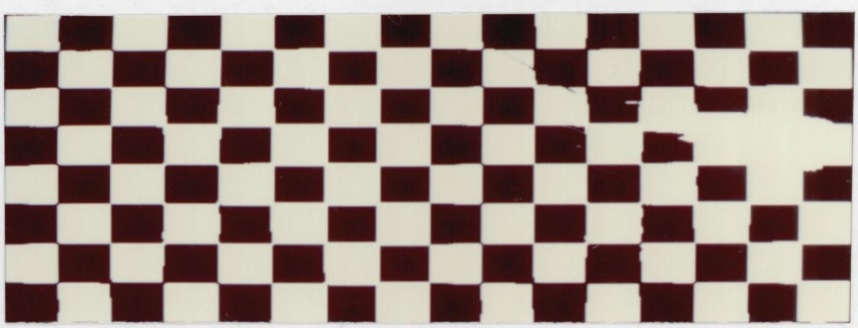
(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.9(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.9(b) using Methods 1, 2, 3, 4, 7 and 9 respectively.

Figure 10.10

	Original	Method 1	Method 2	Method 3	Method 4	Method 7	Method 9
Number of foreground pixels	32768	32889	32841	33014	33155	33876	34766
Number of background pixels	32768	32647	32695	32522	32381	31660	30770
Mean pixel value	0.5000	0.5018	0.5011	0.5038	0.5059	0.5169	0.5305
Variance of pixel values	0.2500	0.2500	0.2500	0.2500	0.2500	0.2497	0.2491
Number of boundary pixels	10528	4583	10742	10462	10417	10348	10110
Number of background pixels classified as foreground	-	11446	323	538	759	1342	2668
Number of foreground pixels classified as background	-	11325	250	292	372	234	670
Number of misclassified pixels	-	22771	573	830	1131	1576	3338
Mean Square Error (MSE)	-	0.3475	0.0087	0.0127	0.0173	0.0240	0.0509
MSE / Variance(original)	-	1.3898	0.0350	0.0507	0.0690	0.0962	0.2037
MSE / Variance(noise)	-	1.0000	1.0001	1.0011	1.0020	1.0120	1.0186
Maximum Error Measure	-	0.6949	0.0175	0.0253	0.0345	0.0481	0.1019

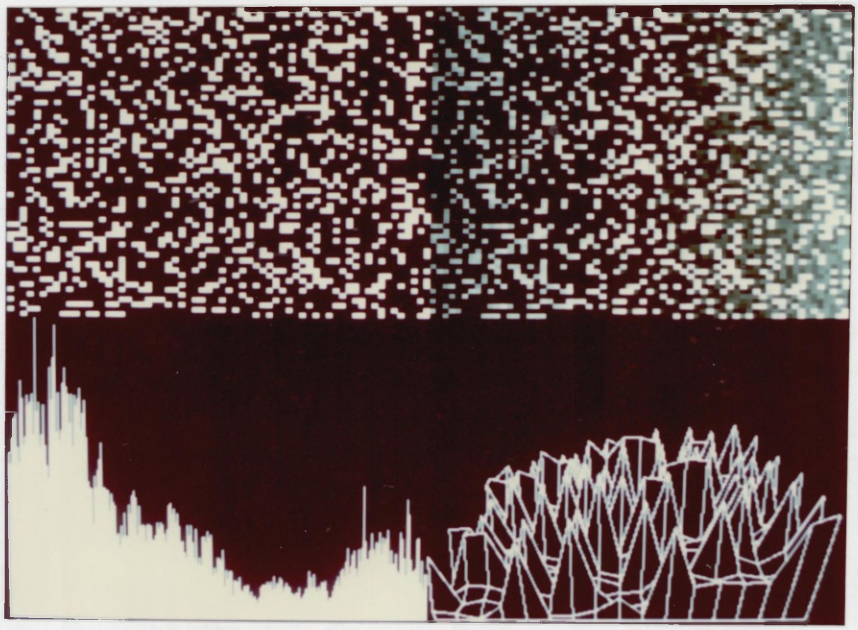
Table 10.5

The figures in table 10.5 suggest that Method 2 (moving window) is the most successful at segmenting the image shown in figure 10.9(b). The measure MEM shows Method 1 to be the most successful, with a Maximum Error of under 2%, followed by Method 3 ($B_{0.5}$, planar) with a Maximum Error of 2.5%, with Method 1 (global threshold) performing the worst, with a Maximum Error of almost 70%. However, because Method 1 misclassified approximately equal numbers of foreground and background pixels, it is considered to be the most successful method by the measure (MSE/Variance(noise)) and the second most successful using the differences between the mean pixel values of the original and reconstructed images as the measure.

Method 3 performs the best if the differences between the number of boundary pixels in the original and the restored images, are used as the error measures.

Given that Method 1 misclassified 22771 pixels while Method 2 misclassified only 573 pixels, and that a subjective analysis of figures 10.10(a) and 10.10(b) would surely consider figure 10.10(b) to be a better approximation to figure 10.9(a) than figure 10.10(a) is, it is again apparent that (MSE/Variance(noise)) is not a good error measure for our purposes.

(a)



(b)

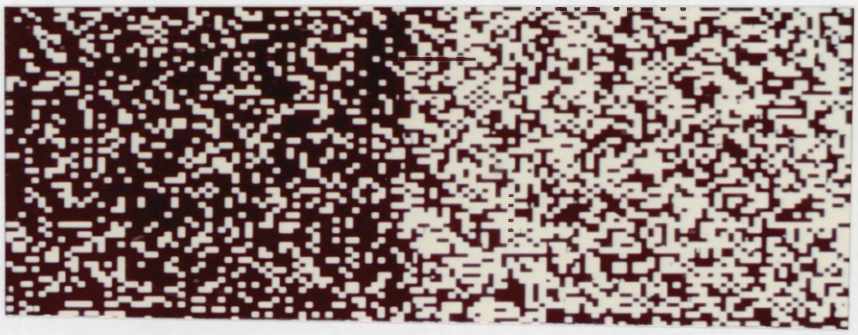
(c)

(d)

Noisy squares: (a) shows some squares, (b) shows the image in (a) degraded by allowing the background and foreground pixels each to take on one of a range of values, and then illuminating the image unevenly, (c) shows the histogram of the pixel values in (b), and (d) shows a wireframe plot of (b).

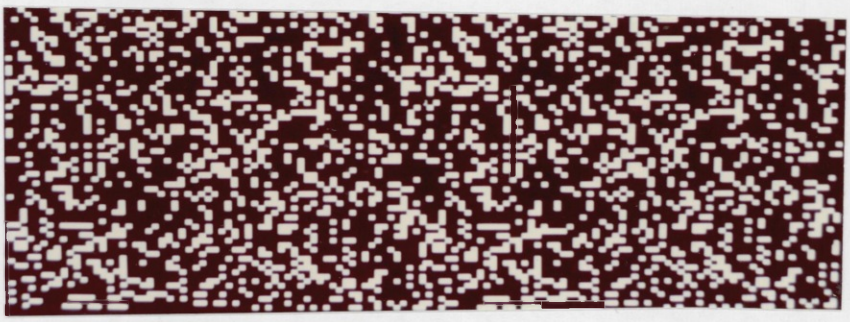
Figure 10.11

(a)



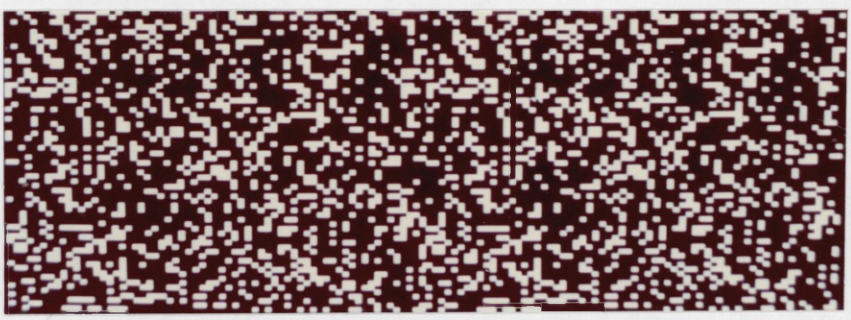
(b)

(c)



(d)

(e)



(f)

Segmentations of figure 10.11(b): (a) to (f) show the segmentations (restorations) of the image shown in figure 10.11(b) using Methods 1, 2, 3, 4, 7 and 9 respectively.

Figure 10.12

	Original	Method 1	Method 2	Method 3	Method 4	Method 7	Method 9
Number of foreground pixels	16592	17124	33101	16645	16683	16628	16715
Number of background pixels	48944	48412	32435	48891	48853	48908	48821
Mean pixel value	0.2532	0.2613	0.5051	0.2540	0.2546	0.2537	0.2551
Variance of pixel values	0.1891	0.1930	0.2500	0.1895	0.1898	0.1893	0.1900
Number of boundary pixels	33817	34289	45167	33919	33923	33873	34023
Number of background pixels classified as foreground	-	768	16509	53	91	36	123
Number of foreground pixels classified as background	-	236	0	0	0	0	0
Number of misclassified pixels	-	1004	16509	53	91	36	123
Mean Square Error (MSE)	-	0.0153	0.2519	0.0008	0.0014	0.0005	0.0019
MSE / Variance(original)	-	0.0810	1.3323	0.0043	0.0073	0.0029	0.0099
MSE / Variance(noise)	-	1.0043	1.3367	1.0008	1.0014	1.0005	1.0019
Maximum Error Measure	-	0.0605	0.9950	0.0032	0.0055	0.0022	0.0074

Table 10.6

The figures in table 10.6 clearly show Method 7 (entropy, planar) to work the best for the image processed, followed by Methods 3 ($B_{0.5}$, planar), 4 ($B_{0.5}$, cubic) and 9 (entropy, cubic), with all four methods having a Maximum Error of less than 1%. By contrast, Method 2 (moving window) performs the worst, with a Maximum Error of 99.5%.

This set of results suggests that while moving window median thresholding may be appropriate for segmenting images with approximately equal numbers of foreground and background pixels, it does not perform well for other images. We see in table 10.6 that it classified approximately half of the pixels as being foreground whereas only about a quarter actually are foreground.

10.3 Conclusion

The results obtained in this chapter can be used to evaluate both the segmentation achieved by various techniques, and the measures used to evaluate the segmentations.

It has been shown in section 10.2 that histogram based measures, such as the differences between the means or the variances of the pixels in the original and restored images, are not good error measures for this application. This is because images with approximately equal numbers of foreground and background pixels misclassified will have approximately the same moments as the original image, and so will be considered to have very little error.

Mean Square Error (MSE) was seen to perform quite well, but (MSE/Variance(noise)) did not. One reason for this is that images in which there are approximately equal numbers of misclassified foreground and background pixels, will be considered better than images with the same number of misclassified pixels, but where all these pixels belong to the same region (i.e. foreground or background).

As expected, the differences between the number of boundary pixels in the original and the restored images, is not a good measure in this application. This is because the measure does not only tend to zero as the restored image tends to the original image.

Clearly, in each table in section 10.2, the measures (number of misclassified pixels), MSE, (MSE/Variance(original)) and MEM agree

on the most successful of the segmentation techniques used to restore the degraded image given. This is because the the number of pixels in the images, the variance of the pixel values in the original image, and the number of pixels in the smaller of the two regions in the original image, remained constant for each table (i.e. each original image). The problem now is determining which of these measures is the most appropriate in this application.

Clearly, the number of misclassified pixels is not the most appropriate of these measures as it is essentially the same as MSE, except that it is dependent on image size.

MSE is considered less appropriate than the other two measures, MEM and (MSE/Variance(original)), because it does not satisfy condition 3(iii) of section 10.1 and they do.

Of these two measures, MEM is preferred when comparing binary images, the case dealt with in this chapter. The reasons are as given in section 10.1. However, in the more general case of comparing two images which are not necessarily binary, (MSE/Variance(original)) may well be more appropriate.

Using MEM as the error measure, we see that the tessellation techniques perform well in recovering the binary images used, and that Method 3 ($B_{0.5}$, planar) performs particularly well, being the "best" method twice and the "second best" the other 4 times. It also had a Maximum Error of less than 10% in 4 of the 6 sets of results. The technique is not perfect, but it appears to give consistently good results.

Moving window median thresholding, with the size of the window being the same as the average size of the tiles used for the tessellation techniques, had mixed success. It was the best technique twice, but it was also the worst technique twice. While it achieved a Maximum Error of less than 2% twice, it also achieved a Maximum error of more than 95% twice. Not unexpectedly, moving window median thresholding performed very well on the two images containing equal numbers of foreground and background pixels, but performed very poorly on the images dominated by the background.

Global thresholding did not fare well, with a Maximum Error of over 50% for 5 of the 6 sets of experiments.

The tessellation techniques generally performed slightly better when using planar interpolation than when using cubic interpolation with the derivatives of the threshold surface approximated at the vertices of the triangles.

It is concluded that the tessellation techniques of image segmentation show great promise, and that the use of Bumpiness measure $B_{0.5}$ as the clustering criterion and planar interpolation of the threshold surface, seems to produce good results quite consistently.

The investigation of other tessellations, clustering criteria, optimisation techniques, threshold evaluation techniques and interpolation techniques (all of which are very important) for this application, seems justified and shows promise of producing even better results.

CHAPTER 11

CONCLUSION

11.1 Some General Comments

The aim of this research was to investigate segmentation techniques for images which have poor contrast and may be illuminated unevenly.

Chapter 4 reviewed some of the current approaches to image segmentation and showed that none of the techniques currently in use was appropriate. It was concluded that more research should be done into optimisation-partitioning techniques as these were little understood, but potentially very useful. The nature of the images to be processed suggested the use of thresholding, but the possible uneven illumination of the images made it difficult to determine for what regions of the image the threshold values should be computed.

Chapter 5 suggested the use of the Dirichlet Tessellation to divide the image into regions for which threshold values could be computed. Although the Dirichlet Tessellation is not easy to compute directly, it can be derived from its dual, the Delaunay Triangulation. Because a given set of generating points defines a unique Dirichlet Tessellation, this is a particularly convenient tessellation for this application.

Chapter 6 introduced several measures for evaluating the suitability of individual tiles for our applications. It also showed how these measures could be combined to give a goodness of

tessellation measure, indicating how good a particular tessellation of a given image was for our purposes.

The next step was to position the generating points so that the goodness of tessellation measure was maximised for a given image. Chapter 7 showed that none of the commonly used optimisation techniques could maximise the problem at hand. A new approach, based on simulated annealing, was implemented. Appendix 2 contains the listing of a Fortran program that was used in this research to implement the optimisation method proposed and described in chapter 7.

Chapter 8 showed that Otsu's threshold evaluation technique was suitable for our purposes, and extended it to yield multiple threshold values. It went on to show how threshold surfaces could be interpolated through the threshold values computed for the generating points, yielding one or more non-intersecting, continuous threshold surfaces for the whole image. Appendices 3 and 4 contain the listings of Pascal program segments that were used to implement two of the interpolation techniques proposed in chapter 8.

Chapter 9 compared the results of applying the technique developed during the course of this research with those of other techniques, subjectively.

Chapter 10 contained a more objective evaluation of the performance of the technique developed during the course of this research.

11.2 Contribution of this Research

The whole approach to image segmentation proposed here is original.

The major contributions are as follows:

(1)

(i) The use of the Dirichlet Tessellation to divide an image into regions for which threshold values can be computed

(ii) The discussion of the properties that make the Dirichlet Tessellation suitable to the application

(iii) The discussion of the implementation details of the technique

(iv) The bumpiness measures described in chapter 6

(v) The optimisation technique described in chapter 7

(vi) The single-linkage clustering algorithm given in (3) of section 4.3.2.

While it is not claimed that the optimisation technique developed here is applicable to a large class of problems, it is well suited to a class of similar optimisation problems. Without it, the

implementation of the segmentation technique developed here would not have been possible.

11.3 Conclusion

As stated in section 1.3, the aim of this research was to develop a segmentation technique that can segment images with the following characteristics:

(i) The images may have low contrast

(ii) The images may be illuminated unevenly

(iii) The images may contain either large or small objects, or both, of any shape, without these details needing to be known or specified a priori

(iv) The image may contain any number of objects

(v) The regions occupied by the individual objects in the image need not be contiguous

(vi) Objects may not have some areas that are lighter and others that are darker than the surrounding background

(vii) The property distinguishing the objects from their backgrounds is pixel intensity rather than some other property, such as texture

(1)

(viii) Neither the objects of interest nor their backgrounds are highly textured.

The technique proposed in chapter 5 and described in chapters 6 to 8 meets these specifications.

