Contrast Enhancement using Grey Scale Transformation Techniques

By

Ralph Dale-Jones

Submitted for the degree of Doctor of Philosophy

to the Higher Degrees Committee

University of Warwick

Department of Engineering

University of Warwick, Coventry, U. K.

March 1993

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my thanks to my supervisor Dr. T. Tjahjadi, for his kindness, encouragement, and support, and for his suggestions and comments. I would also like to thank Dr. R. Staunton for his help while Dr. Tjahjadi was on sabbatical in Canada.

There are also many fellow students (and now Drs) at the University of Warwick to whom I would like to express my thanks: L. C. Carpinetti, C. C. Chan, Dr. T. Judge, M. Gongora, Dr. P. Gosling, Dr. C. Quan, W. Chiu, T. Partridge and Dr. K. G. Zheng.

Finally I would like to express my sincere thanks to Dr. M. Li.

Declaration

This thesis is presented in accordance with the regulations for the degree of Doctor of Philosophy by the Higher Degree Committee at the University of Warwick. The thesis has been composed and written by myself based on the research undertaken by myself. The research materials have not been submitted in any previous application for a higher degree. All sources of information are specifically acknowledged in the content.

R. Dale-Jones

Summary

The object of this thesis has been to examine grey scale transformation techniques in order to incorporate them into a system for automatically selecting a technique to enhance the contrast in a given image.

In order to include existing techniques in the system it was necessary to examine each in detail, and to understand under what conditions it gave good results. It was found that a number of techniques had only a limited scope or suffered from some problem in its design. This led to the development of a new technique based on the display capabilities of a monitor; the adaptation of another technique, global histogram equalisation, to make it applicable to a wider range of images and the modification of the local histogram equalisation algorithm to smooth different sized regions of the image to the same degree.

The resultant algorithms, together with those existing in the literature, were included in the system. The system provides an interactive environment for selecting grey scale transformation techniques. The usual method of choosing a contrast enhancement technique is to apply it, look at the result, discard it if the result is not suitable, or if there is a parameter value to be set, modify its value, and try the technique again. Here a more systematic approach is tried using ideas from Knowledge Based Systems and Object Oriented Systems. A model of the way contrast enhancement techniques are selected is encoded into the system and is used with information obtained by analysing the image (either automatic analysis done by the system, or interactive analysis done with the aid of the user) to select the most appropriate techniques. The techniques selected by the system have to fulfill three quite demanding criteria, ensuring that the system is a reliable and useful tool.

Chapter 1

Introduction

1.1 Introduction

The subject of this thesis is contrast enhancement using grey scale transformation techniques. Contrast enhancement is a sub-area of a field called digital image processing. Digital image processing is a wide ranging discipline with many application areas. Perhaps the most well known of these is its use in the interpretation of images from space, but it is equally important in industry (e.g. quality control), medicine (e.g. radiography) and other areas.

The origin of digital image processing lies in the 1920s, when it was developed as an aid to improving digitised newspaper pictures sent across the Atlantic by submarine cable (Gonzalez and Wintz 1987). These early systems were primitive by today's standards, but it was not until the space programme in the 1960s that major advances began to be made (Gonzalez and Wintz 1987).

In the early space missions analogue methods were used to enhance photographs (Sheldon 1987). It was Robert Nathan who suggested digitising video signals and using computers to remove defects (Sheldon 1987). It was from this beginning that digital image processing grew into a major area of research and application.

One aspect of digital image processing is digital image enhancement. The general purpose of digital image enhancement is to process an image to increase its usefulness. This may mean simply making the image more pleasant to look at (e.g.

television pictures); on the other hand the image may be required for some form of analysis e.g. the pictures from the Surveyor mission to Mars were processed to improve their contrast. The processed pictures contained much more detail of the lunar surface, and helped scientists choose a landing site for the Apollo mission (Green 1983).

Digital image enhancement is concerned with enhancing images affected by three types of degradation. The first class consists of images degraded by some form of noise; for example television pictures often suffer from this problem when transmitted in stormy weather. The second class consists of images that have been degraded by blurring, for example photographs taken during motion, or when the camera lens is not focused correctly. The third class consists of images with poor contrast. For example, if an image with a large dynamic range (i.e. it has a large number of distinguishable grey levels) is recorded on a medium with a small dynamic range (i.e. one only capable of displaying a few distinct greys), the contrast is reduced and the details in the image are obscured, particularly in very dark regions (Lim 1984).

It is this last area of digital image enhancement, contrast enhancement, that is the subject of the thesis. There are several definitions of contrast (Hall 1979). One that is widely used (O'Gorman and Brotman 1985) is given by

$$C = \frac{I - I_B}{I_B} \tag{1.1}$$

where I_B is the background grey level, and I is the pixel's grey level. Thus, in this definition, stretching a grey level away from its local background increases the contrast of the image, and in general makes the detail in the image more visible. This equation immediately poses two problems: how large should the separation $I - I_B$ between grey levels be, and how is the local background grey level I_B approximated. These are two of the most significant problems in contrast enhancement, for if $I - I_B$ is too large, or I_B is incorrectly approximated artefacts may be introduced into the image. To a certain extent these two problems are addressed in this thesis (the first problem in chapters 3 and 6, the second problem in chapter 5).

The main content of the thesis concerns grey scale transformation techniques. These techniques are comparatively simple to design, easy to program, and computationally extremely quick. As a consequence these methods are the most widely used in contrast enhancement, and (mainly because of their speed) the most convenient and useful methods. However this type of algorithm suffers from a number of drawbacks, the most glaring of which is the fact that they usually make no attempt to measure the actual contrasts in the image (Beghadi 1988). They are also often described as *ad hoc* methods, because they are designed for a particular purpose.

In this thesis two existing and widely used grey scale transformations, histogram equalisation and local histogram equalisation are examined in detail. Their shortcomings are pointed out and then they are modified to try to make them more applicable. Also a new grey scale transformation technique is introduced, which uses equation 1.1 as its basic premise. The purpose of this algorithm is to adapt the contrast of the image to suit the display capabilities of the monitor where it is being viewed.

All the transformations are then combined with those existing in the literature to produce a system for choosing the best grey scale transformation technique to enhance a given image. The aim of the thesis has been to examine existing contrast enhancement techniques in order to find their strengths and weaknesses, and to incorporate the resulting knowledge into this system.

1.2 Organisation of the thesis

Chapter two is a review of the contrast enhancement techniques in the literature; the emphasis is on grey scale transformation techniques and histogram modification techniques, but some other important types of transformation are also introduced.

Chapter three presents a grey scale transformation technique. This technique is designed to enhance the contrasts according to the display capabilities of the computer monitor where the image is being shown. Two experiments were used to determine the required (minimum) separation of adjacent grey level bins against a specified background grey level. Two algorithms based on these results have been implemented, and their results are discussed.

In chapter four the global histogram equalisation algorithm is discussed. It is shown that for certain types of grey scale distribution the algorithm is of no use. Four variants of the algorithm are presented. These algorithms extend the histogram equalisation technique to a wider range of grey scale distributions. The results of these algorithms are shown, and discussed.

An analysis of the local histogram equalisation transformation is presented in chapter five. The emphasis is on how the transformation distorts the image because of the way it is defined. As a result of this analysis a modified version of the algorithm, to overcome these problems, is suggested.

Chapter six is the culmination of the thesis. The algorithms presented in the previous chapters are brought together and used in a system for automatically selecting grey scale contrast enhancement techniques. This system uses ideas from knowledge based systems and object oriented systems in its implementation.

1.3 Note on monitors

Three monitors are used to display images in this thesis: the GDM-1604A40, GDM-1604-40, and Hitachi LR47156. Each is capable of displaying 256 grey levels. There are many different types of monitor available; these three happened to be in use at the University of Warwick during the course of this thesis. It was because of the different display capabilities of these monitors that the algorithm in Chapter three was developed.

Other monitors, particularly those used in dedicated image processing systems have a much better display capability than the ones used in this thesis. That is, adjacent grey levels, even at the low end of the grey level scale, can be easily distinguishable by the eye, and this is explored further in chapter 3. However, this does not detract from the relevance of the algorithms, which were designed for poor quality displays.

1.4 Note on the presentation of images

There were three alternatives available for the presentation of images. The simplest method is to use a PostScript laser printer. Unfortunately the output does not accurately represent the image because the printer is only capable of 16 grey levels, and these are achieved by using a dithering algorithm (i.e. black pixels are obtained by a high density of dots per pixel, a white pixel is obtained by printing no dots, and a grey pixel is achieved by varying the density between black and white). The result only very approximately represents the image, and usually appears blurred, though this improves if the printer has a high resolution (i.e. it is capable of printing a very high density of dots per pixel). A further problem is that an image may appear different when displayed on a monitor, but it will be exactly the same when printed out using the printer.

The second method is to use a special printer capable of displaying 256 grey levels on a type of thermal paper. These pictures are much more accurate than the results from a PostScript printer, although they may still be slightly blurred due to poor resolution. However the results still do not show the differences between displays.

The final option is photography. This is the best method for showing the difference between images displayed on different monitors, if the same degree of exposure is used throughout. However because of the limitations of the film, it is not always possible to get an accurate representation of the image (particularly in very dark or light areas).

In chapter 3 photography is used to show how the appearance of the image differs on different monitors. Chapter 5 also uses photography because printed images do not represent the image accurately. The photographs were taken using an ISO 100 black and white film with shutter speed of 1/8s, and an aperture setting of f 11. All photographs were taken in darkness. The developed negatives were printed on matt paper. The photographs were developed so that as much detail as possible was visible in the resultant prints. This has caused a slight reduction in the contrast; however, if the contrast was increased, detail visible on the screen would be lost in the development process. The prints shown in the thesis are the closest approximate to the images viewed on the screen, and faithfully show the effect of the algorithms. In chapters 4 and 6 PostScript images are used because the results are adequate to show the difference between the original and the processed image.

Chapter 2

A review of contrast enhancement techniques

2.1 Introduction

Contrast enhancement techniques can be broadly classified into two groups; global enhancement techniques and local enhancement techniques. Global enhancement techniques (also known as grey scale transformations) operate over the whole image, attempting to increase the contrast by spacing out the occupied grey levels, whereas local enhancement techniques operate over small (usually overlapping) sections of the image known as windows. The particular transformation used may be the same. For example, the well known histogram equalisation transformation may be used as either a global or a local operator. The results, however, may be vastly different. This difference is due to the fact that the global enhancement technique transforms every pixel of the same grey level to the same value, while the local transformation may map pixels of the same grey level to widely different places in the grey level scale due to local variations in the histogram.

Global enhancement techniques are easy to implement, extremely fast to compute, and the results are usually free from artefacts. On the other hand, they make no attempt to measure the contrasts in the image (Beghadi 1988); they assume that the global histogram is a good indicator of the contrasts in the image (although it contains no spatial information). This may be the case if the image is made up of distinct objects occupying non-overlapping regions of the grey scale; but, in general it is not true for complex images, which may contain many regions with overlapping grey scales. There is also a limited amount of space in the grey scale to space out the grey levels, so the transformation usually requires a loss of contrast in some areas.

Local algorithms also have several disadvantages: they are complicated, difficult to program, computationally very slow, and in some cases may introduce artefacts into the image. The artefacts are caused by incorrectly identifying the local background grey level. On the other hand, local enhancement techniques usually successfully enhance the detail in small areas of the image.

2.2 Grey scale transformations

Grey scale transformations are almost always applied globally, although in some cases it is both desirable and possible to extend them to operate locally. They are commonly used to spread a small range of grey levels over a much larger range. The pixels in the image in this grey level range will then be assigned new, more widely spread grey level values. Thus, if such pixels are adjacent in the image, they will become more easily distinguishable. This type of transformation maps individual pixels to values that are independent of the values of the surrounding pixels (Andrews 1976). Such a transformation is called an intensity or point process.

The image f occupies a discrete set of K grey levels, i.e. $f(x,y) \epsilon [0, \dots K-1]$. The grey scale transformation τ is here assumed to map the input image to an output image with the same number of grey levels. The transformation satisfies the following conditions (Kautsky *et al.* 1984, Gonzalez and Fittes 1977, Hummel 1975):

1. τ: $[0, 1, \dots, K-1] \rightarrow [0, 1, \dots, K-1]$.

2. τ is single valued.

3. τ is monotonic i.e. $i \leq j$ implies $\tau(i) \leq \tau(j)$.

Unfortunately the way a grey scale transformation is defined limits its usefulness. Since τ is monotonic and single valued every pixel in the image of the same value is transformed to the same value. The grey levels need to be spaced apart to become visible, so it is clear that it is unlikely that all bins can be separated because there will be insufficient space to fit them all in: some grey levels must be lost in order to create the space to separate the others.

2.2.1 Contrast stretching

This technique stretches the grey level range of the image over the whole of the available scale. The following stages are involved (Green 1983, Rosenfeld and Kak 1982):

1. Identify a lower bound and an upper bound of the grey levels used in the image, b_{low} and b_{high} respectively.
2. Map all the pixels with grey level $f(x,y)$, $f(x,y) < b_{low}$ to grey level 0.
3. Map all the pixels with grey level $f(x,y)$, $f(x,y) > b_{high}$ to grey level $K-1$.
4. Map pixels with grey level $f(x,y)$, $b_{low} \leq f(x,y) \leq b_{high}$ to a value specified by a linear transformation τ. For example, if b_k is the k^{th} grey level, then

$$\tau(b_k) = \frac{b_k - b_{low}}{b_{high} - b_{low}}(K-1)$$

   would be an appropriate transformation.

More generally, selected regions of the grey level scale can be stretched, and others compressed or left alone. This is best done interactively using computer graphics (e.g. the xv program that runs under X Windows), but may be set explicitly if required. For example the three transformation in Figure 2.1 (a) - (c) show possible stretching algorithms enhancing the dark, bright, and mid-regions of intensity respectively (Andrews 1976).

This technique (often known as windowing) is limited in its usefulness since it usually requires some parts of the image to be compressed in order to enhance

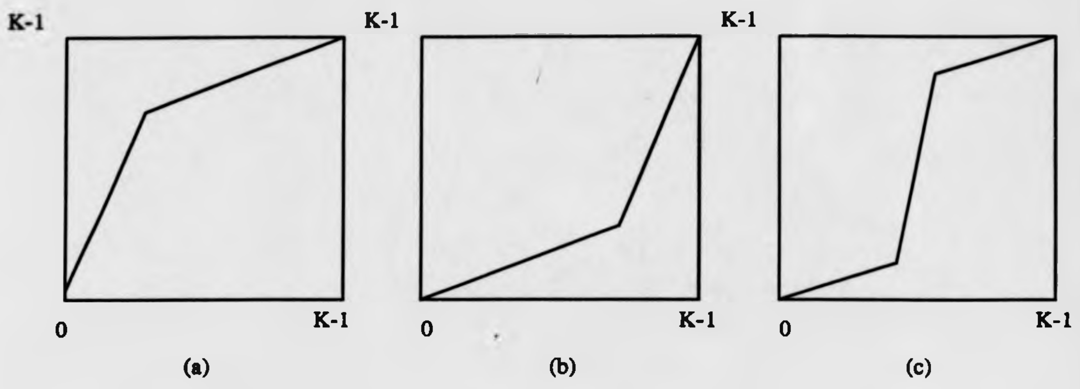


Figure 2.1: (a) Stretching the dark levels, (b) stretching the bright levels and (c) stretching the mid-region grey levels.

the desired area sufficiently. It is also limited by the amount a region can be stretched: large separations between the bins requires the use of a large amount of the grey level range, which in turn means more parts of the grey scale must be compressed to create space.

2.2.2 Histogram modification techniques

Histogram modification is a grey scale transformation used to give an image a specified distribution of grey levels (Hall 1971, Andrews 1972, Rosenfeld and Kak 1982). The histogram of the image shows how the pixels in the image are distributed amongst the available grey levels. The discrete normalised histogram or discrete density function of the image is given by the K-dimensional vector (see Kautsky *et al.* 1985 for this notation) $\mathbf{p} = (p(0), p(1), \ldots, p(K-1))$

$$p(i) = \frac{n_i}{L} \tag{2.1}$$

where $i = 0, \ldots K-1$, n_i is the number of the pixels in the image with grey level i, and L is the total number of pixels in the image. The value $p(i)$ is referred to as the grey level bin at i.

The histogram indicates how much of the available grey scale is used in the image, and where, in the grey scale, the largest concentrations of pixels are. The first point is important because the overall contrast of the image depends on it. The second sometimes indicates where the most relevant information in the image is.

The histogram does not contain spatial information, i.e. how the pixels are related to each other in two-dimensional co-ordinate space. Thus a single bin in the histogram may contain pixels that are very closely related spatially, or pixels that are from very widely scattered areas of the image. This lack of spatial information is a source of problems in grey scale transformations because information from entirely different contexts are treated together.

In general histogram modification techniques seek to redistribute the grey level bins in the original histogram into a new histogram which in some way enhances the contrasts in the image. For example, the histogram equalisation algorithm (see section 2.2.2) is designed to stretch the bins in the histogram in proportion to their information content (Andrews 1976). However, whether a specified shape of histogram stretches the grey scale effectively depends on whether it stretches out the relevant information in the image (i.e. the information the user is interested in viewing). This is not usually known beforehand, although in some instances it is possible to guess, and some distributions are known to favour certain types of images (e.g. for chest radiographs a finite Raleigh distribution has been found to be the most useful (Cocklin *et al.* 1983)).

If the desired histogram is known, the problem is then to find the transformation τ that maps the original histogram to the required histogram. This is known as the histogram modification problem (Hummel 1975).

The histogram modification problem

The histogram of the transformed data is denoted by $\tau \otimes \mathbf{p} = \mathbf{q}$, and is defined as the K-dimensional vector $(q(0), q(1), \ldots, q(K-1))$, where

$$q(i) = \sum_{(j|\tau_j = i)} p(j) \quad \text{where} \quad \tau_j = \tau(j) \quad j = 0, 1, \ldots, K-1$$

The value $q(i)$ is equal to the sum of the grey level bins which are transformed to grey level j (Kautsky *et al.* 1985). The histogram modification problem can now be formulated as follows:

Given an input histogram $\mathbf{p}$ and a reference histogram $\mathbf{q}$, find τ such that $\tau \otimes \mathbf{p} = \mathbf{q}$.

This problem can also be expressed in terms of the cumulative distribution function of the reference and input histograms. Given input histogram $\mathbf{p}$, its cumulative distribution function $D_{\mathbf{p}}$ is defined as a continuous piecewise linear function on $[0,1]$ (Kautsky *et al.* 1985). In order to do this an artificial point is inserted into the scale at 0, and the existing grey levels $[0,1,\dots,K-1]$ are shifted up along the scale to $[1,2,\dots,K]$. The cumulative distribution function is then defined as:

$$D_{\mathbf{p}}(j/K) = \sum_{i=1}^{j} \frac{p(i)}{L} \qquad D_{\mathbf{p}}(0) = 0, \quad j = 0,1,\dots,K$$

By continuous and piecewise linear it is meant that between:

$$D_{\mathbf{p}}\left(\frac{j-1}{K}\right) \text{ and } D_{\mathbf{p}}\left(\frac{j}{K}\right)$$

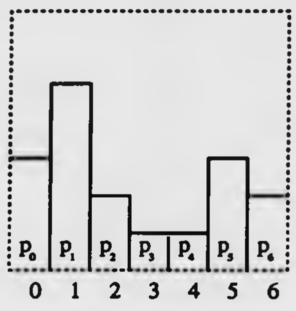
is a straight line of gradient:

$$\frac{\left(D_{\mathbf{p}}\left(\frac{j}{K}\right) - D_{\mathbf{p}}\left(\frac{j-1}{K}\right)\right)}{\frac{1}{K}} = \frac{\left(\sum_{i=1}^{j} \frac{p(i)}{L} - \sum_{i=1}^{j-1} \frac{p(i)}{L}\right)}{\frac{1}{K}} = \frac{p(j)\,.\,K}{L}$$

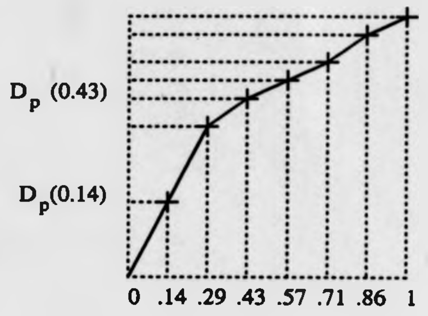
An example of this is shown in Figure 2.2.

The histogram modification problem can be expressed in terms of the cumulative distribution functions. Given input histogram $\mathbf{p}$ and a reference histogram $\mathbf{q}$, find τ such that $D_{\tau\otimes\mathbf{p}} = D_{\mathbf{q}}$

The optimal solution (in the sense that it minimises the error between the cumulative histogram of the transformed data and that of the reference, over all single valued, monotone, discrete transformations of the data), to this problem was described by Kautsky *et al.* (Kautsky *et al.* 1985).



(a)



(b)

Figure 2.2: (a) Original histogram with grey levels $[0, 1, \ldots, 6]$ and (b) piecewise linear cumulative distribution with grey levels $[0, 1, \ldots, 7]$.

If a discrete τ exists such that $D_{\tau \otimes \mathbf{p}}$ matches $D_{\mathbf{q}}$ *exactly*, then for all $k = 1, 2, \ldots, K$ levels in $\mathbf{q}$ there exists an integer $j_k \epsilon\ [1, 2 \ldots K]$ in the input histogram $\mathbf{p}$, such that

$$D_{\mathbf{p}}\left(\frac{j_k}{K}\right) = D_{\mathbf{q}}\left(\frac{k}{K}\right)$$

that is, if T:

$$T : D_{\mathbf{p}} \rightarrow D_{\mathbf{q}}$$

and

$$T\ :\ \left(\frac{j_k}{K}\right) \ \rightarrow\ \frac{k}{K}$$

then

$$T = {D_{\mathbf{q}}}^{-1} \circ D_{\mathbf{p}}$$

so:

$$j_k = KT^{-1}\left(\frac{k}{K}\right) = K{D_{\mathbf{p}}}^{-1} \circ D_{\mathbf{q}}\left(\frac{k}{K}\right)$$

using the above equation for T. This process is illustrated schematically in Figure 2.3.

This is for an exact solution. In general $j_k = KT^{-1}\left(\frac{k}{K}\right)$ is not an integer, and some rounding is required. Typically, procedures for generating the algorithm work from "left to right" (i.e. rounding down as in Hummel (Hummel 1975)).

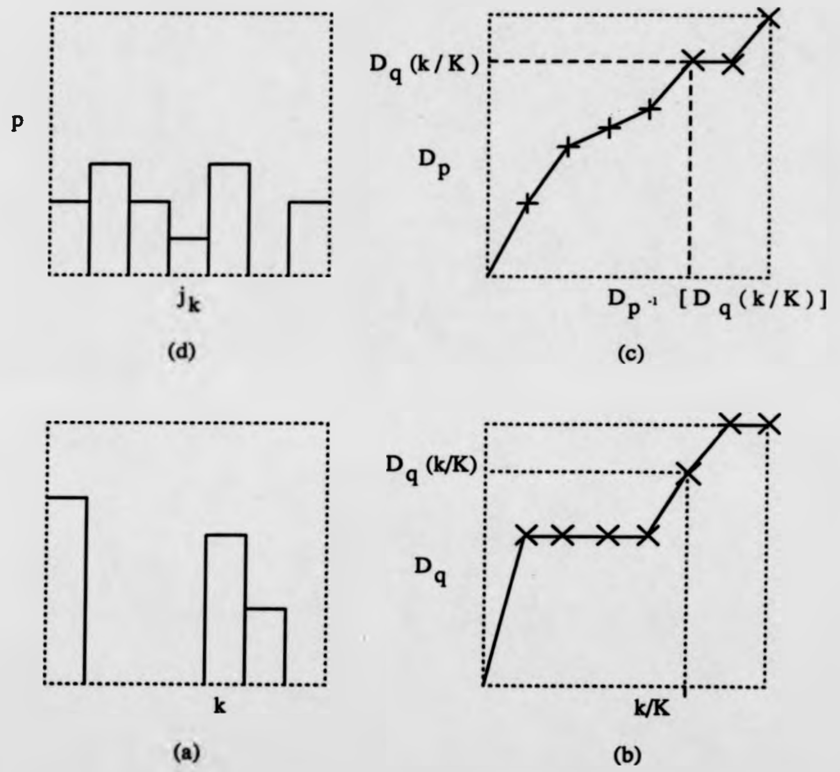


Figure 2.3: (a) Reference histogram $\mathbf{p}$, (b) position of the bin at k in the cumulative distribution of $\mathbf{p}$, (k/K), and the value of $D_{\mathbf{p}}$ at k/K, (c) matching the cumulative histograms of $\mathbf{p}$ and $\mathbf{q}$, (d) the bin j_k in $\mathbf{p}$ that gives the value $D_{\mathbf{q}}(k/K)$ in $D_{\mathbf{p}}$.

From left to right the algorithm is as follows:

$$\tau_j = \min\left[k \mid D_{\mathbf{p}}\left(\frac{j}{K}\right) \leq D_{\mathbf{q}}\left(\frac{k}{K}\right)\right] \tag{2.2}$$

and from "right to left" (i.e. rounding up):

$$\tau_j = \max\left[k \mid D_{\mathbf{q}}\left(\frac{k-1}{K}\right) \leq D_{\mathbf{p}}\left(\frac{j-1}{K}\right)\right] \tag{2.3}$$

As an illustration of these algorithms, and the effect of rounding, consider the following data (from Gonzalez and Wintz 1987):

Levels	p (original)	$D_{\mathbf{p}}$	q (reference)	$D_{\mathbf{q}}$
p_1	0.19	0.19	0	0
p_2	0.25	0.44	0	0
p_3	0.21	0.65	0	0
p_4	0.16	0.81	0.15	0.15
p_5	0.08	0.89	0.2	0.35
p_6	0.06	0.95	0.3	0.65
p_7	0.03	0.98	0.2	0.85
p_8	0.02	1	0.15	1

Table 2.1: Original and reference histograms and their cumulative distributions.

Their histograms are shown in Figure 2.4.

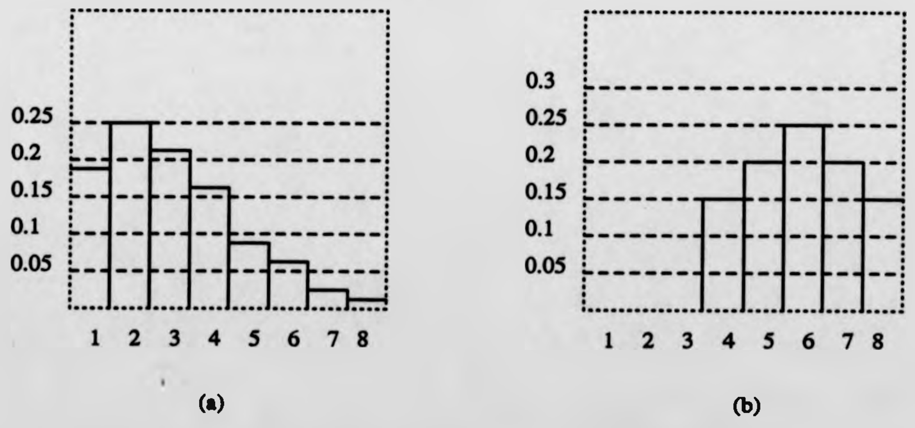


Figure 2.4: (a) Original histogram **p**, and (b) **q** reference histogram

Using equation 2.2:

$$\tau_1 = \min\left[k \mid D_p\left(\frac{1}{8}\right) \leq D_q\left(\frac{k}{8}\right)\right] = \min\left[k \mid 0.19 \leq 0.35\ (k=5)\right]$$

so $\tau_1 \rightarrow 5$, and $\tau_{2,3} \rightarrow 6$ and $\tau_4 \rightarrow 7$ and $\tau_{5,6,7,8} \rightarrow 8$

Using equation 2.3:

$$\tau_8 = \max\left[k \mid D_q\left(\frac{k-1}{8}\right) \leq D_p\left(\frac{8-1}{8}\right)\right] = \max\left[k \mid 0.85(k=8) \leq 0.98\right]$$

so $\tau_{8,7,6} \rightarrow 8$, $\tau_{5,4} \rightarrow 7$, $\tau_3 \rightarrow 6$, $\tau_2 \rightarrow 5$ and $\tau_1 \rightarrow 5$.

The resulting histograms are shown in Figure 2.5 (a) and (b) respectively.

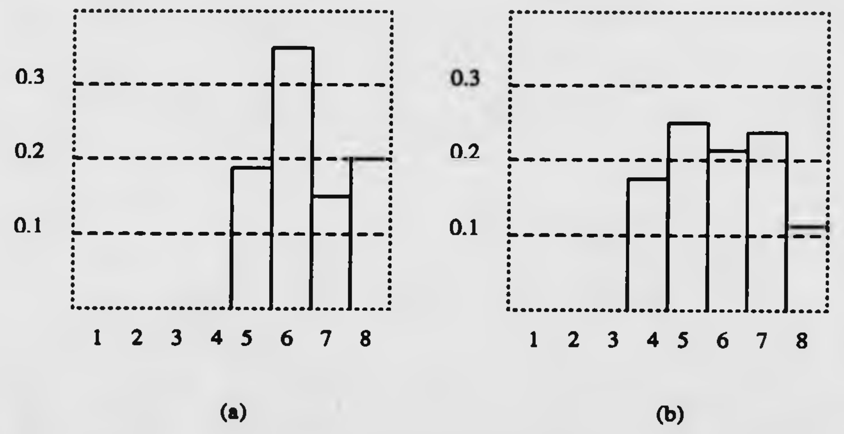


Figure 2.5: (a) Result using equation 2.2, and (b) result using equation 2.3

The transform that minimises the error between the reference and actual histogram for all τ is given (Kautsky *et al.* 1984) by:

$$\tau_j{}^* = k \text{ for all } j \text{ such that } x_{k-1} < j - 1/2 \leq x_k$$
$$\text{where } x_k = KT^{-1}\left(\frac{k}{K}\right), T^{-1} = D_p{}^{-1}{}_o D_q \tag{2.4}$$

Using the data given in Table 2.1, the values of $\tau_j{}^*$ are calculated as follows:

For $k = 1,2,3$, $x_{k-1}, x_k = 0$, so no j (other than 0) is mapped to these values.

If $k = 4$ $x_3 = 0$; $x_4 = 8.D_p{}^{-1}{}_o D_q\left(\frac{4}{8}\right) = 8.D_p{}^{-1}(0.15) = \frac{3}{4}$

${D_p}^{-1}(0.15)$ is calculated by interpolation on the piecewise linear D_q, as shown in Figure 2.6.

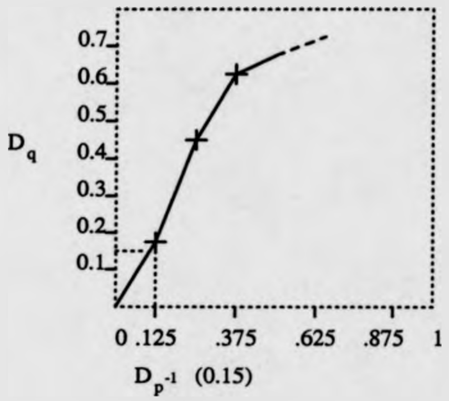


Figure 2.6: Interpolation on the piecewise linear D_p.

The required j are found from $0 < j - 1/2 < 3/4$. Thus $j = 1$ is mapped to 4. Similarly $j = 2 \rightarrow 5; j = 3 \rightarrow 6; j = 4,5 \rightarrow 7; j = 6,7,8 \rightarrow 8$, giving the histogram shown in Figure 2.7.

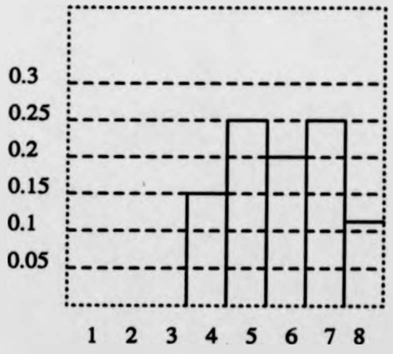


Figure 2.7: Result of using the minimum error transform τ^*.

Histogram equalisation

The histogram equalisation transformation is designed to produce an image which has a uniform grey level distribution (Hall 1971, Andrews 1972) i.e. the reference histogram is flat, all the grey levels occurring equally often. For this reason it is often known as the histogram flattening algorithm (Rosenfeld and Kak 1982).

The required transformation can be obtained by using equation 2.4 with **p**, the reference histogram, flat i.e.

$$p(i) = \frac{L}{K} \quad \text{for } i = 0, 1, \ldots, K-1$$

The distribution function, or cumulative distribution function of **p** is given by (Hummel 1975)

$$D_{\mathbf{p}}(i) = \frac{1}{2}p(i) + \sum_{j=0}^{i-1} p(j) \tag{2.5}$$

for $i = 0\ldots, K-1$, and $\tau : [0\ldots K-1] \rightarrow [0\ldots K-1]$. The histogram equalisation transformation is defined as (Hummel 1975)

$$\tau(i) = (K-1) * D_{\mathbf{p}}(i) \tag{2.6}$$

for $i = 0\ldots, K-1$.