It is not claimed that the technique developed and implemented is an optimal method of segmentation. It is the general approach to image segmentation proposed, through partition and optimisation, that is seen as being the most significant contribution of this research.

It is suggested that the method shows sufficient promise for further work to be done on the optimisation technique and measures of goodness of tessellation, for various classes of segmentation.

APPENDIX A

Following is the listing of the source code of a Fortran-77 program that was used during the course of this research to perform single-linkage (nearest-neighbour) clustering on the pixels of a given image. It does 8-neighbourhood clustering using the algorithm developed during the course of this research and which is described in (3) of section 4.3.2 of this thesis.

The program reads from devices 5 and 11 and writes to devices 6 and 12, as follows:

5 - Reads NREGNS, the number of regions into which the image's pixels are to be clustered, in subroutine INIT.

6 - Writes the value of NREGNS, the number of regions that the image is to be segmented into, in subroutine INIT.

- Writes the maximum stack depth used in the quick-sort in subroutine SORT.

- Prints the connections that would be made, in order, if the clustering process was to continue until only one region remained, in subroutine TIDYUP.

11 - Reads the image from this unit, in row major order, 32 pixels per line, in hexidecimal (2 digits per pixel), in subroutine READIN.

12 - Writes out the values of the pixels in the clustered image in subroutine FINISH.

- Prints the connections that would be made, in order, if the clustering process was to continue until only one region remained, in subroutine TIDYUP.

The program is written in standard Fortran-77 for computers with 64 bit words and which use the ASCII character set. While it should run on any 64 bit, ASCII machine with a Fortran-77 compiler, it is written to run efficiently on a Cray-1S.

```
      CALL INIT
      CALL PROCES
C
      STOP
      END
C
C --- ************************************************************
C
C     This subroutine:
C
C         - reads in NREGNS, the number of regions that the
C             image's pixels are to be clustered into
C         - calls READIN to read in the image
C         - initialises TREE, the variable that stores the numbers
C             of the pixels in each cluster.
C
C --- ************************************************************
C
      SUBROUTINE INIT
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      COMMON  /ISIZE/NREGNS
C
      READ(5,*) NREGNS
      WRITE(6,1000) NREGNS
C
      CALL READIN
C
      DO 10 I=0,65 535
10      TREE(I) = -1
C
      RETURN
1000  FORMAT (1X,'Partitioning image into ',I3,' regions.')
      END
C
C --- ************************************************************
C     This subroutine reads in the image from device 11. The image
C     is read in in row major order, 32 pixels per line, with each
C     pixel being represented by 2 hexadecimal digits.
C --- ************************************************************
C
      SUBROUTINE READIN
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      CHARACTER    DUMMY,DAT(0:63)
      INTEGER      DEC (0:127)
      INTEGER      OFFSET
C
      DO 5 I=48,57
5       DEC(I) = I - 48
C
```

```
      DO 6 I=65,70
6       DEC(I) = I - 55
C
      DO 30 I=0,255
        OFFSET=0
        DO 20 J=1,8
          READ(11,1000) DUMMY,DAT
          DO 10 K=0,31
10          IMAGE(I,OFFSET+K) = 16 * DEC(ICHAR(DAT(2*K))) +
     +                                DEC(ICHAR(DAT(2*K+1)))
20        OFFSET=OFFSET+32
30      CONTINUE
C
      RETURN
1000  FORMAT (65A)
      END
C
C --- ************************************************************
C     This is the main driver subroutine. This is where all the
C     action is !!
C --- ************************************************************
C
      SUBROUTINE PROCES
C
      CALL ALLEDG
      CALL SORT
      CALL WORK
      CALL FINISH
      CALL TIDYUP
C
      RETURN
      END
C
C --- ************************************************************
C
C     This subroutine calculates all the possible edges in the
C     least spanning forest and stores them in EDGES. All that
C     remains is to calculate which of them are the edges wanted.
C
C     Each value in edges represents the numbers of the pixels the
C     edge runs between and the length of the edge itself. This is
C     done by setting the value equal to 4 294 967 296 times the
C     length of the edge, plus 65 536 times the number of one of
C     the pixels, plus the number of the other pixel that the edge
C     runs between. This means that the edges can be sorted into
C     order of increasing length by sorting the values in EDGES.
C
C --- ************************************************************
C
      SUBROUTINE ALLEDG
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      INTEGER CNT
```

```
C
      CNT=0
C
      DO 10 I=0,255
        DO 10 J=0,254
          CNT=CNT + 1
          EDGES(CNT)=4 294 967 296*ABS(IMAGE(I,J)-IMAGE(I,J+1)) +
     +                      65 536*    (256*I+J)                  +
     +                                 (256*I+J+1)
10        CONTINUE
C
      DO 20 J=0,255
        DO 20 I=0,254
          CNT=CNT + 1
          EDGES(CNT)=4 294 967 296*ABS(IMAGE(I,J)-IMAGE(I+1,J)) +
     +                      65 536*    (256*I+J)                  +
     +                                 (256*(I+1)+J)
20        CONTINUE
C
      DO 30 I=0,254
        DO 30 J=0,254
          CNT=CNT + 1
          EDGES(CNT)=4 294 967 296*ABS(IMAGE(I,J)-IMAGE(I+1,J+1))+
     +                      65 536*    (256*I+J)                   +
     +                                 (256*(I+1)+J+1)
30        CONTINUE
C
      DO 40 I=0,254
        DO 40 J=0,254
          CNT=CNT + 1
          EDGES(CNT)=4 294 967 296*ABS(IMAGE(I,J+1)-IMAGE(I+1,J))+
     +                      65 536*    (256*I+J+1)                 +
     +                                 (256*(I+1)+J)
40        CONTINUE
C
      RETURN
      END
C
C --- ************************************************************
C     This subroutine sorts the values in EDGES into ascending
C     order, using a quick-sort, thus sorting the edges into
C     order of increasing length.
C --- ************************************************************
C
      SUBROUTINE SORT
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      INTEGER STACK(5000,2),P,TOP,BOT,T,B
      LOGICAL UP
C
      P          = 1
      MAXP       = 1
      STACK(1,1) = 1
```

```
      STACK(1,2) = 260 610
C
10    TOP = STACK(P,1)
      BOT = STACK(P,2)
      P   = P - 1
15    T   = TOP
      B   = BOT
      UP  = .FALSE.
C
20    IF (T .EQ. B) GOTO 50
C
      IF (EDGES(T) .GT. EDGES(B)) THEN
        ITEMP    = EDGES(T)
        EDGES(T) = EDGES(B)
        EDGES(B) = ITEMP
        UP       = .NOT. UP
      ENDIF
C
      IF (UP) THEN
        B=B-1
      ELSE
        T=T+1
      ENDIF
      GOTO 20
C
C
50    IF (TOP .LT. (T-1)) THEN
        P=P+1
        STACK(P,1) = TOP
        STACK(P,2) = T-1
      ENDIF
      IF (BOT .GT. (B+1)) THEN
        TOP = B+1
        GOTO 15
      ENDIF
      IF (P.GT.MAXP) MAXP=P
C
      IF (P .GT. 0) GOTO 10
C
      WRITE(6,1000) MAXP
      RETURN
1000  FORMAT (1X,'Maximum stack depth in Sort was ',I4)
      END
C
C --- ************************************************************
C     This subroutine calculates which of the edges in EDGES
C     forms the minimum spanning forest of order NREGNS.
C --- ************************************************************
C
      SUBROUTINE WORK
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      COMMON  /ISIZE/NREGNS
```

```
C
      INTEGER DONE,P,SOURCE,SINK,SMALL
      COMMON  /WDATA/DONE,P
C
      DONE = 0
      P    = 0
C
10    P = P+1
      SOURCE = SMALL(MOD((EDGES(P)/65536),65536))
      SINK   = SMALL(MOD(EDGES(P),65536))
      IF (SOURCE.NE.SINK) THEN
        DONE = DONE + 1
        IF (SOURCE.LT.SINK) THEN
          TREE(SINK) = SOURCE
        ELSE
          TREE(SOURCE) = SINK
        ENDIF
      ENDIF
C
      IF (DONE.NE.(65536-NREGNS)) GOTO 10
C
      DO 20 I=0,255
        DO 20 J=0,255
20        IMAGE(I,J) = SMALL(256*I+J)
C
      RETURN
      END
C
C --- ************************************************************
C     This function returns the smallest numbered pixel in the
C     region containing the pixel numbered I.
C --- ************************************************************
C
      INTEGER FUNCTION SMALL (I)
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
C
10    IF (TREE(I).EQ.-1) THEN
        SMALL = I
        RETURN
      ELSE
        I = TREE(I)
        GOTO 10
      ENDIF
C
      END
C
C --- ************************************************************
C     This subroutine outputs the results to device 12.
C --- ************************************************************
C
      SUBROUTINE FINISH
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
```

```
      COMMON  /INFO/IMAGE,EDGES,TREE
C
      COMMON  /ISIZE/NREGNS
C
      INTEGER   MAP (0:255),P
      CHARACTER LINE(0:511)
C
      MAP(0) = 0
      P      = 0
      DO 10 I=0,255
        DO 10 J=0,255
          IF (IMAGE(I,J).GT.MAP(P)) THEN
            P      = P+1
            MAP(P) = IMAGE(I,J)
          ENDIF
10      CONTINUE
C
      DO 30 K=1,P
        DO 20 J=0,255
          DO 20 I=0,255
            IF (IMAGE(I,J).EQ.MAP(K)) IMAGE(I,J) = K
20          CONTINUE
30      CONTINUE
C
      DO 50 I=0,255
        DO 40 J=0,255
          K1 = IMAGE(I,J) / 16
          K2 = MOD(IMAGE(I,J),16)
          IF (K1.LE.9) THEN
            LINE(2*J) = CHAR(K1+48)
          ELSE
            LINE(2*J) = CHAR(K1+55)
          ENDIF
          IF (K2.LE.9) THEN
            LINE(2*J+1) = CHAR(K2+48)
          ELSE
            LINE(2*J+1) = CHAR(K2+55)
          ENDIF
40        CONTINUE
        WRITE(12,1000) LINE
50      CONTINUE
C
      RETURN
1000  FORMAT (1X,64A)
      END
C
C --- ************************************************************
C     This subroutine gives the rest of the links in the
C     completely linked image, writing them to devices 6 and 12.
C --- ************************************************************
C
      SUBROUTINE TIDYUP
C
      INTEGER IMAGE(0:255,0:255),EDGES(260 610),TREE(0:65 535)
      COMMON  /INFO/IMAGE,EDGES,TREE
```

```
C
      COMMON  /ISIZE/NREGNS
C
      INTEGER DONE,P,SOURCE,SINK,SMALL
      COMMON  /WDATA/DONE,P
C
10    P = P+1
      SOURCE = SMALL(MOD((EDGES(P)/65536),65536))
      SINK   = SMALL(MOD(EDGES(P),65536))
      IF (SOURCE.NE.SINK) THEN
        DONE = DONE + 1
        I1 = SOURCE / 256
        I2 = MOD(SOURCE,256)
        J1 = SINK / 256
        J2 = MOD(SINK,256)
        WRITE(6, 1000) I1,I2,J1,J2
        WRITE(12,2000) I1,I2,J1,J2
        IF (SOURCE.LT.SINK) THEN
          TREE(SINK) = SOURCE
        ELSE
          TREE(SOURCE) = SINK
        ENDIF
      ENDIF
C
      IF (DONE.NE.65 535) GOTO 10
C
      RETURN
1000  FORMAT (1X,'Connect (',I3,',',I3,') to (',I3,',',I3,').')
2000  FORMAT (1X,4I4)
      END
```

APPENDIX B

Following is the listing of the source code of a Fortran-77 program that was used during the course of this research to optimise the Dirichlet Tessellation of a given image, using the technique described in chapter 7. It provides the user with a choice of four clustering criteria, and also gives the option of reading in the the initial positions of the generating points or arranging them in a square grid (except for the four that are positioned in the corners of the image).

The program reads from devices 5,11 and 12 and writes to devices 6, 21 and 22, as follows:

5 - The program reads in two integers, in free format, at the beginning of execution, in subroutine INIT. The first value may be 0 or 1 (0 meaning that the generating points should be arranged in a square grid formation initially, and 1 meaning that the initial positions of the generating points is to be read from device 12), and the second may be 0, 1, 2 or 3, indicating the clustering criterion to be used (bumpiness B_0, bumpiness $B_{0.5}$, entropy and variance of the histogram, respectively).

6 - During execution, in subroutine PROCES, the program prints the maximum jump size for the optimisation at each iteration, as well as printing the current goodness of tessellation measure, on this device.

11 - The program reads in the image, in subroutine READIN, in row major order, 32 pixels per row, in hexadecimal form, from this device.

12 - If the second parameter read in from device 5 was 1, the program reads in the initial positions of the generating points from this device, in subroutine INIT.

21 - At the end of the optimisation process, in subroutine ENDOFF, the program writes the final, "optimal" positions of the generating points to this device.

22 - At the end of the optimisation process, in subroutine ENDOFF, the program writes the number of jumps of each size made during the execution of the optimisation process, to this device.

The program is written in standard Fortran-77 for computers with 64 bit words and which use the ASCII character set. While it should run on any 64 bit, ASCII machine with a Fortran-77 compiler, it is written to run efficiently on a CRAY-1S.

```
      CALL INIT
      CALL PROCES
      CALL ENDOFF
C
      STOP
      END
C
C --- ******************************************************
C     This subroutine reads in the image from device 11, in
C     row major order, in hexadecimal, 2 digits per pixels,
C     32 pixels per row of input, into the array IMAGE.
C --- ******************************************************
C
      SUBROUTINE READIN (IMAGE)
C
      CHARACTER    DUMMY,DAT(0:63)
      INTEGER      IMAGE(0:255,0:255)
      INTEGER      OFFSET,DEC(0:127)
C
      DO 10 I=48,57
   10   DEC(I) = I - 48
C
      DO 20 I=65,70
   20   DEC(I) = I - 55
C
      DO 50 I=0,255
        OFFSET=0
        DO 40 J=1,8
          READ(11,1000) DUMMY,DAT
          DO 30 K=0,31
   30       IMAGE(I,OFFSET+K) = 16 * DEC (ICHAR (DAT(2*K  ))) +
     +                                DEC (ICHAR (DAT(2*K+1)))
   40     OFFSET=OFFSET+32
   50   CONTINUE
C
      RETURN
 1000 FORMAT (65A)
      END
C
C --- ******************************************************
C     This function returns the bumpiness measure, B0, of
C     the histogram COUNT.
C --- ******************************************************
C
      FUNCTION BUMP ( )
C
      INTEGER COUNT (0:255),N
      COMMON  /CDATA/ COUNT,N
C
      REAL TEMP1(0:255),TEMP2(0:255)
C
      DO 10 I=0,255
   10   TEMP1(I) = FLOAT(COUNT(I))/FLOAT(N) - (1.0/256.0)
      DO 20 I=0,255
   20   TEMP2(I) = MAX(TEMP1(I),0.0)
```

```
C
      R = 0.0
      DO 50 I=0,255
        IF (TEMP2(I).GT.0.0) THEN
          DO 30 J=I,255
   30       TEMP1(J) = TEMP2(I)*TEMP2(J) / FLOAT(J-I+1)
          DO 40 J=I,255
   40       R = R+TEMP1(J)
        ENDIF
   50   CONTINUE
C
      BUMP = R
      RETURN
      END
C
C --- ******************************************************
C     This function returns the bumpiness measure, B0.5, of
C     the histogram COUNT.
C --- ******************************************************
C
      FUNCTION BUMP2 ( )
C
      INTEGER COUNT (0:255),N
      COMMON  /CDATA/ COUNT,N
C
      REAL TEMP1(0:255),TEMP2(0:255)
C
      DO 10 I=0,255
   10   TEMP1(I) = FLOAT(COUNT(I))/FLOAT(N) - (1.0/256.0)
      DO 20 I=0,255
   20   TEMP2(I) = MAX(TEMP1(I),-0.5*TEMP1(I))
C
      R = 0.0
      DO 50 I=0,255
        DO 30 J=I,255
   30     TEMP1(J) = TEMP2(I)*TEMP2(J) / FLOAT(J-I+1)
        DO 40 J=I,255
   40     R = R+TEMP1(J)
   50   CONTINUE
C
      BUMP2 = R
      RETURN
      END
C
C --- ******************************************************
C     This function returns the entropy of the histogram
C     COUNT.
C --- ******************************************************
C
      FUNCTION ENTPY ( )
C
      INTEGER COUNT (0:255),N
      COMMON  /CDATA/ COUNT,N
C
      REAL TEMP (0:255)
```

```
C
      DO 10 I=0,255
   10   TEMP(I) = FLOAT(COUNT(I)) / FLOAT(N)
C
      R = 0
      DO 20 I=0,255
        IF (TEMP(I).NE.0) THEN
          R = R + TEMP(I) * LOG(TEMP(I))
        ENDIF
   20   CONTINUE
C
      ENTPY = -R
      RETURN
      END
C
C --- ******************************************************
C     This function returns the variance of the histogram
C     COUNT.
C --- ******************************************************
C
      FUNCTION VARNCE ( )
C
      INTEGER COUNT (0:255),N
      COMMON  /CDATA/ COUNT,N
C
      REAL TEMP1(0:255)
C
      DO 10 I=0,255
   10   TEMP1(I) = FLOAT(COUNT(I))/FLOAT(N)

      R = 0.0
      DO 20 I=0,255
   20   R = R + TEMP1(I)*TEMP1(I)
C
      VARNCE = R
      RETURN
      END
C
C --- ******************************************************
C     This function generates a random number such that
C     0.0 <= RND < 1.0 .
C --- ******************************************************
C
      FUNCTION RND ( )
C
      INTEGER X,Y,Z
      COMMON  /RNDVAR/X,Y,Z
C
      X=171*MOD(X,177) - 2 *(X/177)
      Y=172*MOD(Y,176) - 35*(Y/176)
      Z=170*MOD(Z,178) - 63*(Z/178)
C
      IF (X.LT.0) X=X+30269
      IF (Y.LT.0) Y=Y+30307
      IF (Z.LT.0) Z=Z+30323
```

```
C
      RND=MOD(REAL(X)/30269.0+REAL(Y)/30307.0+REAL(Z)/30323.0,1.0)
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine moves the generating point N a random
C     distance not greater than STEP in the X and Y axis
C     directions, but ensures that the generating point still
C     lies in the image space and that no two generating
C     points have the same co-ordinate.
C --- ********************************************************
C
      SUBROUTINE MOVPNT (N)
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER JUMPS(-255:255)
      COMMON  /JDATA/JUMPS
C
      INTEGER XOFF,YOFF,X,Y,TX,TY
C
   10 N   =INT(RND()*REAL(NPOINT-4)) + 3
      XOFF=INT(RND()*REAL(2*JUMP+1)) - JUMP
      YOFF=INT(RND()*REAL(2*JUMP+1)) - JUMP
      JUMPS(XOFF) = JUMPS(XOFF) + 1
      JUMPS(YOFF) = JUMPS(YOFF) + 1
C
      X=POINTS(N,1)
      Y=POINTS(N,2)
C
      IF (X+XOFF.GT.255) THEN
        TX=X + NINT(REAL(XOFF*(255-X))/REAL(JUMP))
      ELSE IF (X+XOFF.LT.0) THEN
        TX=X + NINT(REAL(XOFF*X)/REAL(JUMP))
      ELSE
        TX=X + XOFF
      ENDIF
C
      IF (Y+YOFF.GT.255) THEN
        TY=Y + NINT(REAL(YOFF*(255-Y))/REAL(JUMP))
      ELSE IF (Y+YOFF.LT.0) THEN
        TY=Y + NINT(REAL(YOFF*Y)/REAL(JUMP))
      ELSE
        TY=Y + YOFF
      ENDIF
C
      DO 20 I=1,NPOINT
        IF (POINTS(I,2).GT.TY) THEN
          GOTO 30
```

```
        ELSE IF ((POINTS(I,1).EQ.TX).AND.(POINTS(I,2).EQ.TY)) THEN
          GOTO 10
        ENDIF
   20 CONTINUE
C
   30 POINTS(N,1)=TX
      POINTS(N,2)=TY
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine sorts the generating points into order
C     of increasing Y value, again. It utilises the
C     information that only generating point N may be out of
C     order.
C --- ********************************************************
C
      SUBROUTINE SORT (N,INDEX)
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER LIST(64),INDEX(64),X,Y
C
      DO 10 I=1,NPOINT
   10   LIST(I)=I
C
      NEWN=N
      X=POINTS(N,1)
      Y=POINTS(N,2)
      DO 20 I=(N+1),NPOINT
        IF (Y.GT.POINTS(I,2)) NEWN=I
   20   CONTINUE
C
      IF (NEWN.EQ.N) THEN
        DO 30 I=(N-1),1,-1
          IF (Y.LT.POINTS(I,2)) NEWN=I
   30     CONTINUE
      ENDIF
C
      IF (N.NE.NEWN) THEN
        IF (NEWN.LT.N) THEN
          DO 40 J=1,2
            DO 40 I=(N-1),NEWN,-1
   40         POINTS(I+1,J)=POINTS(I,J)
          DO 45 I=(N-1),NEWN,-1
   45       MESURE(I+1)  =MESURE(I)
          DO 46 I=(N-1),NEWN,-1
   46       LIST  (I+1)  =I
          DO 50 I=(N-1),NEWN,-1
            DO 50 J=0,NEYBOR(I,0)
```

```
   50          NEYBOR(I+1,J)=NEYBOR(I,J)
            ELSE
              DO 60 J=1,2
                DO 60 I=N,(NEWN-1)
   60             POINTS(I,J)=POINTS(I+1,J)
              DO 65 I=N,(NEWN-1)
   65           MESURE(I)  =MESURE(I+1)
              DO 66 I=N,(NEWN-1)
   66           LIST  (I)  =I+1
              DO 70 I=N,(NEWN-1)
                DO 70 J=0,NEYBOR(I+1,0)
   70             NEYBOR(I,J)=NEYBOR(I+1,J)
            ENDIF
            POINTS(NEWN,1)=X
            POINTS(NEWN,2)=Y
            LIST  (NEWN)  =N
          ENDIF
C
          DO 80 I=1,NPOINT
   80     INDEX(LIST(I))=I
C
          RETURN
          END
C
C --- ********************************************************
C     This function indicates whether the queue is empty or
C     not. It is part of the triangulation software.
C --- ********************************************************
C
          LOGICAL FUNCTION EMPTY ( )
C
          INTEGER QUEUE(0:9,2),HEAD,TAIL
          COMMON  /QINFO/QUEUE,HEAD,TAIL
C
          EMPTY=HEAD .EQ. TAIL
          RETURN
          END
C
C --- ********************************************************
C     This subroutine gets the head of the queue.
C --- ********************************************************
C
          SUBROUTINE GETHD (SOURCE,SINK)
C
          INTEGER SOURCE,SINK
C
          INTEGER QUEUE(0:9,2),HEAD,TAIL
          COMMON  /QINFO/QUEUE,HEAD,TAIL
C
          LOGICAL EMPTY
C
          IF (EMPTY ()) THEN
            PRINT*,'Queue underflow.'
            STOP 7777
          ELSE
```

```
        SOURCE=QUEUE(HEAD,1)
        SINK  =QUEUE(HEAD,2)
        HEAD  =MOD(HEAD+1,10)
      ENDIF
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine adds the new source and sink to the
C     tail of the queue.
C --- ********************************************************
C
      SUBROUTINE ADDTL (SOURCE,SINK)
C
      INTEGER SOURCE,SINK
C
      INTEGER QUEUE(0:9,2),HEAD,TAIL
      COMMON  /QINFO/QUEUE,HEAD,TAIL
C
      LOGICAL EMPTY
C
      QUEUE(TAIL,1)=SOURCE
      QUEUE(TAIL,2)=SINK
      TAIL=MOD(TAIL+1,10)
      IF (EMPTY ()) THEN
        PRINT*,'Queue overflow.'
        STOP 7777
      ENDIF
C
      RETURN
      END
C
C --- ********************************************************
C     This function sees if it can remove a redundant entry
C     from the queue and if it can, it does.
C --- ********************************************************
C
      LOGICAL FUNCTION CANKIL (SOURCE,SINK)
C
      INTEGER SOURCE,SINK
C
      INTEGER QUEUE(0:9,2),HEAD,TAIL
      COMMON  /QINFO/QUEUE,HEAD,TAIL
C
      LOGICAL FOUND
C
      FOUND=.FALSE.
      I=HEAD
      IF (I.EQ.TAIL) GOTO 20
   10   FOUND=FOUND .OR. ((SOURCE.EQ.QUEUE(I,1))
     +                .AND. (SINK  .EQ.QUEUE(I,2)))
        NEXT=MOD(I+1,10)
        IF (FOUND) THEN
          DO 15 J=1,2
```

```
   15        QUEUE(I,J)=QUEUE(NEXT,J)
          ENDIF
          I=NEXT
          IF (I.NE.TAIL) GOTO 10
C
   20  IF (FOUND) TAIL=MOD(TAIL+9,10)
       CANKIL=FOUND
C
       RETURN
       END
C
C --- ********************************************************
C     This function gives the index of the next vertex to
C     be processed.
C --- ********************************************************
C
       INTEGER FUNCTION PINPON ( )
C
       PARAMETER (MAXNEY=64)
       INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
       REAL    MESURE(64)
       INTEGER NPOINT,JUMP
       COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
       INTEGER OFFSET,ORGN,POSX,NEGX,POSY,NEGY
       COMMON  /SEARCH/OFFSET,ORGN,POSX,NEGX,POSY,NEGY
C
       INTEGER TEST,X,Y,SIDE
       LOGICAL FINISH
C
   10  TEST=0
       DO 20 SIDE=1,2
         IF (TEST.EQ.0) THEN
           IF (OFFSET.GT.0) THEN
             OFFSET=-OFFSET
             TEST  =ORGN+OFFSET
             IF (TEST.GE.1) THEN
               X=POINTS(TEST,1)
               Y=POINTS(TEST,2)
               IF (Y.LT.NEGY) TEST=0
             ELSE
               TEST=0
             ENDIF
           ELSE
             OFFSET=1-OFFSET
             TEST  =ORGN+OFFSET
             IF (TEST.LE.NPOINT) THEN
               X=POINTS(TEST,1)
               Y=POINTS(TEST,2)
               IF (Y.GT.POSY) TEST=0
             ELSE
               TEST=0
             ENDIF
           ENDIF
         ENDIF
```

```
   20   CONTINUE
C
      IF (TEST.EQ.0) THEN
        FINISH=.TRUE.
      ELSE
        FINISH=(X.LE.POSX) .AND. (X.GE.NEGX)
      ENDIF
C
      IF (.NOT.FINISH) GOTO 10
      PINPON=TEST
C
      RETURN
  100 FORMAT (1X,'Offset = ',I4)
  200 FORMAT (1X,'Origin = ',I4)
  300 FORMAT (1X,'PosX   = ',I4)
  400 FORMAT (1X,'NegX   = ',I4)
  500 FORMAT (1X,'PosY   = ',I4)
  600 FORMAT (1X,'NegY   = ',I4)
      END
C
C --- ********************************************************
C     This function finds the nearest vertex to ORGN.
C --- ********************************************************
C
      FUNCTION NEARST ( )
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER OFFSET,ORGN,POSX,NEGX,POSY,NEGY
      COMMON  /SEARCH/OFFSET,ORGN,POSX,NEGX,POSY,NEGY
C
      INTEGER CLOSE,POINT,LIMIT,PINPON
      REAL    SQDIST,SQLIM
C
      DIST2 (A,B) = A*A + B*B
C
      OFFSET=0
      CLOSE=0
C
      SQLIM=1E50
      LIMIT=1 000 000
C
      NEGX=0
      NEGY=0
      POSX=255
      POSY=255
C
   10 POINT=PINPON ()
      IF (POINT.NE.0) THEN
        SQDIST=DIST2(FLOAT(POINTS(ORGN,1) - POINTS(POINT,1)),
     +               FLOAT(POINTS(ORGN,2) - POINTS(POINT,2)))
```

```
        IF (SQDIST.GE.SQLIM) GOTO 10
C
        SQLIM=SQDIST
        LIMIT=INT(SQRT(SQLIM)) + 1
        POSX=MIN(255,POINTS(ORGN,1)+LIMIT)
        NEGX=MAX(0,  POINTS(ORGN,1)-LIMIT)
        POSY=MIN(255,POINTS(ORGN,2)+LIMIT)
        NEGY=MAX(0,  POINTS(ORGN,2)-LIMIT)
        CLOSE=POINT
        GOTO 10
      ENDIF
C
      NEARST=CLOSE
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine forms triangles. It tries to find the
C     vertex VERTEX that will complete the optimal, minimal
C     surround triangle on SOURCE and SINK. CLKWSE indicates the
C     direction of the triangulation, and FOUND will inform the
C     calling routine whether a VERTEX was found or not.
C
C     This subroutine uses the fact that the radius of the
C     circumcircle going through the vertices of the triangle
C     (A,B,C) is given as follows :
C                        Radius = A / (2.0 * Sin(A))
C     and the circumcentre is given by :
C                        XCircum = XMid - YHalf*CosA/SinA
C                        YCircum = Ymid + XHalf*CosA/SinA.
C --- ********************************************************
C
      SUBROUTINE TRIANG (SOURCE,SINK,VERTEX,FOUND,CLKWSE)
C
      INTEGER SOURCE,SINK,VERTEX
      LOGICAL FOUND,CLKWSE
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER OFFSET,ORGN,POSX,NEGX,POSY,NEGY
      COMMON  /SEARCH/OFFSET,ORGN,POSX,NEGX,POSY,NEGY
C
      INTEGER X1,X2,X3,Y1,Y2,Y3,INDEX,RADIUS,XCIRC,YCIRC,
     +        SQRA,SQRB,SQRC,PINPON
      REAL    XMID,YMID,XHALF,YHALF,MINCSA,COSATS,COSA,SINA,
     +        COSB,RATIO,DELTA,TEMP,COSBLS
C
      IDIST2 (I,J) = I*I + J*J
C
      X1 = POINTS(SOURCE,1)
```

```
      X2 = POINTS(SINK,   1)
      Y1 = POINTS(SOURCE,2)
      Y2 = POINTS(SINK,   2)
C
      XMID  = FLOAT(X1+X2) / 2.0
      YMID  = FLOAT(Y1+Y2) / 2.0
      XHALF = FLOAT(X2-X1) / 2.0
      YHALF = FLOAT(Y2-Y1) / 2.0
C
      POSX = 255
      POSY = 255
      NEGX = 0
      NEGY = 0
C
      DELTA  = 2.0E-6
      MINCSA = 2.0
      COSBLS = 2.0
      COSATS = MINCSA + DELTA
      SQRA   = IDIST2((X1-X2),(Y1-Y2))
      OFFSET = 0
      ORGN   = (SOURCE+SINK+1) / 2
      INDEX  = ORGN
      FOUND  = .FALSE.
C
   10 IF ((INDEX.NE.SOURCE).AND.(INDEX.NE.SINK)) THEN
        X3 = POINTS(INDEX,1)
        Y3 = POINTS(INDEX,2)
        IF (Y2*(X3-X1)+Y1*(X2-X3)+Y3*(X1-X2).LT.0) THEN
          SQRB = IDIST2((X3-X2),(Y3-Y2))
          SQRC = IDIST2((X3-X1),(Y3-Y1))
          COSA = FLOAT(SQRB+SQRC-SQRA) /
     +          (2.0 * SQRT(FLOAT(SQRB)*FLOAT(SQRC)))