The reason for choosing a flat histogram as the reference histogram is that the resulting image has *maximum entropy* - it presents to the viewer every grey level in equally likely mode (Andrews 1976). A given pixel in the image can take on one of K values, whose probabilities of occurrence, $(p(0), p(1), \ldots, p(K-1))$ are given in the original histogram. The entropy H, is a measure of how much "choice" is involved in the selection of the pixel's grey level value. H is given by (Shannon and Weaver 1949)

$$H = -\sum p(i)\log p(i)$$

H is a maximum when all the $p(i)$ are equal (i.e. a flat histogram). This is a maximum entropy mode, but may or may not be desirable.

However, if the transformation is single valued (as τ was defined earlier), grey level bins can be merged but not broken apart. Thus if the original histogram has large peaks the resulting histogram will only have an approximately flat histogram (Hummel 1977). Some work has been done to try and overcome this problem. Hummel (Hummel 1977) used a tie-breaking method to assign different pixels in the same bin to different values in the equalized histogram, so that a perfectly flat histogram was obtained. One method is to randomly choose the bins to allocate extra pixels to. Another is to use a 3×3 neighbourhood as a guide i.e. the n points with the smallest neighbourhood average are assigned to grey level g, and the remaining n' points are assigned to grey level g'. However, the results did not prove substantially different from histogram equalisation (Hummel 1977). Gonzalez and Fittes (Gonzalez and Fittes 1977) requantised the image into many more values, using local information, then equalized the resultant histogram, and requantised the equalized histogram to the original quantization levels. This method gives a more nearly flat histogram.

The result of histogram equalisation is an image with improved contrast, as the points in densely populated regions of the histogram are forced to occupy a larger number of grey level values (Ketcham 1976, Rosenfeld and Kak 1982). At the same time, points in sparser regions of the scale are forced to occupy fewer grey levels. However, the stretched regions are more populous than the compressed regions so the overall effect is one of contrast enhancement (Rosenfeld and Kak 1982).

A more detailed discussion of this algorithm, and some suggested modifications are to be found in chapter 3.

Histogram specification

Histogram specification is the name given (by Gonzalez and Fittes 1977, Gonzalez and Wintz 1987) to the process of specifying a histogram for the image, and then transforming it using the following method:

1. Transform the input histogram $\mathbf{p}$ to its equalised histogram $\hat{\mathbf{p}}$.
2. Transform the reference histogram $\mathbf{q}$ to its equalised histogram $\hat{\mathbf{q}}$.
3. For each value k in the input histogram, find its transformed (equalised) value $\hat{k}$.
4. Match $\hat{k}$ to the same $\hat{k}$ in $\hat{\mathbf{q}}$.

5. Find the level j in $\mathbf{q}$ such that $\hat{j} = \hat{k}$.

6. Map k to j.

According to Gonzalez and Wintz (Gonzalez and Wintz 1977) it should always be possible to match up the values at step (4) because the two histograms have approximately uniform densities, i.e. the grey levels occur equally often. This method can be replaced by the method described in the previous section using the cumulative distribution functions (and equation 2.4), since it gives the optimal result for matching an input to a reference histogram.

The required histogram is specified either by giving a known distribution, or by drawing its shape using computer graphics. Gonzalez and Fittes (Gonzalez and Fittes 1977) originally suggested a four parameter method to specify the histogram, in order to keep the method simple and quick. However, as these authors suggest, the parameters can easily be increased to produce the desired shape more accurately.

The are two main problems with the algorithm:

1. How is the shape of the desired histogram chosen?

2. If there are only a few grey levels in the original, it is inaccurate.

The second problem cannot be solved, but the first problem can be be given a qualified answer. A given area of the histogram can be stretched by specifying a ramp shape (O'Gorman and Brotman 1985). For example the ramp shape in Figure 2.8 (a) will enhance the low grey levels.

The gradient of the line really depends on the average size of the bins in the lower part of the histogram. For example, consider a histogram with k bins of average size g between 0 and k. Then the stretch can be controlled by the parameters a and b in Figure 2.8 (a). If $a = kg$ and $b = 2k$ then the gradient $g/2$ will cause the bins to be one grey level apart, and all the bins will be stretched, because at k the reference histogram has size gk. If a is reduced to $kg/2$ then half of the bins would be separated by three bins. Thus if b is fixed at $2k$ and a is

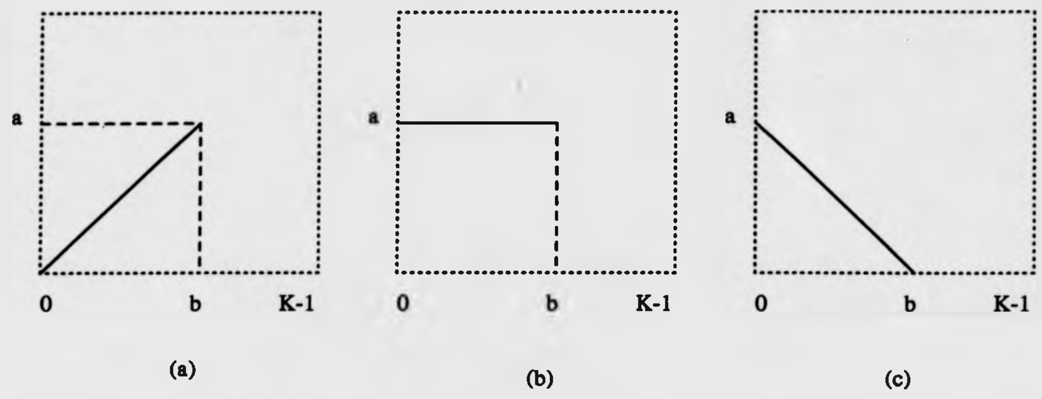


Figure 2.8: Specified output histogram shape for enhancing: (a) the low grey levels, (b) all grey levels equally, and (c) the high grey levels.

reduced from kg to 0 then a smaller number of bins will be stretched by a larger amount. In general the bins may be of any size, so the stretch will be unevenly distributed - large for the larger bins, and nothing for the smaller bins.

To compress bins a transform similar to Figure 2.8 (b) can be used. In the above example the k bins of size g can be compressed by half by setting $a = 2kg$ and $b = k/2$.

By piecing together straight lines of various gradients it is thus possible to construct the required histogram. It is, of course, not easy to do so - but it is an interactive technique - where eventually it should be possible to determine the required histogram.

Histogram hyperbolisation

Histogram hyperbolisation was proposed by Frei (Frei 1977). In this method the transformation is based upon the histogram of the image to be processed and the nature of human brightness perception, and is designed to produce images with a uniform distribution of *perceived* brightness levels. A rudimentary model of brightness perception is used to predict that the distribution of the *displayed* brightness levels should be hyperbolic (Frei 1977). Frei derivation was corrected by Nahin (Nahin 1979), and that analysis is described here.

The *measured* brightness of the image is denoted by the random variable $I(x,y)$, the random variable $J(x,y)$ denotes the *displayed* brightness level, and the random variable $B(x,y)$ denotes the brightness *perceived* by a human. The relationship between B and J can be written as

$$B = g(J)$$

where g is a monotonic increasing function. In a digital system

$$0 \leq i \leq n_I$$
$$0 \leq j \leq n_J$$

where n_J is not necessarily equal to n_I. B is taken to be a uniformly distributed random variable. Its probability density function is thus:

$$\begin{aligned} p_B(b) &= \frac{1}{g(n_J) - g(0)} \quad && g(0) \leq b \leq g(n_J) \\ &= 0 && \text{otherwise} \end{aligned} \tag{2.7}$$

J has the same probability of lying between 0 and j as B has of lying between $g(0)$ and $g(J)$, so using equation 2.7

$$\int_0^i p_J(u)du = \int_{g(0)}^{g(j)} p_B(u)du = \frac{g(j) - g(0)}{g(n_J) - g(0)} \tag{2.8}$$

where $p_J(j)$ is the density function of J. If the same assumption is made about the relationship between J and I, then using equation 2.8:

$$\int_0^i p_I(u)du = \int_0^j p_J(u)du = \frac{g(j) - g(0)}{g(n_J) - g(0)} \tag{2.9}$$

where $p_I(i)$ is the density function of I. The left hand side is the cumulative distribution function of I, and is easily computed from the brightness histogram of the original image. Solving equation 2.9 for j

$$j(i) = g^{-1}\left(g(0) + [g(n_J - g(0)] \int_0^i p_I(u)du\right). \tag{2.10}$$

This mapping maps the entire range of I into J.

It only remains to find a suitable function for g. Frei uses a form of the logarithmic Weber-Fechner law of human vision

$$g(J) = \ln(J + c)$$

with c a constant (Deutsch 1967). The inverse function is: $g^{-1}(J) = \exp(J) - c$. Substituting into equation 2.10 the result is:

$$j(i) = c\left\{\exp\left[\ln\left(1 + \frac{n_J}{c}\right)\int_0^i p_I(u)du\right] - 1\right\}.$$

In Frei's examples, the constant c is 2.4 (to give s = -0.5, p 287 Frei 1977).

Frei (Frei 1977) reports that all images processed using histogram hyperbolisation have been consistently considered of superior intelligibility than their histogram equalised counterparts. This is supported by Hummel (Hummel 1977) who says that the method works well for satellite images with inherently poor contrast, due to their relatively small amount of detail. However, the algorithm suffers from the usual problems of grey scale transformations.

Histogram smoothing

Histogram smoothing is the result of work carried out by Kautsky *et al.* over a number of years (Kautsky *et al.* 1980, 1981, 1982, 1984). This technique provides a means of balancing gains and losses of information between the original and reference histogram. The authors define a ***smoothed equidistributing regrading*** as a transformation resulting in a compromise between the original histogram and the histogram which is the result of histogram equalisation. Similarly, a ***smoothed weighted regrading*** results in a compromise between the extremes of the original histogram and a given reference histogram. For example, a smoothed equidistributing regrading is a smoothed weighted regrading with a flat reference histogram.

The most successful methods were based on nonlinear "padding". These methods operate by adding artificial "pixels" to pad out the grey levels with few or no entries, thus "smoothing" the original or reference histogram. Two methods were used. For a smoothed equidistributing regrading the paddings methods match certain paddings of the input histogram to flat reference histograms (Kautsky *et al.* 1984). The padding is controlled by a parameter λ, $0 \leq \lambda \leq 1$. When $\lambda = 0$ there is no padding and the equidistributing regrading is obtained. When $\lambda = 1$ the original histogram is obtained. For smoothed weighted regradings both the input and reference histograms are padded, and the padded histograms are matched.

The methods are described in (Kautsky *et al.* 1984). The first method uses constant padding. The input histogram $\mathbf{p}$ and the reference histogram $\mathbf{q}$ are replaced by $\mathbf{p}_\lambda$ and $\mathbf{q}_\lambda$ where

$$p_\lambda(j) = \max(p(j), c_1(\lambda)), \quad j = 1, 2, \ldots, K,$$
$$q_\lambda(j) = \max(q(j), c_2(\lambda)), \quad j = 1, 2, \ldots, K,$$

and

$$c_1(\lambda) = \lambda(\lambda \max_j[p(j)] + 2(1-\lambda)|\mathbf{p}|/K),$$
$$c_2(\lambda) = \lambda(\lambda \max_j[q(j)] + 2(1-\lambda)|\mathbf{q}|/K).$$

where K is the number of grey levels, and $|\mathbf{p}|$ and $|\mathbf{q}|$ are the number of pixels in the histograms $\mathbf{p}$ and $\mathbf{q}$ respectively. The smoothed weighted regrading is determined by matching $D_{\mathbf{p}_\lambda}$ to $D_{\mathbf{q}_\lambda}$ as described earlier. Any choice of the constants c_1 and c_2 such that $c_1(0) = 0 = c_2(0)$ and $c_1 = \max_j p(j)$, $c_2(1) = \max_j q(j)$ would give a range of regradings.

The second method uses padding by an inverse linear function. The smoothed weighted regrading is obtained by matching $D_{\mathbf{p}_\lambda}$ to $D_{\mathbf{q}_\lambda}$ where $\mathbf{p}_\lambda$ and $\mathbf{q}_\lambda$ are given by

$$p_\lambda(j) = \max_i(p(i)/(1 + c_1\ (\lambda)p(i)\ |i-j|)), \quad j = 1, 2, \ldots, K,$$

$$q_\lambda(j) = \max_i(q(i)/(1 + c_2(\lambda)\ q(i)\ |i-j|)), \quad j = 1, 2, \ldots, K,$$

with

$$c_1(\lambda) = m\ \log(1/\lambda)/|\mathbf{p}|,$$

$$c_2(\lambda) = m\ \log(1/\lambda)/|\mathbf{q}|.$$

Both methods give the smoothed equidistributing regrading when the reference histogram **q** is constant.

All the smoothed algorithms described in this section are useful for modifying images when the original histogram modification proves to be too harsh. For example, the *smoothed equidistributing regrading* is extremely useful when the histogram equalisation algorithm proves too harsh (this is illustrated in chapter 4).

2.3 Local contrast enhancement techniques

In this section local contrast enhancement techniques are described. Usually local contrast enhancement techniques determine the transformation of each pixel in the image from some function of the pixels in a surrounding (usually square) neighbourhood. In general local contrast enhancement techniques try to separate the grey level of a given pixel from those in the surrounding neighbourhood, thus making it more distinguishable in the sense of equation 1.1. In this way they introduce a spatial function, where a pixel's transformation varies according to its position in the image. Local enhancement techniques have a number of advantages over global contrast enhancement techniques:

1. It is not necessary to destroy information in the image. This happens in global contrast enhancement techniques when there is insufficient space in the grey level scale to allow for all the grey level bins to be separated. Some grey level bins must be compressed before other information can be enhanced. Local contrast enhancement techniques are not hampered by this problem: the pixel can be placed anywhere in the scale.

2. All pixels in the image can be enhanced, not just a selected range as in all global contrast enhancement techniques.

3. A local algorithm is sensitive to local detail so pixels with the same grey level value may be mapped to different areas of the grey level scale. This can, however, result in the introduction of artefacts.

2.3.1 Local histogram equalisation

Local histogram equalisation (LHE) or local histogram modification (LHM) was invented by Ketcham (Ketcham *et al.* 1974). It is a grey scale transformation which has been extended to a local contrast enhancement technique. In this way it was thought that the problems caused by the global histogram equalisation transformation could be overcome. Global histogram equalisation relies on the global distribution; by calculating the transformation for a window moving over the image, the transformation takes into account variations in local detail.

For each pixel (x, y) the (normalised) histogram or discrete density function of the surrounding square window W of dimension $M \times M$ (M odd) is:

$$p_w(i) = \frac{n_i}{M^2} \tag{2.11}$$

where n_i is the number of pixels of grey level i in the window W. Using equation 2.5 and equation 2.6 the central pixel (x, y) is assigned the value

$$\tau(f(x,y)) = (K-1) * D_{\mathbf{p_w}}(f(x,y))$$

where $D_{\mathbf{p}_w}$ is the cumulative distribution function of the window.

The algorithm usually gives an effective stretch of all local regions, although the result is heavily dependent on the window size chosen before the processing begins. The algorithm is discussed in detail in chapter 5.

2.3.2 Lee's algorithm

Lee (Lee 1980) designed an algorithm that causes a pixel to maintain its local mean, but modifies its variance by a factor of its original variance. The pixel (x, y) is assigned the value

$$\mu_{x,y} + k\ (f(x,y) - \mu_{x,y})$$

where $\mu_{x,y}$ is the mean of a square (it may also be rectangular) window W of dimension $M \times M$ (M odd), centered at the pixel (x, y). The mean is given by:

$$\mu_{x,y} = \frac{1}{M^2} \sum_{k,l} \{x_{k,l} \mid x_{k,l} \in W\} \tag{2.12}$$

The gain, k, is the ratio of the new local standard deviation to the original standard deviation. If $k > 1$, the image will be sharpened, but if $0 \leq k < 1$ the image will be smoothed.

This algorithm can be extended to enhance images with an ill distributed histogram. If τ is a grey scale transformation then the modified algorithm is given by:

$$\tau(\mu_{x,y}) + k\ (f(x,y) - \mu_{x,y})$$

This algorithm is typically used with windows of dimensions 3×3 or 5×5 (Lee 1980). It is fast, simple to compute, and usually gives a satisfactory result. However, the image does not enhance detail as much as the local histogram equalisation. Larger separations can be obtained, but this usually introduces an unacceptable amount of noise into the image due to the small window sizes used.

2.3.3 Local mean correction

Local mean correction (Hummel 1977) attempts to adjust the local average grey level at each point towards the global mean. Let $\mu_0 = \frac{L}{K}$ denote the global mean, and $\mu_{x,y}$ the average grey level value in an $M \times M$ (M odd) square neighbourhood around the pixel at (x, y), as given in equation 2.12. The pixel's grey level value $f(x, y)$ is lightened if $\mu_{x,y} < \mu_0$ and darkened if $\mu_{x,y} > \mu_0$. The new value assigned to the pixel is $f(x, y) - \alpha(\mu_{x,y} - \mu_0)$. High values of α increase the visibility of edges.

This algorithm is similar to Lee's algorithm (above).

2.3.4 Gordon's method

This technique is due to Gordon and Rangayan (Gordan and Rangayan 1984). For a given pixel (x, y) two windows W_M and W_{3M} of dimension m and $3m$ (respectively) are used. For a given m the contrast is defined (Beghadi and LeNegrate 1989) as:

$$C_{x,y} = \frac{|\bar{X}_1 - \bar{X}_2|}{|\bar{X}_1 + X_2|} \tag{2.13}$$

where

$$\bar{X}_1 = \frac{1}{M \times M} \sum_{(i,j) \epsilon W_M} x_{i,j}$$

and

$$\bar{X}_2 = \frac{1}{8M \times M} \sum_{(i,j) \epsilon (W_{3M} - W_M)} x_{i,j}$$

The enhancement transforms the grey level $x_{x,y}$ into $x'_{x,y}$ according to the sign of the difference $(\bar{X}_1 - \bar{X}_2)$.

According to Beghadi and LeNegrate (Beghadi and LeNegrate 1989) Gordon's method is more efficient and powerful than other methods, because it attempts to measure the contrast between a pixel and the surrounding neighbourhood. However, for small neighbourhoods the noise and digitisation effects are enhanced; for large neighbourhoods there is some loss of detail (Beghadi 1986).

2.3.5 Contrast enhancement using edge detection

This algorithm (Beghadi and LeNegrate 1989) attempts to overcome the problems in Gordon's algorithm described in the previous section.

In Gordon's paper the contrast is enhanced by computing the difference between a pixel and those in a surrounding neighbourhood. The advance is to use the fact that the eye is more sensitive to edges. The contrast associated with each pixel is defined with respect to its grey level distance from the mean grey level of the object boundaries (Kittler *et al.* 1985). The pixel is then transformed according to its position from object edges.

The value $C_{x,y}$ in equation 2.13 is now based on the detection of edges. An edge detector is applied to an odd sized window W, sufficiently large to filter out the noise and small enough to take into account the details to be shown. The Laplacian or some other edge detector is applied to each pixel in the window; then the mean grey level $\bar{E}_{x,y}$ is calculated:

$$\bar{E}_{x,y} = \frac{\left(\sum_{(i,j)\epsilon W} \Delta_{i,j}\ x_{i,j}\right)}{\sum_{(i,j)\epsilon W} \Delta_{i,j}}$$

where $\Delta_{i,j}$ is the edge detector and $x_{i,j}$ a pixel.

A large window is used to make the edge operator less sensitive to noise. Then $C_{x,y}$ is given by:

$$C_{x,y} = \frac{|x_{x,y} - \bar{E}_{x,y}|}{x_{x,y} + \bar{E}_{x,y}}$$

It is then transformed according to $C'_{x,y} = g(C_{x,y})$ using a function g satisfying the conditions: $g(C_{x,y}) \geq C_{x,y}$ and $g(C_{x,y})\epsilon[0,1]$ $(C_{x,y}\epsilon[0,1])$. For example $C'_{x,y} = (C_{x,y})^{1/2}$.

The contrast is defined with respect to the position of the pixel's grey level relative to the mean grey level of the boundaries. It is then transformed according to its position relative to the object edges. The noise is not enhanced considerably due to the averaging of all grey levels weighted by the edge values of the pixels in the current window (Beghadi and Le Negrate 1989).

This algorithm is involved, and consequently slow to compute. However, Dash and Chatterji (Dash and Chatterji 1991) have recently shown that a significant increase in speed can be obtained using an interpolative scheme.

2.3.6 Contrast enhancement using sliding histograms

This method was developed by Chochia (Chochia 1988) and applied to aerospace images (Chochia 1989). A statistical model of the image is used to divide the image into a "background" component and an "edge" component (Yan and Sakrison 1977). The idea is to separate the two components. The "background" or mean intensity component is determined from a large histogram (a "fragment") surrounding the pixel. This is done by choosing the appropriate mode in the histogram, and the mean value of the distribution.

Once the "background" component is found it is possible to do smoothing based on an M-type estimator (Huber 1981, Pomalaza-Raez and McGillem 1984), or texture enhancement.

This algorithm is involved. Similar texture enhancement can be obtained by using small windows with the local histogram equalisation algorithm, but this algorithm gives better results.

Chapter 3

Contrast enhancement adapted for a monitor

3.1 Introduction

In this chapter a global contrast enhancement algorithm is presented. The purpose of this algorithm is to enhance the contrast of the image to suit the display capabilities of the monitor where it is viewed. It is quite possible, for example, to have quite large differences between the grey levels of two pixels, but to find that these pixels are not visually distinguishable when displayed on a monitor. Clearly, different monitors also have different display capabilities (Dale-Jones and Tjahjadi 1991)[a]: an image may not appear to be the same. The algorithms described in this chapter attempt to adapt the contrast of an image so that it appears approximately the same when displayed on different monitors.

In order to determine the display capabilities of a monitor, two experiments were conducted to find how large the grey level difference between adjacent pixels must be before they are visible when displayed on a monitor. As is well known (Gonzalez and Wintz 1987, Sheppard 1968, Stevens 1951), the ability of the eye to distinguish adjacent grey levels is limited; a degree of separation is required before a difference is noticed. The discriminative ability of the eye has been measured by exposing an observer to a uniform field of brightness (Gonzalez and Wintz 1987), and a theoretical study of the optimal histogram distribution based on these results has been suggested by Ku (Ku 1984). However, in practice it is the ability of the eye to distinguish adjacent grey levels when

[a]see Appendix L

displayed on a particular monitor that must be measured.

3.2 The contrast sensitivity of the eye

The contrast sensitivity of the eye has been tested experimentally (see Gonzalez and Wintz 1987). An observer is exposed to a uniform field of light of brightness B, with a sharp-edged circular target in the centre of brightness $B + \Delta B$, as shown in Figure 3.1 (a). The brightness of the circular target is increased from

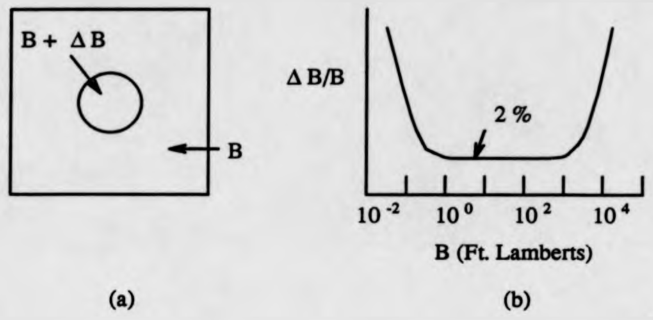


Figure 3.1: Contrast sensitivity with a constant background (Gonzalez and Wintz 1987).

B until its brightness $B + \Delta B$ is just noticeable from the background brightness B. The Weber ratio, $\frac{\Delta B}{B}$, is close to 2 percent over a large range of brightness levels, Figure 3.1 (b). This indicates that in a monitor capable of displaying 256 grey levels only 64 are required to distinguish detail in a local region. However as there are different local regions it is desirable to use the full grey level range so that 64 grey levels can be seen at different points on the screen).

More applicable results can be obtained using the experiment shown in Figure 3.2 (a). The background brightness is set to B_0 and the Weber ratio is found for different values of B. The Weber ratio remains constant in a small range around B_0, the surrounding adaptive brightness. This is called the dynamic range. The dynamic range of the human eye can be achieved with electronic imaging systems if they are correctly adjusted for the background brightness (Gonzalez and Wintz 1987).

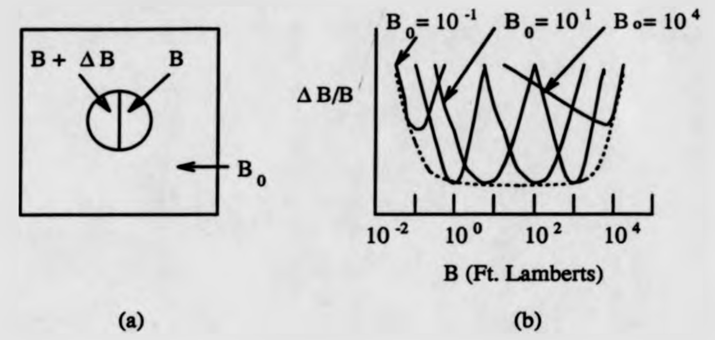


Figure 3.2: Constant sensitivity with a varying background (Gonzalez and Wintz 1987).

In general, for any small point or area in the image, the Weber ratio is much larger than the experimentally obtained values because of the lack of sharply defined boundaries and intensity variations in the background (Gonzalez and Wintz 1987).

3.3 Contrast enhancement on a monitor

Image processing takes place on a computer and it is the computer monitor where the image is displayed. However, the monitors used in computer systems are not always the same, and the appearance of an image frequently differs between monitors. For example, Figure 3.3 and Figure 3.4 shows a hologram of a can (Quan and Bryanston-Cross 1991) displayed on two different Sun workstation monitors, a GDM-1604-40 and an Hitachi LR47156 respectively.

In order to determine the different characteristics of each monitor the experiments described in the previous section were conducted (Dale-Jones and Tjahjadi 1991). The experimental set up for the two experiments is shown in Figure 3.5 (based on Figure 3.1 (a)). The brightness and contrast settings on the monitor (where available), were set at the maximum value to obtain the best results. Other monitors, with better display capabilities, would require adjustment of the contrast and brightness controls using a test card, to make sure the settings were standardised.

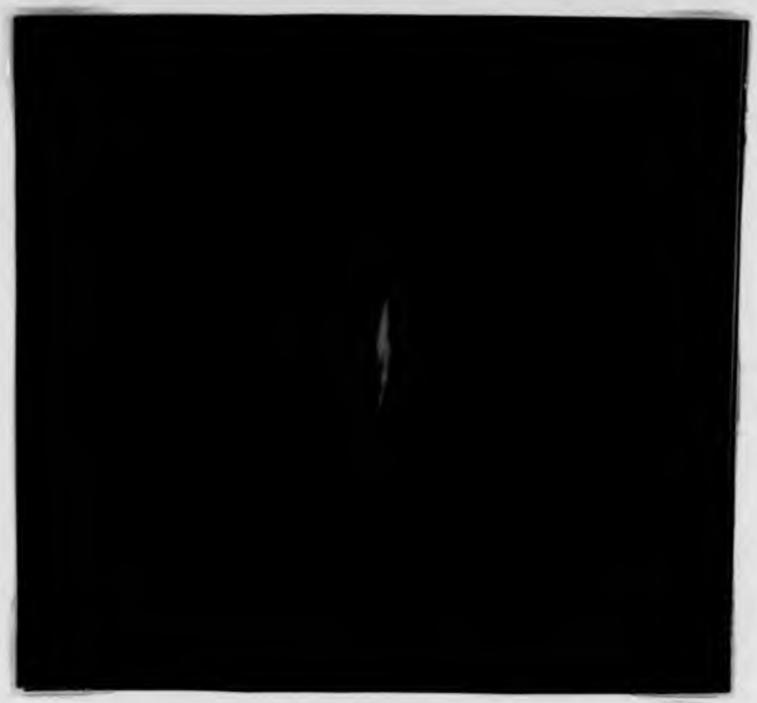


Figure 3.3: Hologram of a can displayed on a GDM-1604-40 monitor.

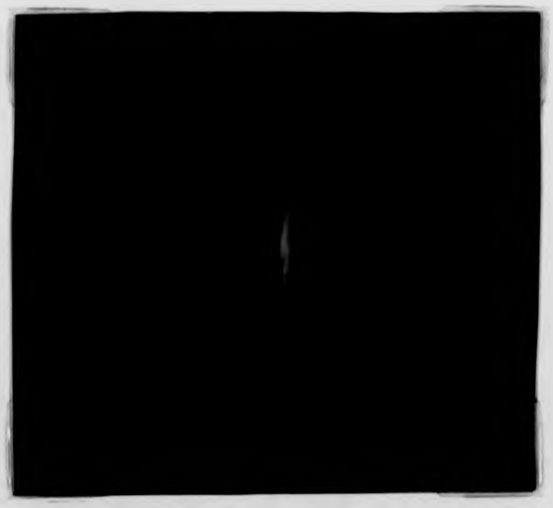


Figure 3.4: Hologram of a can displayed on an Hitachi LR47156 monitor.

The experiments were carried out under constant ambient light.

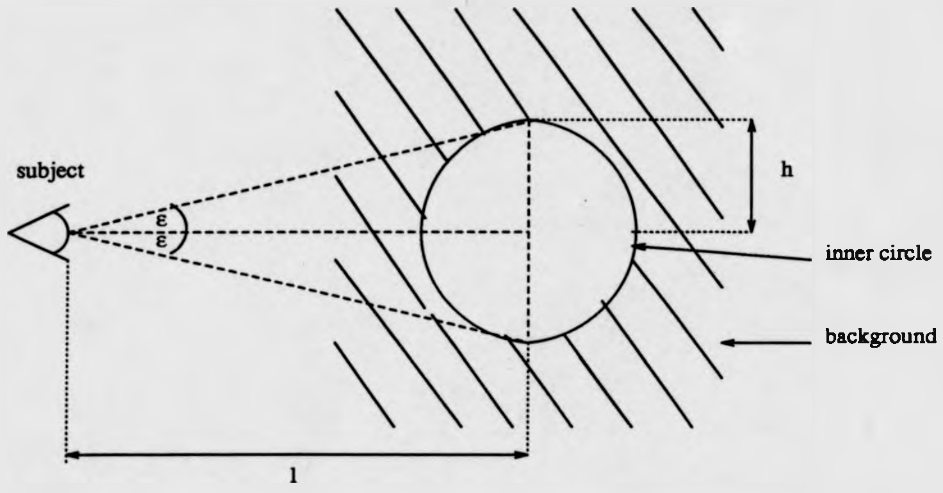


Figure 3.5: The experimental set used to determine the display characteristics of monitors.

The subject was placed at a comfortable viewing distance from the monitor, $l = 43$ cm. The inner circle was placed at the centre of the monitor. The angular subtense to the inner circle was $2\varepsilon = 10$ degrees (see Judd and Wyszecki 1975, p 155). The diameter of the inner circle h (7.5cm) was calculated from the values of l and 2ε. The outer region consisted of the remaining part of the monitor.

The first experiment (Figure 3.1 (a)) was carried out in two parts. In the first part, the grey level of the inner circle was increased from B until the subject could just distinguish the intensity of the inner circle from the background intensity. This 'just noticeable difference' (JND) is called $+\Delta B$. In the second part, the grey level of the inner circle was decreased from B until the subject could just distinguish the intensity of the inner circle from the background intensity. This JND is called $-\Delta B$. In each of these experiments the subject was asked to view the screen only after each increment or decrement of the previous intensity.

The experiments described above were carried out, using a standard observer, on three monitors, a GDM-1604A40, a GDM-1604-40 and an Hitachi LR47156 each capable of displaying 256 grey levels, by varying the background level B between 0 and 255. The grey level range was tested at 5 grey level intervals (i.e. 51 samples, starting at 0 and ending at 255). The experiment was carried out with 2 observers. Each observer was tested three times for each monitor at different periods. The results (see Appendix A.1) show some variance between the observers and among the three readings for each observer in the low grey level values for each monitor. Thus it is the mean of the $+\Delta B$ and $-\Delta B$ values for the two observers that are used in the following algorithm. The resulting mean values from the experiments are shown in Table 3.1. The grey level values within two grey levels of each sampled grey level are considered to have the same value (e.g. 3-7 have the same value as 5, and 8-12 have the same value as 10). Note

Hitachi LR47156				GDM-1604-40				GDM-1604A40			
B	ΔB	B	$-\Delta B$	B	ΔB	B	$-\Delta B$	B	ΔB	B	$-\Delta B$
0-4	6	5-14	5	0-39	46-B	45-49	8	0-49	53-B	55-59	6
5-9	5	14-29	3	40-44	9	50-54	4	50-54	4	60-64	3
10-14	4	30-234	2	45-49	5	55-69	2	55-64	3	65-79	2
15-29	3	235-255	3	50-54	4	70-214	1	65-74	2	80-244	1
30-239	2			55-64	3	215-255	2	75-229	1	245-255	2
240-252	3			65-84	2			230-253	2		
				85-184	1						
				185-253	2						

Table 3.1: A summary of the results of the first experiment.

that the JND values are defined at the extremes of the scale. At the top of the grey level scale (e.g. 0-10), when $-\Delta B$ is undefined (i.e. if it is impossible to get a value within the defined scale) the JND value is equal to $+\Delta B$. Similarly, at the end of the scale (250-255), when $+\Delta B$ is undefined, the JND value is equal to $-\Delta B$.

The second experiment (Figure 3.2 (a)) was also carried out with the three monitors. This experiment, like the first, was carried out in two parts. In the first part the background intensity was set at B_0, and the intensity of the target at $B = B_0$. The intensity of the left hand half circle was increased from B until it was just noticeable, $B_1 = B + \Delta B$. B was then set to B_1, and the just noticeable difference $B_2 = B_1 + \Delta B$ was obtained for the left hand side of the circle. In this way, values of B_1, B_2, B_3, B_4 and B_5 were obtained (in some cases five values cannot be obtained since this would overshoot the grey level scale), for each value of B_0 as it varied in steps of 10 grey levels over the whole grey level range. The second part of the experiment was similar, except that the left hand half circle was decreased in intensity, to obtain the values $B_{-1}, B_{-2}, B_{-3}, B_{-4}$, and B_{-5} (where five values could be obtained). Only five values were measured on either side because this is considered to be a large enough number of bins to separate against a given background intensity. The bins on either side of the background usually contain pixels from the same region of the image, so they need to be separated, while bins further away from the background probably come from a different region of the image. The bins that are more than five grey level values distant form the background intensity are separated from their adjacent bins by using the values obtained in the first experiment (i.e. these bins are considered to be against the background of the adjacent bin).

This experiment was carried out with one observer, sampling every five grey levels (the grey levels within a distance of two on either side are considered to have the same value in the look up table values), with one reading for B_i and B_{-i}, $i = 1, \ldots 5$ respectively. Only one observer was used because the results do not differ much from the results of the first experiment and the dynamic range is constant for most grey level value (except in the low grey level values). The results of the second experiment are in Appendix A.2, and a summary in Table 3.2.

Hitachi LR47156		GDM-1604-40		GDM-1604A40	
B	ΔB	B	ΔB	B	ΔB
0-10	7	0-35	40-B	0-40	54-B
10-30	4	35-45	9	40-50	7
30-75	3	45-50	7	50-55	4
75-160	2	50-55	4	55-60	3
160-255	3	55-70	3	60-255	2
-	-	70-210	2	-	-
-	-	210-255	3	-	-

Table 3.2: A summary of the second experiment.

3.4 Algorithms based on the characteristics of the monitor

The values found in the two experiments can be used to separate grey level bins in the histogram, in order to increase the contrast in the image. The results from the first experiment indicate where the bins immediately to the left and to the right of the background intensity should be placed. The required spacing for the bins are put in a look up table *lut*: $lut[i], i = 0, \ldots, K-1$. The look up table indicates the required spacing of adjacent grey level bins when the background intensity is i. The values in the look up table are the minimum separation required before the bins become visible.

The results from the second experiment can be used to determine the separation of the bins between two and five grey levels on either side of the background. The required spacing for the bins is placed in a second look up table *alut*: $alut[i][j], i, j = 0, \ldots, K-1$. The look up table indicates the spacing required for the bin adjacent to bin j when the background intensity is i.

The values in these tables can be increased slightly if necessary by adding an increment (*inc* in the forthcoming algorithms); it may also be advantageous to increase the look up table values for bins close to the background (e in the forthcoming algorithm), as they are likely to contain the most interesting information.

3.4.1 Global algorithm for one background

The first algorithm operates globally, separating grey level bins close to a background intensity. Only a single background intensity is allowed, as there is usually only sufficient space in the grey level scale to allow for a small amount of separation. The background intensity can be from any region of the image. The algorithm is a grey scale transformation as defined in section 2.2).

The results of the first visual experiment in Table 3.1, show that for two of the three monitors, there is an area of very poor contrast at the beginning of the grey level scale. To avoid placing many grey level bins here, an offset is used to shift the distribution of the histogram beyond this part of the grey level scale. The values of the offset for the monitors are shown in Table 3.3. The

Monitor	Offset
Hitachi LR74156	10
GDM-1604-40	50
GDM-1604A40	60

Table 3.3: The offset values for the monitors.

result of the transformation is to restrict most of the grey level bins to the range $[offset \dots K-1]$. A few grey level bins are allowed in the range $[0 \dots offset]$, if there is sufficient space for separation.

To separate the bins in the histogram within the range $[offset \dots K-1]$ the number of occupied grey levels in the image needs to be at least smaller than the available range $K-1-offset$. In this algorithm, a parameter A ($0 \leq A \leq n_l+n_u$ see below for the definition of n_l and n_u) is used to give an indication of the amount of stretching required. If the background occupies the grey level range $b_l \dots b_u$, and the number of occupied grey levels before b_l and after b_u are n_l and n_u respectively, then

$$c = (K-1-offset) - (b_u - b_l + 1) - n_l - n_u - A \tag{3.1}$$

indicates the number of bins that need to be merged with other grey level bins in the original histogram to provide sufficient space for stretching. If $c > 0$, then there is no need to merge grey level bins to create space. If $c < 0$ then at least $|c|$ bins need to be merged in the original histogram.

If $c < 0$ the space is obtained by merging $|c|$ of the smallest bins in the histogram. To find which of the bins to merge, the limit l is found where there are more than $|c|$ bins less than l (with none of the bins lying in the background range), and less than $|c|$ bins less than $l-1$ (again with none of the bins lying in the background range)[1]. Then these $|c|$ bins less than l are merged.

[1]It is possible for these algorithm to fail if the histogram is very flat.

The transformation τ required is given by:

$$\tau = \begin{cases} i & \text{if } p(i) > l \text{ or } b_l \leq i \leq b_u \\ min_j[|i-j|], j < i, p(j) > l & \text{if } p(i) \leq l \\ 0 & \text{if there is no } j \text{ satisfying part 2 and } i < b_l \\ b_u & \text{if there is no } j \text{ satisfying part 2 and } i > b_u \end{cases}$$

The transformation is in four parts. The first part maps all bins, $p(i)$, larger than l, or bins lying in the background range, to the same place in the histogram. The second part maps bins that are less than l to the same place as the nearest bin with smaller grey level but whose size is greater than l. Parts three and four are for special cases of part two. If the bin is less than l, its grey level is less than any of the background grey levels, and there is no bin satisfying part two, then it is mapped to zero. Similarly, part four ensures that bins with a grey level greater than any bins in the background range, not satisfying part two are mapped to the position of the last bin in the background range.

The result of this transformation is to create sufficient space in the grey level range to separate the grey levels close to the background, as suggested.

A second transformation τ' is applied to the grey level distribution produced by τ, to separate the bins according to the JND values obtained for the monitor. Before calculating the transformation new values of n_l and n_u must be found, as some bins in the histogram have been compressed. The transformation is calculated as follows:

1. Calculate the position of the background. This is given by

$$\tau'(\tau(b_l)) = \tau'(b_l) = offset + \frac{n_l * (K - 1 - offset)}{n_u + n_l + (b_u - b_l + 1)}$$

(It doesn't matter if b_l, b_u or any grey level value between them is used, because the relative positions will be the same). The value $\tau'(b_l)$ gives a measure of the grey level distribution about the background. For example, if more grey levels are occupied in the range $[0 \ldots b_l]$ than $[b_l+1 \ldots K-1]$, then the position of $\tau'(b_l)$ will be close to the value of the $offset$. This calculation ensures that the placement of the background range in the transformed histogram is in proportion to the occupied grey levels on either side.

2. The background grey levels are mapped as follows

$$\tau'(\tau(b_l+k)) = \tau'(b_l+k) = \tau'(b_l)+k \quad \text{for } k=0,\ldots,b_u-b_l-1$$

and (for those bins mapped into b_u by τ)

$$\tau'(\{i \mid \tau(i)=b_u\}) = \tau'(b_l)+(b_u-b_l)$$

3. The spacing of the first bin to the left of the background is

$$\tau'(\{i \mid \tau(i)=b_l-1\}) = \tau'(b_l) - lut[\tau'(b_l)] - inc - e$$

where inc is an increment added to increase the separation and e is separation added in inverse proportion to the distance to the background b_l.

4. The spacing of the first bin to the right of the background is

$$\tau'(\{i \mid \tau(i)=b_u+1\}) = \tau'(b_u) + lut[\tau'(b_u)] + inc + e$$

5. The spacing of the next four bins to the left is

$$\tau'(\{i \mid \tau(i)=b_l-2\}) = \tau'(b_l-1) - alut[\tau'(b_l)][\tau'(b_l-1)] - inc - \max\{0,(e-1)\}$$
$$\tau'(\{i \mid \tau(i)=b_l-3\}) = \tau'(b_l-2) - alut[\tau'(b_l)][\tau'(b_l-2)] - inc - \max\{0,(e-2)\}$$

and in general

$$\begin{aligned}\tau'(\{i \mid \tau(i)=b_l-j\}) = \tau'(b_l-(j-1)) - alut[\tau'(b_l)][\tau'(b_l-(j-1))]\\ -inc-\max\{0,(e-j)\}\end{aligned}$$

for $j = 2,\ldots,5$. The remaining bins to the left are then separated as follows:

$$\tau'(\{i \mid \tau(i) = b_l - j\}) = \tau'(b_l - (j-1)) - lut[\tau'(b_l - (j-1))] - inc - \max\{0, (e-j)\}$$

for $j = 6, \ldots, n_l - 5$.

6. The spacing of subsequent bins to the right is, similarly,

$$\tau'(\{i \mid \tau(i) = b_u + j\}) = \tau'(b_u + (j-1)) + alut[\tau'(b_u)][\tau'(b_u + (j-1))] + inc + \max\{0, (e-j)\}$$

   for $j = 2, \ldots, 5$. The remaining bins to the right are then separated as follows:

$$\tau'(\{i \mid \tau(i) = b_u + j\}) = \tau'(b_u + (j-1)) + lut[\tau'(b_u + (j-1))] + inc + \max\{0, (e-j)\}$$

   for $j = 6, \ldots, n_u - 5$.

7. There may be empty spaces in the grey scale as the result of the transformation τ. Empty grey levels are mapped to the nearest occupied grey level value.

8. A grey level is only given the spacing in steps 5 and 6 if there is still sufficient space in the grey scale to map the remaining bins to the left of the present position without merging. This is also true for bins to the right of the background, calculated in step 6.

The code for this algorithm "global lut stretch with parameter", is in Appendix B.

3.4.2 Global algorithm for more than one region

This algorithm is similar to the first algorithm except that it is used to stretch a number of regions from the image. Each region is identified by a range of grey level values (all the grey level ranges are distinct i.e. they do not overlap), which is representative of the region it is identified with. The largest grey level bin situated within the range of each grey level range is considered to be the background grey level of the corresponding region. The grey level bins on either side of the background are then separated in the histogram.

In practise it is not easy to identify the regions of the image. It needs to be done in an "interactive environment", where regions of the image can be identified using a graphics tool. Such an environment is described in chapter 6 of this thesis. Here it will be assumed that the grey level ranges of the regions have already been identified.

The main restriction on the algorithm is once again a lack of space in the grey level scale for stretching, particularly if the original histogram occupies a considerable portion of the available grey scale. Therefore, as in the first algorithm a parameter A is used to create space in the grey level scale, and the preliminary transformation τ is used to get rid of $|c|$ (see equation 3.1) of the smaller bins in the histogram (these must be bins outside the ranges identifying the regions).

A second transformation τ' is used to stretch the bins in each region away from the corresponding background. Let the number of areas be n, $(0,\ldots,n-1)$, and let the background grey level in each area be given by a_i, $i = 0,\ldots,n-1$. The transformation is then as follows:

1. The position of the background intensities a_i, and all the bins mapped to it by τ, are given by:

$$\tau'(\{i \mid \tau(i) = a_i\}) = offset + \frac{n_{b_i} * (K-1-offset)}{n_{b_i} + n_{u_i} + 1}$$

   where n_{b_i} and n_{u_i} are the number of occupied grey levels before and after the background intensity a_i respectively.

2. The first bin to the left of the first background a_0 is separated as follows:

$$\tau'(\{i \mid \tau(i) = a_0 - 1\}) = \tau(a_0) - lut[\tau'(a_0)] - inc - e$$

   where inc is an increment added to increase the look up table value and

e is an increment added in proportion to the distance of the bin from the background a_i. The next four bins are separated, if there is sufficient space:

$$\tau'(\{i \mid \tau(i) = a_0 - j\}) = \tau'(a_0 - (j-1)) - alut[\tau'(a_0)][\tau'(a_0 - (j-1))]$$
$$-inc - \max\{0, e - j\}$$

where $j = 2, \ldots, 5$. And after this (if there is sufficient space) as:

$$\tau'(\{i \mid \tau(i) = a_0 - j\}) = \tau'(a_0 - (j-1)) - lut[\tau'(a_0 - (j-1))] - inc - \max\{0, e - j\}$$

for $j = 6, \ldots, n_{b_0} - 5$.

3. The first bin to the right of the last peak a_{n-1} is separated as follows:

$$\tau'(\{i \mid \tau(i) = a_{n-1} + 1\}) = \tau'(a_{n-1}) + lut[\tau'(a_{n-1})] + inc + e$$

then the next four bins to the right are separated, if there is sufficient space, as follows:

$$\tau'(\{i \mid \tau(i) = a_{n-1} + j\}) = \tau'(a_{n-1} + (j-1)) - alut[\tau'(a_{n-1})][\tau'(a_{n-1} - (j-1))]$$
$$+inc + \max\{0, e - j\}$$

where $j = 2, \ldots, 5$. And after this (if there is sufficient space) as:

$$\tau'(\{i \mid \tau(i) = a_{n-1} + j\}) = \tau'(a_{n-1} + (j-1)) - lut[\tau'(a_{n-1} + (j-1))]$$
$$+inc + \max\{0, e - j\}$$

for $j = 6, \ldots, n_{u_{n-1}} - 5$.

4. The bins between each pair of background grey levels a_j and a_{j+1}, $j = 0, \ldots, n-2$ are separated in a similar manner. Bins are separated to the left of a_{j+1} and the right of a_j as long as there is sufficient room.

5. The empty bins in the grey scale are mapped to the nearest occupied grey level bin.

The code for this algorithm "multiple lut stretch with parameter" is given in Appendix C.

3.5 Results

In this section the results of the two algorithms on an example image are shown. The results were obtained using a value of $inc = 1$ and $e = 3$. For the first algorithm "global lut stretch with parameter" a slightly modified algorithm is used. Instead of averaging the values for $+\Delta B$ and $-\Delta B$, they are separated into two sets of value, one an average of $+\Delta B$ and the other an average of $-\Delta B$. The bins to the left and right are separated according to the values in the different *lut* arrays. This enables some bins to be separated in the low grey level values below the $offset$. In this way a little more stretching is obtained, although at the cost of placing some pixels in the invisible part of the grey scale. Later in the thesis (Chapter 6) the version of the algorithm in Appendix B is used. This version does not allow any bins to be placed below the $offset$ value.

Figure 3.3 shows a hologram of a can displayed on an GDM-1604-40 monitor, and Figure 3.6 shows its histogram. It is clear from the histogram that the holo-

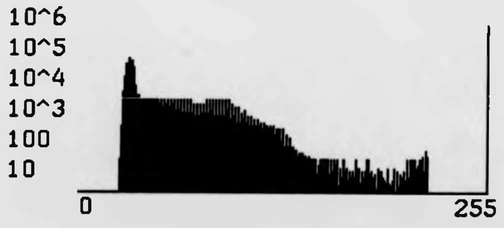


Figure 3.6: Histogram of Figure 3.3.

gram does not fully occupy the grey level range. The image is dark because a lot of the pixels in the image have grey level values below the offset value for this monitor. The black area (grey level range 28-36) is selected as the background. Figure 3.7 shows Figure 3.3 after processing with the first algorithm with the *lut* values for the GDM-1604-40 monitor and $A = 0$. Figure 3.8 shows the histogram of Figure 3.7. Figure 3.9 shows Figure 3.3 after processing with the

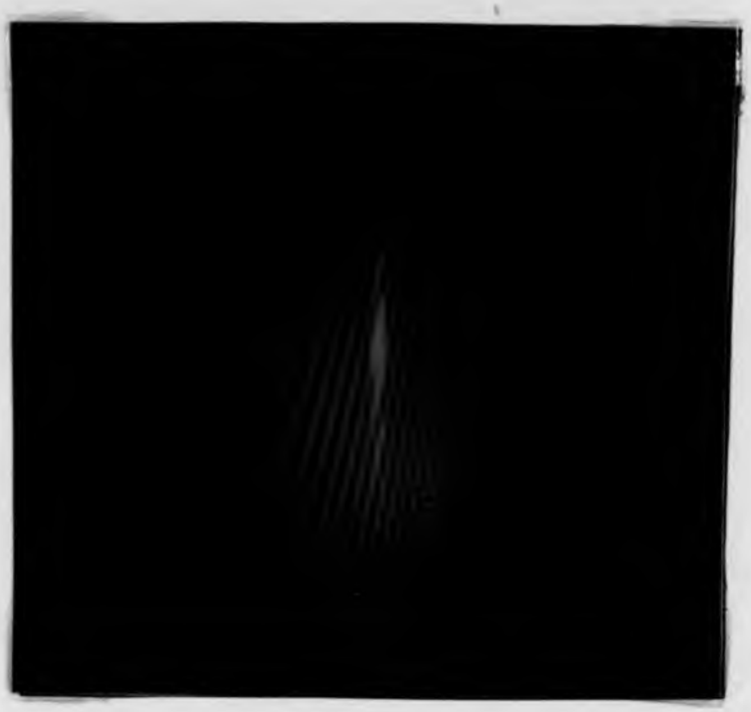


Figure 3.7: Hologram of a can after processing for the GDM-1604-40 monitor with $A = 0$ (displayed on a GDM-1604-40 monitor).

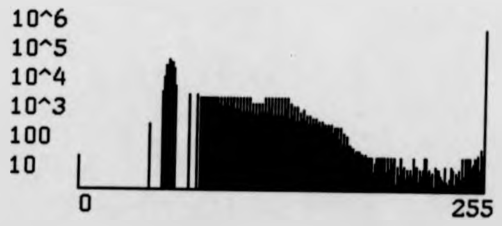


Figure 3.8: Histogram of Figure 3.7.

first algorithm with the *lut* values for the GDM-1604-40 monitor and of $A = 30$ and Figure 3.10 shows its histogram. In Figure 3.7 there is no information loss,

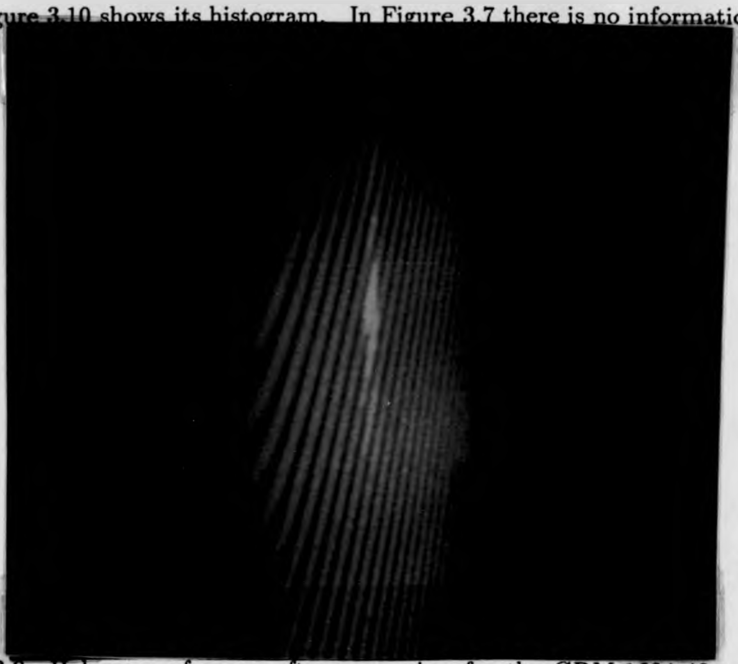


Figure 3.9: Hologram of a can after processing for the GDM-1604-40 monitor with $A = 30$ (displayed on a GDM-1604-40 monitor).

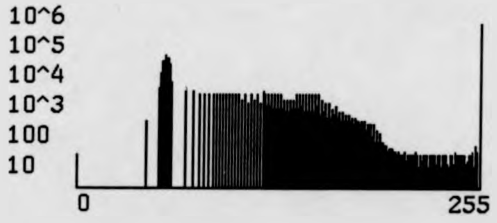


Figure 3.10: Histogram of Figure 3.9.

and the image is fairly clear. Figure 3.9 is clearer; the information lost is negligible (see Figure 3.10) since there are many small grey level bins of no particular interest at the end of the histogram, but there is more stretching.

Figure 3.11 shows Figure 3.3 after processing with the first algorithm with the *lut* values for the Hitachi LR47156 monitor and $A = 0$. The histogram of Figure 3.11 is shown in Figure 3.12.

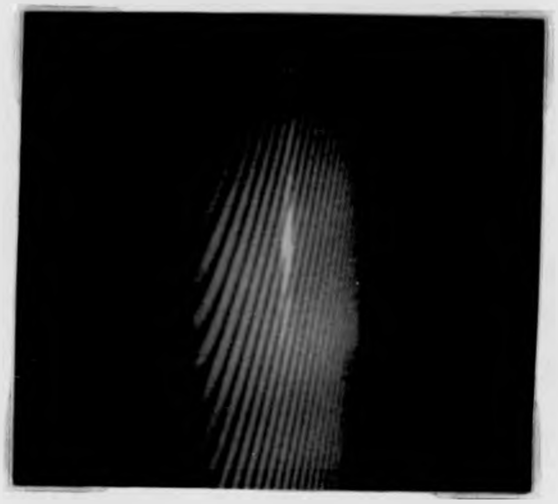


Figure 3.11: Hologram of a can after processing for the Hitachi LR47156 monitor with $A = 0$ (displayed on a Hitachi LR47156 monitor).

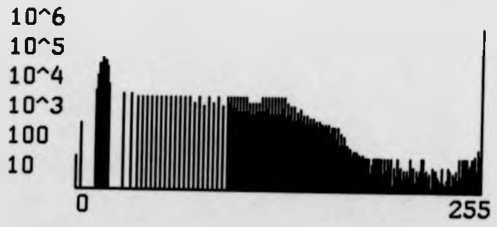


Figure 3.12: Histogram of Figure 3.9.

Figure 3.9 and Figure 3.11 look similar, but as the histograms illustrate, the grey level distributions are not. The difference is further illustrated by looking at Figure 3.13 which shows Figure 3.11 displayed on a GDM-1604-40 monitor: it is only a very small improvement on the original image when displayed on the GDM-1604-40 monitor (Figure 3.3).

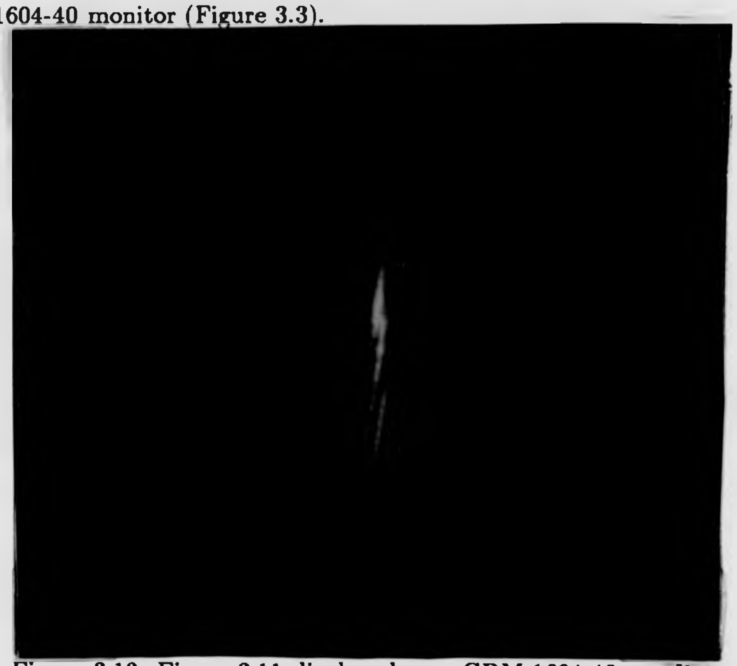


Figure 3.13: Figure 3.11 displayed on a GDM-1604-40 monitor.

The results illustrated above show that the first algorithm is quite effective at enhancing contrasts against a background area, with negligible information loss. The second algorithm can also be used to select various regions of the image and to stretch them together. An example of the use of the second algorithm is given in chapter 6, in section 6.5.2 and Figure 6.19.

3.6 Discussion

In this chapter two experiments have been used to determine the minimum grey level separations required for pixels in adjacent grey level bins to become visible in the image. However, these values only indicate the minimum separation required;

increments are added for bins that are close to the background grey level bin (where it is assumed most of the information is contained), and also a general increment (if larger separations are required). These can be adjusted to increase the contrast.

The experimental results have been used to design two algorithms to separate grey level bins from the background. The algorithms assume that the background grey levels are known (they can be identified using a graphics tool e.g. a mouse).

As the algorithms identify background grey levels and have a parameter A that controls the number of bins destroyed, they are more effective than the usual grey scale transformations, because the latter transform the image according to a fixed criterion i.e. the separation between the grey level bins is an arbitrary amount. The images are also enhanced according to the display capabilities of the monitor; thus the images will appear similar after processing, when displayed on different monitors.

It is clear that in many cases it is desirable to separate grey levels in the image to increase the contrast. However it is also possible to inadvertently increase the signal to noise ratio by doing so. Therefore a method of quantifying the results of contrast enhancement, based on something other than the human eye would be helpful. It would also be useful in comparing the results of different algorithms on the same image. This could be obtained by using some mathematical method or some device to measure contrast from a photographic negative (i.e. putting a camera in front of the screen with the automatic gain control switched off; the image captured would be a faithful representation of the image; the contrast differences could then be measured from the numerical values of the image). In future work such methods should be used.

Chapter 4

Histogram equalisation and inverse histogram equalisation

4.1 Introduction

The global histogram equalisation algorithm is a very well known algorithm in contrast enhancement (Hall 1971, Andrews 1972, Ketcham 1976, Hummel 1975, Rosenfeld and Kak 1982 and Gonzalez and Wintz 1987). The purpose of the algorithm, as mentioned in section 2.2.2 (page 17), is to produce an image which has a completely flat histogram.

In the continuous case it is always possible to obtain a flat histogram, but it is not possible when the data is in discrete form as the transformation is single valued. This sometimes results in a very harsh transformation (a large amount of information in the image is lost), particularly if the image has a heavily skewed distribution. In this chapter the discrete histogram equalisation algorithm is examined in detail; it is then modified to make it applicable to a larger range of histograms (Dale-Jones and Tjahjadi 1992)[a]. The modified algorithms allow the user to balance the amount of stretching given to peaks in the histogram against the amount of information lost. The effects of the algorithms on peaks or Gaussians of various means, standard deviations and population sizes are discussed.

4.2 The histogram equalisation algorithm

A method of calculating the histogram equalisation transformation τ, due to Hummel (Hummel 1975), is

[a]see Appendix L

described in section 2.2.2. The first step is to calculate the cumulative distribution function (equation 2.5); then the grey level bin at i is transformed according to equation 2.6.

When using a computer the transformation is computed iteratively using a computer algorithm (the code for this algorithm is given in Appendix D). In this case the transformation is calculated as follows:

$$\begin{aligned}
\tau[0] &= 0 + \frac{1}{2}p(0)(K-1) \\
\tau[1] &= p(0)(K-1) + \frac{1}{2}p(1)(K-1) \\
\tau[2] &= p(0)(K-1) + p(1)(K-1) + \frac{1}{2}p(2)(K-1) \\
&\cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\
\tau[i] &= \tau[i-1] + \frac{1}{2}p(i-1)(K-1) + \frac{1}{2}p(i)(K-1) \quad 1 \leq i \leq K-1
\end{aligned}$$

where p is the normalised histogram and K is the number of grey levels available. This algorithm places the histogram bin at grey level i a distance of $\frac{1}{2}p(i)(K-1)$ from the bins on either side. The factor

$$p(i)(K-1) \tag{4.1}$$

is the total amount of stretching bin i receives. If a bin is to be separated from its adjacent bins then

$$\frac{1}{2}p(i)(K-1) \geq 1 \Leftrightarrow p(i) \geq \frac{2}{K-1}$$

A slightly different version of the algorithm results if the factor $\frac{1}{2}$ is dropped from $\frac{1}{2}p(i)(K-1)$. The separation of bin i from bin $i-1$ is then $p(i)(K-1)$, and from bin $i+1$, $p(i+1)(K-1)$. A bin is then separated from the bin to its left if $p(i) \geq \frac{1}{K-1}$, and from the bin on the right if $p(i+1) \geq \frac{1}{K-1}$. Note that if there are L pixels in the image then the average number of pixels per bin is $\frac{L}{K}$, and that if $p(i) \geq \frac{1}{K-1} \approx \frac{1}{K}$ (for large K) then $n_i = p(i)L \geq \frac{L}{K}$. Thus, in this case, if a bin in the histogram contains more pixels than the average it is separated, otherwise it is compressed. In this thesis the factor $\frac{1}{2}p(i)(K-1)$ is used in the above iterative algorithm.

As mentioned in chapter 2, the contrast response of some monitors at the lower end of the scale is very poor. This means that adjacent bins at the lower end of the grey level range in the histogram require very large separations to become visible on a monitor. It is therefore common practice to use an offset to shift the histogram to a part of the grey level scale where only a relatively small separation is required for adjacent bins to become visible. A table of offset values for monitors of various types is shown in Table 3.3 If the offset is o_{off}, the above algorithm is modified to:

$$\tau[0] = o_{off} + \frac{1}{2}p_w(0)(K-1-o_{off})$$

$$\cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot$$

$$\tau[i] = \tau[i-1] + \frac{1}{2}p(i-1)(K-1-o_{off}) + \frac{1}{2}p(i)(K-1-o_{off}) \quad 1 \leq i \leq K-1$$

4.3 Problems caused by the histogram equalisation algorithm

As many authors have pointed out (Kautsky *et al.* 1984, Hummel 1977, and Rosenfeld and Kak 1982), there are a number of problems with the histogram equalisation algorithm. The most significant problem is that because the histogram equalisation transformation is single valued it is impossible to obtain the desired flat histogram (i.e. there will always be some bins containing more pixels than the average); even when the transformation is multi-valued, i.e. mapping pixels in the same bin to different values according to some criteria (see section 2.2.2), a flat histogram may be achieved but the resulting image is not significantly different (Hummel 1977).

In the previous section it was shown that a bin is stretched by the amount $p(i)(K-1)$, when $p(i) \geq \frac{2}{K-1}$. However, if $p(i) \leq \frac{2}{K-1}$ the bin is not stretched at all: instead it is assigned to $\tau[i-1] + \frac{1}{2}p(i-1)(K-1)$. If $\frac{1}{2}p(i-1)(K-1) < 1$ (i.e.

the previous bin has not been stretched) then $r[i] = r[i-1]$, merging the pixels of grey level i and $i-1$ together. This results in the following two problems:

1. Very large grey level bins are stretched by very large amounts;
2. Compression or merging of small bins that are close together in the histogram, with a subsequent loss of information.

The basis of the method i.e. stretching the peaks, is of course debatable, but many images contain a lot of information which may be of interest in the peaks. For example, Figure 4.1 (a) shows an image of a coin and, (b) its histogram; Figure 4.2 (a) shows the image after processing using the histogram equalisation algorithm, and (b) its histogram. In Figure 4.2 the coin is clearly visible against the background. The coin is visible because the pixels making up the coin are in the grey level bins that are separated by the histogram equalisation algorithm.

Figure 4.1: (a) Image of a coin, and (b) its histogram (from Gonzalez and Wintz 1987).

What is much more questionable, however, is the compression of the smaller bins. For example, Figure 4.3 (a) shows the hologram of a can Figure 3.3 af-

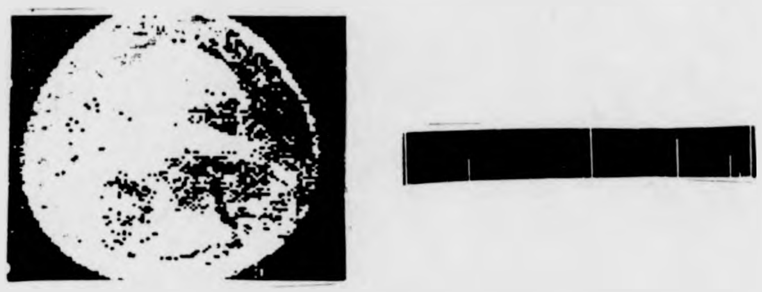


Figure 4.2: (a) Image of a coin after processing using histogram equalisation, and (b) its histogram (from Gonzalez and Wintz 1987).

ter processing using the histogram equalisation algorithm, and (b) its hologram. (Note : The images in this chapter are printed out on a PostScript laser printer; no $offset$ value is used). The histogram of the can Figure 3.3 (b) has a large peak in the low grey level values, and tails off towards the higher grey level values. The peak does in fact correspond to a homogeneous area in the image, the background, so that the histogram equalisation algorithm introduces artificial noise into the image when the peak is separated. The smaller bins in the high grey level values of the histogram are merged together and compressed. As these pixels belong to the same area of the image then the contrast information in this region is lost. The overall contrast of the image is much too high because the peak, which contains no significant information, has been stretched over too large a range, while the remaining bins are compressed into a small number of bins of high grey level value.