          IF (COSA.LT.COSATS) THEN
            COSB = FLOAT(SQRA+SQRC-SQRB) /
     +                (2.0 * SQRT(FLOAT(SQRA)*FLOAT(SQRC)))
            IF (ABS(COSA-MINCSA).GT.DELTA) THEN
              COSBLS = COSB
              VERTEX = INDEX
              MINCSA = COSA
              COSATS = COSA + DELTA
              FOUND  = .TRUE.
              SINA   = SQRT(1.0 - COSA*COSA)
C
              IF (SINA.LT.1.0E-4) THEN
                RADIUS = 255
              ELSE
                RADIUS =
     +            INT(MIN(255.0,(1.0 + SQRT(FLOAT(SQRA))/(2.0*SINA))))
              ENDIF
C
              IF (SINA.LT.1.0E-20) THEN
                RATIO=255.0
              ELSE
                RATIO=COSA/SINA
              ENDIF
```

```
C
              XCIRC = NINT(MAX(0.0,MIN(255.0,(XMID-YHALF*RATIO))))
              YCIRC = NINT(MAX(0.0,MIN(255.0,(YMID+XHALF*RATIO))))
C
              POSX = MIN(255,XCIRC+RADIUS)
              POSY = MIN(255,YCIRC+RADIUS)
              NEGX = MAX(0,  XCIRC-RADIUS)
              NEGY = MAX(0,  YCIRC-RADIUS)
            ELSE
              IF (FOUND) THEN
                IF (CLKWSE) THEN
                  IF (COSB.LT.COSBLS) THEN
                    COSBLS = COSB
                    VERTEX = INDEX
                  ENDIF
                ELSE
                  IF (COSB.GT.COSBLS) THEN
                    COSBLS = COSB
                    VERTEX = INDEX
                  ENDIF
                ENDIF
              ELSE
                FOUND  = .TRUE.
                VERTEX = INDEX
                COSBLS = COSB
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      INDEX = PINPON ()
C
      IF (INDEX.NE.0) GOTO 10
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine calculates the surround of generating
C     point CENTRE.
C --- ********************************************************
C
      SUBROUTINE SURRND (CENTRE)
C
      INTEGER CENTRE
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER QUEUE(0:9,2),HEAD,TAIL
      COMMON  /QINFO/QUEUE,HEAD,TAIL
C
```

```
      INTEGER OFFSET,ORGN,POSX,NEGX,POSY,NEGY
      COMMON  /SEARCH/OFFSET,ORGN,POSX,NEGX,POSY,NEGY
C
      INTEGER SOURCE,SINK,VERTEX
      LOGICAL FOUND,CLKWSE,CANKIL,EMPTY
C
      HEAD   = 0
      TAIL   = 0
      N      = 0
      ORGN   = CENTRE
      VERTEX = NEARST ()
C
      IF (VERTEX.NE.0) THEN
        CALL ADDTL(CENTRE,VERTEX)
        CALL ADDTL(VERTEX,CENTRE)
        NEYBOR(CENTRE,1) = VERTEX
        N = 1
      ENDIF
C
      IF (EMPTY ()) THEN
        NEYBOR(CENTRE,0) = 0
        RETURN
      ENDIF
C
   10 CALL GETHD(SOURCE,SINK)
      CLKWSE = SOURCE .EQ. CENTRE
      CALL TRIANG(SOURCE,SINK,VERTEX,FOUND,CLKWSE)
C
      IF (FOUND) THEN
        IF (N.GE.MAXNEY) THEN
          PRINT*,' Too many neighbours.'
          STOP 7777
        ELSE
          N=N+1
          NEYBOR(CENTRE,N)=VERTEX
          IF (CLKWSE) THEN
            IF (.NOT.CANKIL(VERTEX,CENTRE)) CALL ADDTL(CENTRE,VERTEX)
          ELSE
            IF (.NOT.CANKIL(CENTRE,VERTEX)) CALL ADDTL(VERTEX,CENTRE)
          ENDIF
        ENDIF
      ENDIF
C
      IF (.NOT.EMPTY ()) GOTO 10
C
      NEYBOR(CENTRE,0) = N
      IF (N.GE.2) THEN
        IF (NEYBOR(CENTRE,N).EQ.NEYBOR(CENTRE,N-1))
     +      NEYBOR(CENTRE,0) = N - 1
      ENDIF
C
      RETURN
      END
C
C --- ********************************************************
```

```
C     This function indicates whether the point (X,Y) is
C     nearer C the point in DIST(0,1:2) than it is to any
C     other point in DIST.
C --- ********************************************************
C
      LOGICAL FUNCTION INTILE (X,Y)
C
      INTEGER X,Y
C
      PARAMETER (MAXNEY=64)
C
      INTEGER DIST(0:MAXNEY,2),NUMBER
      COMMON  /DINFO/DIST,NUMBER
C
      INTEGER DIFF(0:MAXNEY,2),DISTSQ
C
      DO 10 I=0,NUMBER
   10   DIFF(I,1) = (DIST(I,1)-X) * (DIST(I,1)-X)
C
      DO 20 I=0,NUMBER
   20   DIFF(I,2) = (DIST(I,2)-Y) *(DIST(I,2)-Y)
C
      DISTSQ = DIFF(0,1) + DIFF(0,2)
      DO 30 I=1,NUMBER
        IF ((DIFF(I,1) + DIFF(I,2)).LT.DISTSQ) THEN
          INTILE = .FALSE.
          RETURN
        ENDIF
   30   CONTINUE
C
      INTILE = .TRUE.
C
      RETURN
      END
C
C --- ********************************************************
C     This procedure computes the histogram for generating
C     point CENTRE's tile.
C --- ********************************************************
C
      SUBROUTINE TILE (CENTRE)
C
      INTEGER CENTRE
C
      INTEGER IMAGE(0:255,0:255)
      COMMON  /IINFO/IMAGE
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER DIST(0:MAXNEY,2),NUMBER
      COMMON  /DINFO/DIST,NUMBER
```

```
C
      INTEGER COUNT (0:255),N
      COMMON  /CDATA/ COUNT,N
C
      INTEGER OPT1,OPT2
      LOGICAL INC
      COMMON  /OPTNS/OPT1,OPT2,INC
C
      INTEGER YMIN,YMAX,ROWMIN,ROWMAX,COL,X,Y,
     +        TOP,BOTTOM,MID
      LOGICAL STOP,INTILE
C
      DO 10 I=0,255
   10   COUNT(I) = 0
C
      N = 0
      DIST(0,1) = POINTS(CENTRE,1)
      DIST(0,2) = POINTS(CENTRE,2)
      NUMBER    = NEYBOR(CENTRE,0)
      DO 20 J=1,2
        DO 20 I=1,NUMBER
   20     DIST(I,J) = POINTS(NEYBOR(CENTRE,I),J)
C
      COL    = POINTS(CENTRE,1)
      X      = COL
      BOTTOM = 0
      TOP    = POINTS(CENTRE,2) + 1
      STOP   = .FALSE.
C
   30   MID = (TOP+BOTTOM) / 2
        Y   = MID
        IF (INTILE(X,Y)) THEN
          IF (MID.EQ.0) THEN
            STOP = .TRUE.
          ELSE
            Y = MID - 1
            IF (INTILE(X,Y)) THEN
              TOP = MID
            ELSE
              STOP = .TRUE.
            ENDIF
          ENDIF
        ELSE
          BOTTOM = MID
        ENDIF
        IF (.NOT.STOP) GOTO 30
C
      ROWMIN = MID
C
      BOTTOM = POINTS(CENTRE,2)
      TOP    = 256
      STOP   = .FALSE.
C
   40   MID = (TOP+BOTTOM) / 2
        Y   = MID
```

```
        IF (INTILE(X,Y)) THEN
          IF (MID.EQ.255) THEN
            STOP = .TRUE.
          ELSE
            Y = MID + 1
            IF (INTILE(X,Y)) THEN
              BOTTOM = MID
            ELSE
              STOP = .TRUE.
            ENDIF
          ENDIF
        ELSE
          TOP = MID
        ENDIF
        IF (.NOT.STOP) GOTO 40
C
      ROWMAX = MID
C
      DO 50 I=ROWMIN,ROWMAX
   50   COUNT(IMAGE(I,COL)) = COUNT(IMAGE(I,COL)) + 1
      N = N + ROWMAX - ROWMIN + 1
C
      J    = COL
      YMIN = ROWMIN
      YMAX = ROWMAX
      IF (J.LE.0) GOTO 120
C
   60   YMIN = MAX(YMIN-2,0)
        J    = J-1
        STOP = .FALSE.
        X    = J
        Y    = YMIN
        IF (INTILE(X,Y)) THEN
   70     IF (YMIN.EQ.0) THEN
            STOP = .TRUE.
          ELSE
            Y = YMIN - 1
            IF (INTILE(X,Y)) THEN
              YMIN = Y
            ELSE
              STOP = .TRUE.
            ENDIF
          ENDIF
          IF (.NOT.STOP) GOTO 70
        ELSE
   80     IF ((YMIN.GT.YMAX+1).OR.(YMIN.EQ.255)) THEN
            J = -1
          ELSE
            YMIN = YMIN + 1
            Y    = YMIN
            STOP = INTILE(X,Y)
          ENDIF
          IF ((.NOT.STOP).AND.(J.GE.0)) GOTO 80
        ENDIF
C
```

```
      IF (J.GE.0) THEN
        YMAX = MAX(YMIN,YMAX)
        STOP = .FALSE.
        Y    = YMAX
        IF (INTILE(X,Y)) THEN
 90       IF (YMAX.EQ.255) THEN
            STOP = .TRUE.
          ELSE
            Y = YMAX + 1
            IF (INTILE(X,Y)) THEN
              YMAX = Y
            ELSE
              STOP = .TRUE.
            ENDIF
          ENDIF
          IF (.NOT.STOP) GOTO 90
        ELSE
100       YMAX = YMAX - 1
          Y    = YMAX
          STOP = INTILE(X,Y)
          IF (.NOT.STOP) GOTO 100
        ENDIF
        DO 110 I=YMIN,YMAX
110       COUNT(IMAGE(I,J)) = COUNT(IMAGE(I,J)) + 1
        N = N + YMAX - YMIN + 1
      ENDIF
      IF (J.GT.0) GOTO 60
C
120 CONTINUE
C
      J    = COL
      YMIN = ROWMIN
      YMAX = ROWMAX
      IF (J.GE.255) GOTO 190
130   YMIN = MAX(YMIN-2,0)
      J    = J+1
      STOP=.FALSE.
      X    = J
      Y    = YMIN
      IF (INTILE(X,Y)) THEN
140     IF (YMIN.EQ.0) THEN
          STOP = .TRUE.
        ELSE
          Y = YMIN - 1
          IF (INTILE(X,Y)) THEN
            YMIN = Y
          ELSE
            STOP = .TRUE.
          ENDIF
        ENDIF
        IF (.NOT.STOP) GOTO 140
      ELSE
150     IF ((YMIN.GT.YMAX+1).OR.(YMIN.EQ.255)) THEN
          J = 256
        ELSE
```

```
            YMIN = YMIN + 1
            Y    = YMIN
            STOP = INTILE(X,Y)
          ENDIF
          IF ((.NOT.STOP).AND.(J.LE.255)) GOTO 150
        ENDIF
C
        IF (J.LE.255) THEN
          YMAX = MAX(YMIN,YMAX)
          STOP = .FALSE.
          Y    = YMAX
          IF (INTILE(X,Y)) THEN
  160       IF (YMAX.EQ.255) THEN
              STOP = .TRUE.
            ELSE
              Y = YMAX + 1
              IF (INTILE(X,Y)) THEN
                YMAX = Y
              ELSE
                STOP = .TRUE.
              ENDIF
            ENDIF
            IF (.NOT.STOP) GOTO 160
          ELSE
  170       YMAX = YMAX - 1
            Y    = YMAX
            STOP = INTILE(X,Y)
            IF (.NOT.STOP) GOTO 170
          ENDIF
          DO 180 I=YMIN,YMAX
  180       COUNT(IMAGE(I,J)) = COUNT(IMAGE(I,J)) + 1
          N = N + YMAX-YMIN+1
        ENDIF
        IF (J.LT.255) GOTO 130
C
  190 CONTINUE
C
C --- Calculates the histogram's waviness.
C
      IF      (OPT2.EQ.0) THEN
        MESURE(CENTRE) = BUMP()
      ELSE IF (OPT2.EQ.1) THEN
        MESURE(CENTRE) = BUMP2()
      ELSE IF (OPT2.EQ.2) THEN
        MESURE(CENTRE) = ENTPY()
      ELSE IF (OPT2.EQ.3) THEN
        MESURE(CENTRE) = VARNCE()
      ENDIF
C
      RETURN
      END
C
C --- ********************************************************
C     This subroutine is called at the beginning of the run.
C     It initialises all the necessary variables and also
```

```
C     calls READIN to read in the image. If OPT1 = 1, it will
C     read in the initial positions of the generating points
C     from device 12.
C --- ******************************************************
C
      SUBROUTINE INIT
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER IMAGE(0:255,0:255)
      COMMON  /IINFO/IMAGE
C
      INTEGER X,Y,Z
      COMMON  /RNDVAR/X,Y,Z
C
      INTEGER JUMPS(-255:255)
      COMMON  /JDATA/JUMPS
C
      INTEGER OPT1,OPT2
      LOGICAL INC
      COMMON  /OPTNS/OPT1,OPT2,INC
C
      READ(5,*) OPT1,OPT2
      WRITE(6,1000) 1,OPT1
      WRITE(6,1000) 2,OPT2
      INC = (OPT2.EQ.2)
C
      CALL READIN(IMAGE)
C
      JUMP   = 255
      NPOINT = 16
C
      X      = 123
      Y      = 231
      Z      = 312
C
      DO 10 I=-255,255
10      JUMPS(I) = 0
C
      IF (OPT1.EQ.0) THEN
        POINTS(1,1) = 0
        POINTS(1,2) = 0
        POINTS(2,1) = 255
        POINTS(2,2) = 0
C
        POINTS(3,1) = 96
        POINTS(4,1) = 160
        DO 20 I=3,4
20        POINTS(I,2) = 32
        DO 30 I=5,8
          POINTS(I,1) = 32 + (I-5)*64
```

```
   30     POINTS(I,2) = 96
        DO 40 I=9,12
          POINTS(I,1) = 32 + (I-9)*64
   40     POINTS(I,2) = 160
        POINTS(13,1)  = 96
        POINTS(14,1)  = 160
        DO 50 I=13,14
   50     POINTS(I,2) = 224
        POINTS(15,1)  = 0
        POINTS(15,2)  = 255
        POINTS(16,1)  = 255
        POINTS(16,2)  = 255
      ELSE
        DO 60 I=1,16
   60     READ(12,*) POINTS(I,1),POINTS(I,2)
      ENDIF
C
      DO 70 I=1,NPOINT
        CALL SURRND(I)
   70   CALL TILE(I)
C
      RETURN
1000  FORMAT (1X,'Option',I2,' has value',I2)
      END
C
C --- ********************************************************
C     This function calculates the goodness of tessellation of
C     the image.
C --- ********************************************************
C
      FUNCTION CALMES ( )
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      TEMP = 0.0
      DO 10 I=1,NPOINT
   10   TEMP = TEMP + MESURE(I)
      CALMES = TEMP
C
      RETURN
      END
C
C --- ********************************************************
C     This is the subroutine that actually does all the
C     processing.
C --- ********************************************************
C
      SUBROUTINE PROCES
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
```

```
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER OPT1,OPT2
      LOGICAL INC
      COMMON  /OPTNS/OPT1,OPT2,INC
C
      INTEGER OLDPNT(64,2),OLDNEY(64,0:MAXNEY)
      REAL    OLDMES(64)
      COMMON  /LOCAL/OLDPNT,OLDNEY,OLDMES
C
      PARAMETER (ISIZE=128 + 64*(MAXNEY+1) + 64)
      INTEGER A(ISIZE),B(ISIZE)
      EQUIVALENCE (A,POINTS),(B,OLDPNT)
C
      INTEGER INDEX(64),WHICH
      REAL    TOTAL,TARGET
      LOGICAL DONE(64)
C
      TARGET = CALMES ()
C
   10 WRITE(6,200) JUMP
C
        DO 20 I=1,NPOINT
   20     DONE(I) = .FALSE.
C
        DO 30 I=1,ISIZE
   30     B(I) = A(I)
C
        CALL MOVPNT(WHICH)
        CALL SORT  (WHICH,INDEX)
C
        DO 50 I=1,NPOINT
          NEYBOR(INDEX(I),0) = OLDNEY(I,0)
          DO 50 J=1,OLDNEY(I,0)
   50       NEYBOR(INDEX(I),J) = INDEX(OLDNEY(I,J))
C
        CALL SURRND (INDEX(WHICH))
        CALL TILE   (INDEX(WHICH))
        DONE(INDEX(WHICH)) = .TRUE.
C
        DO 60 I=1,OLDNEY(WHICH,0)
          K = INDEX(OLDNEY(WHICH,I))
          CALL SURRND (K)
          CALL TILE   (K)
   60     DONE(K) = .TRUE.
C
        DO 70 I=1,NEYBOR(INDEX(WHICH),0)
          K = NEYBOR(INDEX(WHICH),I)
          IF (.NOT.DONE(K)) THEN
            CALL SURRND (K)
            CALL TILE   (K)
           ENDIF
   70    CONTINUE
```

```
C
        TOTAL = CALMES ()
        IF ((TOTAL.NE.TARGET).AND.((TOTAL.GT.TARGET).EQV.INC)) THEN
          TARGET = TOTAL
          JUMP   = MIN(JUMP+20,255)
          WRITE(6,300) TARGET,JUMP
        ELSE
          DO 80 I=1,ISIZE
   80       A(I) = B(I)
          JUMP = JUMP - 1
        ENDIF
C
        IF (JUMP.GT.0) GOTO 10
C
      WRITE(6,400) TARGET
C
      RETURN
  100 FORMAT (1X,'Initial bumpiness = ',F25.20)
  200 FORMAT (1X,'Jump size = ',I4)
  300 FORMAT (1X,'Bumpiness = ',F25.20,5X,'Jump size = ',I4)
  400 FORMAT (1X,'Final bumpiness = ',F25.20)
      END
C
C --- ***********************************************************
C     This subroutine stores the result of the optimisation on
C     device 21 and the number of jumps of each length on
C     device 22.
C --- ***********************************************************
C
      SUBROUTINE ENDOFF
C
      PARAMETER (MAXNEY=64)
      INTEGER POINTS(64,2),NEYBOR(64,0:MAXNEY)
      REAL    MESURE(64)
      INTEGER NPOINT,JUMP
      COMMON  /INFO/POINTS,NEYBOR,MESURE,NPOINT,JUMP
C
      INTEGER JUMPS(-255:255)
      COMMON  /JDATA/JUMPS
C
      DO 10 I=1,NPOINT
   10   WRITE(21,100) POINTS(I,1),POINTS(I,2)
C
      DO 20 I=-255,255
   20   WRITE(22,100) I,JUMPS(I)
C
      DO 30 I=1,NPOINT
   30   WRITE(6,100) POINTS(I,1),POINTS(I,2)
C
      RETURN
  100 FORMAT (1X,2I10)
      END
```

APPENDIX C

Following is the listing of the source code of a segment of a non-standard Pascal program written to compute and display the interpolated threshold surface, using planar interpolation, as described in section 8.3.3. It is written in Oregon Software's Pascal 2.1 for an LSI-11, a 16 bit mini-computer.

The program segment plots the threshold surface as an image with a spatial resolution of 256x256 pixels.

The parameters for procedure PLANE and their uses are as follows:

Coordinates - the (x,y) co-ordinates of the generating points, their threshold values (z), and the partial derivatives of the surface to be interpolated (not used).

NumberOfPoints - the number of generating points.

XOffSet - the x-offset of the window in the framestore where the threshold surface is to be drawn.

YOffSet - the y-offset of the window in the framestore where the threshold surface is to be drawn.

Because of the accesses that need to be made to the values held in the frame-store, there are some machine-dependent features in this program segment, namely the use of the variables XREG, YREG and

DATAREG in procedure PLANE. To assign a value V to position (X,Y) of the frame-store, the following statements are necessary:

```
xreg    :=X;
yreg    :=Y;
datareg:=V
```

and to interrogate the value of position (X,Y) of the frame-store, DATAREG is interrogated after the statements:

```
xreg:=X;
yreg:=Y
```

```
Program Tile21;

{$nomain}

Const MaxPoints     = 64;   {maximum number of generating points }
                            {  allowed                           }
      QueueSize     = 300;  {maximim queue size for triangulation}
      MaxDegeneracy = 20;   {maximum degeneracy allowed          }

Type Ordinate    = 0..255;
     PointIndex  = 1..MaxPoints;
     IntTuple    = record
                     x,y       : Ordinate;
                     ddx,ddy,z : real
                   end;
     IntCoords   = array [PointIndex] of IntTuple;
     Rectangle   = record
                     MinX,MaxX,MinY,MaxY : integer
                   end;

     QueueIndex  = 0..QueueSize;
     DegenFactor = 0..MaxDegeneracy;
     PointArray  = array [DegenFactor] of PointIndex;
     LinkType    = record
                     source,sink : PointIndex;
                   end;
     QueueType   = record
                     link      : array [QueueIndex] of LinkType;
                     Head,Tail : QueueIndex
                   end;
     List        = array [0..511] of integer;

(*****************************************************************)

Procedure Plane (Var Coordinates       : IntCoords;
                     NumberOfPoints,
                       XOffSet,YOffSet : integer);       external;

(*****************************************************************)

Function Min (a,b : integer) : integer;

begin
    if (a >= b) then Min:=b
                else Min:=a
end;

(*****************************************************************)

Function Max (a,b : integer) : integer;

begin
    if (a >= b) then Max:=a
                else Max:=b
end;
```

```
(*****************************************************************)

Function Min3 (a,b,c : integer) : integer;

Var t : integer;

begin
    if (a < b) then t:=a
               else t:=b;
    if (c < t) then Min3:=c
               else Min3:=t
end;

(*****************************************************************)

Function Max3 (a,b,c : integer) : integer;

Var t : integer;

begin
    if (a > b) then t:=a
               else t:=b;
    if (c > t) then Max3:=c
               else Max3:=t
end;

(*****************************************************************)

Procedure Plane;

Var Queue                 : QueueType;
    SearchOffset,
     PosX,PosY,NegX,NegY  : integer;
    Orgn                  : PointIndex;
    ch                    : char;

    xreg    origin 164400B : integer;
    yreg    origin 164402B : integer;
    datareg origin 164404B : integer;

(*===============================================================*)
(* This procedure sets the pixel at position (x,y) in the        *)
(* 256x256 window where the threshold surface is being drawn     *)
(* to the value z.                                               *)
(*===============================================================*)

Procedure PutAPixel  (x,y,z : integer);

begin
    xreg   :=x + XOffSet;
    yreg   :=y + YOffSet;
    datareg:=z
end;
```

```
(*=================================================================*)
(* This procedure initialises the queue used in the               *)
(* triangulation routine.                                          *)
(*=================================================================*)

Procedure InitQueue;

begin
    Queue.Head:=0;
    Queue.tail:=0
end;

(*=================================================================*)
(* This function returns a value indicating whether the queue     *)
(* is empty or not.                                                *)
(*=================================================================*)

Function Empty : boolean;

begin
    Empty:=Queue.Head = Queue.Tail
end;

(*=================================================================*)
(* This function returns the length of the queue.                 *)
(*=================================================================*)

Function Length : integer;

Var Interval : integer;

begin
    Interval:=Abs(Queue.Head - Queue.Tail);
    if (Queue.head <= Queue.Tail) then Length:=Interval
    else Length:=QueueSize - Interval
end;

(*=================================================================*)
(* This procedure gets the edge at the front of the queue.        *)
(*=================================================================*)

Procedure GetHead (Var HeadLink : LinkType);

begin
    if Empty then writeln('Queue underflow')
    else with Queue do
       begin
        HeadLink:=Link[Head];
        Head:=(Head + 1) mod Queuesize
       end
end;

(*=================================================================*)
(* This procedure adds an edge to the back of the queue.          *)
(*=================================================================*)
```

```
Procedure AddToTail (NewSource,NewSink:PointIndex);

begin
    with Queue.Link[Queue.Tail] do
       begin
        Source:=NewSource;
        Sink:=NewSink
       end;
    Queue.Tail:=(Queue.Tail + 1) mod QueueSize;
    if Empty then writeln('Queue overflow')
end;

(*=============================================================*)
(* This function determines whether there are any redundant    *)
(* edges in the queue or not.                                  *)
(*=============================================================*)

Function AbleToErase (SearchSource,
                      SearchSink : PointIndex) : boolean;

Var Pointer : QueueIndex;
    Found   : boolean;

begin
    found:=false;
    with Queue do
       begin
        Pointer:=Head;
        while (Pointer <> Tail) do
           begin
            found:= found or ((SearchSource=Link[Pointer].Source)
                          and (SearchSink  =Link[Pointer].Sink ));
            if found then
                Link[Pointer]:=Link[(Pointer+1) mod QueueSize];
            Pointer:=(Pointer + 1) mod QueueSize
           end;
        if found then Tail:=(Tail + QueueSize - 1) mod QueueSize
       end;
    AbleToErase:=found
end;

(*=============================================================*)
(* This function returns the number of the next generating     *)
(* point to be triangulated.                                   *)
(*=============================================================*)

Function PingPong : integer;

Var TestPoint,x,y,i : integer;
    finished        : boolean;
    side            : integer;

begin
    repeat
```

```
        TestPoint:=0;
        for side:=1 to 2 do
            if (TestPoint = 0) then
                if (SearchOffset > 0) then
                   begin
                    SearchOffset:=0 - SearchOffset;
                    TestPoint:=orgn + SearchOffset;
                    if (TestPoint >= 1) then
                       begin
                        x:=Coordinates[TestPoint].x;
                        y:=Coordinates[TestPoint].y;
                        if (y < NegY) then TestPoint:=0
                       end
                    else TestPoint:=0
                   end
                else
                   begin
                    SearchOffset:=1 - SearchOffset;
                    TestPoint:=orgn + SearchOffset;
                    if (TestPoint <= NumberOfPoints) then
                       begin
                        x:=Coordinates[TestPoint].x;
                        y:=Coordinates[TestPoint].y;
                        if (y > PosY) then TestPoint:=0
                       end
                    else TestPoint:=0
                   end;
         if (TestPoint = 0)
             then Finished:=true
             else Finished:=(x <= PosX) and (x >= NegX)
       until Finished;
    PingPong:=TestPoint
end;

(*=============================================================*)
(* This function returns the number of the generating point    *)
(* nearest to generating point number ORGN.                    *)
(*=============================================================*)

Function Nearest : integer;

Var ClosestFound         : PointIndex;
    SearchPoint,Limit    : integer;
    SqrDistance,SqrLimit : real;

begin
    SearchOffset:=0;
    SqrLimit    :=1E38;
    Limit       :=maxint;

    NegX:=0;        NegY:=0;
    PosX:=255;      PosY:=255;

    repeat
        repeat
```

```
            SearchPoint:=PingPong;
            if (SearchPoint <> 0) then
                SqrDistance:=sqr(Coordinates[orgn].x -
                                 Coordinates[SearchPoint].x+0.0) +
                             sqr(Coordinates[orgn].y -
                                 Coordinates[SearchPoint].y+0.0)
          until (SqrDistance < SqrLimit) or (SearchPoint = 0);

        if (SearchPoint <> 0) then
           begin
            SqrLimit:=SqrDistance;
            Limit:=trunc(sqrt(SqrLimit)) + 1;
            PosX:=Min(255,Coordinates[orgn].x + Limit);
            NegX:=Max(0,  Coordinates[orgn].x - Limit);
            PosY:=Min(255,Coordinates[orgn].y + Limit);
            NegY:=Max(0,  Coordinates[orgn].y - Limit);
            ClosestFound:=SearchPoint
           end
       until (SearchPoint = 0);
    Nearest:=ClosestFound
end;

(*=============================================================*)
(* This procedure computes the next triangle (or triangles in  *)
(* event of a degeneracy) to be included in the triangulation. *)
(*=============================================================*)

Procedure Triangle (    BaseLink   : LinkType;
                    Var VertexList : PointArray;
                    Var Degeneracy : DegenFactor);

Var x1,x2,x3,y1,y2,y3                : integer;
    SqrA,SqrB,SqrC                   : real;
    Index,Radius,XCircumCentre,
     YCircumCentre                   : integer;
    XMid,YMid,XHalf,YHalf,
     MinimumCosA,CosATest,CosA,SinA,
     CosB,Ratio,Delta,Temp           : real;
    VertexIndex,Slot                 : DegenFactor;
    SlotSet                          : boolean;
    CosBList                         : array [DegenFactor] of real;

begin
    SearchOffset:=0;
    orgn:=round((BaseLink.Source + BaseLink.Sink)/ 2.0);
    x1:=Coordinates[BaseLink.Source].x;
    x2:=Coordinates[BaseLink.Sink  ].x;
    y1:=Coordinates[BaseLink.Source].y;
    y2:=Coordinates[BaseLink.Sink  ].y;
    SqrA:=sqr(x1-x2+0.0) + sqr(y1-y2+0.0);
    XMid :=(x1+x2) / 2.0;
    YMid :=(y1+y2) / 2.0;
    XHalf:=(x2-x1) / 2.0;
    YHalf:=(y2-y1) / 2.0;
    PosX :=255;      PosY:=255;
```

```
NegX :=0;         NegY:=0;
Delta:=2E-6;
MinimumCosA:=2.0;
CosBList[1]:=2.0;
CosATest:=MinimumCosA + Delta;
Degeneracy:=0;
Index:=orgn;

while (index <> 0) do
   begin
    if (index <> BaseLink.Source) and
       (index <> BaseLink.Sink)   then
       begin
        x3:=Coordinates[index].x;
        y3:=Coordinates[index].y;
        if ((y2*(x3-x1+0.0) +
             y1*(x2-x3+0.0) +
             y3*(x1-x2+0.0)) < 0.0) then
           begin
            SqrB:=sqr(x3-x2+0.0) + sqr(y3-y2+0.0);
            SqrC:=sqr(x3-x1+0.0) + sqr(y3-y1+0.0);
            CosA:=(SqrB+SqrC-SqrA) / (2.0 * sqrt(SqrB*SqrC));
            if (CosA < CosATest) then
               begin
                CosB:=(SqrA+SqrC-SqrB) /
                      (2.0 * sqrt(SqrA*SqrC));
                If (abs(CosA - MinimumCosA) > Delta) then
                   begin
                    CosBList[1]:=CosB;
                    VertexList[1]:=Index;
                    MinimumCosA:=CosA;
                    CosATest:=CosA + Delta;
                    Degeneracy:=1;
                    SinA:=sqrt(1.0 - sqr(CosA));
                    If (SinA < 1E-4) then Radius:=255
                    else
                       begin
                        Temp:=sqrt(SqrA)/(2.0*SinA);
                        if (Temp < 254.0)
                            then Radius:=1+trunc(Temp)
                            else Radius:=255
                       end;
                    if (SinA = 0.0)
                        then Ratio:=255.0
                        else Ratio:=CosA/SinA;
                    Temp:=XMid - YHalf*Ratio;
                    if (temp < 0.0) then XCircumcentre:=0
                    else if (temp > 255)
                        then XCircumcentre:=255
                        else XCircumcentre:=round(temp);
                    Temp:=YMid + XHalf*Ratio;
                    if (temp < 0.0)
                        then YCircumcentre:=0
                        else if (temp > 255)
                            then YCircumcentre:=255
```

```
                                else YCircumcentre:=round(temp);
                            PosX:=Min(255,XCircumcentre+radius);
                            PosY:=Min(255,YCircumcentre+radius);
                            NegX:=Max(0,  XCircumcentre-radius);
                            NegY:=Max(0,  YCircumcentre-radius)
                           end
                       else
                           begin
                            SlotSet:=false;
                            Slot:=Degeneracy+1;
                            for VertexIndex:=1 to Degeneracy do
                            if (CosB>=CosBList[VertexIndex]) and
                               (not SlotSet) then
                               begin
                                Slot:=VertexIndex;
                                SlotSet:=true
                               end;
                            for VertexIndex:=Degeneracy downto Slot do
                               begin
                                CosBList[VertexIndex+1]:=
                                    CosBList[VertexIndex];
                                VertexList[VertexIndex+1]:=
                                    VertexList[VertexIndex]
                               end;
                            CosBList  [Slot]:=CosB;
                            VertexList[Slot]:=Index;
                            Degeneracy      :=Degeneracy + 1
                           end
                    end
                end
            end;
         Index:=PingPong
        end
end;

(*=============================================================*)
(* This function indicates whether the point (x,y) lies on the  *)
(* right hand side of the vector (x2-x1,y2-y1) passing through  *)
(* (x1,y1) or not.                                              *)
(*=============================================================*)

Function OnRightSideOfLine (x,x1,x2,y,y1,y2 : Ordinate) : boolean;

begin
    OnRightSideOfLine:=(y2*1.0*(x-x1) +
                        y1*1.0*(x2-x) +
                        y *1.0*(x1-x2)) > 0.0
end;

(*=============================================================*)
(* This function determines whether the point (x,y) lies in the *)
(* triangle with generating points (x1,y1), (x2,y2) and (x3,y3) *)
(* or not.                                                      *)
(*=============================================================*)
```

```
Function PointInTriangle (x,x1,x2,x3,y,y1,y2,y3 : Ordinate)
                                                        : boolean;

begin
    if OnRightSideOfLine(x,x1,x2,y,y1,y2)
        then PointInTriangle:=false
        else if OnRightSideOfLine(x,x2,x3,y,y2,y3)
            then PointInTriangle:=false
            else if OnRightSideOfLine(x,x3,x1,y,y3,y1)
                then PointInTriangle:=false
                else PointInTriangle:=true
end;

(*============================================================*)
(* This procedure calculates the parameters for the plane     *)
(* passing through the generating points VERTEX1, VERTEX2 and *)
(* VERTEX3, using partial pivoting, and then interpolates the *)
(* threshold values for the pixels lying within the triangle  *)
(* formed by the three generating points.                     *)
(*============================================================*)

Procedure PlanarPatch (Vertex1,Vertex2,Vertex3 : PointIndex);

Var Matrix                : array [1..3,1..4] of real;
    Index                 : array [1..3]      of integer;
    a,b,c,factor          : real;
    i,j,x1,x2,x3,y1,y2,y3 : integer;
    MinX,MaxX,MinY,MaxY   : integer;

begin
    with Coordinates[Vertex1] do
       begin
        Matrix[1,1]:=x;
        Matrix[1,2]:=y;
        Matrix[1,3]:=1.0;
        Matrix[1,4]:=z
       end;
    with Coordinates[Vertex2] do
       begin
        Matrix[2,1]:=x;
        Matrix[2,2]:=y;
        Matrix[2,3]:=1.0;
        Matrix[2,4]:=z
       end;
    with Coordinates[Vertex3] do
       begin
        Matrix[3,1]:=x;
        Matrix[3,2]:=y;
        Matrix[3,3]:=1.0;
        Matrix[3,4]:=z
       end;

    for i:=1 to 3 do
        Index[i]:=i;
```

```
j:=1;  factor:=Matrix[1,1];
for i:=2 to 3 do
    if (Matrix[i,1] > factor) then
       begin
        factor:=Matrix[i,1];
        j:=i
       end;
Index[1]:=j;
Index[j]:=1;

for i:=2 to 3 do
   begin
    factor:=-(Matrix[Index[i],1] / Matrix[Index[1],1]);
    for j:=1 to 4 do
        Matrix[Index[i],j]:=Matrix[Index[i],j] +
                    factor*Matrix[Index[1],j];
   end;
factor:=Matrix[Index[1],1];
for i:=1 to 4 do
    Matrix[Index[1],i]:=Matrix[Index[1],i] / factor;

if (abs(Matrix[Index[3],2]) > abs(Matrix[Index[2],2])) then
   begin
    j:=Index[2]; Index[2]:=Index[3]; Index[3]:=j
   end;
factor:=-(Matrix[Index[3],2] / Matrix[Index[2],2]);
for i:=2 to 4 do
    Matrix[Index[3],i]:=Matrix[Index[3],i] +
                factor*Matrix[Index[2],i];
factor:=-(Matrix[Index[1],2] / Matrix[Index[2],2]);
for i:=2 to 4 do
    Matrix[Index[1],i]:=Matrix[Index[1],i] +
                factor*Matrix[Index[2],i];
factor:=Matrix[Index[2],2];
for i:=2 to 4 do
    Matrix[Index[2],i]:=Matrix[Index[2],i] / factor;

for i:=1 to 2 do
   begin
    factor:=-(Matrix[Index[i],3] / Matrix[Index[3],3]);
    for j:=3 to 4 do
        Matrix[Index[i],j]:=Matrix[Index[i],j] +
                    factor*Matrix[Index[3],j];
   end;
Matrix[Index[3],4]:=Matrix[Index[3],4] / Matrix[Index[3],3];
Matrix[Index[3],3]:=1.0;

a:=Matrix[Index[1],4];
b:=Matrix[Index[2],4];
c:=Matrix[Index[3],4];

x1:=Coordinates[Vertex1].x;
x2:=Coordinates[Vertex2].x;
x3:=Coordinates[Vertex3].x;
MinX:=Min3(x1,x2,x3);
```

```
    MaxX:=Max3(x1,x2,x3);
    y1:=Coordinates[Vertex1].y;
    y2:=Coordinates[Vertex2].y;
    y3:=Coordinates[Vertex3].y;
    MinY:=Min3(y1,y2,y3);
    MaxY:=Max3(y1,y2,y3);

    for i:=MinX to MaxX do
        for j:=MinY to MaxY do
            if PointInTriangle(i,x1,x2,x3,j,y1,y2,y3) then
                PutAPixel(i,j,round(a*i + b*j + c))
end;

(*=================================================================*)
(* This procedure computes the triangles in the Delaunay          *)
(* Triangulation of the generating points and then calls          *)
(* PlanarPatch to interpolate the threshold surface over each     *)
(* triangle.                                                      *)
(*=================================================================*)

Procedure DelaunayTriangulation;

Var BaseLink     : LinkType;
    VertexList   : PointArray;
    Degeneracy   : DegenFactor;
    DegenIndex   : DegenFActor;
    Vertex       : integer;
    LastVertex   : PointIndex;

begin
    InitQueue;
    orgn:=NumberOfPoints div 2;
    Vertex:=Nearest;
    if (Vertex <> 0) then
       begin
        AddToTail(NumberOfPoints div 2,Vertex);
        AddToTail(Vertex,NumberOfPoints div 2);
        while not Empty do
           begin
            GetHead(BaseLink);
            Triangle(BaseLink,VertexList,Degeneracy);
            LastVertex:=BaseLink.Sink;
            if (Degeneracy <> 0) then
               begin
                for DegenIndex:=1 to Degeneracy do
                   begin
                    Vertex:=VertexList[DegenIndex];
                    PlanarPatch
                            (BaseLink.Source,LastVertex,Vertex);
                    if not AbleToErase(LastVertex,Vertex)
                        then AddToTail(Vertex,LastVertex);
                    LastVertex:=Vertex
                   end;
                if not AbleToErase(Vertex,BaseLink.Source) then
                    AddToTail(BaseLink.Source,Vertex)
```

```
                end
            end
        end
end;

(*=================================================================*)

begin
    DelaunayTriangulation
end;
```