Clearly, to make the histogram equalisation algorithm more flexible it needs to be modified so that the stretching given to a peak is balanced against the amount of information lost through compressing the smaller bins in the histogram. This

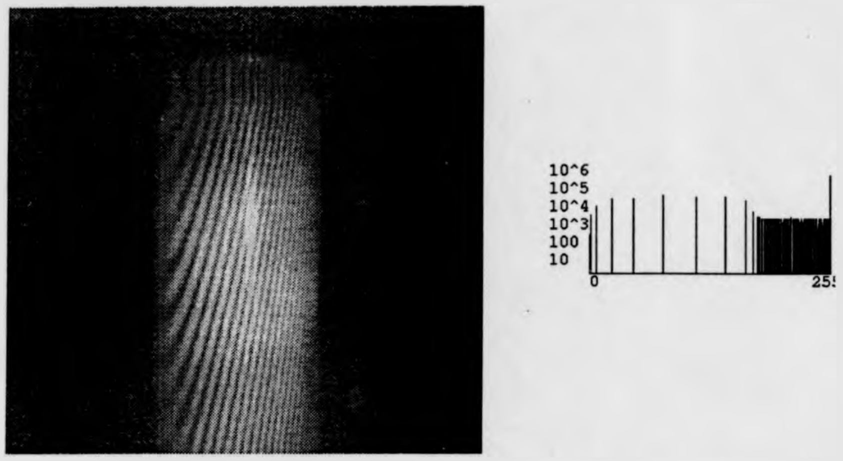


Figure 4.3: (a) Figure 3.3 after histogram equalisation, and (b) its histogram.

was suggested by Kautsky *et al.* (Kautsky *et al.* 1980, 1981, 1982, 1984); this method, *smoothed equidistributing regrading* (see section 2.2.2 p. 22-24), results in a histogram that is a compromise between the original histogram and the histogram that is the result of histogram equalisation. In this chapter algorithms are introduced that complement and extend the histogram equalisation algorithm. Following Kautsky *et al.* each algorithm allows the amount of stretching to be balanced by the amount of information lost.

4.4 Peak smoothing

In this algorithm the user can choose how much of a large peak in the histogram to stretch apart. A peak is defined as a range of adjacent grey level bins $\{i, i+1, \ldots, i+k,\ k \geq 1\}$ each containing at least $A : 0 \leq A \leq 1$ percent of the pixel population (A is a parameter: for example in the histogram equalisation algorithm $A = \frac{2}{K-1}$); one of the bins in the range is the apex of the peak. The user selects the size, A, of one of the bins in the peak. All the bins in the peak (and in other peaks in the histogram) that contain a greater proportion of the total pixel population than A are then stretched in proportion to their size, while smaller bins are retained, rather than merged (as in the histogram equalisation algorithm), by mapping each of them to a separate bin in the transformed histogram.

Let the normalised histogram be $p(i)$, $i : 0, 1 \ldots K-1$, where K is the number of quantisation levels. Let the size of the bin selected by the user be A, and the number of bins with a pixel population less than A (excluding the empty bins) be n. The bins in the new histogram are distributed as follows. If $p(i) > A$ then it will occupy the range of levels:

$$\frac{p(i)}{\sum_i \{p(i) | p(i) > A\}} * (K - 1 - n)$$

The rationale of this algorithm is that the peaks are stretched according to the amount of the original histogram the user wishes to retain. If A is a very large value most of the original histogram is retained, and the peaks are stretched over a very limited range. On the other hand, as A becomes smaller, less and less of the original histogram is retained, and there is more space to stretch the peaks across. For very low values of A, the algorithm approaches the histogram equalisation algorithm. The algorithm thus allows the user to balance the amount of information retained against the amount of stretching, and thus mitigates against the worst excesses of histogram equalisation.

The algorithm can be compared to the smoothed equidistributing regrading developed by Kautsky *et al.* However, peak smoothing always stretches bins with value greater than A over the maximum range possible, whereas the smoothed equidistributing regrading will, at one extreme, approach the original histogram. Thus if the original histogram does not fully occupy the grey level range the histogram will not be stretched as much. The smoothed equidistributing algorithm has a parameter $\lambda : 0 \leq \lambda \leq 1$, that is adjusted to determine the amount of smoothing. If $\lambda = 0$ the result is histogram equalisation, and if $\lambda = 1$ the result is the original histogram. Intermediate values of λ result in a smoothed histogram between the two extremes. The results of processing the holographic image in Figure 3.3 with these two algorithms are compared in Figure 4.4 and Figure 4.5. The histogram of the smoothed equidistributing regrading in Figure 4.4 (b) is close to the original histogram (Figure 3.3 (b)). The peak in the original his-

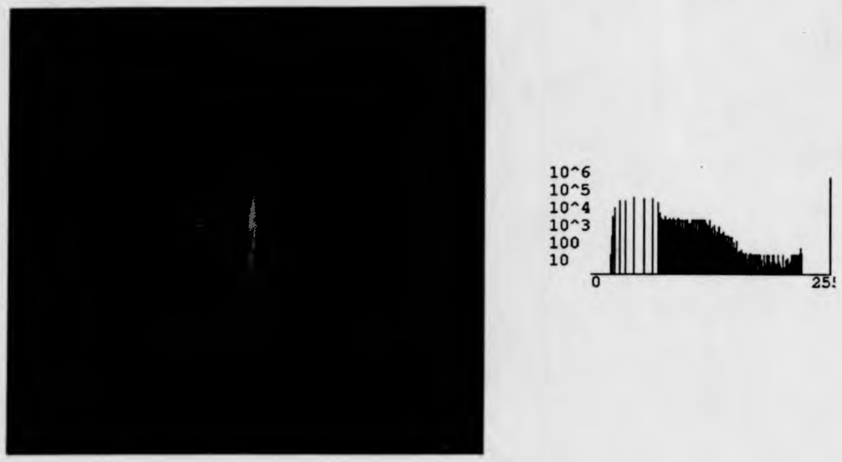


Figure 4.4: (a) Result of processing the hologram of a can with the smoothed equidistributing regrading with $\lambda = 0.27$, and (b) its histogram.

togram is not separated as much as in the histogram equalisation algorithm, so that there is less noise in the background area of the image. Also, no information is lost because the smaller grey level bins in the original histogram are retained, rather than merged as in the histogram equalisation algorithm. Note however that the new histogram does not occupy the full grey level range available. The result of the peak smoothing algorithm with a value of $A = 0.065$ shown in Figure 4.5, is comparable. There is slightly more noise in the background area of the image because the peak in the original histogram is stretched over a larger range. However, the overall contrast of the image is better because the new histogram uses the full grey level range available.

The code for the smoothed equidistributing regrading is given in Appendix E, and for the peak smoothing algorithm in Appendix F.

4.5 Inverse equalisation and smoothed variants

Inverse histogram equalisation does the reverse of histogram equalisation, compressing the large peaks, while spreading the smaller bins in the histogram in proportion to their size. Let A be some percentage of the pixel population,

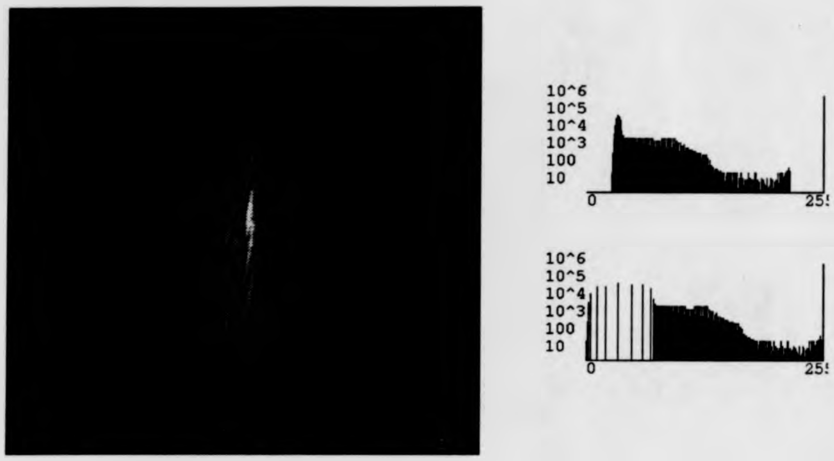


Figure 4.5: (a) Result of processing the hologram of a can with the peak smoothing algorithm with $A = 0.065$, and (b) its histogram.

$0 \leq A \leq 1$. Then, every peak greater than A is counted, and each is compressed into a single bin in the transformed histogram. If there are n such peaks, then n bins are filled in the transformed histogram. This leaves $(K - 1 - n)$ bins to be used for stretching out the remaining bins. That is, if $p(i) < A$ it is assigned the range:

$$\frac{p(i)}{\sum_i \{p(i) | p(i) < A\}} * (K - 1 - n)$$

The method described above is a basic, rather crude version of the inverse histogram equalisation algorithm (in the same way as the histogram equalisation algorithm is a crude version of the peak smoothing algorithm). This algorithm can have severe effects on the histogram in two ways. First, if A is a large value it will cause merging of the smaller bins. Second, if A is a small value it may cause an unacceptable loss of information due to merging of the larger bins. Therefore two smoothed versions of the algorithm are given: smoothed inverse histogram equalisation variants (1) and (2).

Variant (1) is used to balance the compression of smaller bins in the histogram against the amount of stretching. First, a second parameter B, $0 \leq B \leq 1$ is chosen. All bins that have a smaller pixel population than B are retained. If there are q bins smaller than B then each is assigned one bin in the transformed histogram. A is chosen in the same way as in the inverse histogram equalisation algorithm; all bins with a larger pixel population than A are compressed. Each bin with a pixel population greater than B and less than A is then stretched according to its size, i.e.

$$\frac{p(i)}{\sum_i \{p(i)|B < p(i) < A\}} * (K - 1 - n - q)$$

Variant (2) is used to to balance the amount of stretching of the smaller bins against loss of information due to the compression of larger bins. The user again selects two parameters A and B. All the bins greater than A are compressed as in inverse equalisation, and all the bins below B are retained. Bins that are between A and B are stretched in inverse proportion to their size. If there are n peaks with value greater than A, and q bins below B, then there are $(K - 1 - n - q)$ bins remaining. If $p(i)$ is such that $B < p(i) < A$ it is assigned the range:

$$\frac{\sum_i \{p(i)|B < p(i) < A\}}{p(i)} * (K - 1 - n - q)$$

The rationale behind the inverse equalisation algorithm is that large peaks in a histogram sometimes consist of pixels from a uniform area of the image e.g. the background. The histogram equalisation algorithm will force the bins in these peaks apart. The pixels in these bins are all from the same uniform area of the image, but they are now assigned widely different grey level values. Thus artificial noise is introduced into uniform areas of the image. The inverse equalisation algorithm will compress each large peak in the original histogram to a single level in the transformed histogram, and stretch out the smaller bins. Thus, it is also useful when there are smaller bins in the histogram which contain important information, as these areas will be stretched. The two smoothed versions allow the user to balance the amount of information lost by merging in the histogram against the amount of stretching.

The code for the inverse equalisation algorithm and its two variants is given in Appendix G.

4.6 Effects of the algorithms on various Gaussians

The effects of the algorithms on various histograms are now considered. To facilitate analysis, peaks in the histogram are considered as Gaussian distributions with population $P : 0 \leq P \leq 1$ (as a percentage of the total), standard deviation σ, and mean μ.

A histogram that has a dominant Gaussian has a peak in the histogram that contains a large proportion of the pixel population. The peak is generally due to the fact that there are large areas of the image that contain pixels from a small range of grey level values. Two types of dominant Gaussians are considered: Gaussians with a large population and small standard deviation, and Gaussians with a large population and a large standard deviation.

4.6.1 Large population, small standard deviation

Consider a Gaussian with a large population and a small standard deviation. This implies a very sharp peak. The result of the histogram equalisation algorithm on this type of distribution is to stretch the distribution over approximately $P*(K-1)$ grey levels, and to compress the remaining bins into $((K-1) - P)*(K-1)$ grey levels. Two examples are given. Figure 3.3 shows a holographic image of a can. Because of some deficiencies of the camera the full grey level range of the image is not used, so that the overall contrast is poor, and in particular, it is difficult to distinguish the object from the background. Figure 4.6 shows an image of "Trevor". Both images have a dominant Gaussian in the low grey level values due to the preponderance of background pixels in the image, Figure 3.3 (b) and Figure 4.6 (b). The statistics for the dominant Gaussians are shown in Table 4.1. The results of applying the histogram equalisation algorithm to

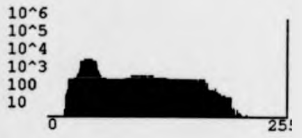
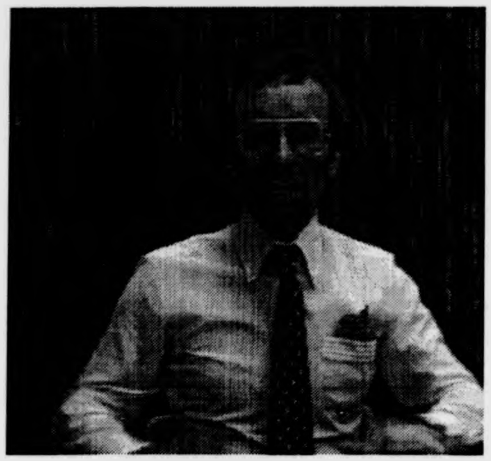


Figure 4.6: (a) Image of "Trevor", and (b) its histogram

Large P, small σ			
Image	μ	P	σ
Figure 3.3	32	0.72	2.0
Figure 4.6	43	0.64	6.5

Table 4.1: Statistics of the dominant Gaussians in Figure 3.3 and Figure 4.6.

Figure 3.3 and Figure 4.6 are shown in Figure 4.3 and Figure 4.7 respectively. Figure 4.3 (b) shows that 11 bins from the dominant Gaussian of Figure 3.3



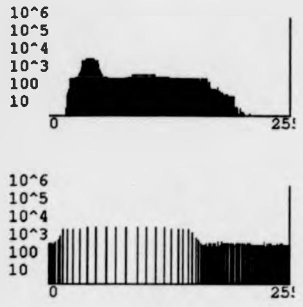


Figure 4.7: (a) Result of applying the histogram equalisation algorithm to Figure 4.6, and (b) its histogram.

occupy 185 grey levels, an average of 16.8 grey levels per bin[1]. Consequently the remaining 182 grey level bins in the histogram occupy only 70 grey levels. Since the Gaussian consists of pixels from a uniform area of the image the separation introduces artificial noise. The high grey level values have been compressed into a small number of grey level values at the end of the grey scale, so that many of the contrasts which correspond to the fringe patterns are lost, and the overall contrast of the image is too high.

The result (Figure 4.7) of applying the histogram equalisation algorithm to Figure 4.6 shows a similar, if better result. Figure 4.6 has a larger standard deviation and a smaller population than Figure 3.3, indicating a less dominant role for the Gaussian. In Figure 4.7, 28 bins from the Gaussian occupy 156 grey levels, a stretch ratio of 5.5. However, the remaining 160 bins occupy 100 grey levels. In this case the dominant Gaussian contains information that can only be seen after separation. Histogram equalisation enhances this information, but destroys contrasts in other regions notably the shirt and face. The smaller stretch

[1]This value is hereafter known as the stretch ratio.

ratio is due to the larger standard deviation.

The peak smoothing algorithm produces better results for both Figure 3.3 and Figure 4.6, since it limits the amount of stretching of the dominant Gaussian, and allows many of the smaller bins to be retained. The results are shown in Figure 4.5 and Figure 4.8. Figure 4.5 is the result of using peak smoothing to stretch peaks

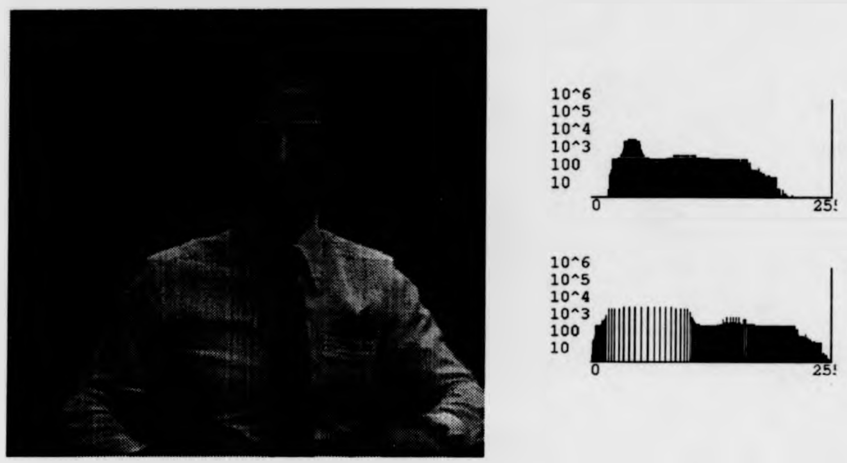


Figure 4.8: (a) Result of processing Figure 4.6 with the peak smoothing algorithm with a value of $A = 0.116$, and (b) its histogram.

within one standard deviation of the mean. This stretches 6 bins over 70 grey levels, a stretch ratio of 11.7. Similarly Figure 4.8 stretches 17 bins over 80 grey levels, a stretch ratio of 4.7. The stretch ratios are similar to those of the results of the histogram equalisation algorithm, but peak smoothing produces much better contrast, because none of the smaller bins are merged. Note that there is no need for any compression because there are plenty of empty bins in the histogram. If the histogram is full, then some compression is necessary to produce stretching.

Inverse equalisation can be applied to Figure 3.3 to compress the Gaussian, and spread out the remaining information in proportion to the bin size. In this image, A is chosen so that the majority of the Gaussian, within one standard deviation of the mean, is compressed. Inverse equalisation gives good enhancement of the border region but at the expense of compressing the smaller bins. The result is shown in Figure 4.9. Less compression of the smaller bins is obtained by

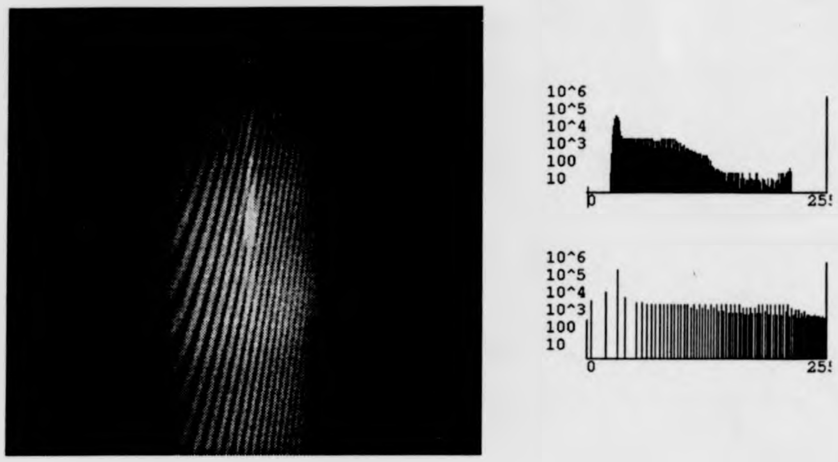


Figure 4.9: (a) Result of inverse histogram equalisation of Figure 3.3 with a value of $A = 0.143$, and (b) its histogram.

using variant (1) of the smoothed inverse equalisation algorithm, with values of $A = 0.143$ and $B = 0.024$ shown in Figure 4.10.

The results of applying the inverse equalisation algorithm and its smoothed variants to Figure 4.6 are not shown, because they do not enhance the background pixels contained in the main Gaussian. However they can be used to enhance smaller Gaussians in the histogram.

4.6.2 Large population, large standard deviation

A Gaussian with a large population and a large standard deviation implies a broad peak that contains most of the image population. Histogram equalisation will produce a good result for this type of image, because most bins in the histogram will be stretched by a small amount, and few bins will be compressed. Figure 4.11 shows a photometric image. The image can therefore be considered as being made up of two Gaussians, one containing the photometric image, and the other the background information. The statistics for these two Gaussians are shown in Table 4.2. Table 4.2 indicates that the background peak with $\mu = 91$ is a dominant Gaussian with a large standard deviation.

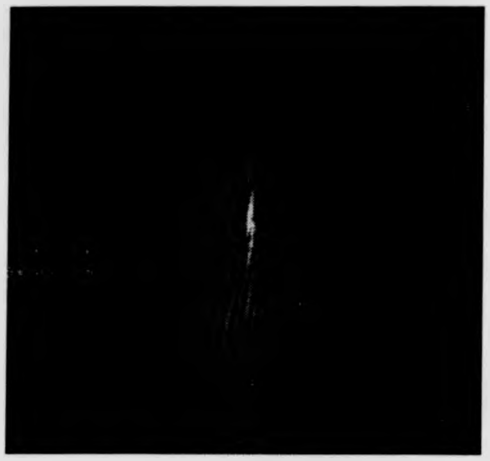
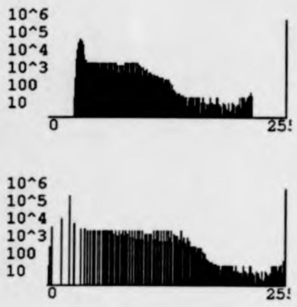



Figure 4.10: (a) Result of smoothed inverse equalisation (variant (1)) on Figure 3.3 with values of $A = 0.143$ and $B = 0.024$, and (b) its histogram.

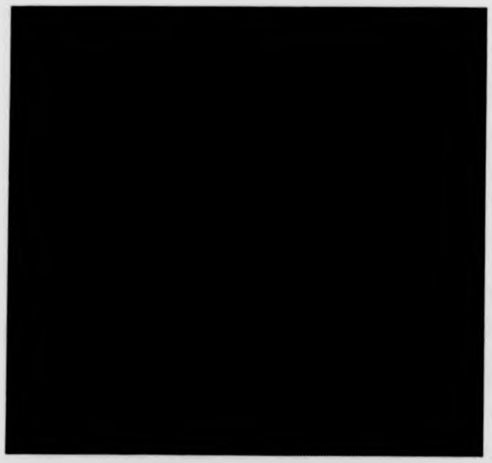
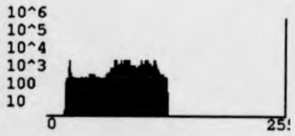
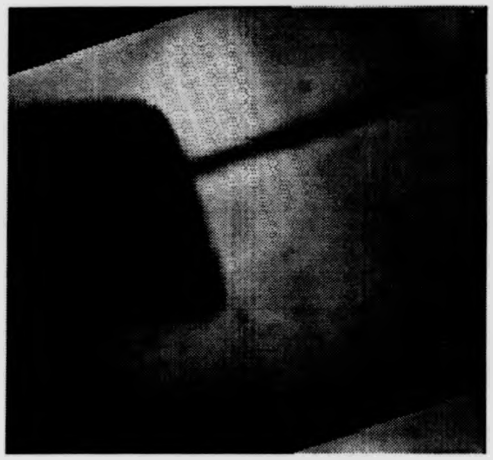



Figure 4.11: (a) Photometric image, and (b) its histogram.

Large P, large σ		
μ	P	σ
37	0.15	11.7
91	0.85	17.0

Table 4.2: Statistics of the two Gaussians in Figure 4.11.

The result of applying the histogram equalisation algorithm to Figure 4.11 is shown in Figure 4.12. Figure 4.12 shows that 60 bins from the dominant Gaussian



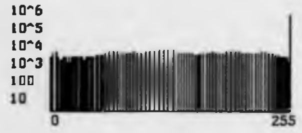


Figure 4.12: (a) Result of processing Figure 4.11 with the histogram equalisation algorithm, and (b) its histogram.

occupy 200 grey levels, a stretch ratio of 3.3. The remaining 50 bins occupy 56 grey levels[2]. Histogram equalisation stretches the dominant Gaussian effectively, and the image has a good contrast between the two regions.

The result of using the peak smoothing algorithm does not produce a better image. It is only when the dominant Gaussian has a small standard deviation that histogram equalisation produces an unacceptable destruction of contrast information, due to an excessive amount of stretching of the Gaussian.

The inverse equalisation algorithm can be applied to Figure 4.11 to enhance the information in the smaller Gaussian (see Figure 4.13). A has to be chosen so that it is higher than all the bins in the Gaussian corresponding to the dark rhombus shaped object in the image. A hole in the middle of this area is now clearly visible. However most of the information in the higher bins in the histogram has been destroyed. The first variant of the smoothed inverse equalisation algorithm produces a similar result because it does not preserve bins larger than

[2]The leftmost Gaussian has three bins that contain more than the average number of grey levels.

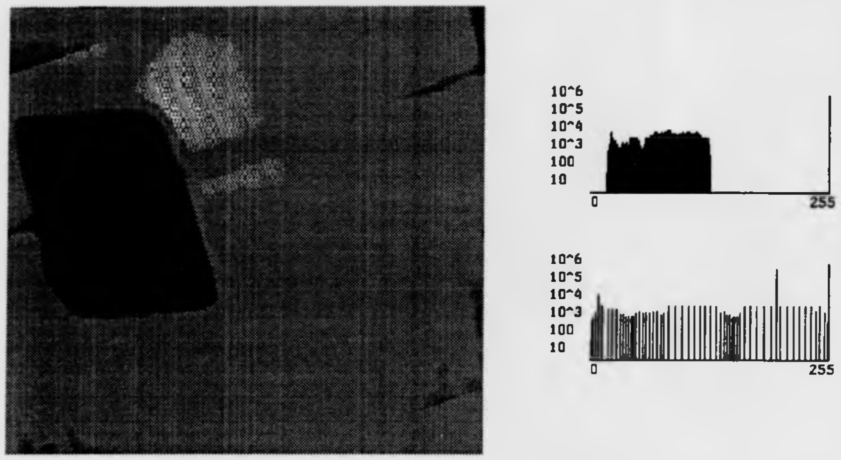


Figure 4.13: (a) Result of applying the inverse equalisation to Figure 4.11 with $A = 0.312$, and (b) its histogram.

the value of A. However, if the second variant is used with the values of A and B shown in Figure 4.14, more of the background Gaussian is preserved. The value of A is chosen so that most of the background Gaussian is preserved. Since the bins below A are stretched in inverse proportion to their size (as long as they are above B) the hole is clearly visible.

4.6.3 Two or more Gaussians.

The previous examples contained a dominant Gaussian, and some smaller Gaussians. Histograms that contain no dominant Gaussian, but a number of Gaussians are now considered. From the previous analysis histogram equalisation is known to favour Gaussians with large populations and small standard deviations. If a histogram is made up of mixed Gaussians, then these will be stretched depending on their standard deviation and population size. An approximation to the stretch ratio defined earlier can be obtained using the following formula:

$$SR = \frac{P * (K - 1)}{4 * \sigma}$$

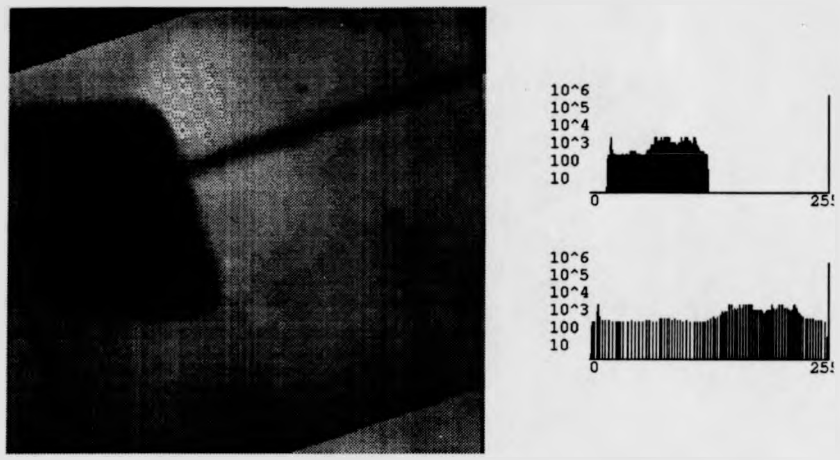


Figure 4.14: (a) Result of smoothed inverse equalisation (variant (2)) on Figure 4.11 with values of $A = 0.903$ and $B = 0.099$, and (b) its histogram.

The standard deviation is multiplied by 4, because most of the Gaussian's population lies within two standard deviations of the mean. An image that has a histogram with mixed Gaussians, the semiconductor image, is shown in Figure 4.15. Figure 4.15 consists of two dominant Gaussians, corresponding to the

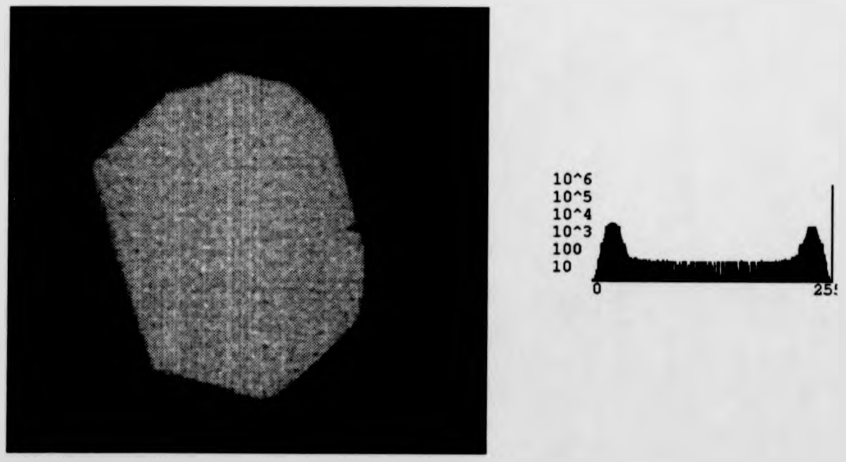


Figure 4.15: (a) Semiconductor image with two Gaussians and (b) its histogram.

dark background and the white semiconductor. The statistics of this image are shown in Table 4.3. The statistics in Table 4.3 indicate that the first peak will

μ	P	σ	SR
22	0.66	5.0	8.4
234	0.3	5.9	3.3

Table 4.3: Statistics of the two dominant Gaussians in Figure 4.15.

be stretched by a large amount, and the second peak by a smaller amount. The result of equalisation of the image is shown in Figure 4.16. Both Gaussians are

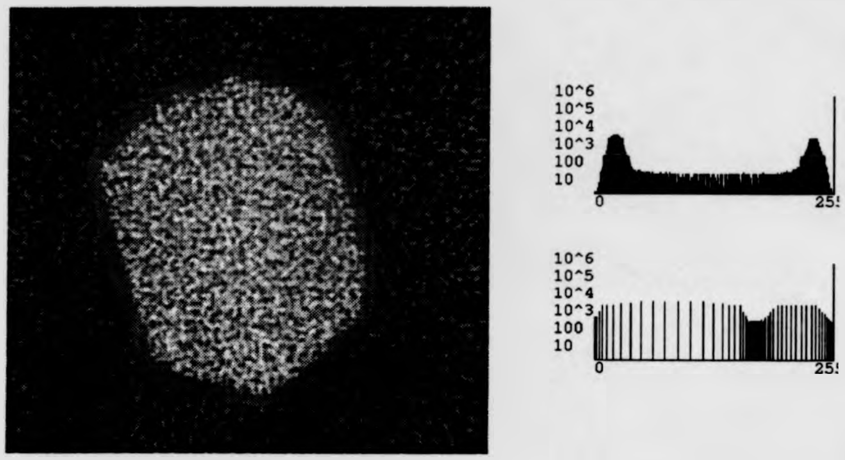


Figure 4.16: (a) Result of applying the histogram equalisation algorithm to Figure 4.15, and (b) its histogram.

stretched effectively, but the compression of the smaller bins in the histogram causes a loss of information in the border region of the semiconductor.

Using the peak smoothing algorithm, A must be chosen so that there is sufficient space to stretch the larger bins in the histogram. Most of the histogram is occupied by smaller bins, so A must be a small value in order to compress bins slightly larger than A, and to stretch bins much larger than A. The result is shown in Figure 4.17.