APPENDIX D

Following is the listing of the source code of a segment of a non-standard Pascal program written to compute and display the interpolated threshold surface, using bi-cubic interpolation, as described in section 8.3.3. of this thesis. It is written in Oregon Software's Pascal 2.1 for an LSI-11, a 16 bit mini-computer.

The program segment plots the threshold surface as an image with a spatial resolution of 256x256 pixels.

The parameters for procedure CUBIC and their uses are as follows:

Coordinates - the (x,y) co-ordinates of the generating points, their threshold values (z), and the partial derivatives of the surface to be interpolated.

NumberOfPoints - the number of generating points.

XOffSet - the x-offset of the window in the framestore where the threshold surface is to be drawn.

YOffSet - the y-offset of the window in the framestore where the threshold surface is to be drawn.

Because of the accesses that need to be made to the values held in the frame-store, there are some machine-dependent features in this

program segment, namely the use of the variables XREG, YREG and DATAREG in procedure CUBIC. To assign a value V to position (X,Y) of the frame-store, the following statements are necessary:

```
xreg    :=X;
yreg    :=Y;
datareg:=V
```

and to interrogate the value of position (X,Y) of the frame-store, DATAREG is interrogated after the statements:

```
xreg:=X;
yreg:=Y
```

```
Program Tile20;

{$nomain}

Const MaxPoints     = 64;  {maximum number of generating points}
                           {  allowed                          }
      QueueSize     = 300; {length of queue for triangulation  }
      MaxDegeneracy = 20;  {maximum degeneracy allowed         }

Type Ordinate    = 0..255;
     PointIndex  = 1..MaxPoints;
     IntTuple    = record
                     x,y       : Ordinate;
                     ddx,ddy,z : real
                   end;
     IntCoords   = array [PointIndex] of IntTuple;
     Rectangle   = record
                     MinX,MaxX,MinY,MaxY : integer
                   end;
     DerivPair   = record
                     ddx,ddy : real
                   end;
     CubicType   = record
                     a,b,c,d,p,q,r,s : real;
                     Deriv           : DerivPair
                   end;

     QueueIndex  = 0..QueueSize;
     DegenFactor = 0..MaxDegeneracy;
     PointArray  = array [DegenFactor] of PointIndex;
     LinkType    = record
                     source,sink : PointIndex;
                   end;
     QueueType   = record
                     link      : array [QueueIndex] of LinkType;
                     Head,Tail : QueueIndex
                   end;
     List        = array [0..511] of integer;

(*****************************************************************)

Procedure Cubic (Var Coordinates       : IntCoords;
                     NumberOfPoints,
                       XOffSet,YOffSet : integer);       external;

(*****************************************************************)

Function Min (a,b : integer) : integer;

begin
    if (a >= b) then Min:=b
                else Min:=a
end;

(*****************************************************************)
```

```
Function Max (a,b : integer) : integer;

begin
    if (a >= b) then Max:=a
                else Max:=b
end;

(*****************************************************************)

Function Min3 (a,b,c : integer) : integer;

Var t : integer;

begin
    if (a < b) then t:=a
               else t:=b;
    if (t < c) then Min3:=t
               else Min3:=c
end;

(*****************************************************************)

Function Max3 (a,b,c : integer) : integer;

Var t : integer;

begin
    if (a > b) then t:=a
               else t:=b;
    if (t > c) then Max3:=t
               else Max3:=c
end;

(*****************************************************************)

Function Sign (a : real) : integer;

begin
    if (a >= 0) then Sign:=1
                else Sign:=-1
end;

(*****************************************************************)

Procedure Cubic;

Var Queue                  : QueueType;
    SearchOffset,
     PosX,PosY,NegX,NegY   : integer;
    Orgn                   : PointIndex;
    ch                     : char;

    xreg    origin 164400B : integer;
    yreg    origin 164402B : integer;
```

```
    datareg origin 164404B : integer;

(*================================================================*)
(* This procedure sets the pixel at position (x,y) in the        *)
(* 256x256 window where the threshold surface is being drawn     *)
(* to the value z.                                                *)
(*================================================================*)

Procedure PutAPixel  (x,y,z : integer);

begin
    xreg   :=x + XOffSet;
    yreg   :=y + YOffSet;
    datareg:=z
end;

(*================================================================*)
(* This procedure initialises the queue used in the               *)
(* triangulation routine.                                         *)
(*================================================================*)

Procedure InitQueue;

begin
    Queue.Head:=0;
    Queue.tail:=0
end;

(*================================================================*)
(* This function returns a value indicating whether the queue    *)
(* is empty or not.                                               *)
(*================================================================*)

Function Empty : boolean;

begin
    Empty:=Queue.Head = Queue.Tail
end;

(*================================================================*)
(* This function returns the length of the queue.                 *)
(*================================================================*)

Function Length : integer;

Var Interval : integer;

begin
    Interval:=Abs(Queue.Head - Queue.Tail);
    if (Queue.head <= Queue.Tail) then Length:=Interval
    else Length:=QueueSize - Interval
end;

(*================================================================*)
(* This procedure gets the edge at the front of the queue.        *)
```

```
(*=============================================================*)

Procedure GetHead (Var HeadLink : LinkType);

begin
    if Empty then writeln('Queue underflow')
    else with Queue do
       begin
        HeadLink:=Link[Head];
        Head:=(Head + 1) mod Queuesize
       end
end;

(*=============================================================*)
(* This procedure adds an edge to the back of the queue.       *)
(*=============================================================*)

Procedure AddToTail (NewSource,NewSink:PointIndex);

begin
    with Queue.Link[Queue.Tail] do
       begin
        Source:=NewSource;
        Sink:=NewSink
       end;
    Queue.Tail:=(Queue.Tail + 1) mod QueueSize;
    if Empty then writeln('Queue overflow')
end;

(*=============================================================*)
(* This function determines whether there are any redundant    *)
(* edges in the queue or not.                                  *)
(*=============================================================*)

Function AbleToErase (SearchSource,SearchSink : PointIndex)
                                                    : boolean;

Var Pointer : QueueIndex;
    Found   : boolean;

begin
    found:=false;
    with Queue do
       begin
        Pointer:=Head;
        while (Pointer <> Tail) do
           begin
            found:= found or ((SearchSource=Link[Pointer].Source)
                        and (SearchSink  =Link[Pointer].Sink ));
            if found then
                Link[Pointer]:=Link[(Pointer+1) mod QueueSize];
            Pointer:=(Pointer + 1) mod QueueSize
           end;
        if found then Tail:=(Tail + QueueSize - 1) mod QueueSize
       end;
```

```
    AbleToErase:=found
end;

(*=================================================================*)
(* This function returns the number of the next generating       *)
(* point to be triangulated.                                       *)
(*=================================================================*)

Function PingPong : integer;

Var TestPoint,x,y,i : integer;
    finished        : boolean;
    side            : integer;

begin
    repeat
        TestPoint:=0;
        for side:=1 to 2 do
            if (TestPoint = 0) then
                if (SearchOffset > 0) then
                   begin
                    SearchOffset:=0 - SearchOffset;
                    TestPoint:=orgn + SearchOffset;
                    if (TestPoint >= 1) then
                       begin
                        x:=Coordinates[TestPoint].x;
                        y:=Coordinates[TestPoint].y;
                        if (y < NegY) then TestPoint:=0
                       end
                    else TestPoint:=0
                   end
                else
                   begin
                    SearchOffset:=1 - SearchOffset;
                    TestPoint:=orgn + SearchOffset;
                    if (TestPoint <= NumberOfPoints) then
                       begin
                        x:=Coordinates[TestPoint].x;
                        y:=Coordinates[TestPoint].y;
                        if (y > PosY) then TestPoint:=0
                       end
                    else TestPoint:=0
                   end;

        if (TestPoint = 0)
            then Finished:=true
            else Finished:=(x <= PosX) and (x >= NegX)

       until Finished;

    PingPong:=TestPoint
end;

(*=================================================================*)
```

```
(* This function returns the number of the generating point      *)
(* nearest to generating point number ORGN.                      *)
(*=============================================================*)

Function Nearest : integer;

Var ClosestFound          : PointIndex;
    SearchPoint,Limit     : integer;
    SqrDistance,SqrLimit  : real;

begin
    SearchOffset:=0;
    SqrLimit    :=1E38;
    Limit       :=maxint;

    NegX:=0;        NegY:=0;
    PosX:=255;      PosY:=255;

    repeat
        repeat
            SearchPoint:=PingPong;
            if (SearchPoint <> 0) then
                SqrDistance:=sqr(Coordinates[orgn].x -
                                 Coordinates[SearchPoint].x+0.0) +
                             sqr(Coordinates[orgn].y -
                                 Coordinates[SearchPoint].y+0.0)
           until (SqrDistance < SqrLimit) or (SearchPoint = 0);

        if (SearchPoint <> 0) then
           begin
            SqrLimit:=SqrDistance;
            Limit:=trunc(sqrt(SqrLimit)) + 1;
            PosX:=Min(255,Coordinates[orgn].x + Limit);
            NegX:=Max(0,  Coordinates[orgn].x - Limit);
            PosY:=Min(255,Coordinates[orgn].y + Limit);
            NegY:=Max(0,  Coordinates[orgn].y - Limit);
            ClosestFound:=SearchPoint
           end
       until (SearchPoint = 0);

    Nearest:=ClosestFound
end;

(*=============================================================*)
(* This procedure computes the next triangle (or triangles in  *)
(* event of a degeneracy) to be included in the triangulation. *)
(*=============================================================*)

Procedure Triangle (    BaseLink   : LinkType;
                    Var VertexList : PointArray;
                    Var Degeneracy : DegenFactor);

Var x1,x2,x3,y1,y2,y3               : integer;
    SqrA,SqrB,SqrC                  : real;
    Index,Radius,XCircumCentre,
```

```
     YCircumCentre                    : integer;
    XMid,YMid,XHalf,YHalf,
     MinimumCosA,CosATest,CosA,SinA,
     CosB,Ratio,Delta,Temp            : real;
    VertexIndex,Slot                  : DegenFactor;
    SlotSet                           : boolean;
    CosBList                          : array [DegenFactor] of real;

begin
    SearchOffset:=0;
    orgn:=round((BaseLink.Source + BaseLink.Sink)/ 2.0);
    x1:=Coordinates[BaseLink.Source].x;
    x2:=Coordinates[BaseLink.Sink  ].x;
    y1:=Coordinates[BaseLink.Source].y;
    y2:=Coordinates[BaseLink.Sink  ].y;
    SqrA:=sqr(x1-x2+0.0) + sqr(y1-y2+0.0);
    XMid :=(x1+x2) / 2.0;
    YMid :=(y1+y2) / 2.0;
    XHalf:=(x2-x1) / 2.0;
    YHalf:=(y2-y1) / 2.0;
    PosX :=255;       PosY:=255;
    NegX :=0;         NegY:=0;
    Delta:=2E-6;
    MinimumCosA:=2.0;
    CosBList[1]:=2.0;
    CosATest:=MinimumCosA + Delta;
    Degeneracy:=0;
    Index:=orgn;

    while (index <> 0) do
       begin
        if (index <> BaseLink.Source) and
           (index <> BaseLink.Sink) then
           begin
            x3:=Coordinates[index].x;
            y3:=Coordinates[index].y;
            if ((y2*(x3-x1+0.0) + y1*(x2-x3+0.0) +
                 y3*(x1-x2+0.0)) < 0.0) then
               begin
                SqrB:=sqr(x3-x2+0.0) + sqr(y3-y2+0.0);
                SqrC:=sqr(x3-x1+0.0) + sqr(y3-y1+0.0);
                CosA:=(SqrB+SqrC-SqrA) / (2.0 * sqrt(SqrB*SqrC));
                if (CosA < CosATest) then
                   begin
                    CosB:=(SqrA+SqrC-SqrB) /
                          (2.0 * sqrt(SqrA*SqrC));
                    If (abs(CosA - MinimumCosA) > Delta) then
                       begin
                        CosBList[1]:=CosB;
                        VertexList[1]:=Index;
                        MinimumCosA:=CosA;
                        CosATest:=CosA + Delta;
                        Degeneracy:=1;
                        SinA:=sqrt(1.0 - sqr(CosA));
                        If (SinA < 1E-4) then Radius:=255
```

```
                        else
                           begin
                            Temp:=sqrt(SqrA)/(2.0*SinA);
                            if (Temp < 254.0)
                                then Radius:=1+trunc(Temp)
                                else Radius:=255
                           end;
                        if (SinA = 0.0)
                            then Ratio:=255.0
                            else Ratio:=CosA/SinA;
                        Temp:=XMid - YHalf*Ratio;
                        if (temp < 0.0)
                            then XCircumcentre:=0
                            else if (temp > 255)