For this image the inverse equalisation algorithm can be used in two ways. First the information in the first Gaussian can be compressed, and a large stretch given to the smaller Gaussian (see Figure 4.18). Or both Gaussians can be compressed to stretch the smaller bins (see Figure 4.19). The smoothed inverse equal-

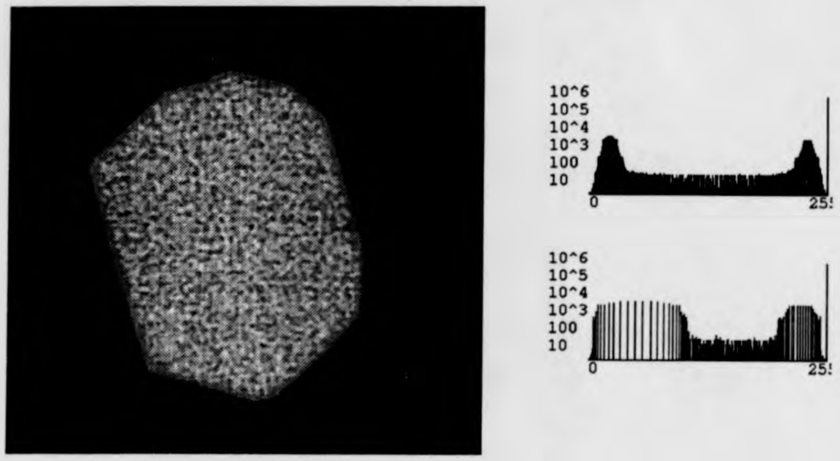


Figure 4.17: (a) Result of applying the peak smoothing algorithm to Figure 4.15 with $A = 0.004$, and (b) its histogram.

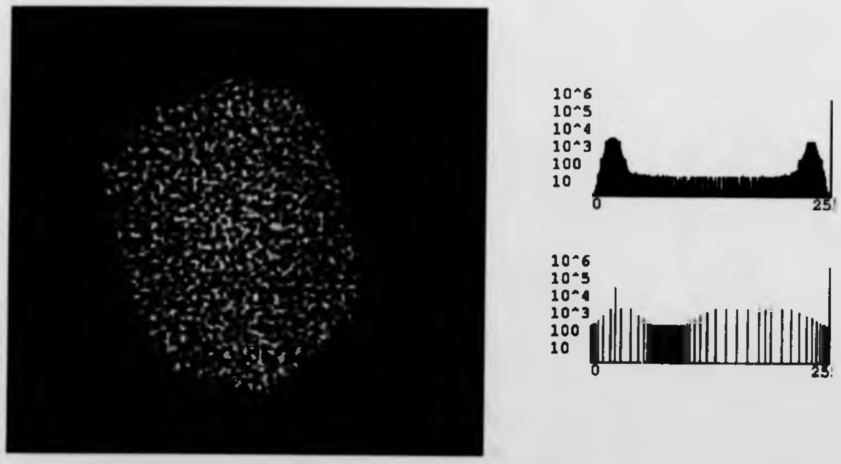
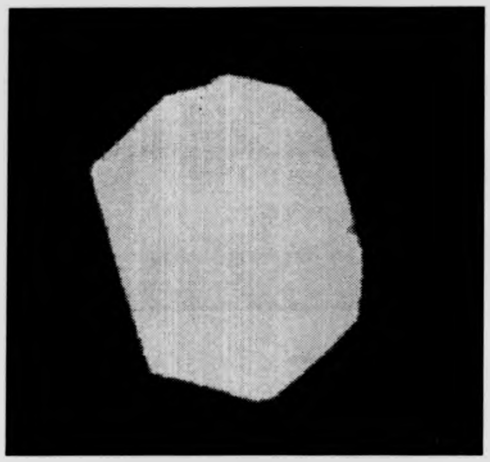
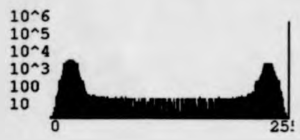


Figure 4.18: (a) Result of applying the inverse equalisation to Figure 4.15 with $A = 0.359$, and (b) its histogram.

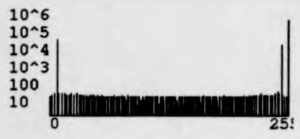


Figure 4.19: (a) Result of applying the inverse equalisation to Figure 4.15 with $A = 0.015$, and (b) its histogram.

isation algorithm (variant 1) can be used to reduce the amount of destruction of smaller bins in the histogram caused by the inverse equalisation algorithm, as shown in Figure 4.18. The result is shown in Figure 4.20. The result of applying the second variant of the smoothed inverse equalisation algorithm is not shown because there is not sufficient space in the histogram to produce good stretching.

4.7 Discussion

The algorithms presented in this chapter have certain limitations, due to their global nature. When using a global algorithm, it is not possible to stretch all regions of the histogram sufficiently, since there is not enough space available in the histogram. Only a certain range of grey levels can be stretched, and other ranges must be compressed in order to create space. However, the algorithms allow the user to balance the amount of compression against the amount of stretching.

There are many cases where the algorithms are not effective. For example, if there are two Gaussians of similar height, it is not possible to selectively stretch one at the expense of the other. Also if a Gaussian is mixed, i.e. it contains pixels from many different areas, it is not possible for the algorithm to make such

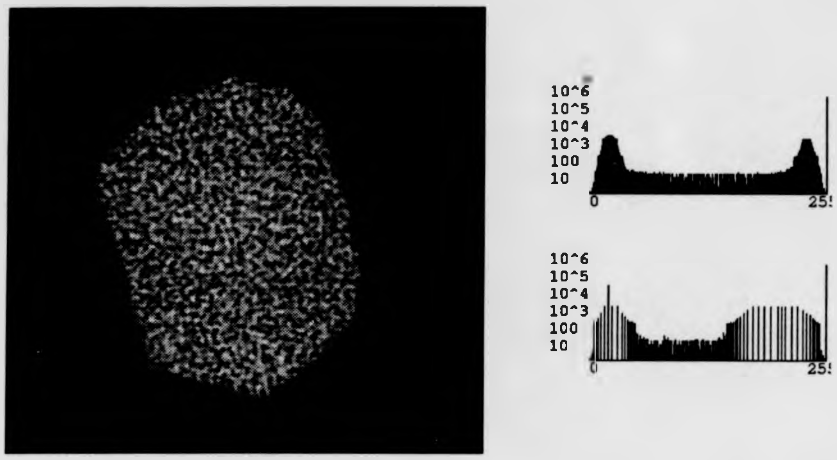


Figure 4.20: (a) Result of applying the smoothed inverse equalisation algorithm (variant 1) to Figure 4.15 with $A = 0.359$ and $B = 0.005$, and (b) its histogram.

a distinction. In general the algorithms will work best when the Gaussians are from distinct regions of the image, and are separated in the histogram. Table 4.4 summarises the cases when the algorithms produce significant enhancement.

Algorithm	Best Case
Histogram equalisation	Used to stretch large Gaussians. Best for Gaussians with large populations and large standard deviations
Peak smoothing	Balances the stretching of large Gaussians against the compression of smaller bins
Inverse equalisation	Used for compressing large Gaussians with small standard deviations, and for stretching smaller Gaussians at the expense of large Gaussians
Smoothed inverse equalisation, variant (1)	Used to balance compression of the smaller bins caused by inverse equalisation
Smoothed inverse equalisation, variant (2)	Used to balance compression of larger bins by inverse equalisation against stretching of smaller bins

Table 4.4: Summary of results.

Chapter 5

A study and modification of the local histogram equalisation algorithm

5.1 Introduction

Local histogram equalisation (LHE) or local histogram modification (LHM) was developed by Ketcham *et al.* (Ketcham *et al.* 1974). The transformation is exactly the same as in the global histogram equalisation algorithm (page 17). The difference is that the transformation of *each pixel* in the image is determined from a histogram of the pixels in a surrounding neighbourhood of some pre-determined size. If the local neighbourhood is sufficiently small it will ensure that even the detail in the smallest regions is enhanced. Thus the algorithm is generally employed when the global histogram equalisation algorithm does not enhance the detail in small regions of the image (Gonzalez and Wintz 1987).

In its initial form the transformation of each pixel was determined by calculating the histogram in a square neighbourhood of a given size. As might be imagined, the need to calculate a new histogram for every pixel in the image is computationally expensive. The efficiency of the algorithm can be increased since the histograms of adjacent pixels have a large number of pixels in common, but it is still slow, particularly as the computation time increases with the size of the image. Recently a number of authors (Leszczynski and Shalev, 1989, Pizer *et al.* 1987), have published fast algorithms for computing local histogram equalisation.

However, speed is not the only problem with the algorithm. The size of the neighbourhood chosen affects the results. Small neighbourhoods enhance the detail greatly, and are particularly effective at revealing information in small regions of the image, but they create artificial noise in large homogeneous areas. On the other hand, a large neighbourhood does not enhance small regions, but produces an image that is fairly close to the original. The reason for this is that a small neighbourhood usually identifies the region where the pixel comes from; thus the local histogram equalisation transformation will stretch this region, and the pixel will be separated from the region background (i.e. the mode of the local histogram) in the sense of equation 1.1. In this case the local histogram equalisation algorithm finds the correct local background. Also since the histogram only contains a small number of pixels, occupying a relatively small number of grey levels, the stretch is quite large. In contrast, a large window will probably contain a number of regions, and only rarely does it give a good approximation to the local background grey level. Also, because it contains a large number of pixels in a large number of grey levels the stretch is comparatively small. What is needed then, is some "smooth" compromise (after Kautsky *et al.* 1984) between the two extremes - one which will give a good approximation to the local background in most cases and which will enhance all regions of the image equally.

In this chapter the local histogram equalisation algorithm is examined in detail. An adaptation of the algorithm is suggested that involves varying the window size over different regions of the image (Dale-Jones and Tjahjadi 1993). This enables each region of the image to be enhanced equally.

5.2 Calculation of LHE over a square neighbourhood

A square neighbourhood is defined as follows. Let $f(x, y)$ be a square image of dimension $N \times N$, i.e. $x = 0 \ldots N-1$, $y = 0 \ldots N-1$. A square window, W, around a point (x, y) is a subimage or neighbourhood, of dimension $M \times M$ as shown in Figure 5.1. If M is odd, (x, y) is the central point of the window.

The square neighbourhood of a point (x, y) may only be defined if, assuming

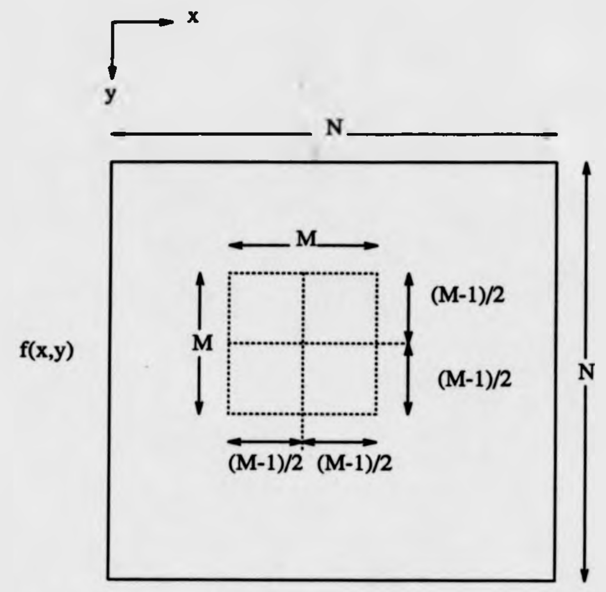


Figure 5.1: A square window of dimension $M \times M$

M odd, $(x,y) \in S = \{(m,n) \mid \frac{M-1}{2} < m,n < N - \frac{M-1}{2}\}$, for the window will not fit over the image in other cases. The neighbourhood of a point $(x,y) \in S'$, shown in Figure 5.2, is defined according to its position. If (x,y) is in the top left hand corner A, that is,

$$(x,y) \in A = \left\{(m,n) \mid m \leq \frac{M-1}{2}, n \leq \frac{M-1}{2}\right\}$$

then the window is:

$$\{(m,n) \mid m \leq M, n \leq M\}$$

Similarly if,

$$\begin{aligned}
(x,y) \in B &= \left\{(m,n) \mid m \geq N - \frac{M-1}{2}, n \leq \frac{M-1}{2}\right\} \\
(x,y) \in C &= \left\{(m,n) \mid m \leq \frac{M-1}{2}, n \geq N - \frac{M-1}{2}\right\} \\
(x,y) \in D &= \left\{(m,n) \mid m \geq N - \frac{M-1}{2}, n \geq N - \frac{M-1}{2}\right\} \\
(x,y) \in E &= \left\{(m,n) \mid \frac{M-1}{2} < m < N - \frac{M-1}{2}, n \leq \frac{M-1}{2}\right\}
\end{aligned}$$

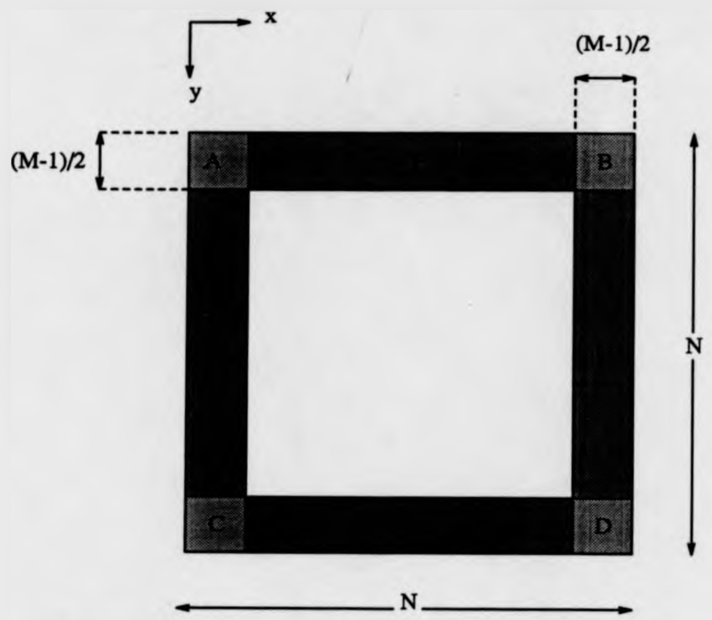


Figure 5.2: Regions requiring a modified window

$$(x,y) \in F = \left\{(m,n) \mid m \leq \frac{M-1}{2}, \frac{M-1}{2} < n < N - \frac{M-1}{2}\right\}$$
$$(x,y) \in G = \left\{(m,n) \mid m \geq N - \frac{M-1}{2}, \frac{M-1}{2} < n < N - \frac{M-1}{2}\right\}$$
$$(x,y) \in H = \left\{(m,n) \mid \frac{M-1}{2} < m < N - \frac{M-1}{2}, n \geq N - \frac{M-1}{2}\right\}$$

the windows are respectively:

$$\{(m,n)|m \geq N - M, n \leq M\}$$
$$\{(m,n)|m \leq M, n \geq N - M\}$$
$$\{(m,n)|m \geq N - M, n \geq N - M\}$$
$$\{(m,n)|x - \frac{M-1}{2} \leq m \leq x + \frac{M-1}{2}, n \leq M\}$$
$$\{(m,n)|m \leq M, y - \frac{M-1}{2} \leq n \leq y + \frac{M-1}{2}\}$$
$$\{(m,n)|m \geq N - M, y - \frac{M-1}{2} \leq n \leq y + \frac{M-1}{2}\}$$
$$\{(m,n)|x - \frac{M-1}{2} \leq m \leq x + \frac{M-1}{2}, n \geq N - M\}$$

In LHE the transformation of the central pixel in the window, W, is found by equalising the histogram of W. Let K be the number of grey levels in the image, i.e. $f(x,y) = 0 \ldots K-1$. The discrete (normalised) histogram or discrete density function of W is given by equation 2.11. The distribution function, D_w, of W is given (Hummel 1975) by (see equation 2.5)

$$D_w(i) = \frac{1}{2}p_w(i) + \sum_{j=0}^{i-1} p_w(j)$$

for $i = 0 \ldots, K-1$. The histogram equalisation transformation over W is defined (Hummel 1975) as (see equation 2.6)

$$\tau(i) = (K-1) * D_w(i)$$

for $i = 0 \ldots, K-1$. Thus the central pixel (x,y) is assigned the value

$$\tau(f(x,y)) = (K-1) * D_w(f(x,y)) \tag{5.1}$$

The code for the LHE algorithm used in this chapter is given in Appendix H.

5.3 The effects of window size on the LHE algorithm

5.3.1 Smoothing effect

An image can be considered as a union of homogeneous regions. For example, the image in Figure 5.3 (a) (displayed on a GDM-1640-40 monitor, as are all the other images in this chapter), consists of a background, a shirt, a face, a tie, a handkerchief and a few others. These regions may themselves be made up of other regions (e.g. a pattern on the tie). The size of the window chosen is an important parameter in LHE. Large windows will, more often than not, have a histogram that is a mixture of the histograms from a number of regions. Such

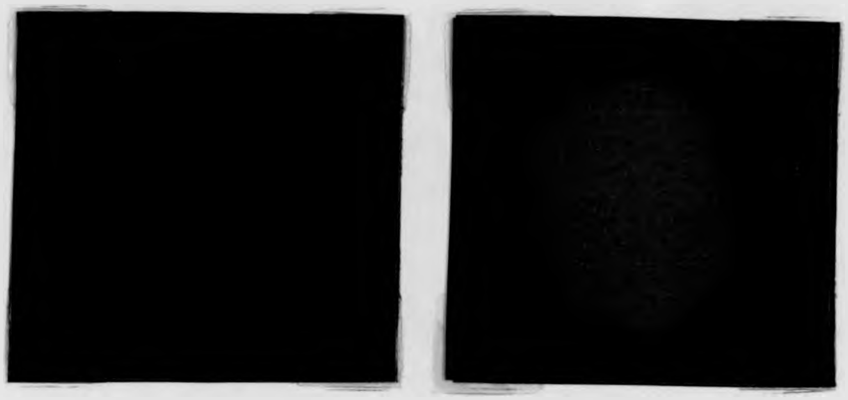


Figure 5.3: (a) Figure 4.6 and (b) Figure 4.15 displayed on a GDM-1604-40 monitor.

a histogram is less likely to have a dominant mode (i.e. a mode that contains a large percentage of the total pixel population) than the histogram obtained from a smaller window, and the mode may also not belong to the same region as the pixel. Therefore according to equation 4.1 there will be restricted stretching (or smoothing) in a few areas of the histogram (and compression elsewhere). On the other hand, smaller windows are more likely to contain pixels from a single region, and to have a dominant mode. Thus equalising the histogram will cause large stretching (according to equation 4.1 again).

Two models are used to illustrate this point. Consider a sub-image split into two parts A and B, as shown in Figure 5.4. A is assumed to be made up of pixels from one region of the image. The distribution of A can thus be approximated by a normal distribution, mean μ. Let B have any distribution, mean μ'. For example, B could be made up of pixels from a number of regions in the image. Its distribution would then be the sum of the distributions of these regions. For simplicity it is assumed that the histograms of A and B have no grey levels in common, and that if a sample of points is taken from A it will have the same

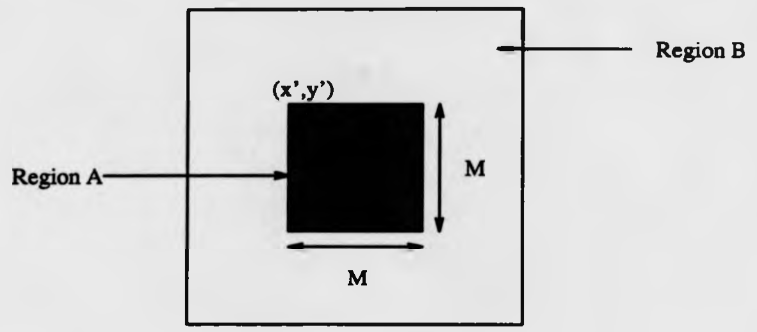


Figure 5.4: Model consisting of two regions: A of dimension $M \times M$ surrounded by another region B.

distribution as A (but scaled). Let region A have dimension $M \times M$ (M odd). To access the effects of varying the window size on the LHE transformation of region A, two points are considered, (x, y) and (x', y'). These two points are chosen because when an $M \times M$ window is positioned at (x, y) it will contain the maximum number of pixels from region A and, conversely, at (x', y') the minimum number of pixels from the region. Their behaviour will thus give an indication of the behaviour of pixels throughout the region. See Appendix I for an analysis of points in region A, other than these two. It is sufficient to study the behaviour of the mode μ in the histogram of the window $\mathbf{p}_w$, i.e. the histogram bin $p_w(\mu)$ at grey level μ, since all other bins in the histogram will be stretched by a smaller amount. Thus it is assumed that $f(x, y) = f(x', y') = \mu$. It is considered that $p_w(\mu)$ is the local background grey level in the sense of equation 2.11; thus if this is stretched then the pixels in the same region will be stretched away from it.

A window of size $M \times M$ is fitted over (x, y) and (x', y'). By equation 4.1, (x, y) is stretched by the amount:

$$p_w(\mu)(K - 1)$$

since the window, when placed at (x, y), contains only pixels from region A. At

(x', y') the stretching is:

$$\frac{\left[\left(\frac{M-1}{2}\right)+1\right]^2}{M^2} p_w(\mu)(K-1) \approx \frac{1}{4} p_w(\mu)(K-1)$$

It is less because there are pixels from both A and B, and A and B have no grey levels in common.

The window size is now increased to lM (odd), where l is a positive rational number. In this case (x, y) is stretched by the amount:

$$\frac{p_w(\mu)M^2}{l^2M^2}(K-1) = \frac{p_w(\mu)}{l^2}(K-1) \tag{5.2}$$

At (x', y') the stretching is also:

$$\frac{p_w(\mu)}{l^2}(K-1)$$

if $\frac{lM-1}{2} \geq M-1$. If $\frac{lM-1}{2} < M-1$ (i.e. $l \approx< 2$) the stretch is

$$\frac{\left[\left(\frac{lM-1}{2}\right)+1\right]^2}{l^2M^2} p_w(\mu)(K-1) \approx \frac{1}{4} p_w(\mu)(K-1)$$

If $l \approx\geq 2$, it is clear that if $p_w(\mu)(K-1) \leq 2l^2$ then all pixels in region A of grey level μ will not be separated by the equalisation algorithm. More generally, as was shown in Chapter 2, adjacent grey levels need to be separated by an amount ΔB to be visible. The value of ΔB depends on the grey level and on the contrast response of the monitor. If $\mu < \mu'$, then as l increases $\tau[\mu]$ will approach zero. Conversely if $\mu > \mu'$, then $\tau[\mu]$ will approach $K-1$ as l increases. The value of ΔB can thus be approximated, and the region will no longer be discernible if:

$$p_w(\mu)(K-1) \leq 2l^2\Delta B \tag{5.3}$$

Note that μ may be in a different position relative to μ' in different parts of the image, as μ' varies according to the local context. This can lead to the same area in the image being mapped to different places in the grey level scale. It does guarantee, however, that the local area is pushed away from the predominant background.

The above model can be extended to that shown in Figure 5.5. If a window of size $M \times M$ is fitted over (x, y) then (x, y) is stretched by $p_w(\mu)(K-1)$. If

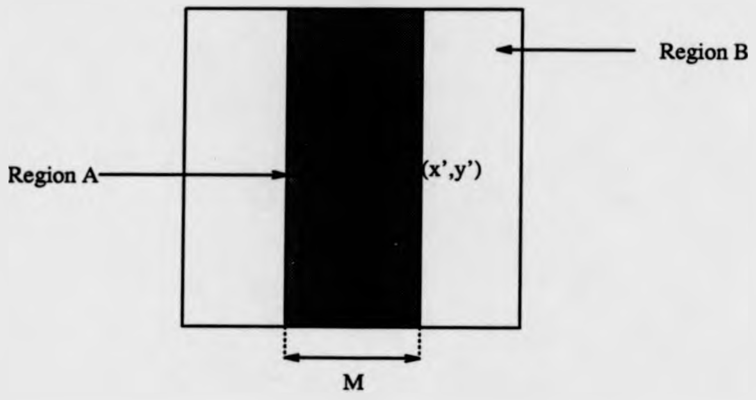


Figure 5.5: Extended model.

the window size is increased to $lM \times lM$, where l is a positive rational number ($l > 1$), then a pixel at (x, y) of grey level μ is stretched by:

$$\frac{lM^2}{l^2M^2} p_w(\mu)(K-1) = \frac{p_w(\mu)}{l}(K-1)$$

Similarly at (x', y'), if $\frac{lM-1}{2} \geq M-1$ then the stretching is given by

$$\frac{p_w(\mu)}{l}(K-1)$$

If $\frac{lM-1}{2} \leq M-1$ (i.e. $l \approx< 2$) then the stretch is

$$\frac{lM\left(\frac{lM-1}{2}+1\right)}{l^2M^2} \approx \frac{1}{2} p_w(\mu)(K-1)$$

In this model region A will no longer be stretched (for $l \approx\geq 2$) if:

$$p_w(\mu)(K-1) \leq 2l\Delta B$$

These models can be used to analyse the tie and handkerchief in Figure 4.6 (a). The tie represents region A in Figure 5.5, and the shirt region B. For region A a sample indicated a standard deviation $\sigma = 11$, and mean $\mu = 35$. In this case region B can also be approximated by a normal distribution, since the pixels in B are all from the shirt area. For B, $\sigma = 12$ and $\mu = 112$. There is no overlap between the two histograms. From the sample $p_w(\mu)$ was estimated as 0.0719. M is approximately 25. Thus for windows of size 51, and 75 the mode is separated from the adjacent bins by 4 and 3 grey levels respectively. If a Hitachi LR47156 monitor is used to display the image, then the required spacing is about 4 to 7 in the lower grey level values, so that the detail in the tie should be visible, even with window as large as 75×75. If an offset of 50 is used for the GDM-1604-40, then the detail will also just be visible. This is indeed the case (see Figure 5.6).

The handkerchief in Figure 4.6 (a) can be approximated by region A in the first model. For region A in Figure 5.4 a sample indicated $\sigma = 6.5$ and $\mu = 58$. From the sample $p_w(\mu)$ was estimated as 0.061, and M is approximately 12. From equation 5.3 with $\Delta B = 3$, l must be ≥ 1.6 for the detail in the handkerchief to be visible. This implies that a window of size $lM = 12 \times 1.6 = 19 \times 19$ or larger will not enhance the handkerchief.

Figure 5.6 shows Figure 4.6 (a) after LHE with windows of size 15×15, 31×31, 47×47 and 63×63. As the window size increases the image appears to get smoother, and detail within smaller regions disappear, e.g. the tie and handkerchief. The results from the two models show that as the window size increases a small region is stretched by a smaller and smaller amount, until it is finally compressed. Therefore, although a large window results in a smooth image, it has the undesirable effect of compressing smaller regions.

5.3.2 Boundary variation

The size of the window used in LHE also affects the transformation near boundaries. For example, Figure 5.7 shows Figure 4.15 after transformation with windows of size 15×15, 31×31, 47×47 and 63×63. As the window size increases

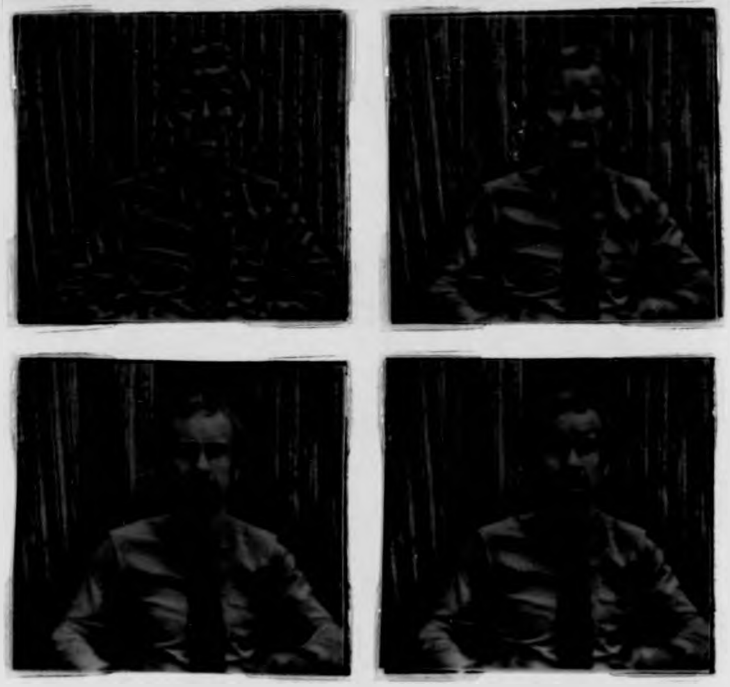


Figure 5.6: LHE of Figure 4.6 (a) using windows of size (a) 15×15, (b) 31×31, (c) 47×47 and (d) 63×63 (clockwise from the top left), (displayed on a GDM-1604-40 monitor).

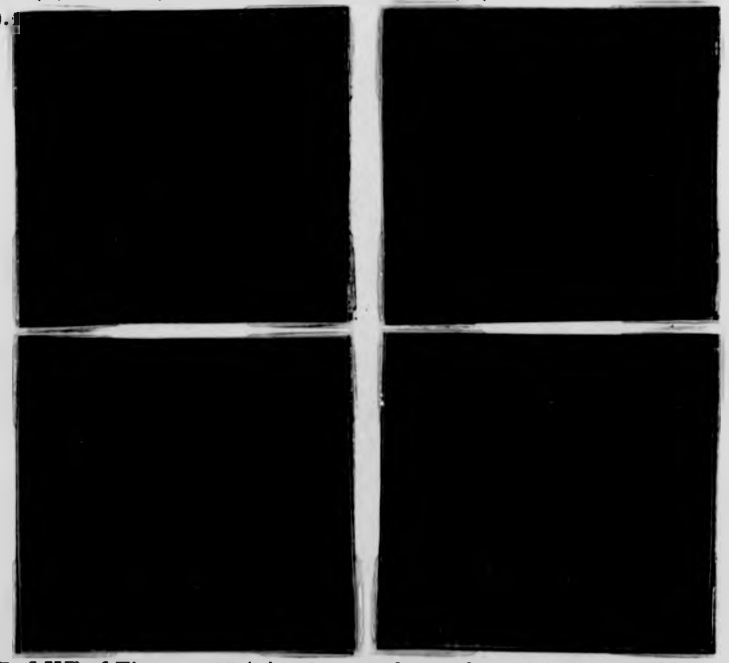


Figure 5.7: LHE of Figure 4.15 (a) using windows of size (a) 15×15, (b) 31×31, (c) 47×47 and (d) 63×63 (clockwise from the top left), (displayed on a GDM-1604-40 monitor).

the transition (edge) between the semi-conductor object and the background becomes broader, producing a distinctive smoothed gradient. The model shown in Figure 5.8, can be used to illustrate this point. It is again assumed that A is a

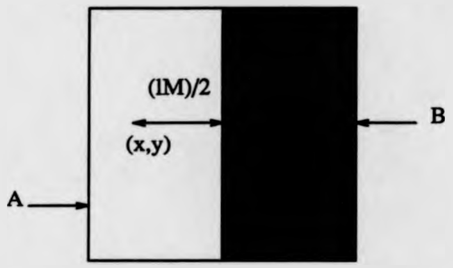


Figure 5.8: Two regions meeting at a boundary.

distinct region with a normal distribution, mode μ, and that any sample taken from the region has the same distribution (but scaled). The point (x, y) in A is at a distance $\frac{lM}{2}$ from region B, where l is a positive rational number. It is assumed to have grey level μ. If a window of size $lM \times lM$ is placed over (x, y), where l is a positive rational number and lM is odd, then (x, y) is stretched by the amount:

$$p_w(\mu)(K-1)$$

Now consider points (x', y') where $x' = x + k$, $k = 0, \ldots \frac{(M-1)}{2}$, $y' = y$, and (x', y') $= \mu$. Then (x', y') is stretched by:

$$\left[\left(\frac{(lM-1)}{2}+1\right)lM + \left(\frac{(lM-1)}{2}-k\right)lM\right]\frac{p_w(\mu)(K-1)}{l^2M^2}$$
$$= -\left(\frac{p_w(K-1)}{lM}\right)k + p_w(\mu)(K-1)$$

This is a straight line of gradient $-\frac{p_w(\mu)(K-1)}{lM}$. Clearly, the stretching decreases over a distance of $\frac{(lM-1)}{2}$ until the boundary edge. If B also has a normal distribution, then this effect will be observed on either side of the dividing line. If M is small then the gradient of the line is at its largest, so that there is a sharp transition over a small distance. On the other hand, as M increases there is a gradual smooth change over a large number of pixels.

5.4 Proposed algorithm

The two problems associated with LHE have been noted above. Large windows produce smooth results in many parts of the image where it is desirable, but unfortunately, they compress smaller regions. Boundary variation is caused by variations in the LHE mapping as a window crosses the border between two regions. Clearly, boundary variation can be avoided by using small sized windows. However, this has the undesirable effect of causing noise due to large stretching, especially when the mode of the histogram is large. Here, we seek a solution that will avoid boundary variation, enhance small regions, and have a degree of smoothness.

A method is proposed to smooth each region of the image to a pre-determined value, by varying the window size over each pixel. There are two steps in the algorithm:

1. An estimate of the region a pixel belongs to is obtained from a small window surrounding a pixel. The grey level range $[r_l, \ldots, r_u]$ in which the majority of the pixels in the window lie characterises the region.

2. The small window is increased in size. As the window is enlarged the region will become subsumed into a larger histogram by the addition of pixels from other regions of the image. The window size is increased until the stretch given to the region after applying histogram equalisation is equal to the preset value of the smoothness.