                                then XCircumcentre:=255
                                else XCircumcentre:=round(temp);
                        Temp:=YMid + XHalf*Ratio;
                        if (temp < 0.0)
                            then YCircumcentre:=0
                            else if (temp > 255)
                                then YCircumcentre:=255
                                else YCircumcentre:=round(temp);
                        PosX:=Min(255,XCircumcentre+radius);
                        PosY:=Min(255,YCircumcentre+radius);
                        NegX:=Max(0,  XCircumcentre-radius);
                        NegY:=Max(0,  YCircumcentre-radius)
                       end
                    else
                       begin
                        SlotSet:=false;
                        Slot:=Degeneracy+1;
                        for VertexIndex:=1 to Degeneracy do
                        if (CosB>=CosBList[VertexIndex]) and
                           (not SlotSet) then
                           begin
                            Slot:=VertexIndex;
                            SlotSet:=true
                           end;
                        for VertexIndex:=Degeneracy downto Slot do
                           begin
                            CosBList[VertexIndex+1]:=
                                CosBList[VertexIndex];
                            VertexList[VertexIndex+1]:=
                                VertexList[VertexIndex]
                           end;
                        CosBList  [Slot]:=CosB;
                        VertexList[Slot]:=Index;
                        Degeneracy      :=Degeneracy + 1
                       end
                 end
              end
           end;
        Index:=PingPong
       end
end;
```

```
(*=================================================================*)
(* This function indicates whether the point (x,y) lies on the  *)
(* right hand side of the vector (x2-x1,y2-y1) passing through  *)
(* (x1,y1) or not.                                               *)
(*=================================================================*)

Function OnRightSideOfLine (x,x1,x2,y,y1,y2 : Ordinate) : boolean;

begin
    OnRightSideOfLine:=(y2*1.0*(x-x1) +
                        y1*1.0*(x2-x) +
                        y *1.0*(x1-x2)) > 0.0
end;

(*=================================================================*)
(* This function determines whether the point (x,y) lies in the *)
(* triangle with generating points (x1,y1), (x2,y2) and (x3,y3) *)
(* or not.                                                       *)
(*=================================================================*)

Function PointInTriangle (x,x1,x2,x3,y,y1,y2,y3 : Ordinate)
                                                        : boolean;

begin
    if      OnRightSideOfLine(x,x1,x2,y,y1,y2)
        then PointInTriangle:=false
        else if OnRightSideOfLine(x,x2,x3,y,y2,y3)
            then PointInTriangle:=false
            else if OnRightSideOfLine(x,x3,x1,y,y3,y1)
                then PointInTriangle:=false
                else PointInTriangle:=true
end;

(*=================================================================*)
(* This procedure calculates the parameters for the cubic       *)
(* passing through the points (x0,y0) and (x1,y1).              *)
(*=================================================================*)

Procedure FitCubic (    x0,x1,y0,y1   : real;
                        z0,z1         : real;
                    Var Deriv0,Deriv1 : DerivPair;
                    Var Cubic         : CubicType);

Var x10,y10,z10,CommonTerm : real;

begin
    x10:=x1 - x0;
    y10:=y1 - y0;
    z10:=z1 - z0;
    CommonTerm:=abs(x10) + abs(y10);

    with Cubic do
       begin
        a:=z0;
```

```
        b:=Deriv0.ddx*x10 + Deriv0.ddy*y10;
        c:=3*z10 - 2*b - Deriv1.ddx*x10 - Deriv1.ddy*y10;
        d:=z10 - b - c;
        q:=3 * d * Sign(x10) / CommonTerm;
        s:=3 * d * Sign(y10) / CommonTerm;
        p:=Deriv1.ddx - Deriv0.ddx - q;
        r:=Deriv1.ddy - Deriv0.ddy - s;
        Deriv:=Deriv0
       end
end;

(*=============================================================*)

Procedure EvaluateCubic (    V      : real;
                             Cubic  : CubicType;
                         Var Z      : real;
                         Var VDeriv : DerivPair);

begin
    with Cubic do
       begin
        Z          :=A + V*(B + V*(C + V*D));
        VDeriv.ddx:=Deriv.ddx + V*(P + V*Q);
        VDeriv.ddy:=Deriv.ddy + V*(R + V*S)
       end
end;

(*=============================================================*)
(* This procedure computes the cubic patch over the triangle    *)
(* whose vertices are given by VERTEX1, VERTEX2 and VERTEX3.    *)
(*=============================================================*)

Procedure BiCubicPatch (Vertex1,Vertex2,Vertex3 : PointIndex);

Var x1,x2,x3,x,
     y1,y2,y3,y          : Ordinate;
    v0,v1,v2,
     z1,z2,z3,z,
     a1,a2,b1,b2,c1,c2,
    TwiceArea            : real;
    TriBox               : Rectangle;
    P1Derivs,P2Derivs,
     P3Derivs            : DerivPair;
    P2ToP1Cubic,
     P3ToP1Cubic,
     CrossCubic          : CubicType;
    Deriv21,Deriv31      : DerivPair;
    z21,z31              : real;

begin
    With Coordinates[Vertex1] do
       begin
        x1:=x; y1:=y; z1:=z
       end;
```

```
With Coordinates[Vertex2] do
   begin
    x2:=x; y2:=y; z2:=z
   end;

With Coordinates[Vertex3] do
   begin
    x3:=x; y3:=y; z3:=z
   end;

With TriBox do
   begin
    MaxX:=Max3(x1,x2,x3);
    MaxY:=Max3(y1,y2,y3);
    MinX:=Min3(x1,x2,x3);
    MinY:=Min3(y1,y2,y3);

    a1:=x2*1.0*y3 - x3*1.0*y2;
    a2:=x3*1.0*y1 - x1*1.0*y3;
    b1:=y2-y3;
    b2:=y3-y1;
    c1:=x3-x2;
    c2:=x1-x3;

    TwiceArea:=a1 + x1*b1 + y1*c1;

    P1Derivs.ddx:=Coordinates[Vertex1].ddx;
    P1Derivs.ddy:=Coordinates[Vertex1].ddy;
    P2Derivs.ddx:=Coordinates[Vertex2].ddx;
    P2Derivs.ddy:=Coordinates[Vertex2].ddy;
    P3Derivs.ddx:=Coordinates[Vertex3].ddx;
    P3Derivs.ddy:=Coordinates[Vertex3].ddy;

    FitCubic(x2,x1,y2,y1,z2,z1,P2Derivs,P1Derivs,P2ToP1Cubic);
    FitCubic(x3,x1,y3,y1,z3,z1,P3Derivs,P1Derivs,P3ToP1Cubic);

    for X:=MinX to MaxX do
    for Y:=MinY to MaxY do
        if (Coordinates[Vertex1].x=X) and
           (Coordinates[Vertex1].y=y) then
            PutAPixel(X,Y,trunc(Coordinates[Vertex1].z))
        else
        if (Coordinates[Vertex2].x=X) and
           (Coordinates[Vertex2].y=y) then
            PutAPixel(X,Y,trunc(Coordinates[Vertex2].z))
        else
        if (Coordinates[Vertex3].x=X) and
           (Coordinates[Vertex3].y=y) then
            PutAPixel(X,Y,trunc(Coordinates[Vertex3].z))
        else
        If PointInTriangle(X,x1,x2,x3,Y,y1,y2,y3) then
           begin
            v1:=(a1 + b1*X + c1*Y) / TwiceArea;
            v2:=(a2 + b2*X + c2*Y) / TwiceArea;
            v0:=v2 / (1-v1);
```

```
                EvaluateCubic(v1,P2ToP1Cubic,z21,Deriv21);
                EvaluateCubic(v1,P3ToP1Cubic,z31,Deriv31);
                FitCubic(v1*x1 + (1-v1) * x3,
                         v1*x1 + (1-v1) * x2,
                         v1*y1 + (1-v1) * y3,
                         v1*y1 + (1-v1) * y2,
                         z31,z21,
                         Deriv31,Deriv21,CrossCubic);

                With CrossCubic do
                    Z:=a + v0*(b + v0*(c + v0*d));
                PutAPixel(x,y,trunc(z))
               end
       end
end;

(*=============================================================*)
(* This procedure computes the triangles in the Delaunay      *)
(* Triangulation of the generating points and then calls      *)
(* BiCubicPatch to interpolate the threshold surface over each *)
(* triangle.                                                  *)
(*=============================================================*)

Procedure DelaunayTriangulation;

Var BaseLink     : LinkType;
    VertexList   : PointArray;
    Degeneracy   : DegenFactor;
    DegenIndex   : DegenFActor;
    Vertex       : integer;
    LastVertex   : PointIndex;

begin
    InitQueue;
    orgn:=NumberOfPoints div 2;
    Vertex:=Nearest;

    if (Vertex <> 0) then
       begin
        AddToTail(NumberOfPoints div 2,Vertex);
        AddToTail(Vertex,NumberOfPoints div 2);
        while not Empty do
           begin
            GetHead(BaseLink);
            Triangle(BaseLink,VertexList,Degeneracy);
            LastVertex:=BaseLink.Sink;
            if (Degeneracy <> 0) then
               begin
                for DegenIndex:=1 to Degeneracy do
                   begin
                    Vertex:=VertexList[DegenIndex];
                    BiCubicPatch
                        (BaseLink.Source,LastVertex,Vertex);
                    if not AbleToErase(LastVertex,Vertex) then
```

```
                        AddToTail(Vertex,LastVertex);
                      LastVertex:=Vertex
                     end;
                  if not AbleToErase(Vertex,BaseLink.Source) then
                      AddToTail(BaseLink.Source,Vertex)
                end
            end
        end
end;

(*=============================================================*)

begin
    DelaunayTriangulation
end;
```

REFERENCES

[1] Ahuja N., An B., Schachter B.

"Image Representation using Voronoi Tessellation"

Computer Vision, Graphics and Image Processing;

Volume 29; 1985; Pages 286-295

[2] Ahuja N., Rosenfeld A

"A Note on the use of Second-Order Gray-Level Statistics for Threshold Selection"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; Number 12; December 1978; Pages 895-898

[3] Anderberg M.R.

"Cluster Analysis for Applications"

Academic Press; New York; 1973

[4] Bailey T., Jain A.K.

"A Note on Distance-weighted K-nearest Neighbour Rules"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; 1978; Pages 311-313

[5] Baily N.A., Rimkus D., Nalcioglu O.

"Normalization (Division) versus Subtraction of Digitized Images"

Medical Physics; Volume 10; 1983; Pages 25-28

[6] Balakrishnan A.V., Neustadt L.W.

"Computing Methods in Optimization Problems"

Academic Press; New York; 1964

[7] Biemond I., Links L.H., Boekee D.E.
"Image Modeling and Quality Criteria"
IEEE Transactions on Acoustics, Speech and Signal Processing;
Volume ASSP-27; 1979; Pages 649-652

[8] Bogren H.G., Bürsch J.H., et al
"Intravenous Angiocardiography using Digital Image Processing: Experience with Axial Projections in Normal Pigs with Experimentally Generated Left-to-Right Shunts"
SPIE; Volume 314; Digital Radiography; 1981;
Pages 287-293

[9] Bower A.
"Computing Dirichlet Tessellations"
The Computer Journal; Volume 24; 1981; Pages 162-166

[10] Brennecke R., Brown T.K., et al
"Digital Processing of Videoangiocardiographic Image Series using a Minicomputer"
Proceedings on Computer Cardiology;
IEEE Computer Society; 1976; Pages 255-260

[11] Brennecke R., Brown T.K., et al
"Computerized Video-Image Preprocessing with Applications to Cardio-Angiographic Roentgen-Image Series"
Digitale Bildverarbeitung; Editor: Nagel, H.H.;
Springer-Verlag; 1977; Pages 244-262

[12] Brody W.R.

"Hybrid Subtraction for Improved Arteriography"

Radiology; Volume 141; 1981; Pages 828-831

[13] Burley D.M.

"Studies in Optimisation"

Intertext Books; Heath and Reach; 1974

[14] Caceci M.S., Cacheris W.P.

"Fitting Curves to Data: The Simplex Method is the Answer"

Byte; May 1984; Pages 340-362

[15] Chow C.K., Kaneko T.

"Automatic Boundry Detection of the Left Ventricle from Cineangiograms"

Comput. Biomed. Res.; Volume 5; August 1972; Pages 388-410

[16] Crummy A.B., Strother C.M., et al

"Computerised Fluoroscopy: Digital Subtraction for Intravenous Angiocardiography and Arteriography"

American Journal of Radiology; Volume 135; December 1980; Pages 1131-1140

[17] Danzig G.B., Eaves B.C.

"Studies in Optimization"

The Mathematical Association of America"; 1974

[18] Davis L.S., Rosenfeld A., Weszka J.S.
"Region Extraction by Averaging and Thresholding"
IEEE Transactions on Systems, Man and Cybernetics;
Volume SMC-5; May 1975; Pages 383-388

[19] Delattre M., Hansen P.
"Bicriterion Cluster Analysis"
IEEE Transactions on Pattern Analysis and Machine
Intelligence; Volume PAMI-2; 1980; Pages 277-291

[20] Delaunay B.
"Sur la Sphére Vide"
Bulletin de L'Academie des Sciences de L'URSS;
Classe de Sciences Mathématiques et Naturtelles;
Volume 6; 1934; Pages 793-800

[21] De Souza P.
"Edge Detection using Sliding Statistical Tests"
Computer Vision, Graphics and Image Processing;
Volume 23; 1983; Pages 1-14

[22] Dixon L.C.W.
"Nonlinear Optimisation"
The English Universities Press Limited; London; 1972

[23] Dudani S.A.
"The Distance-Weighted K-nearest Neighbour Rule"
IEEE Transactions on Systems, Man and Cybernetics;
Volume SMC-6; 1976; Pages 325-327

[24] Eklundh J.O. Yamamoto H., Rosenfeld A.
"A Relaxation method for Multispectral Pixel Classification"
IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 72-75

[25] Enzmann D.R., Djang W.T., et al
"Low-Dose, High-Frame-Rate versus Regular-Dose, Low-Frame-Rate Digital Subtraction Angiography"
Radiology; Volume 146; March 1983; Pages 669-676

[26] Everitt B.
"Cluster Analysis"
Heinemann Educational Books Limited; London; 1974

[27] Farrelle P.M., Jain A.K.
"Recursive Block Coding - A New Approach to Transform Coding"
IEEE Transactions on Communications; Volume COM-34; 1986 Pages 161-179

[28] Fernando G.M.X.
"Variable Thresholding of Images with Application to Ventricular Angiograms"
Ph.D. thesis; Imperial College of Science and Technology; London; 1985

[29] Fredendall G.L., Behrend W.L.
"Picture Quality - Procedures for Evaluating Subjective Effects of Interference"
Proceedings of the IRE; Volume 48; 1960; Pages 1030-1034

[30] Fu K.S., Mui J.K.

"A Survey on Image Segmentation"

Pattern Recognition; Volume 13; 1981; Pages 3-16

[31] Gill P.E., Murray W., Wright M.H.

"Practical Optimization"

Academic Press; London; 1981

[32] Goldberg M., Boucher P.R., Shlien S.

"Image Compression Using Adaptive Vector Quantization"

IEEE Transactions on Communications; Volume COM-34; 1986

Pages 180-187

[33] Goldberg M., Shlien S.