If the algorithm is applied to a large region of the image, the window size required to smooth the region is very large. On the other hand, in small regions of the image, or near edges, only a small window will be needed to obtain the desired value. Thus, the two problems caused by using a fixed window size, inconsistent stretching and boundary variation, are overcome.

5.4.1 Estimation of the region a pixel belongs to

The first step in the algorithm is to obtain an estimate of the region to which a pixel belongs (see Tomita and Tsuji 1977, and Nagao and Matsuyama 1979). The variance $v_0, \ldots, v_8$ of nine 3×3 windows surrounding the pixel is calculated (see Figure 5.9),

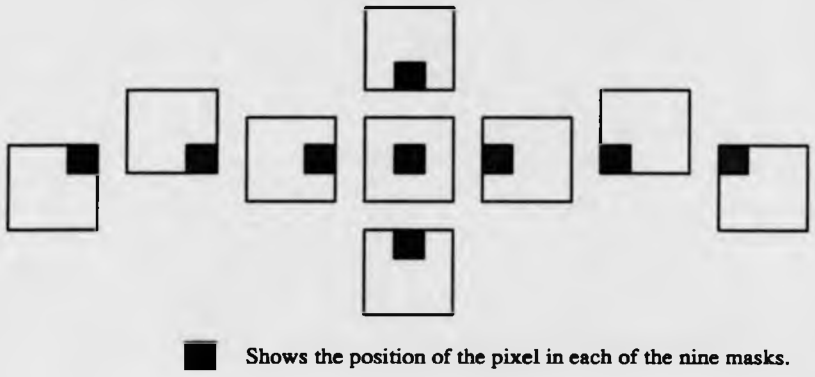


Figure 5.9: The nine masks used to determine the variance.

$$v_i = \frac{1}{8}\left[\sum_{j=0}^{8} z_{ij} - \frac{1}{9}\left(\sum_{j=0}^{8} z_{ij}\right)^2\right]$$

where $i = 0, \ldots, 8$, and the z_{ij}, $j = 0, \ldots, 8$, are the nine pixels in the window v_i. This is only possible for pixels (x, y) where $1 \leq x \leq N-2$ and $1 \leq y \leq N-2$. For pixels not satisfying this condition (i.e. pixels around the edge of the image), the region chosen is shown in Figure 5.10. Otherwise, the pixel is considered to

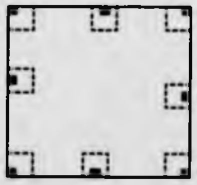


Figure 5.10: Pixels around the edge of the image and the regions they are assigned.

belong to the 3×3 window with the smallest variance, i.e.

$$v_k = \min_i v_i$$

where $i = 0, \dots, 8$. The mean of this region:

$$\mu = \frac{1}{9}\left(\sum_{j=1}^{9} x_{kj}\right)$$

where x_{kj}, $j = 0, \dots, 9$ runs over the pixels in the k^{th} window, and its standard deviation:

$$\sigma = \sqrt{v_k}$$

are calculated. Most of the pixels in the window lie in the grey level range $[\mu - \sigma, \dots, \mu + \sigma]$, and this range is used to characterise the region.

This estimate, using a 3×3 window, is not always large enough to characterise a region. If it is not, the 3×3 neighbourhood with the smallest variance is increased to a 5×5 window in the manner shown in Figure 5.11 (a). Pixels around the edge of the image have their region expanded similarly, in the manner

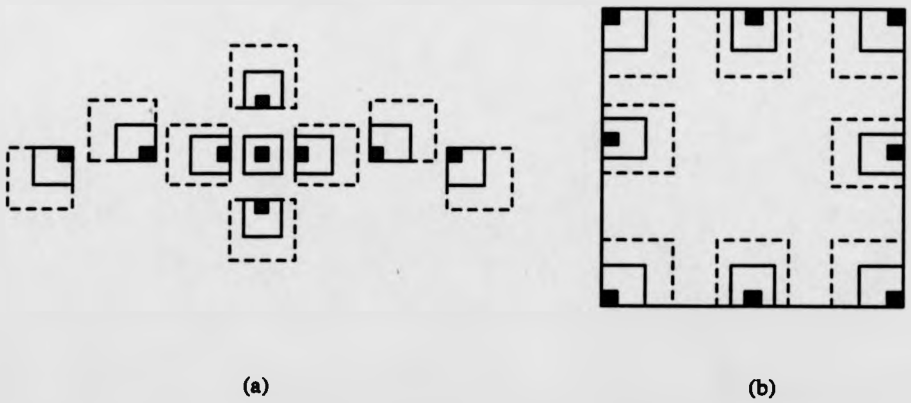


Figure 5.11: Expansion of each 3×3 window to a 5×5 estimate of the region.

shown in Figure 5.11 (b).

5.4.2 Smoothing the region

Let p_w be the normalised histogram of the 3×3 or 5×5 window selected. Two methods of estimating the stretch given to the region are suggested.

1. If n is the number of occupied bins between $\mu - \sigma$ and $\mu + \sigma$, then an initial estimate of the stretch given to the region is obtained by s,

$$s = \frac{\sum_{i=l}^{u} p_w(i)(K - 1 - o_{off})}{n}$$

the average stretch (or stretch ratio) of the area.

2. An estimate of the stretch given to the mode is obtained by finding the largest bin l between $\mu - \sigma$ and $\mu + \sigma$. The stretch given to that bin is then:

$$s = p_w(l)(K - 1 - o_{off})$$

The value of s controls the smoothness of the image. Clearly, the lower s is, the smoother a region will be. The object of the algorithm is to smooth all regions to a preset value of s, S.

As mentioned earlier the value of ΔB varies over the grey level range, but if a large enough offset is chosen the response of the monitor over the remaining grey level values is nearly constant. In method 1, this offset is found, and a preset value of S is used. Method 2, however, is more flexible. If the largest bin is at l, $l \in \{\mu - \sigma, \ldots \mu + \sigma\}$, then the position it is mapped to, $T[l]$, can be obtained from

$$\tau[l] = o_{off}\frac{1}{2}p_w(l)(K - 1 - o_{off}) + \sum_{i=0}^{l-1} p_w(i)(K - 1 - o_{off})$$

A look up table can be used to find the required separation between two adjacent bins at this position. This is then the required value of S. Hence, if necessary, the offset can be adjusted.

If, in either method, the initial window does not give the required value of S, the window size W is increased, $W = 2k + 1$, $k = 1, \ldots, N/2 - 1$ (where N is the dimension of the (square) image), and a new value of s is calculated and compared (in method 2, a new value of S must be found also). The way the window is increased is described in Appendix J. If s_k is the ratio obtained from a window of size $2k + 1$, and s_{k-1} is the ratio from a window of size $2(k - 1) + 1$,

$k > 1$ then the window of size $k-1$ where

$$s_{k-1} \leq S \leq s_k \tag{5.4}$$

is chosen.

If the initial value of s (obtained from a 3×3 or 5×5 window) is $\leq S$ then this window immediately satisfies the criterion of equation 5.4. On the other hand, if no two windows satisfy the criterion then the largest possible window is selected (i.e. global histogram equalisation).

The code for the algorithm is given in Appendix K.

5.5 Experimental results

The results of applying method 1 to Figure 5.3 (a) and (b) are now considered. In the examples an initial window size of dimension 3×3 is used. For Figure 5.3 (a) a typical point in each of six regions is chosen, and the size of the window chosen for $S = 2.6$ and $S = 3.0$ are compared in Table 5.1. The resultant images are

Region	window size	
	$S = 2.6$	$S = 3.0$
Tie	33	29
Handkerchief	21	21
Shirt	79	59
Head	15	13
Background	255	255
Arm/Background	55	51

Table 5.1: Table showing the results with the stretch ratio $S = 2.6$ and $S = 3.0$.

shown in Figure 5.12 (a) and (b). Figure 5.12 (a) shows that the shirt area is as smooth as the 63×63 window used in Figure 5.6 (d) while the tie and handkerchief are almost as clear as the 15×15 window used in Figure 5.6 (a). Note that it was impossible to get the background to the required degree of smoothing except in the small area underneath the arm, because the background is stretched too much even if a window covering the whole image is used. Figure 5.12 (b) shows

Figure 5.12: (a) Result of processing Figure 5.3 (a) with $S = 2.6$ and (b) $S = 3.0$ (displayed on a GDM-1604-40 monitor).

a slightly less smooth image. Most of the window sizes in Table 5.1 have been decreased to obtain a more stretched image. Figure 5.13 (a) and (b) are grey maps of Figure 5.12 (a) and (b) respectively, showing how the window size varies over the pixel in the image. These maps are obtained by mapping the pixel's window size W (which lies in the range $0 - 255$), to the value $255 - W$. Thus pixels with a small window are bright, while those with large windows are grey. Figure 5.13 (a) and (b) show that the background has mostly large windows, and the edges have the smallest windows.

The result of applying method 1 to Figure 5.3 (b) with a stretch ratio of $S = 2.6$ is shown in Figure 5.14. The resulting image has a sharp transition between the background and the foreground object rather than the smoothed gradient of Figure 5.7, because the window size decreases near the transition, as illustrated. Thus the stretch is uniform throughout. Figure 5.15 shows the grey map of Figure 5.14.

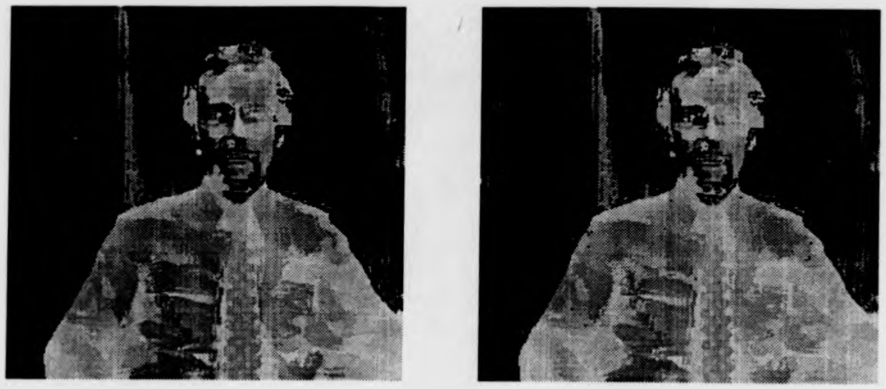


Figure 5.13: (a) Grey maps of Figure 5.12 (a) and (b) respectively, showing how the window size varies. Light pixels have small windows, and dark pixels large ones.

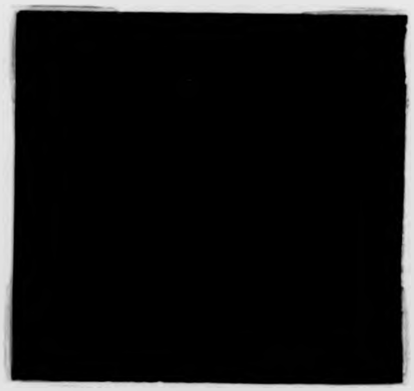


Figure 5.14: Result of processing Figure 1 (b) with method 1 using $S = 2.6$ (displayed on a GDM-1604-40 monitor).

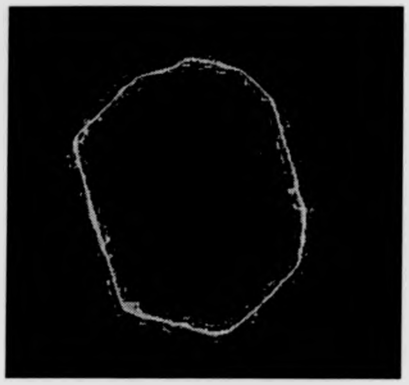


Figure 5.15: Grey map of Figure 5.14, showing how the window size varies. Light pixels have small windows, and dark pixels large ones.

The results of applying method 2 to Figure 5.3 (a) and Figure 4.15 (a) with an offset of 50 and the look up table shown in Table 5.2 are shown in Figure 5.16 (a)

Grey level range	ΔB
50-55	6
56-70	5
71-210	4
211-255	3

Table 5.2: Look up table used for method 2.

and (b) respectively. The results are as smooth as method 1, but the stretching is less, because the separation is calculated for the largest bin rather than as an average over the whole range.

5.6 Discussion

LHE is, in general, a useful technique for highlighting or enhancing local detail in the image. A window is chosen that is small enough to cover the region when it is at the centre: this guarantees that the majority of pixels will be pushed away from the mode - or background of the region. As has been noted, there are problems at the region boundaries when the window size is large, because the

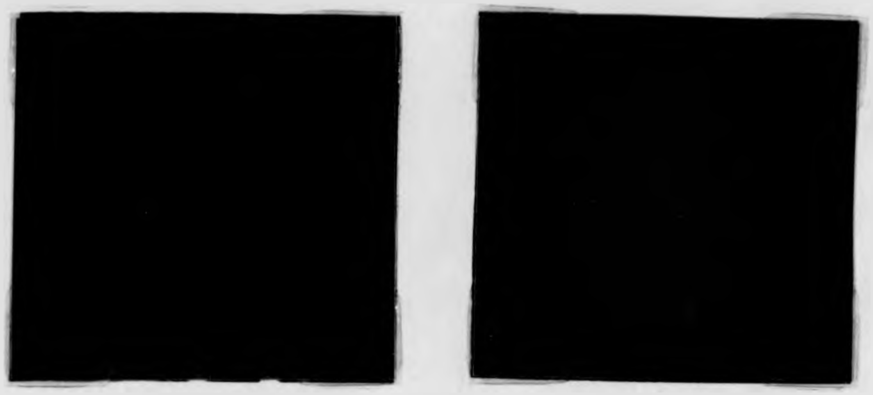


Figure 5.16: Result of processing Figure 5.3 (a) and Figure 4.15 (b) with method 2 (displayed on a GDM-1604-40 monitor).

histogram is bimodal at the edges, and the mode is no longer distinct.

If the image is made up of similarly sized objects/regions then it is a fairly straightforward task to enhance the image (although there may still be boundary variation). The problems arise when there are several regions of different sizes in the image. It is then impossible to choose a window that will enhance each region to a similar degree. A large window will not enhance the small areas, but will give a good result for large areas, and vice versa. Another problem is caused by the fact that the position of a region in the transformed histogram is affected by the surrounding region. Thus the same region may be given a different transformation in different parts of the image.

The modified algorithm suggested in this chapter initially estimates the region on the basis of a 3×3 or 5×5 window. This window size is then increased until the region is enhanced by a preset amount. Thus the window size chosen will echo the region size. If the region is large, a large window is required before the region is smoothed. On the other hand, if the region is small, only a small window will be required to smooth the region. And, as a boundary is approached, the window size will automatically decrease in size, because the mixture of pixels in the window's histogram means that a smaller window is required to reach the preset value.

There are some problems with the algorithm. It is possible for the same region to be mapped to different parts of the grey level scale; the initial estimate of a region can lead to errors - it may be too large to pick out very fine detail and too small to give a good approximation to the distribution of a large smooth area; it is also computationally very slow. It is hoped that these problems can be overcome in the future.

Chapter 6

Automatic analysis and selection of contrast enhancement techniques

6.1 Introduction

In this chapter the work done in the earlier parts of the thesis is used to design and build a system to analyse and enhance images with poor contrast. The system concentrates almost entirely on global grey scale transformation techniques, because these methods are computationally quick and to a certain extent can be adapted to the image (by a judicious choice of the parameter values associated with the algorithm).

As chapter 2 illustrates, there are a number of global contrast techniques available. Each technique gives different results depending on the underlying principle with which it was designed. For example, histogram hyperbolisation (Frei 1977), produces an image with a hyperbolic distribution, as, according to the originator of the method this is the distribution most suitable to the visual perception of the eye.

However the effect an algorithm has on any given image will vary greatly. Whether the algorithm enhances the image to our satisfaction depends on at least the following three points:

1. Whether the algorithm improves the contrast in areas of the image (i.e. either the whole image, or some part of it) where we would like to have contrast enhancement.

2. Conversely, the algorithm must not cause too much loss of contrast in any area of the image.

3. Whether the algorithm improves the contrasts i.e. whether the contrasts in the area are clearly visible, but still have contextual meaning.

It is usually possible to have a rough idea of the effect an algorithm will have on an image. However, it is very difficult to predict with any degree of precision whether a particular algorithm will satisfy the three conditions stated above. And, since there are a number of contrast enhancement algorithms available, it is even more difficult to know which of them will be the most suitable algorithm. Consequently the usual method of choosing a contrast enhancement technique is to apply the algorithm, look at the result, discard it if the result is not suitable, or if the algorithm requires a parameter value to be set, modify the parameter value, and try the algorithm again. Here a more systematic approach is tried using ideas from "Knowledge Based Systems" (KBS) or "Expert System" (see for example Forsyth 1986). Such systems use stored "knowledge" of a subject to solve typical problems in an area. In contrast enhancement the problem is comparatively simple to formulate (1-3). The most important information needed is to know which areas of the image the user is interested in enhancing; then rules are required to judge if a given algorithm enhances these areas satisfactorily. A model of the way contrast enhancement techniques are selected is encoded into the system and this knowledge is used to choose the algorithms available.

The advantages of applying this methodology to contrast enhancement are as follows:

(i) An environment where images can be displayed and analysed simultaneously.

(ii) The system can be partly automatic. It can therefore calculate data, for example from the histogram, that the user would not know directly.

(iii) It is not possible for a computer to decide automatically what information in an image is interesting and what information is not, because this depends entirely on what the user is looking for. The advantage of a KBS is that it acts as an interface between the user and the image processing programs that it contains. Visual information which a computer has no knowledge of can be given to the system by asking the user the appropriate questions. This information is then used in the selection of the results together with automatic data (which in turn it would be very difficult for the user to know) collected by the computer.

The advantage of the user's visual input can be seen in the following example. It is possible to do automatic analysis of a histogram to determine whether it is heavily skewed (Schowengerdt and Wang 1989). The fact that it is heavily skewed does not mean that the information in the skew can be compressed, as this area may or may not contain information that the user is interested in enhancing. In this case a KBS can provide information that cannot normally be supplied to automatic algorithms.

(iv) It can include all the possible algorithms to give as wide a choice of algorithm as possible.

(v) It is possible for inexperienced users to be guided through the system.

(vi) The parameters associated with the algorithm can be modified interactively.

Research on KBSs for image processing has been going on for sometime, particularly in Japan (Matsuyama 1989). Some systems are general purpose image processing systems (Taniguchi *et al.* 1990, Toriu *et al.* 1987), which calculate image processing processes e.g. how to binarise an image. However, more detailed work (despite its title) has also been published (Schowengerdt and Wang 1989). This paper describes a method of contrast enhancement using an analysis of the histogram. A number of parameters are calculated and then a stretch is applied. The system does not use any other algorithms (excepting histogram equalisation, as a default algorithm) stating that they are all combinations of linear stretches, although this is not in fact the case. The system proposed here is different in the following points:

(i) It has a user oriented approach. The user is first consulted about what part of the image needs enhancing, and then uses this information to select the most appropriate algorithm.

(ii) The system contains contrast enhancement techniques previously published, and some that have been specifically written where there was no known algorithm that could produce effective results.

(iii) Three criteria for choosing grey scale transformation techniques.

In the remaining part of the chapter the design and implementation of the system are described. The main points discussed are, the requirements for the system, the way the knowledge is stored in it, when and why a particular algorithm is selected, and the implementation of the system using object oriented methods. Finally, to conclude the chapter, two examples of the system analysing and choosing algorithms to process images are given.

6.2 Requirements for the system

In section 6.1 the three basic requirements for the system were stated (1-3). In order to fulfill (1), the user needs to be able to tell the system which areas of the image are of interest; this is achieved by using a *dialogue* where the user is asked questions and replies directly or by using some associated tools.

In order to fulfill points (2) and (3) every algorithm is tested to make sure that it stretches 50 percent of each *area of interest* selected by the user; that the average stretch per bin of each area is at least greater than the required *average lut stretch*, and finally that no area of the grey scale is *excessively stretched.* These three criteria are discussed in detail in the following section.

6.2.1 Dialogue

A dialogue is used to gather information about the image which cannot be obtained directly (i.e. using automatic functions). The image is displayed while the dialogue proceeds. Most of the questions may be answered directly. However, in order to determine information about certain regions of the image, it is necessary to select a typical part of that region by drawing a square on the image using the "mouse".

The dialogue consists of a fixed number of questions, although the exact series of questions may vary as some questions are prompted by the answers to others. The more difficult questions are accompanied by explanations. If there is some uncertainty about the answer to a question then a response of "dont know" will cause a further question to be asked. This question should make clear what is expected.

The questions asked are as follows:

1. What type of monitor is being used?

   This sets the offset value, and the appropriate look up table of values. The monitors and their offsets are given in Table 3.3.

2. How many quantisation levels are used?

   This question is not currently necessary because the available monitors can only display 256 grey levels. However, this may change in future.

3. Do you require a particular shape of histogram?

   This question is asked in case the user has a particular shape of histogram in mind. As mentioned in section 2.2.2, certain shapes will enhance the contrasts in various regions.

4. Are there any uniform areas in the image?

   By "uniform" is meant an area of the image, typically the background, that should occupy only a single grey level, but due to noise in the digitisation process occupies a much larger range than it should. There are, of course, some areas that may appear to be uniform, but in fact contain information that is simply not visible.

5. How many uniform areas are there?

   For each area the user is asked to select a typical part using the mouse. If it is not known whether the area contains any information, the area is enhanced (i.e. it is stretched) to discover this fact. The area is compressed into a single grey level (the central grey level in the area) if it contains no information; otherwise it becomes an *area of interest* (see below).

6. Are there any areas of the image that are of interest?

7. How many *areas of interest* are there?

   The user is asked to select a typical part of each *area of interest* using the mouse. Information about the area is stored, and used later in the selection of algorithms.

The information obtained from these questions is stored in the system. It is used later on to determine which of the available algorithms are best suited to enhance the image. In the process of checking whether each algorithm enhances the *areas of interest*, the system also checks which areas of the grey scale are compressed by each algorithm. A further question is asked - whether the user minds if these areas are compressed. If the user answers "dont-know", the compressed areas are displayed. The system then asks the question again. If the user does not want to destroy the information shown, the algorithm is discarded or (in some cases) substituted with a smoothed version of the algorithm.

A schematic representation of the dialogue between the user and the system is shown in Figure 6.1.

6.2.2 Criteria for selecting algorithms

There must be some criteria to judge whether an algorithm has successfully enhanced the image. Three criteria are used:

1. The percentage stretch. At least 50 percent of each region of the image selected by the user as an *area of interest* must be stretched by an algorithm.

2. The average stretch per bin given to an *area of interest* must be at least greater than the average *lut* value of the area. These terms are defined below in the section entitled *Average lut stretch.*

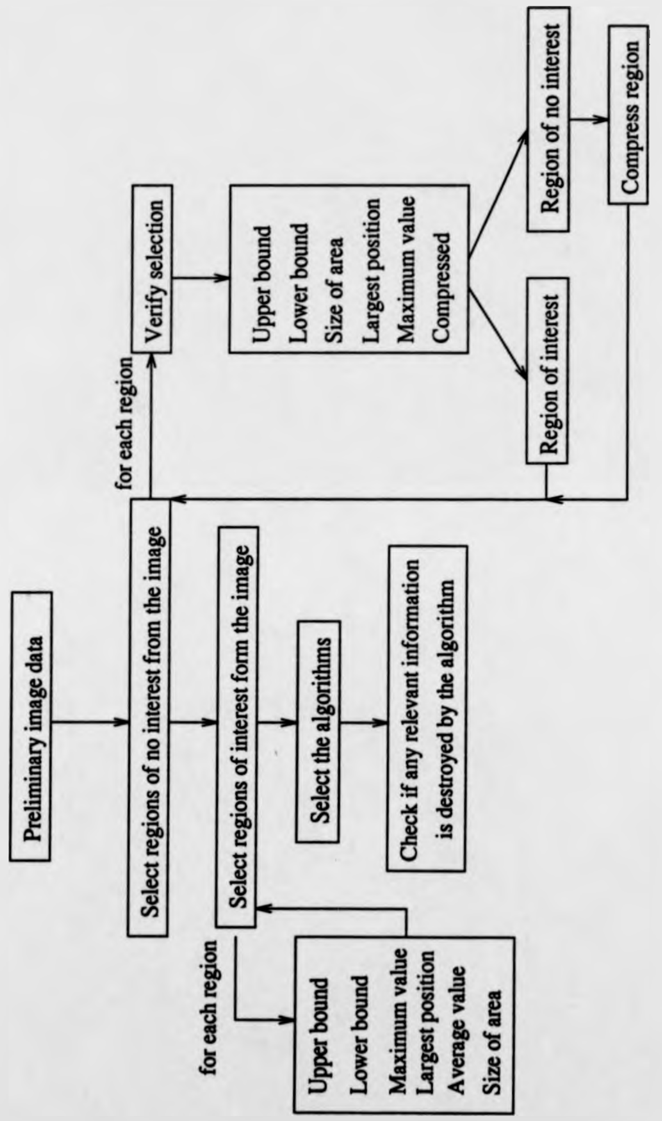


Figure 6.1: The dialogue between the user and the system.

3. No region of the image (including any *area of interest*) must be stretched too much by an algorithm. This is referred to as "excessive stretching".

Percentage stretch

Each algorithm in the system enhances certain intervals of the grey scale. Let there be k of these for each algorithm, and let the range of each interval be given by $[l_i, \ldots, u_i]$, for $i = 0, \ldots, k-1$, where l_i and u_i are the upper and lower bound of the interval respectively.

The overlap between each *area of interest* and the intervals stretched by the algorithm are now found. If there are r *areas of interest* selected by the user, with ranges given by $[L_j, \ldots, U_j]$, $j = 0, \ldots, r-1$, where L_i and U_i are the upper and lower bounds of the intervals respectively, then for each j the number of grey levels in the intersection of the interval $[L_j, \ldots, U_j]$ and $[l_i, \ldots, u_i]$ for $i = 0, \ldots k-1$ is calculated. Let the j^{th} *area of interest* have Z_j grey levels in its intersections. If the algorithm is to fulfill this criterion $\frac{Z_j}{U_j - L_j}$ must be greater than 0.5 for each j, $j = 0, \ldots, r-1$.

As the value of 0.5 is arbitrary (i.e. it is not an absolute test) the algorithms that do not pass this test but do stretch each region by some amount are stored separately. If there are no algorithms passing this test then these algorithms are selected.

Average lut stretch

For each interval of the grey scale stretched by an algorithm the *stretch ratio* and the *lut ratio* are calculated. The *stretch ratio* is the average stretch per bin. Using the above notation the *stretch ratio*, s_i, given to the i^{th} interval $[l_i, \ldots, u_i]$ is:

$$s_i = \frac{\tau[u_i] - \tau[l_i]}{b_i} \tag{6.1}$$

where τ is the grey scale transformation, and b_i is the number of occupied bins in the i^{th} interval. The *lut ratio* is a measure of the separation required between the bins in the i^{th} interval $[l_i, \ldots, u_i]$ when they are mapped by the algorithm to a new place in the grey level scale $[\tau[l_i], \ldots, \tau[u_i]]$. The *lut ratio* of the i^{th} is defined as:

$$l_i = \frac{1}{\tau[u_i] - \tau[l_i]} \sum_{i=\tau[l_i]}^{i=\tau[u_i]} lut[i] \qquad (6.2)$$

For each *area of interest* selected by the user the *average stretch ratio* and the *average lut ratio* are calculated and compared. The *average stretch ratio* is the average of the *stretch ratio*s of all the intervals with which an *area of interest* has some grey levels in common. Similarly the *average lut ratio* is the average of the *lut ratio*s of all the intervals with which an *area of interest* has some grey levels in common.

The *average stretch ratio* of each *area of interest* is compared to its *average lut ratio*. If the *average stretch ratio* is larger than the *average lut ratio* for each interval then the algorithm satisfies this criterion.

Excessive stretching

An algorithm causes excessive stretching if a stretched interval $[\tau[l_i], \ldots, \tau[u_i]]$ occupies so much space that the remaining bins in the histogram have to be "squeezed" into a very small region of the grey level range. An example of this is shown in Figure 6.2 (a). This is the result of applying the histogram hyperbolisation algorithm to Figure 4.15 with no offset value. The white region of Figure 4.15 has been stretched over such a large area that the rest of the image loses its meaning.

Each interval stretched by an algorithm is checked for excessive stretching. The following five steps are involved (illustrated in Figure 6.3).

1. Let n be the number of occupied bins in the interval $[l_i, \ldots, u_i]$.
2. Let *stretch* be the grey level range the interval is stretched over i.e. $stretch = \tau[u_i] - [\tau[l_i]]$.
3. Let m be the number of occupied bins in the histogram minus those in the

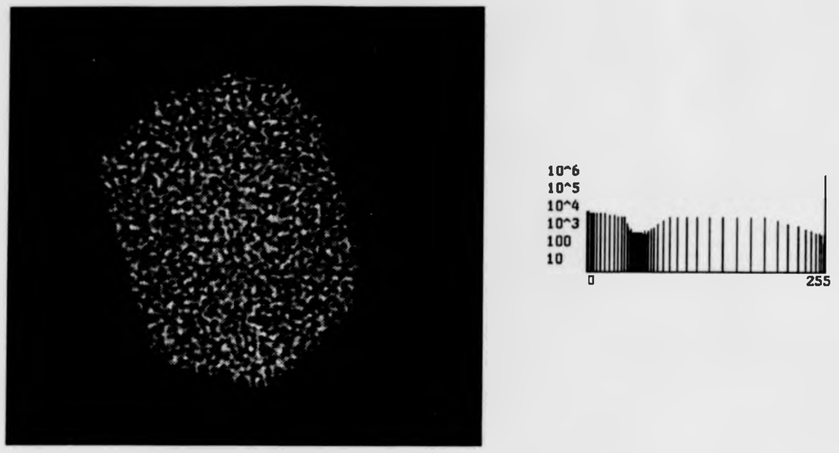


Figure 6.2: Excessive stretching of the semi-conductor image: (a) image, (b) histogram.

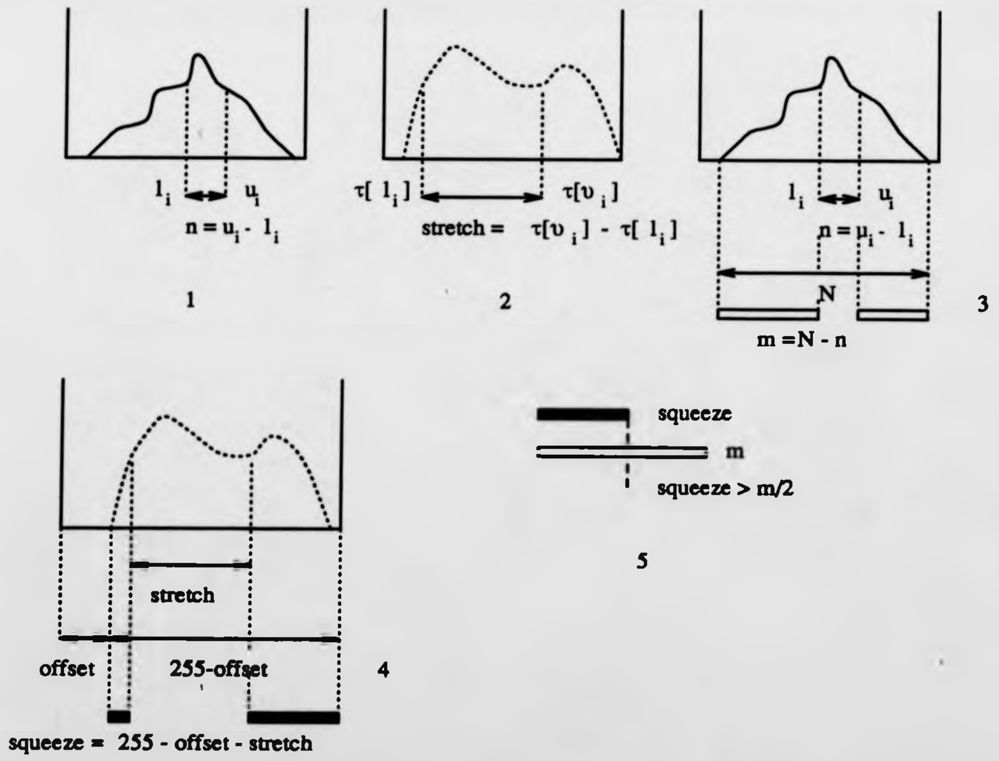


Figure 6.3: Stages (1-5) to determine if there is an excessive stretch.

interval, i.e. $m = N - n$ where N is the total number of occupied bins in the histogram.

4. Let *squeeze* be the grey level range available for stretching, $squeeze = 255 - offset - stretch$.

5. The remaining m bins have to be squeezed into *squeeze* grey levels. The criterion is that $squeeze > \frac{m}{2}$. That is, there must be enough space for at least half of the bins not in the interval.

If the algorithm passes these three criteria it can then be confidently expected to enhance the image well.

6.3 Description of the system

6.3.1 Overview

The system consists of two main parts. These are called the image processing module and the advisory system respectively. The image processing module is a graphics program concerned with processing, analysing, and displaying images; the advisory system carries out a dialogue with the user, asking questions, receiving answers, calling the image processing module when appropriate to display and analyse the image, and reasoning with data obtained from the user and from data it already contains about each algorithm, to select algorithms to enhance the image. The two parts operate simultaneously on a SUN workstation, which has a graphics package (Sunwindows or Sunview) (see the Sunview Reference Manual 1986) to display images.

The system is divided into two parts not from choice, but because it is difficult to implement in one computer language. The image processing module is written in "C", an excellent language for writing algorithms and using graphics. Unfortunately "C" does not support operations on "words" required for the advisory program. The advisory program is written in the object oriented language Flavors from Franz Common LISP (see Franz Lisp Reference Manual 1987). LISP supports operations on "words", but its operations on numbers are extremely cumbersome, and it does not support a graphics package.

6.3.2 The image processing module

The purpose of the image processing module is to support (i.e. obtain data from the image) the dialogue between the user and the advisory system. It does this in the following ways:

1. It displays the image and the histogram of the image.
2. It allows the user to select regions of the image and to determine if the information they contain is relevant. On the other hand, if the region is of no interest it can be compressed into a single grey level.
3. The areas of the grey scale compressed by each algorithm can be displayed.
4. The user can compare the result of using the algorithm with different parameter values.
5. The results of different algorithms can be displayed and compared.

6.3.3 The Advisory system

The function of the advisory system is to carry out an interactive dialogue with the user, asking questions about the image, doing automatic processing, and gathering information until it selects the most appropriate algorithm(s). As with all such systems, the main issues are, how the knowledge is structured or represented (in this case the knowledge is in the form of the questions asked) and how the system inferences or reaches its conclusions.

Knowledge representation

The knowledge contained in this system is in the form of the questions asked. The questions are attached to slots that contain the answers to the questions, and the slots are arranged in a frame structure. When the questions are answered the resulting facts are stored in a frame structure. Rules are then applied to these facts to choose the algorithms best suited to enhancing the image.

The idea of a frame was first introduced in an influential article by Minsky (Minsky 1975). Basically a frame is a data structure representing a stereotyped situation. The frame is made up of slots or terminals which store information about the situation. Each slot is filled if the conditions attached to it are fulfilled. The frames can also be arranged in a hierarchical structure, with some frames sharing terminals (see for example Waterman 1986). In this application the frames model the way grey scale transformation techniques are chosen. This has already been described in Figure 6.1. The frames are used to build up a picture of the regions the user wants to enhance; this requires a frame to store the relevant information about the region; later on this information is used to select the relevant algorithms.

The greatest advantage of a frame structure is that it is a very neat and structured way of representing information, particularly when (as in this case) the form of the problem fits the representation[1] (see below *Function of the frames*). There are three levels of frames (see Figure 6.4) in the system: at the top is the *top frame*, beneath it is a *contrast frame*, and below this are a number of *contrast algorithm frames*. Each level inherits all the slots from the level(s) above. Frames

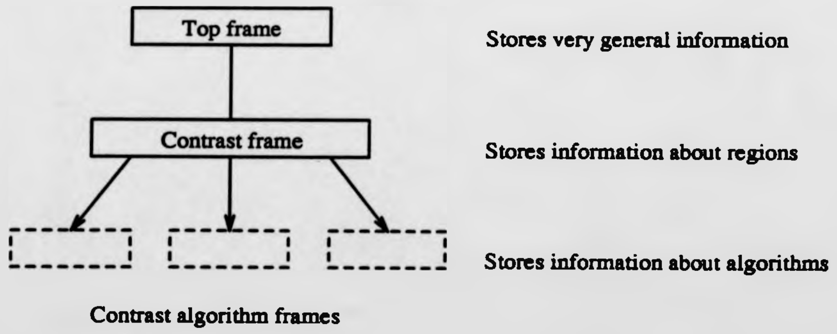

Top frame
Stores very general information
Contrast frame
Stores information about regions
Stores information about algorithms
Contrast algorithm frames

Figure 6.4: The frame structure.

[1]A previous attempt to construct a knowledge based system for image enhancement using rules (see Dale-Jones 1989) proved to be extremely difficult.

may contain a variety of different slots. The most common type of slots is an empty slot. This is simply a slot with a question attached to it; the slot will be filled by the answer to the question. The main types of slot used in system are shown in Table 6.1 (other types of slots are mixtures of the ones described there).

Type of slot	Function
Slot	Slot with question attached. Filled when the question is answered
Dependent slot	Slot that is only relevant when the values in other slots are equal to some desired value
Slot with procedure	Slot with an attached procedure. Often used to do some automatic calculations (e.g. find the number of occupied grey levels in the histogram)
Frame slot	A slot with a sub-frame attached to it. This is used most often when there are several sub-frames attached. Typically stores information about regions

Table 6.1: The different types of slots in the system.

The function of the frames

As mentioned earlier there are three levels of frames in the system. The top level frame is called the *top frame*, beneath is the *contrast frame*, and at the bottom are the *contrast algorithm frames.*

The *top frame* is used to store very general knowledge about the image. The information in the top frame may be used by slots lower down in the system, to determine what value they should have. The slots in the *top frame* are shown in Table 6.2.

Name of slot	Type of slot	Function
Number of pixels	Slot with procedure	To determine the number of pixels in the image
Monitor offset	Slot	To determine the offset required for the monitor

Table 6.2: Slots in the *top frame.*

The *contrast frame* is used to obtain and store information about areas of the image the user is or is not interested in. First of all, as much space as possible is created in the histogram by compressing areas of no interest to the user (e.g. background areas). This is achieved by asking the user to select such regions, and then verifying whether or not the region is of interest. If the region is of no interest then this area of the histogram is compressed. The user is then asked to select regions of the image that are of interest. Information about all regions selected are stored in attached frames. The slots in the *contrast frame* are shown in Table 6.3. The slots in the frames which are attached to slots other (frame

Name of slot	Type of slot	Function
Quantisation levels	Slot	To determine the number of quantisation levels used.
Largest bin	Slot with procedure	Determine the largest bin in the histogram
Occupied bins	Slot with procedure	Determine the number of occupied grey level bins
Uniform areas	Frame slot	Determine the number of uniform areas
Size of area	Slot with procedure	Determine the number of occupied bins in this area
Lower bound	Slot with procedure	Determine the lower bound of the area
Upper bound	Slot with procedure	Determine the upper bound of the area
Largest position	Slot with procedure	Determine the position of the largest bin
Maximum value	Slot with procedure	Determine the largest bin in the area
Compressed	Slot with procedure	Whether the area is compressed
Areas of interest	Frame slot	Determine the number of areas the user is interested in enhancing
Upper bound	Slot with procedure	Determine the upper bound of the area
Lower bound	Slot with procedure	Determine the lower bound of the area
Maximum value	Slot with procedure	Determine the largest bin in the area
Largest position	Slot with procedure	Determine the position of the largest bin
Average value	Slot with procedure	Determine the average bin value in area
Number occupied	Slot with procedure	Determine the number of occupied bins in the area
Background peak	Slot with procedure	Whether or not there is a background peak
Total areas	Slot with procedure	The number of areas the user is interested in enhancing (these may be from the *uniform areas* slot or from the *areas of interest* slot

Table 6.3: Slots in the *contrast frame.*

slots) are distinguished from other slots by indenting them.

The *contrast algorithm frames* are the lowest level frames. They are used to store information about each contrast algorithm. The algorithms are: *histogram equalisation* and its variant *smoothed histogram equalisation* (Kautsky *et al.* 1984), *histogram hyperbolisation* (Frei 1977), *inverse histogram equalisation* and its two variants which are called *smoothed inverse histogram equalisation* variants 1 and 2 respectively (see chapter 4), *histogram specification* (Hummel 1975 and Gonzalez and Fittes 1977), *global contrast stretching* (see chapter 2), *multiple contrast stretching* (contrast stretching of more than one area of the grey scale), and *simple lut stretch* (similar to the algorithm in chapter 3), *simple lut stretch* *with parameter* (the first algorithm in chapter 3), *multiple lut stretch* (similar to the second algorithm in chapter 3), and *multiple lut stretch with parameter* (the second algorithm in chapter 3).

The three algorithms *histogram equalisation*, *histogram hyperbolisation* and *inverse histogram equalisation* have many slots in common: these are shown in Table 6.4. These algorithms are different from the others (not counting their smoothed variants) because it is difficult to tell which areas of the grey scale they

Name of slot	Type of slot	Function
Distribution	Frame slot	Determine the parts of the grey scale which are stretched by the algorithm
Lower bound	Slot with procedure	The lower bound of the stretched area
Upper bound	Slot with procedure	The upper bound of the stretched area
Mapped lower bound	Slot with procedure	The position the lower bound is mapped to by the algorithm
Mapped upper bound	Slot with procedure	The position the upper bound is mapped to by the algorithm
Stretch ratio	Slot with procedure	How much the region is stretched per bin
Lut ratio	Slot with procedure	The required *lut* stretch per bin in the mapped interval
Uniform area stretch	Frame slot	Determine the stretch given to uniform areas the user is interested in
Percentage stretch	Slot with procedure	The percentage of the area stretched
Average *lut* stretch	Slot with procedure	The average Lut ratio for the area
Average stretch ratio	Slot with procedure	The average Stretch ratio for the area
Areas of interest stretch	Frame slot	Determine the stretch given to the areas that the user selected
Percentage stretch	Slot with procedure	The percentage of the area stretched
Average *lut* stretch	Slot with procedure	The average Lut ratio for the area
Average stretch ratio	Slot with procedure	The average Stretch ratio for the area
Excessive stretch	Slot with procedure	Whether the region is stretched excessively
Enough stretch	Slot with procedure	Whether enough (> 50 percent) of the region is stretched

Table 6.4: Slots shared by algorithms.

will stretch. This information is determined and stored in the *distribution* slot (this is a frame slot, with a sub-frame for each area stretched by the algorithm). The *distribution* slot also stores the *stretch ratio* and *lut ratio* respectively. Altogether, these slots contain all the information needed to decide if the algorithm is suitable to the user's requirements. The *frame slots* below, *uniform area stretch* and *area of interest stretch* calculate the *average stretch ratio*, *average lut ratio*, and the *percentage stretch* given to the areas the user is interested in stretching.

6.3.4 The selection process

The selection process decides which algorithms will be chosen to process the image. The algorithms fall into three groups according to the selection process.

First group

In the first group are the following: *histogram equalisation*, *histogram hyperbolisation*, and *inverse histogram equalisation*. The selection process for these three algorithms is illustrated in Figure 6.5. As was mentioned in the previous section

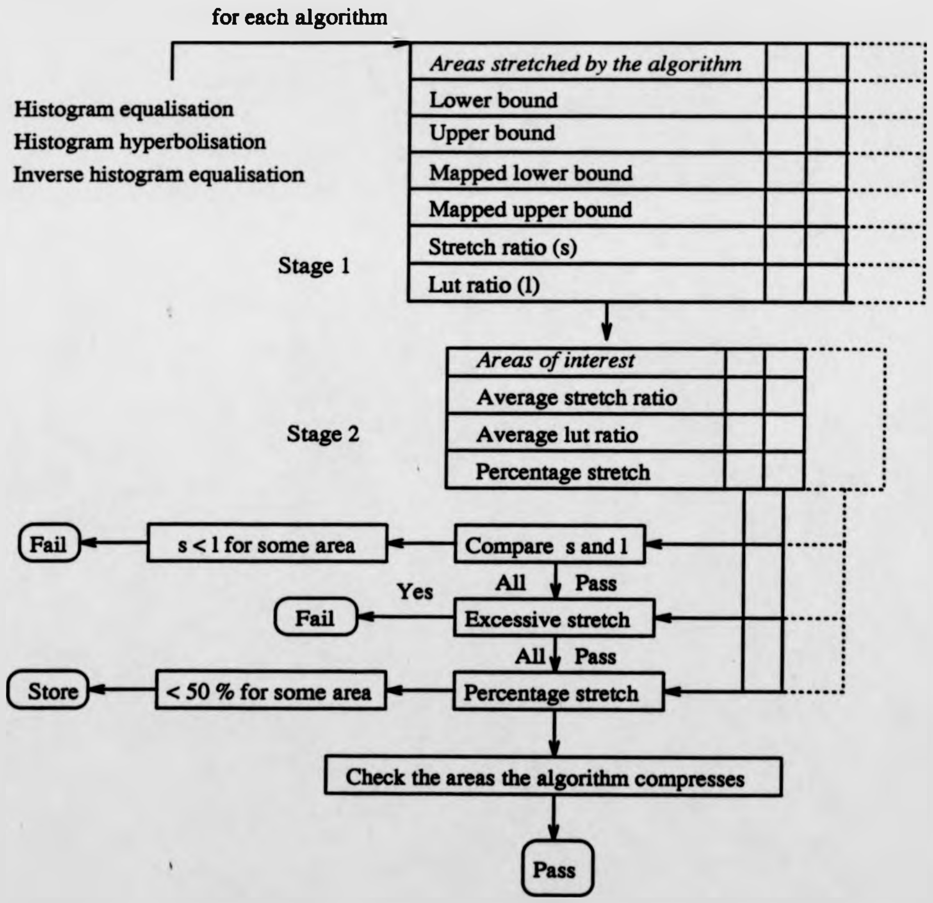


Figure 6.5: The selection process.

it is difficult to judge which areas of the grey scale will be stretched by these algorithms. Thus, this information is first determined and stored in the *distribution slot* in the algorithm's frame (stage 1 in Figure 6.5). Then each *area of interest* is examined in turn to see whether it is enhanced by the algorithm: the *average lut stretch*, *average stretch ratio* and *percentage stretch* of each area is determined. Each area must fulfill the *average lut stretch* and *excessive stretch* criteria of section 6.2.2: if it does not it is discarded. If an algorithm fails the third criterion of section 6.2.2 (i.e. *percentage stretch*) it is stored separately, and may be suggested later in the selection process if no algorithm is selected using the usual procedure.

Once one of the above algorithms has been selected there is one more stage to pass through before the algorithm is used to process the image. Each algorithm compresses some bins in the histogram in order to create room to stretch the other areas. The area the algorithm destroys is displayed to the user, and the user is asked whether the information destroyed is important. If the information is not important the algorithm is used to process the image. If the information is important then in the case of the *histogram equalisation algorithm* and the *inverse equalisation algorithm* the smoothing algorithms associated with them are suggested. In the case of the *histogram hyperbolisation algorithm* there is no smoothed algorithm to suggest, so the algorithm fails.

For the *histogram equalisation algorithm* it is easy to find which bins are destroyed - it is simply those above the average bin value. For *inverse histogram equalisation* algorithm, recall from chapter 4 that a parameter (a bin size) is required before the algorithm is used; all bins larger than this size are destroyed. The parameter chosen initially for the algorithm is the average value of all the bins in the *areas of interest*; in this way most information is enhanced, although some of course will be destroyed. In fact the bins destroyed by the algorithm are of two types. The first type is made up of bins larger than the parameter; the second type consists of bins that are too small. The user is asked separately about the two different types of bins that are destroyed, because there are two smoothed variants of the algorithm, one used to preserve the larger bins, and one to preserve the smaller bins.

Second group

In the second group of algorithms are *smoothed histogram equalisation*, and *smoothed inverse histogram equalisation* version 1 and 2 respectively. These algorithms are smoothed variants of the algorithms in the first group. They are chosen in one of the following two cases:

1. When the corresponding algorithm in the first group passes the *percentage stretch* and *average lut ratio* criteria but fails the *excessive stretch* criteria. This happens when one or more *areas of interest* is stretched too much by the algorithm in the first group. The smoothed variants of the algorithm will still stretch the areas but can, by a judicious choice of parameter, usually avoid excessive stretching.

2. When the user finds that the corresponding algorithm in the first group destroys important information. Again using smoothing, some of this information can be preserved.

All three algorithms require a choice of parameter before processing. The initial value suggested for *smoothed histogram equalisation* is $\lambda = 0.3$. This value gives a fairly smooth result. Version one of the *inverse histogram equalisation* algorithm has two parameters. The first is the value above which all bins are destroyed. This parameter is set to the same value as in the original algorithm (i.e. the average of all bins in *areas of interest*). The second parameter is used to balance the compression of smaller bins in the histogram against the amount of stretching. The initial value of this parameter is chosen so that all bins compressed in the *inverse histogram equalisation* algorithm are retained. Version two of the *inverse histogram equalisation* also has two parameters. The first parameter determines the bin size above which compression occurs. This is initially set to the average of all bins in *areas of interest*. The second parameter is the bin size below which bins are retained. As in the first variant this value is chosen so that all bins compressed in the *inverse histogram equalisation* algorithm are retained.

After the initial processing the user is advised on how the parameter values can be modified in order to obtain more or less stretching as required.

Third or miscellaneous group

In the third group of algorithms are *global contrast stretching*, *multiple contrast stretching*, *simple lut stretch*, *simple lut stretch with parameter*, and *multiple lut stretch with parameter*. These algorithms each require different calculations to determine if they can be used to enhance the image.

Global contrast stretching

This algorithm gives a linear stretch of the bins in the histogram (see section 2.2.1). The algorithm is chosen if the linear stretch is large enough to fulfill the *average lut stretch* criterion. The other two criteria are automatically fulfilled since the algorithm stretches the entire region, and does not compress any bins in the histogram. The linear stretch given to the bins in the histogram is:

$$\frac{r - offset}{n} \tag{6.3}$$

where r is the number of quantisation levels used, $offset$ is the monitor offset, and n is the number of occupied grey levels in the image. If this value stretches greater or equal to the *lut ratio* of each *area of interest* this algorithm is selected.

Multiple contrast stretching

This algorithm attempts to stretch the regions selected by the user while leaving all the remaining bins intact. It is similar to the interactive method described in section 2.2.1. Before the algorithm is used all *areas of interest* are reduced to non-overlapping intervals of the grey scale[2]. The algorithm is selected if the stretch given to the bins in each non-overlapping interval is as large as the *lut ratio* in equation 6.3. The stretch given to the non-overlapping intervals is:

$$\frac{r - offset - n + t}{n}$$

where r, $offset$, and n are the same as in equation 6.3, and t is the number of occupied grey levels in all non-overlapping *areas of interest.* If all the *areas of interest* are stretched by the required *lut ratio*, this algorithm is selected.

Simple lut stretch

This algorithm is similar to the first algorithm described in chapter 3. It can only be used when there is one *area of interest.* Therefore all overlapping *areas of interest* are eliminated before this test is applied. The algorithm stretches the bins on either side of the largest bin (assumed to be the background grey level of the region) in the single *area of interest* according to the *lut* values for the monitor. All other bins in the histogram are retained. Clearly there has to be sufficient space in the histogram before this algorithm can be chosen. It is only chosen if there is sufficient space for each bin in the *area of interest* to be stretched by one bin on either side. If it fails this test the algorithm is not selected. If it passes this test a note is made of the number of bins that are stretched by the algorithm (i.e. the *lut* values vary and so the initial test is only a rough measure of the number of bins stretched by the algorithm). Thus the *percentage stretch* given by the algorithm is determined.

Simple lut stretch with parameter

This algorithm is the first algorithm given in Chapter 3. It is selected when there is one *area of interest*, but there is not sufficient space in the grey scale for a *simple lut stretch.* Instead a number of the smaller bins outside the interval of

[2]Note that it is not necessary to eliminate the overlapping intervals in the algorithms in the first and second group because their results are independent of this condition.

the *area of interest* are compressed to make room for the stretch. The parameter for the algorithm, the number of bins to compress, is determined by calculating the amount of space required to stretch all the bins in the region by giving them a *simple lut stretch* as described above.

Multiple lut stretch

This algorithm is similar to the second algorithm in Chapter 3. It can only be used when there is more than one *area of interest* selected (if only one area has been selected then one of the two preceding algorithms is selected instead). The algorithm uses non-overlapping intervals. It assumes that the largest bin in each non-overlapping *area of interest* is the background, and separates the adjacent bins according to the *lut* values. The algorithm is only selected if there is sufficient space to stretch the *areas of interest* without compressing any bins. This criteria is approximated by assuming that each bin in an *area of interest* receives a stretch of 2 bins. If the remaining space is not enough to accommodate the bins outside the *areas of interest* without compression, the algorithm is not selected. In this case the following algorithm *multiple lut stretch with parameter* is chosen instead.

Multiple lut stretch with parameter

This algorithm is the same as the second algorithm in Chapter 3. It is used when the *multiple lut stretch* algorithm fails because there is not enough space in the grey scale to stretch the *areas of interest* without causing some compression. The algorithm creates space for stretching by compressing a sufficient number of smaller bins not contained in any *area of interest.*

6.3.5 Ordering the algorithms

The selected algorithms fall into one of two groups. The first consists of those (e.g. *global contrast stretching*) algorithms that give a complete stretch of all *areas of interest.* The second group consists of algorithms that give only a percentage stretch of the areas of interest (e.g. *histogram equalisation*).

The order in which the algorithms are presented to the user is in the order of their percentage stretch. Thus the algorithms in the first groups are presented first, followed by the second group in the order of their (total) percentage stretch. Finally, the algorithms placed in a separate list because they cause excessive stretching are presented.

6.3.6 Modification of algorithms after selection

As mentioned earlier, some algorithms have associated parameters. When the results of such an algorithm are presented to the user, the user is asked whether it would be preferable if the parameter was changed in order to increase or decrease the stretching, or to balance the stretching against compression. There are three windows available for the user to compare the results. If more windows are needed before the final decision is made, then one or more of these windows must be deleted manually (i.e. using the mouse).

In addition some algorithms have another algorithm attached them. If the original algorithm causes too much stretching of the *areas of interest* or too much compression of other bins in the histogram then this algorithm is suggested. The algorithms and their parameter values and attached algorithms are shown in Table 6.5.

6.4 Implementation

6.4.1 The image processing module

The image processing module is written in "C", and incorporates the graphics available on the Sun Workstation (Sunview or Sunwindows). These work stations have colour monitors capable of displaying 256 grey levels. The new Sun 4 generation of computer are also very quick: unfortunately however, because LISP is only currently available on the SUN 3 computer, the system cannot make use of the extra speed.

Algorithm	Parameter(s)	Attached algorithm(s)
Histogram equalisation	None	Smoothed histogram equalisation
Histogram hyperbolisation	None	None
Inverse histogram equalisation	$A : 0 \leq A \leq 1$	smoothed inverse histogram equalisation, variants 1 and 2
Smoothed histogram equalisation	$\lambda : 0 < \lambda < 1$	itself
Smoothed inverse histogram equalisation variant 1	$A : 0 \leq A \leq 1$ $B : 0 < B \leq 1$	itself
Smoothed inverse histogram equalisation variant 2	$A : 0 \leq A \leq 1$ $B : 0 < B < 1$	itself
Global contrast stretching	None	None
Multiple contrast stretching	None	None
Simple lut stretch	None	Simple lut stretch with parameter
Simple lut stretch with parameter	$A : 0 \leq A \leq 256$	itself
Multiple lut stretch	None	None
Multiple lut stretch with parameter	$A : 0 \leq A \leq 256$	itself

Table 6.5: Algorithms with parameters and attached algorithms.

The Sunwindows or Sunview environment provides functions to display images. However, a great disadvantage of using this type of program is that it will only accept input from its own input/output devices (e.g. the mouse). Thus it is not possible to to get the image processing module to accept data directly from an external source (e.g. a LISP program). The way the image processing module and the advisory system communicate is explained in section 6.4.3.

6.4.2 The advisory system

The advisory system is written in a version of Object Oriented Lisp (Franz Common Lisp) called Flavors. The main reason for using an object oriented language is that it supports many of the requirements of a frame based system. The principal requirements are a method of storing data in self contained blocks; a structured system (i.e. some blocks of data are related to others), and a way of attaching procedures to the data. All object oriented languages (see for example Stroustrup 1988, Pinson and Lewis 1988, and Goldberg and Robson 1983), fulfill these requirements. Objects are used to store data. The objects can be structured: complex objects can be built from a mixture of basic objects, or on top of a simple object. Finally, objects can share procedures, known as "methods".

Basic objects

There are three basic types of object in the system: *frame*, *slot* and *question*. Frames are collections of slots: they have methods controlling the input of information into the system, and the selection of the algorithms to be used for processing. The relationship between the *frame* objects is shown in Figure 6.6. The most basic frame is shown at the top: the frames below become progressively more complex. The frames divide naturally into different levels. Frames lower down in the system have many things in common, but also some specific information not shared by other frames.

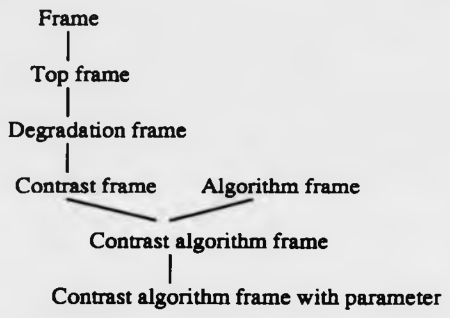


Figure 6.6: The relationship between the frame objects.

Slots are used to store data obtained by asking the user questions. The most basic type of slot object is called *default slot*, and all other slot objects are derived from it or its derivatives. The relationship between the different types of *slot* objects is shown in Figure 6.7. The main type of slot objects, *slot*, *slot with procedure*, *dependent slot* and *frame slot*, implement the behaviour of the slots described in Table 6.1. The function of the other slot objects is described in Table 6.6

Questions are used to obtain information from the user. The most basic question object is *question*, and all the others are either derived from it or from its derivatives. The relationship between the different question objects is shown in Figure 6.8. The function of the different types of question is shown in Table 6.7

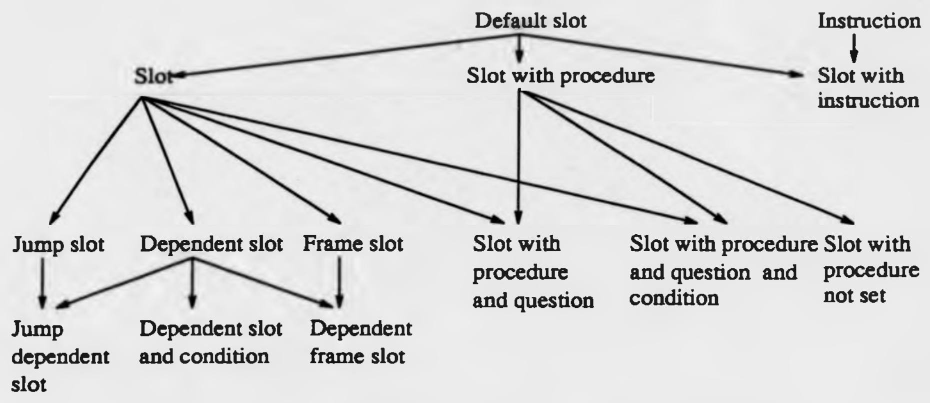


Figure 6.7: The relationship between the slot objects.

Type of slot	Function
Jump Slot	Temporarily jumps out of the system to do something and then returns
Jump dependent slot	Same as *jump slot* except that the jump is made only if certain conditions are fulfilled
Dependent frame slot	*A frame slot* created only under certain conditions
Slot with procedure and question	Asks a question and then executes a procedure
Slot with procedure and question and condition	Asks a question and executes a procedure only if the appropriate answer is given
Slot with procedure not set	Same as *slot with procedure* except that the value of the slot is not set to the result of calling the attached procedure
Dependent slot and condition	Same as *dependent slot* but returns a value indicating the slot is valid
Slot with instruction	Slot with instructions attached

Table 6.6: The other slot objects in the system.

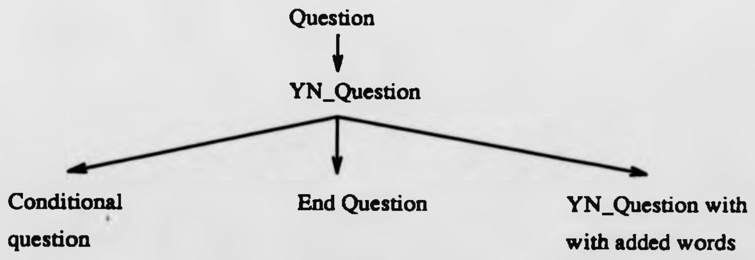


Figure 6.8: The relationship between the question objects.

Type of question	Function
Question	Stores the question to be asked
YN Question	Stores the question to be asked and the value to be put in the slot (to which it is attached) when the question is answered
YN Question with added words	Puts a special value into the text of the question e.g. from a previous slot
End question	Puts a default value into a slot if the question cannot be answered
Conditional question	Asks another question if the answer to the first question is a certain value

Table 6.7: The different types of *question* object in the system.

Automatic procedures

Although the advisory system is written in LISP, it also contains some procedures in "C". These procedures are compiled separately, and loaded into the LISP program when it is initialised. The "C" functions are usually used when a great deal of numerical calculation is required.

The image processing module could also be include in the LISP program in this way. However, it would not be easy to pass between the two parts without directly inputing data to the image processing module i.e. by using the mouse or pressing a button. The preferred method, here, is to automatically pass data to the image processing module using a file (see section 6.4.3), as then the user is free to pass between the two separate parts of the system.

6.4.3 Communication between the image processing module and the advisory system

The communication between the image processing module and the advisory system is effected by a series of files. The image processing module continually senses a file for data. When a value is put in the file by the advisory system it is read by the image processing module, causing it to call a function. For example, after the advisory system has selected the algorithms to be used to process the image, each one is called in turn, and displayed.

The image processing module communicates with the advisory system in the same way. Once it has finished its processing, it puts a value in a file that is continually being checked for new data by the advisory system.

6.5 Two examples

Two examples of how the system operates are now given. Both examples are conducted on a Hitachi LR47156 monitor. This monitor has an offset value of 10. The results of processing are produced from a laser printer and give only an approximate representation of the actual image, but are sufficiently good to illustrate the workings of the system.

6.5.1 First example

The first example is the image of trevor in Figure 4.6. For the purpose of illustration it is assumed that the user is interested in information in the tie area of the image. The system now analyses the image, obtaining data which it cannot generate itself by prompting the user with questions, and automatically calculating some data directly from the image.

First of all some general questions are asked about the image, and the system does some automatic analysis of the histogram. The information obtained by the system is shown in Table 6.8. The system uses this information in its later analysis, and in its selection of algorithms from the knowledge base. The system

no of pixels	65536
monitor offset	10 (Hitachi LR47156)
quantisation levels	256
largest bin	2870
occupied bins	189

Table 6.8: General image information from Figure 4.6.

now asks if there are any areas of the image that appear to contain no information (e.g. background levels). One area is selected using the mouse, the black background of the image, as shown in Figure 6.9. The system now asks if the area

Figure 6.9: The area of the background selected.

contains any relevant information. In this case the answer is "dont-know". The area selected is now given a linear stretch across the grey scale. The resulting image is displayed (see Figure 6.10), and the earlier question is repeated. This

Figure 6.10: The area selected in Figure 6.9 after being given a linear stretch to find if it contains any relevant information.

area does contain information (a curtain), so the answer to the question is yes, causing this area to become an *area of interest.* The system analyses the area, and obtains the data shown in Table 6.9.

The system now asks for *areas of interest* to be selected. The tie area in Figure 6.11 is selected. The system automatically analyses this region, and obtains

lower bound	37
upper bound	55
maximum value	2870
largest position	43
compressed	FALSE
occupied bins	19

Table 6.9: Data from the background region selected in Figure 6.9.

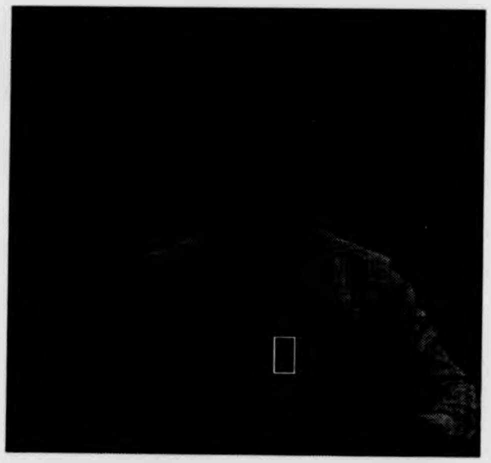


Figure 6.11: The area of interest selected.

lower bound	21
upper bound	62
maximum value	2870
largest position	43
average value	1192
bins occupied	36

Table 6.10: Data from the selected area of Figure 6.11.

Algorithm	Stretch ratio		Lut ratio		% stretch		Pass or Fail
	1	2	1	2	1	2	
Histogram equalisation	5	5	2	2	63	100	Fails : compresses relevant information
Histogram hyperbolisation	2	2	2	2	29	66	Fails : % stretch too small
Inverse histogram equalisation	3	3	2	2	39	50	Fails : % stretch too small
Smoothed histogram equalisation	-	-	-	-	-	-	Passes
Smoothed inverse histogram equalisation variant 1	-	-	-	-	-	-	Fails : because basic inverse equalisation fails
Smoothed inverse histogram equalisation variant 1	-	-	-	-	-	-	Fails : because basic inverse equalisation fails
Global contrast stretching	1.3	1.3	2	2	100	100	Fails : stretch ratio too small
Multiple contrast stretching	2.4	2.4	2	2	100	100	Passes
Simple lut stretch	-	-	-	-	-	-	Fails : not enough space available
Simple lut stretch with parameter	-	-	-	-	-	-	Passes
Multiple lut stretch	-	-	-	-	-	-	Fails : only one combined region
Multiple lut stretch with parameter	-	-	-	-	-	-	Fails : only one combined region

Table 6.11: Results for the algorithms in the knowledge base.

the data shown in Table 6.10.