"A Clustering Scheme for Multispectral Images"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; 1978; Pages 86-92

[34] Gonzalez R.C., Wintz P.

"Digital Image Processing"

Addison-Wesley Publishing Company; Reading, Ma.; 1979

[35] Gowda K.C., Krishna G.

"Disaggregative Clustering using the Concept of Mutual

Nearest Neighbourhood"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; 1978; Pages 888-895

[36] Gowda K.C., Krishna G.

"Agglomerative Clustering using the Concept of Mutual Nearest Neighbourhood"

Pattern Recognition; Volume 10; 1978; Pages 105-112

[37] Granlund G.H.

"In Search of a General Picture Processing Operator"

Computer Graphics and Image Processing; Number 8; 1978; Pages 155-173

[38] Granlund G.H., Knutsson H., Wilson R.

"Image Enhancement"

Fundamentals in Computer Vision; Edited by O.D. Faugeras; Cambridge University Press; 1983

[39] Green P.J., Sibson R.

"Computing Dirichlet Tessellations in the Plane"

The Computer Journal; Volume 21; 1978; Pages 168-173

[40] Haralick R.M.

"The Digital Edge"

IEEE Computer Society Conference on Pattern Recognition and Image Processing; Dallas, Texas; 1981; Pages 285-291

[41] Hueckel M.H.

"An Operator which Locates Edges in Digitized Pictures"

Journal of the ACM; Volume 18; 1971; Pages 113-125

[42] Hueckel M.H.

"A Local Visual Operator which Recognizes Edges and Lines"

Journal of the ACM; Volume 20; 1973; Pages 634-647

[43] Hummel R.A., Rosenfeld A.

"Relaxation Processes for Scene Labeling"

IEEE Transactions on Systems, Man and Cybernetics; Volume SMC-8; 1978; Pages 765-768

[44] Jain A.K.

"Cluster Analysis"

Handbook of Pattern Recognition and Image Processing; Academic Press; Orlando, Florida; 1986; Pages 33-57

[45] Jain A.K., Smith S.P., Backer E.

"Segmentation of Muscle Cell Pictures: A Preliminary Study"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 232-242

[46] Jardine N., Sibson R.

"The Construction of Hierarchic and Non-Hierarchic Classifications"

The Computer Journal; Volume 11; 1968; Pages 177-184

[47] Jardine N., Sibson R.

"Choice of Methods for Automatic Classification"

The Computer Journal; Volume 14; 1971; Pages 404-406

[48] Kapur J.N., Sahoo P.K., Wong A.K.C.
"A New Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram"
Computer Vision, Graphics and Image Processing; Volume 29; Number 3; March 1985; Pages 273-285

[49] Kendall M.G., Buckland W.R.
"A Dictionary of Statistical Terms"
Longman Group Limited; London; 1982

[50] Kindelan M., de Lezo J.S.
"Artery Detection and Tracking in Coronary Angiography"
Digital Image Analaysis; Edited by S. Levialdi; Pitman Publishing Limited; London; 1984

[51] Kirby R.L., Rosenfeld A.
"A Note on the Use of (Gray Level, Local Average Gray Level) Space as an aid in Threshold Selection"
IEEE Transaction on Systems Man and Cybernetics; Volume SMC-9; Number 12; December 1979; Pages 860-864

[52] Knutsson H., Wilson R., Granlund G.H.
"Estimating the Local Orientation of Anisotropic 2-D Signals"
IEEE Conference on Pattern Recognition; 1983; Pages 234-239

[53] Knutsson H., Wilson R., Granlund G.H.
"Anisotropic Filtering Operations for Image Enhancement and their Relation to the Visual System"
IEEE Conference on Pattern Recognition; 1983; Pages 420-423

[54] Knutsson H.E., Wilson R., Granlund G.H.
"Anisotropic Nonstationary Image Estimation and its Applications: Part I - Restoration of Noisy Images"
IEEE Transactions on Communications; Volume COM-31; 1983; Pages 388-397

[55] Kruger R.A., Miller F.J., et al
"Digital Subtraction Angiography using a Temporal Bandpass Filter: Associated Patient Motion Properties"
Radiology; Volume 145; 1982; Pages 315-320

[56] Kruger R.A., Mistretta C.A., Riederer S.J.
"Physical and Technical Considerations of Computerized Fluoroscopy Difference Imaging"
IEEE Transactions on Nuclear Science; Volume NS-28; February 1981; Pages 205-212

[57] Kruskal J.B.
"On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem"
Proceedings of the American Mathematical Society; Volume 7; Number 1; February 1956; Pages 48-50

[58] Kunzi H.P., Tzschach H.G., Zehnder C.A.
"Numerical Methods for Optimization"
Academic Press; New York; 1971

[59] Lai P.F., Ehrich R.W.
"Segmentation of Images with Incompletely Specified Regions"
IEEE transactions on Systems, Man and Cybernetics; Volume SMC-9; 1979; Pages 864-868

[60] Lance G.N., Williams W.T.
"A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems"
The Computer Journal; Volume 9; 1967; Pages 373-380

[61] Lance G.N., Williams W.T.
"Note on the Classification of Multi-Level data"
The Computer Journal; Volume 9; 1967; Pages 381-382

[62] Lance G.N., Williams W.T.
"Note on a New Information-Statistic Classificatory Program"
The Computer Journal; Volume 11; 1968; Page 195

[63] Lee H.C., Fu K.S.

"The GLGS Image Representation and its Application to Preliminary Segmentation and Pre-attentive Visual Search"

IEEE Computer Society Conference on Pattern Recognition and Image Processing; Dallas, Texas; 1981; Pages 256-261

[64] Linnaeus C.

"Genera Plantarum"

Conrad Wishoff; Leyden; 1737

[65] Loo L-N.D., Doi K., Metz C.E.

"A Comparison of Physical Image Quality Indices and Observer Performance in the Radiographic Detection of Nylon Beads"

Physics in Medicine and Biology; Volume 29; 1984; Pages 837-856

[66] McLain D.H.

"Two Dimensional Interpolation from Random Data"

The Computer Journal; Volume 19; 1976; Pages 178-181

[67] Meaney T.F., Weinstein M.A., et al

"Digital Subtraction Angiography of the Human Cardiovascular System"

American Journal of Radiology; Volume 135; December 1980 Pages 1153-1160

[68] Mistretta C.A.

"Current Practice and Future Directions in Digital Subtraction Angiography"

SPIE; Volume 314; Digital Radiography; 1981; Pages 18-23

[69] Mitchell O.R., Carlton S.G.

"Image Segmentation using a Local Extrema Texture Measure"

Pattern Recognition; Volume 10; 1978; Pages 205-210

[70] Mizoguchi R., Shimura M.

"A Nonparametric Algorithm for Detecting Clusters using Hierarchical Structure"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 292-300

[71] Modic M.T., Meredith A.W., et al

"Intravenous Digital Angiography of the Head and Neck: a Clinical Update"

SPIE; Volume 314; Digital Radiography; 1981; Pages 244-247

[72] Monro D.M.

"Contouring Software for the European Centre for Medium Range Weather Forecasts"

Program Logic Manual; 1976

[73] Monro D.M.

"Practical Algorithms for Triangulation of Points in a Plane"

In preparation

[74] Nakagawa Y., Rosenfeld A.

"Edge/Border Coincidence as an aid in Edge Extraction"

IEEE Transactions on Systems, Man and Cybernetics; Volume SMC-8; 1978; Pages 899-901

[75] Nakagawa Y., Rosenfeld A.

"Some Experiments on Variable Thresholding"

Pattern Recognition; Volume 11; 1979; Pages 191-204

[76] Narendra P.M., Goldberg M.

"Image Segmentation with Directed Trees"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 185-191

[77] Nash J.C.

"Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation"

Adam Hilger Limited; Bristol; 1979

[78] Nelson J.A., Miller F.J., et al

"Digital Subtraction Angiography using a Temporal Bandpass Filter: Initial Clinical Results"

Radiology; Volume 145; November 1982; Pages 309-313

[79] Otsu N.

"A Threshold Selection Method from Gray-Level Histograms"

IEEE Transactions on Systems, Man and Cybernetics; Volume SMC-9; Number 1; January 1979; Pages 62-66

[80] Panda D.P., Rosenfeld A.

"Image Segmentation by Pixel Classification in (Gray Level, Edge Value) space"

IEEE Transactions on Computers; Volume C-27; Number 9; September 1978; Pages 875-879

[81] Peleg S.

"A New Probabilistic Relaxation Scheme"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; Number 4; July 1980; Pages 362-369

[82] Perkins W.A.

"Area Segmentation of Images using Edge Points"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 8-15

[83] Pong T.C., Shapiro L.G., Haralick R.M.

"A Facet Model Region Growing Algorithm"

IEEE Computer Society Conference on Pattern Recognition and Image Processing; Dallas, Texas; 1981; Pages 279-284

[84] Prager J.M.

"Extracting and Labeling Boundary Segments in Natural Scenes"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-2; 1980; Pages 16-27

[85] Pratt W.K.

"Digital Image Processing"

John Wiley and Sons; New York; 1978

[86] Pun T.

"A New Method for Grey-Level Picture Thresholding Using the Entropy of the Histogram"

Signal Processing; Volume 2; 1980; Pages 223-237

[87] Pun T.

"Entropic Thresholding: A New Approach"

Computer Graphics and Image Processing; Volume 16; 1981; Pages 210-239

[88] Ridler T.W., Calvard S.

"Picture Thresholding using an Iterative Selection Method"

IEEE Transaction on Systems, Man and Cybernetics; Volume SMC-8; Number 8; August 1978; Pages 630-632

[89] Riederer S.J., Belanger B.F., et al

"Iodine Sensitivity of Digital Imaging Systems"

SPIE; Volume 314; Digital Radiography; 1981; Pages 132-139

[90] Riederer S.J., DiBianca F.A., et al

"Performance Characteristics of a Digital Fluorographic System"

SPIE; Volume 273; Applications of Optical Instrumentation in Medicine IX; 1981; Pages 88-95

[91] Rosenfeld A.

"Iterative Histogram Modification"

IEEE Transactions on Systems, Man and Cybernetics; Volume SMC-8; Number 4; April 1978; Pages 300-302

[92] Rosenfeld A.

"Iterative Methods in Image Analysis"

Pattern Recognition; Volume 10; 1978; Pages 181-187

[93] Rosenfeld A., Hummel R.A., Zucker S.W.

"Scene Labelling by Relaxation Operations"

IEEE Transactions of Systems, Man and Cybernetics; Volume SMC-6; 1976; Pages 420-433

[94] Rosenfeld A., Kak A.C.

"Digital Picture Processing"

Volumes 1 and 2; Academic Press; New York; 1982

[95] Rosenfeld A., Smith R.C.

"Thresholding using Relaxation"

IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-3; Number 5; September 1981; Pages 598-606

[96] Saghri J.A., Tescher A.G.

"Adaptive Transform Coding Based on Chain Coding Concepts"

IEEE Transactions on Communications; Volume COM-34; 1986

Pages 112-117

[97] Schachter B.J., Rosenfeld A., Davis L.S.

"Random Mosaic Models for Textures"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; 1978; Pages 694-702

[98] Seeger J.F., Carmody R.F., et al

"Digital Intravenous Cerebral Angiography"

SPIE; Volume 314; Digital Radiography; 1981;

Pages 239-243

[99] Shannon C.E.

"A Mathematical Theory of Communication - I"

Bell Systems Technical Journal; Volume 27; 1948;

Pages 379-423, 623-656

[100] Sibson R,

"Some Observations on a Paper by Lance and Williams"

The Computer Journal; Volume 14; 1971; Pages 156-157

[101] Sibson R.

"Order Invariant Methods for Data Analysis"

Journal of the Royal Statistical Society; Series B;

Volume 34; 1972; Pages 311-349

[102] Sibson R.

"SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method"

The Computer Journal; Volume 16; 1973; Pages 30-34

[103] Sibson R.

"Locally Equiangular Triangulations"

The Computer Journal; Volume 21; 1978; Pages 243-245

[104] Sklansky J.

"Image Segmentation and Feature Extraction"

IEEE Transactions on Systems, Man and Cybernetics; Volume SMC-8; 1978; Pages 237-247

[105] Slump C., Hoenders B.J.

"The Determination of the Location of the Global Maximum of a Function in the Presence of Several Local Extrema"

IEEE Transactions on Information Theory; Volume IT-31; 1985; Pages 490-497

[106] Stieghorst M.F., Crummy A.B., et al

"Digital Subtraction Angiography in Peripheral Vascular Disease"

SPIE; Volume 314; Digital Radiography; 1981; Pages 299-300

[107] Strother C.M., Stieghorst M.F., et al

"Intraarterial Digital Subtraction Angiography"

SPIE; Volume 314; Digital Radiography; 1981;

Pages 235-238

[108] Takiyama R.

"Multiple Threshold Perceptron"

Pattern Recognition; Volume 10; 1978; Pages 27-30

[109] Therrien C.W., Quatieri T.F., Dudgeon D.E.

"Statistical Model-Based Algorithms for Image Analysis"

Proceedings of the IEEE; Volume 74; 1986; Pages 532-551

[110] Thompson W.B.

"Combining Motion and Contrast for Segmentation"

IEEE Transactions on Pattern Analysis and Machine

Intelligence; Volume PAMI-2; 1980; Pages 543-549

[111] Tomek I.

"An Experiment with the Edited Nearest-Neighbour Rule"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-6; 1976; Pages 448-452

[112] Tsai W-H.

"Moment Preserving Thresholding: A New Approach"

Computer Vision, Graphics and Image Processing;

Volume 29; Number 3; March 1985; Pages 377-393

[113] Venot A., Lebruchec J.F., Roucayrol J.C.

"A New Class of Similarity Measures for Robust Image Registration"

Computer Vision, Graphics, and Image Processing; Volume 28; 1984; Pages 176-184

[114] Venot A., Leclerc V.

"Automated Correction of Patient Motion and Gray Values Prior to Subtraction in Digitised Angiography"

IEEE Transactions on Medical Imaging; Volume MI-3; 1984; Pages 179-186

[115] Watson D.F.

"Computing the N-dimensional Delaunay Tessellation with Application to Voronoi Polytopes"

The Computer Journal; Volume 24; 1981; Pages 167-172

[116] Weszka J.S.

"A Survey of Threshold Selection Techniques"

Computer Graphics and Image Processing; Volume 7; 1978; Pages 259-265

[117] Weszka J.S., Nagel R.N., Rosenfeld A.

"A Threshold Selection Technique"

IEEE Transactions on Computers; Volume C-23; 1974; Pages 1322-1326

[118] Weszka J.S., Rosenfeld A.

"Threshold Evaluation Techniques"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-8; Number 8; August 1978; Pages 622-629

[119] Weszka J.S., Rosenfeld A.

"Histogram Modification for Threshold Selection"

IEEE Transactions on Systems, Man and Cybernetics;

Volume SMC-9; Number 1; January 1979; Pages 38-52

[120] Wichmann B.A., Hill I.D.

"A Pseudo-random Number Generator"

National Physics Laboratory Report DITC 6/82; 1982

[121] Wiener N.

"Cybernetics"

Chapter 3; MIT Press and J. Wiley and Sons; New York 1961

[122] Williams W.T., Clifford H.T., Lance G.N.

"Group-Size Dependence: a Rationale for Choice between Numerical Classifications"

The Computer Journal; Volume 14; 1971; Pages 157-162

[123] Williams W.T., Lance G.N., et al

"Controversey Concerning the Criteria for Taxonometric Strategies"

The Computer Journal; Volume 14; 1971; Pages 162-165

[124] Wilson R.G., Knutsson H., Granlund G.H.
"Image Orientation Estimation and its Applications"
IEE Electronics Division Colloquium on Image Restoration and 2-D Filtering Techniques; London; 3 May 1985; Digest Number 1985/51; Professional Groups E4 and C8

[125] Wishart D.
"A Numerical Classification Method for Deriving Natural Classes"
Nature; Volume 221; 1969; Pages 97-98

[126] Wu A.Y., Hong T-H., Rosenfeld A.
"Threshold Selection Using Quadtrees"
IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-4; Number 1; January 1982; Pages 90-94

[127] Young H.E.
"Interpolation of Irregularly Sampled Surface Data"
M.Sc Thesis; Imperial College of Science and Technology; London; 1985

[128] Young T.Y., Fu K-S.
"Handbook of Pattern Recognition and Image Processing"
Academic Press; Orlando, Florida; 1986

[129] Ziskin M.C., Revesz G., et al
"Spatial Frequency Spectra of Radiographic Images"
Radiology; Volume 98; March 1971; Pages 507-517