The system now determines which algorithms to process the image with. It examines each algorithm in turn, and decides whether they pass the three criteria in section 6.2.2. During this process it asks two further questions about the information destroyed by the *histogram equalisation algorithm*, and the *simple lut stretch with parameter* algorithm. The first question asks whether it matters if the information in the bins whose size is less than 266 is destroyed. The area destroyed is shown - it is a large number of pixels in the shirt area - so the answer to this question is yes. The second question asks whether it matters if the information contained in 32 of the smallest bins is destroyed. The number of pixels in these bins is very small so the answer to this question is no. The answer of yes to the first question eliminates the *histogram equalisation algorithm* and selects its smoothed variant instead. The answer of no to the second question selects the *simple lut stretch with parameter* algorithm.

A summary of the results for the algorithms is shown in Table 6.11 with the *average stretch ratio*, *average lut ratio*, and *percentage stretch* (where relevant) produced by each algorithm. The table also explains the reason why some algorithms failed to be selected.

The three algorithms selected by the system are *smoothed histogram equalisa-*

tion (with a suggested parameter value of $\lambda = 0.3$), *multiple contrast stretching*, and *simple lut stretch with parameter* (with a parameter of $A = 32$). The results of applying the algorithms to Figure 4.6 are shown in Figure 6.12 (a), Figure 6.13

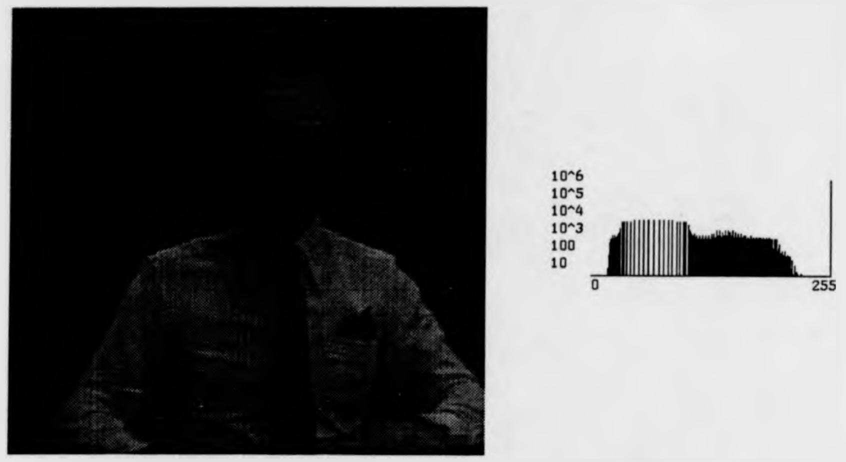


Figure 6.12: (a) Result of applying the *smoothed histogram equalisation algorithm* to Figure 4.6 with $\lambda = 0.3$ and, (b) its histogram.

(a) and Figure 6.14 (a) respectively. The histograms of the processed images are

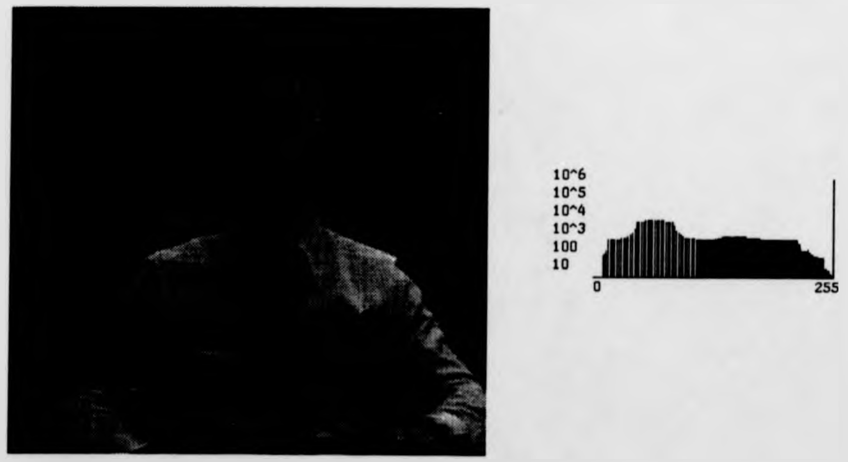


Figure 6.13: (a) Result of applying the *multiple contrast stretching* algorithm to Figure 4.6 (b) its histogram.

shown in Figure 6.12 (b), Figure 6.13 (b) and Figure 6.14 (b).

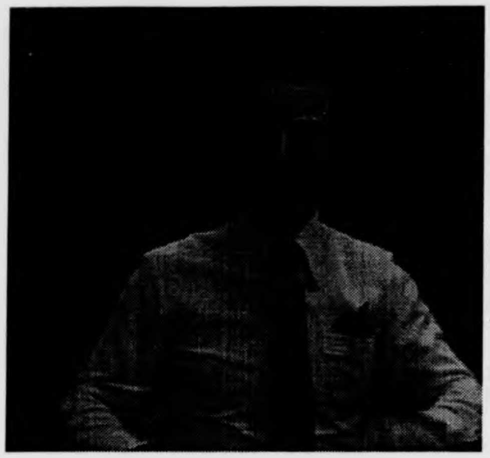
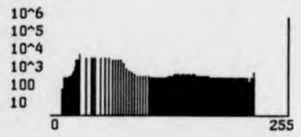



Figure 6.14: (a) Result of applying the *simple lut stretch with parameter* algorithm to Figure 4.6 with $A = 32$ and, (b) its histogram.

6.5.2 Second example

The second example is the photometric image in Figure 4.11. In this illustration the user is assumed to be interested in enhancing the predominantly grey background of the image. The initial data obtained by the system is shown in Table 6.12.

no of pixels	65536
monitor offset	10 (Hitachi LR47156)
quantisation levels	256
largest bin	1428
occupied bins	111

Table 6.12: General image information from Figure 4.11.

In response to the question about areas of the image that appear to contain no information the black object is selected, as shown in Figure 6.15. The system now asks if the area contains any relevant information. The answer, as in the first example is "dont-know". The area selected is given a linear stretch across the grey scale. The resultant image is displayed (Figure 6.16), and the earlier question is repeated. The area does contain some interesting information (i.e. a

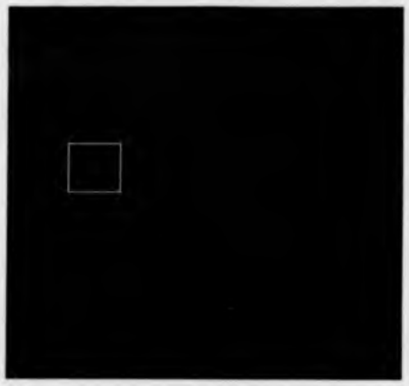


Figure 6.15: The area of the black object in Figure 4.11 selected.

Figure 6.16: The area selected in Figure 6.15 after being stretched linearly across the grey scale

hole) so the answer to the question is yes, causing the area to become an *area of interest*. The system analyses the area and obtains the data shown in Table 6.13.

lower bound	26
upper bound	51
maximum value	393
largest position	44
compressed	FALSE
occupied bins	26

Table 6.13: Data from the background region selected in Figure 6.15.

The system now asks for *areas of interest* to be selected. The area shown in Figure 6.17 is chosen. The system automatically analyses this region and obtains

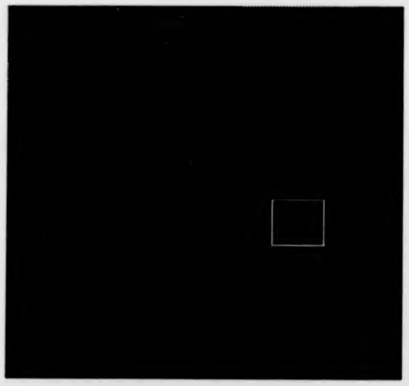


Figure 6.17: The area of interest selected in Figure 4.11.

the data shown in Table 6.14.

The system now determines which algorithms to process the image with. No further questions are asked. A summary of the results for the algorithms in the knowledge base is shown in Table 6.15.

The two algorithms selected by the system are *multiple contrast stretching* and *multiple lut stretch*. The results of the algorithms are shown in Figure 6.18 (a) and Figure 6.19 (a) respectively. The histograms of the images are shown in Figure 6.18 (b) and Figure 6.19 (b) respectively.

lower bound	101
upper bound	110
maximum value	1166
largest position	104
average value	992
bins occupied	10

Table 6.14: Data from the selected area of Figure 6.17.

Algorithm	Stretch ratio		Lut ratio		% stretch		Pass or Fail
	1	2	1	2	1	2	
Histogram equalisation	3	1	2	2	100	32	Fails : % stretch too small
Histogram hyperbolisation	5	-	2	-	100	-	Fails : % stretch too ama ll
Inverse histogram equalisation	-	2	-	2	-	96	Fails : % stretch too small
Smoothed histogram equalisation	-	-	-	-	-	-	Fails : because histogram equalisation fails
Smoothed inverse histogram equalisation variant 1	-	-	-	-	-	-	Fails : because basic inver se equalisation fails
Smoothed inverse histogram equalisation variant 1	-	-	-	-	-	-	Fails : because basic inverse equalisation fails
Global contrast stretching	2.2	2.2	2	3	100	100	Fails : stretch ratio too small
Multiple contrast stretching	4.8	4.8	2	3	100	100	Passes
Simple lut stretch	-	-	-	-	-	-	Fails : more than one area selected
Simple lut stretch with parameter	-	-	-	-	-	-	Fails : more than one area selected
Multiple lut stretch	-	-	-	-	-	-	Passes
Multiple lut stretch with parameter	-	-	-	-	-	-	Fails : because multiple lut stretch passes

Table 6.15: Results for the algorithms in the knowledge base.

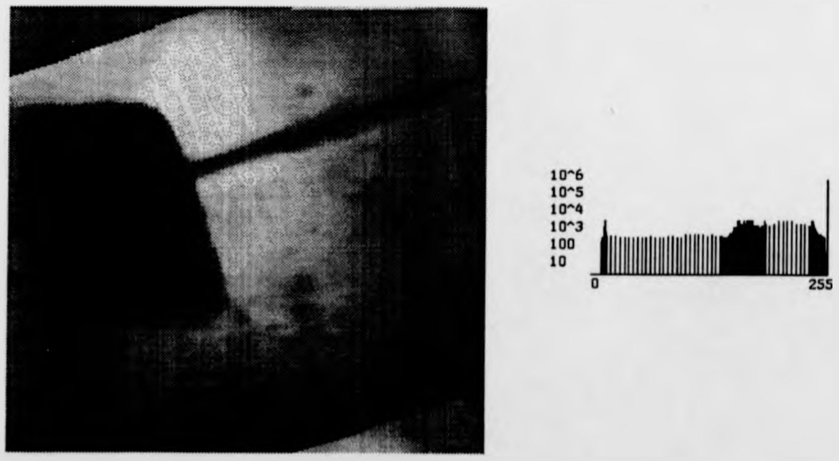
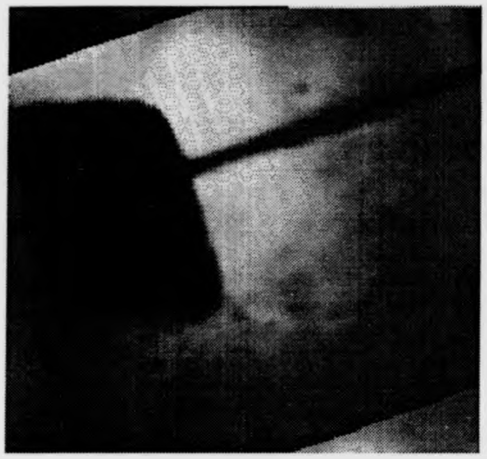


Figure 6.18: (a) The result of processing Figure 4.11 with the *multiple contrast stretching* algorithm, and (b) its histogram.



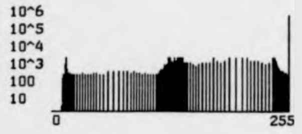


Figure 6.19: (a) The result of processing Figure 4.11 with the *multiple lut stretch* algorithm, and (b) its histogram.

6.6 Discussion

In this chapter a system for automatically selecting grey scale transformation techniques has been presented. The aim of the system is to provide a flexible environment for image analysis, and comparisons between processed images. The normal method of choosing a grey scale transformation technique is to apply it, look at the result, discard it if it is not good enough, or change the value of the parameter value (if there is one) and re-apply the technique. The system attempts to model this process, and to improve upon it.

First it gets the user to identify regions of the image that appear to contain no information. The selected regions are given a linear stretch to show whether or not they contain any relevant information. If the area does contain relevant information, then this region becomes an *area of interest*; if it does not contain any relevant information the part of the grey scale the region occupies is compressed. Compressing the region creates space in the grey scale for stretching. This pre-processing stage is a definite advantage over the normal method of choosing an algorithm.

Secondly, it asks the user to identify areas of interest. In some grey scale transformation techniques (e.g. *multiple contrast stretching*) identifying intervals of the grey scale is required before the technique, but in others (e.g. *histogram equalisation*) this is not the case. For the first group the process of identifying a region is at least easier - it is done by selecting a typical part of the area of interest using the mouse. For the second group it is now possible to determine if the technique does stretch the areas the user wants to enhance, and therefore whether the technique is useful or not.

Thirdly, most techniques compress some part of the grey scale. It is important for the user to know which areas are compressed, because these areas may contain some relevant information (which the user would like to retain, but has not selected as an *area of interest*). The areas the technique compresses are shown to the user; if relevant information is destroyed the technique is discarded or replaced with a smoothed variant (e.g. *histogram equalisation* is replaced by *smoothed histogram equalisation.*

Finally, the system is capable of displaying three processed images at the same time. If the techniques have parameters, then different value of the parameters can be applied and the results compared. Similarly, if more than one algorithm is selected by the system, each algorithm can be applied to the image and the results compared.

Chapter 7

Conclusion

7.1 Introduction

The thesis has described contrast enhancement using grey scale transformation techniques. Global and local contrast enhancement techniques were introduced in chapter 2, with particular emphasis on grey scale transformation techniques. Chapter 3 introduced a new grey scale transformation technique based on the display capabilities of a monitor. Chapters 4 and 5 examine the global and local histogram equalisation algorithms respectively, and suggest modifications to these algorithms. Chapter 6 brought together the work done in the previous chapters, and described the design and implementation of a system for automatically selecting grey scale transformation techniques to enhance the contrast in a given image.

7.2 Aim of the thesis

The aim of the thesis has been to examine grey scale transformation techniques in order to incorporate them into a system for automatically selecting a transformation to enhance the contrast in a given image. During the design of the system it was found that many of the algorithms were of only limited use. This led to a detailed examination of the problems involved and the development of some new algorithms (and modifications of existing algorithms).

7.2.1 The algorithms developed

The two algorithms in Chapter 3 (enhancement against a single background, and more than one background respectively) were developed partly as a result of frustration with the display capabilities of some monitors, and partly out of a desire to have algorithms that separate bins according to a straightforward and common sense criterion. The minimum separation required to distinguish adjacent grey levels against a given background was determined from two experiments. The results do not give an exact value for the required separation, but they give a good basis on which to judge the required separation. In the implementation of the algorithms the separation is increased near the background (because this is where most information is expected to be in a normal distribution), and a general increment can be added if required. The algorithms are also flexible. They have a parameter A which can be varied to increase the number of bins separated on either side of the background. In the author's opinion this makes the algorithm at least as useful as any currently available grey scale transformation technique. A possible area of future research could be determining the maximum allowable separation between adjacent grey levels.

The algorithms in Chapter 4 were developed from the global histogram equalisation algorithm. It is hoped that they complement the algorithms created by Kautsky (Kautsky *et al* 1984) for smoothing histogram equalisation (i.e. smoothed equidistributing regrading). The histogram equalisation transformation is one of the most commonly used grey scale transformation techniques, because it is said to maximise the information content. But, as it is shown in Chapter 4, the transformation is not always useful. The suggested modifications in Chapter 4 extend the usage of the transformation to a wider range of images (and distributions).

The local histogram equalisation transformation described in Chapter 5 is also a very popular algorithm. However, as is shown in this chapter, the algorithm has some unfortunate problems due to the fixed window size it uses. The reason for these problems is discussed in the chapter, and a modified algorithm is suggested. The modified algorithm tries to identify the region to which a pixel belongs, and then varies the window size according to the size of the region. In this way a smooth image is produced where each region is stretched by an equal amount.

However, the algorithm itself has a number of problems which should be the subject of further investigations. Firstly it is not always capable of identifying the region to which the pixel belongs; secondly it is computationally expensive.

7.2.2 System for selecting algorithms

The system in Chapter 6 provides an environment for selecting grey scale transformation techniques to enhance a given image. The system analyses selected regions of the image and uses the information in its selection process. When the algorithms have been selected the processed images can be displayed and compared, and each algorithm can be compared using different parameter values (if there is a parameter associated with the algorithm).

The system is certainly a great improvement on the way algorithms are currently selected. It is no longer necessary to try each algorithm individually because the system has rules encoded into it to eliminate algorithms which do not enhance the required information, or destroy too much information. The algorithms selected by the system have to fulfill three quite demanding criteria, ensuring that the system is a useful and reliable tool.

Appendices

Appendix A

Results of visual experiments

A.1 Experiment 1

A.1.1 Results for the first observer

B	Hitachi LR47156						GDM-1604A40						GDM-1604-40					
	$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
0	5	5	6	-	-	-	54	53	54	-	-	-	48	47	47	-	-	-
5	5	4	5	-	-	-	48	47	48	-	-	-	42	43	41	-	-	-
10	4	4	5	4	5	6	44	44	44	-	-	-	38	38	38	-	-	-
15	4	3	4	4	4	4	39	38	38	-	-	-	34	33	30	-	-	-
20	4	2	3	3	3	3	33	33	34	-	-	-	28	27	26	-	-	-
25	4	2	3	3	3	3	28	28	28	-	-	-	23	23	22	-	-	-
30	3	2	3	2	2	2	24	23	23	-	-	-	17	17	17	-	-	-
35	3	2	2	2	2	2	19	18	18	-	-	-	12	14	13	-	-	-
40	3	2	2	2	2	2	15	13	14	-	-	-	11	9	8	-	-	-
45	3	2	2	2	2	2	11	9	9	-	-	-	8	6	6	8	12	7
50	3	2	2	2	2	2	8	6	6	-	-	-	5	5	4	4	5	4
55	3	2	2	2	2	2	5	4	4	-	8	9	4	3	3	2	3	2
60	3	2	2	2	2	2	3	3	3	13	4	6	3	3	2	2	2	1
65	3	2	2	2	2	2	3	2	2	8	3	2	2	2	2	2	2	1
70	3	2	2	2	2	2	2	2	2	4	2	2	2	2	2	1	1	1
75	3	2	2	2	2	2	1	1	1	3	1	2	2	2	2	1	1	1
80	2	2	2	2	2	2	1	1	1	2	1	1	2	2	2	1	1	1
85	2	2	2	2	2	2	1	1	1	1	1	1	2	1	2	1	1	1
90	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
95	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1

B	Hitachi LR47156						GDM-1604A40						GDM-1604-40					
	$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
100	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
105	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	1	1	1
110	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
115	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
120	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
125	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
130	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
135	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
140	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
145	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
150	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
155	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
160	3	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
165	3	2	2	2	2	2	1	1	1	1	1	1	1	2	2	1	1	1
170	3	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
175	3	2	2	2	2	2	1	1	1	1	1	1	1	1	2	1	1	1
180	3	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	1	1
185	3	2	2	2	2	2	1	1	1	1	1	1	2	2	1	1	1	1
190	3	2	2	2	2	2	1	1	1	1	1	1	2	2	1	1	1	1
195	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	1	1
200	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	1	1
205	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	2	1
210	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	1	1
215	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	1	1
220	3	2	2	2	2	2	1	1	1	1	1	1	2	2	2	1	2	1
225	3	2	2	2	2	2	2	1	1	1	1	1	2	2	2	1	1	1
230	3	2	2	2	2	2	2	1	1	2	1	1	2	2	2	2	2	2
235	3	2	2	2	3	3	2	1	2	2	2	1	2	2	2	2	2	2
240	3	2	2	3	3	3	2	1	2	2	2	1	2	2	2	2	2	2
245	3	2	2	3	2	3	2	2	2	2	2	2	2	2	2	2	2	2
250	3	2	2	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
255	-	-	-	3	3	3	-	-	-	2	2	2	-	-	-	2	2	2

A.1.2 Results for the second observer

B	Hitachi LR47156						GDM-1604A40						GDM-1604-40					
	$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
0	9	6	5	-	-	-	52	55	51	-	-	-	45	45	44	-	-	-
5	9	4	3	5	5	-	48	49	47	-	-	-	37	43	38	-	-	-
10	5	4	2	4	4	5	43	46	41	-	-	-	32	37	34	-	-	-
15	3	3	2	2	3	4	39	40	37	-	-	-	25	31	29	-	-	-

B	Hitachi LR47156						GDM-1604A40						GDM-1604-40					
	$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
20	5	3	2	3	3	3	32	33	32	-	-	-	24	28	22	-	-	-
25	3	2	2	3	2	3	28	29	27	-	-	-	14	23	19	-	-	-
30	4	2	1	2	2	2	22	24	20	-	-	-	13	17	14	-	-	-
35	3	2	2	3	2	2	19	19	17	-	-	-	-	13	9	-	-	-
40	3	2	1	3	2	2	15	17	12	-	-	-	9	9	6	10	-	-
45	2	2	1	2	2	2	7	12	8	-	-	-	5	6	2	8	5	7
50	3	2	1	2	2	2	4	8	4	-	7	-	5	3	3	4	4	3
55	2	2	1	2	2	2	3	3	4	8	3	5	3	3	2	4	2	2
60	3	2	1	2	2	2	3	4	2	4	3	3	3	2	2	3	2	2
65	2	2	1	2	2	2	2	2	2	2	2	2	3	2	2	3	2	1
70	3	1	1	2	2	2	2	2	1	2	1	2	2	2	1	2	1	1
75	2	2	1	2	2	2	1	2	1	2	1	2	2	2	1	2	1	1
80	2	2	1	2	2	2	1	1	1	1	1	1	2	2	1	2	1	1
85	2	2	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
90	2	1	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
95	2	2	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
100	2	2	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
105	2	1	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
110	2	2	1	2	2	2	1	1	1	1	1	1	2	1	1	1	1	1
115	2	2	1	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
120	3	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
125	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
130	3	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
135	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
140	3	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
145	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
150	3	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
155	2	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
160	4	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
165	2	2	2	2	2	2	1	1	1	1	1	1	2	1	2	2	1	1
170	3	2	2	2	2	2	1	1	1	1	1	1	2	1	1	2	1	1
175	2	2	2	2	2	2	1	1	1	1	1	1	2	1	2	2	1	1
180	3	2	2	2	2	2	1	1	1	1	1	1	2	1	2	2	1	1
185	2	2	2	3	2	2	1	1	1	1	1	1	2	1	2	2	1	1
190	3	2	2	2	2	2	1	1	1	1	1	1	2	1	2	2	1	1
195	3	2	2	2	2	2	1	1	1	1	1	1	2	1	2	2	1	1
200	3	2	2	3	2	2	1	1	1	1	1	1	2	1	2	2	1	1
205	3	2	2	3	2	2	1	1	1	1	1	1	2	2	2	2	1	1
210	3	2	2	3	2	2	1	1	1	1	1	1	2	2	2	3	1	1
215	3	2	2	3	2	2	1	1	1	1	1	1	3	1	2	3	1	1
220	3	2	2	3	2	3	1	1	2	1	1	1	2	1	2	3	2	1
225	3	2	2	3	2	2	1	1	2	1	1	1	3	2	2	3	2	1
230	4	2	2	3	2	3	2	1	2	1	1	1	2	1	2	3	2	2
235	3	2	3	3	2	3	2	1	2	1	1	1	3	2	2	2	2	2

B	Hitachi LR47156						GDM-1640A40						GDM-1604-40					
	$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$			$+\Delta B$			$-\Delta B$		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
240	4	2	3	3	3	3	2	1	2	1	1	1	3	1	2	3	2	2
245	3	2	2	3	3	3	2	1	2	1	2	2	3	2	2	2	2	2
250	4	2	2	3	3	3	2	2	2	2	2	2	2	2	2	3	2	2
255	-	-	-	3	3	3	-	-	-	2	2	2	-	-	-	3	2	2

A.1.3 Averaged values

	Hitachi LR47156			GDM-1604A40			GDM-1604-40		
B	$+\Delta B$	$-\Delta B$	ΔB	$+\Delta B$	$-\Delta B$	ΔB	$+\Delta B$	$-\Delta B$	ΔB
0	6	-	6	46	-	46	53	-	53
5	5	5	5	41	-	41	48	-	48
10	4	5	5	36	-	36	44	-	44
15	3	3	3	30	-	30	38	-	38
20	3	3	3	26	-	25	33	-	33
25	3	3	3	21	-	21	28	-	28
30	3	2	3	16	-	16	23	-	23
35	2	2	2	12	-	12	18	-	18
40	2	2	2	9	-	9	14	-	14
45	2	2	2	5	8	7	10	-	10
50	2	2	2	4	4	4	6	-	6
55	2	2	2	3	2	3	4	7	6
60	2	2	2	3	2	3	3	6	5
65	2	2	2	2	2	2	2	3	3
70	2	2	2	2	1	2	2	2	2
75	2	2	2	2	1	2	1	2	2
80	2	2	2	2	1	2	1	1	1
85	2	2	2	1	1	1	1	1	1
90	2	2	2	1	1	1	1	1	1
100	2	2	2	1	1	1	1	1	1
105	2	2	2	1	1	1	1	1	1
110	2	2	2	1	1	1	1	1	1
115	2	2	2	1	1	1	1	1	1
120	2	2	2	1	1	1	1	1	1
125	2	2	2	1	1	1	1	1	1
130	2	2	2	1	1	1	1	1	1
135	2	2	2	1	1	1	1	1	1
140	2	2	2	1	1	1	1	1	1
145	2	2	2	1	1	1	1	1	1
150	2	2	2	1	1	1	1	1	1
155	2	2	2	1	1	1	1	1	1
160	2	2	2	1	1	1	1	1	1
165	2	2	2	2	1	2	1	1	1
170	2	2	2	1	1	1	1	1	1

	Hitachi LR47156			GDM-1604-40			GDM-1604A40		
B	$+\Delta B$	$-\Delta B$	ΔB	$+\Delta B$	$-\Delta B$	ΔB	$+\Delta B$	$-\Delta B$	ΔB
175	2	2	2	1	1	1	1	1	1
180	2	2	2	1	1	1	1	1	1
185	2	2	2	2	1	2	1	1	1
190	2	2	2	2	1	2	1	1	1
195	2	2	2	2	1	2	1	1	1
200	2	2	2	2	1	2	1	1	1
205	2	2	2	2	1	2	1	1	1
210	2	2	2	2	1	2	1	1	1
215	2	2	2	2	2	2	1	1	1
220	2	2	2	2	2	2	1	1	1
225	2	2	2	2	2	2	1	1	1
230	2	2	2	2	2	2	2	1	2
235	2	3	3	2	2	2	2	1	2
240	3	3	3	2	2	2	2	1	2
245	2	3	3	2	2	2	2	2	2
250	3	3	3	2	2	2	2	2	2
255	-	3	3	-	2	2	-	2	2

A.2 Second experiment

B	Hitachi LR47156									
	B_i					B_{-i}				
	1	2	3	4	5	1	2	3	4	5
0	6	9	12	14	17	-	-	-	-	-
10	14	17	20	22	24	5	0	-	-	-
20	23	25	27	29	31	17	14	10	6	0
30	33	35	37	39	41	28	26	24	22	20
40	42	44	46	48	50	38	36	34	32	30
50	52	54	56	58	60	48	46	44	42	40
60	62	64	66	68	70	58	56	54	52	50
70	72	74	76	78	80	68	66	64	62	60
80	82	84	86	88	90	78	76	74	72	70
90	92	94	96	98	100	88	86	84	82	80
100	102	104	106	108	110	98	96	94	92	90
110	112	114	116	118	120	108	106	104	102	100
120	122	124	126	128	130	118	116	114	112	110
130	132	134	136	138	140	128	126	124	122	120
140	142	144	146	148	150	138	136	134	132	130
150	152	154	156	158	160	148	146	144	142	140
160	162	164	166	168	170	158	156	154	152	150
170	172	174	176	178	180	168	166	164	162	160
180	182	184	186	188	190	178	176	174	172	170
190	192	194	196	198	200	188	186	184	182	180

B	Hitachi LR47156									
	B_i					B_{-i}				
	1	2	3	4	5	1	2	3	4	5
200	202	204	206	208	210	198	196	194	192	190
210	212	214	216	218	220	207	205	203	201	199
220	222	224	226	228	230	217	215	213	211	209
230	232	234	236	238	240	227	225	223	221	219
240	243	245	247	249	251	237	235	233	231	229
250	253	255	-	-	-	247	244	242	240	238

B	GDM-1604-40									
	B_i					B_{-i}				
	1	2	3	4	5	1	2	3	4	5
0	46	51	54	57	59	-	-	-	-	-
10	46	51	54	57	59	-	-	-	-	-
20	46	51	54	57	59	-	-	-	-	-
30	46	51	54	57	59	-	-	-	-	-
40	49	52	55	57	59	-	-	-	-	-
50	54	57	59	61	63	46	39	-	-	-
60	62	64	66	68	70	58	56	54	51	48
70	72	73	74	75	76	69	68	67	66	65
80	82	83	84	85	86	79	78	77	76	75
90	91	92	93	94	95	89	88	87	86	85
100	101	102	103	104	105	99	98	97	96	95
110	111	112	113	114	115	109	108	107	106	105
120	121	122	123	124	125	119	118	117	116	115
130	131	132	133	134	135	129	128	127	126	125
140	141	142	143	144	145	139	138	137	136	135
150	151	152	153	154	155	149	148	147	146	145
160	161	162	163	164	165	159	158	157	156	155
170	171	172	173	174	175	169	168	167	166	165
180	181	182	183	184	185	179	178	177	176	175
190	191	193	195	197	199	189	188	187	186	185
200	202	204	206	208	210	199	198	197	196	195
210	212	214	216	218	220	209	208	207	206	205
220	222	224	226	228	230	219	218	217	216	215
230	232	234	236	238	240	228	227	226	225	224
240	242	244	246	248	250	238	237	236	235	234
250	252	254	-	-	-	248	247	246	245	244

B	GDM-1604A40									
	B_i					B_{-i}				
	1	2	3	4	5	1	2	3	4	5
0	53	60	64	67	69	-	-	-	-	-
10	54	60	64	67	69	-	-	-	-	-
20	53	60	64	67	69	-	-	-	-	-

B	GDM-1604A40									
	B_i					B_{-i}				
	1	2	3	4	5	1	2	3	4	5
30	53	60	64	67	69	-	-	-	-	-
40	54	60	63	65	67	-	-	-	-	-
50	56	61	64	67	69	-	-	-	-	-
60	63	66	68	70	72	-	-	-	-	-
70	72	74	76	77	78	67	64	60	53	-
80	81	83	84	85	86	77	75	73	70	67
90	91	92	93	94	95	88	86	84	82	80
100	101	102	103	104	105	98	96	93	91	89
110	111	112	113	114	115	108	106	105	103	101
120	121	122	123	124	125	118	117	116	115	114
130	131	132	133	134	135	128	126	124	122	120
140	141	142	143	144	145	138	137	136	135	134
150	151	152	153	154	155	149	148	147	146	145
160	161	162	163	164	165	159	158	156	154	152
170	171	172	173	174	175	168	167	166	165	165
180	181	182	183	184	185	178	176	175	173	171
190	191	192	193	194	195	189	188	187	186	185
200	201	202	203	204	205	199	198	197	196	195
210	211	212	213	214	215	209	208	207	206	205
220	221	222	223	224	225	219	218	216	214	213
230	232	233	234	235	236	228	227	226	224	223
240	242	243	244	245	246	238	236	234	232	232
250	252	253	254	255	-	248	246	244	242	240

Appendix B

The code for "global lut stretch with parameter"

global lut.code **global_lut.code**

```
/* @(#) Global_lut_stretch */

/* The functions in the file are listed below :

   set_lut()          set the lut scale for the appropriate monitor
   set_lut_hitachiLR() sets the lut scale for the Hitachi LR47156
   set_lut_gdm()      sets the lut scale for the GDM-1604-40
   set_lut_gdmA()     sets the lut scale for the GDM-1604A40
   left_side()        separates grey levels to the left of the background
   right_side()       separates grey levels to the right of the background
   find_nl()          find the number of occupied grey levels between the
                      grey levels A and B
   histogram_info()   find the lowest grey level value in the image, the
                      highest value, and the total number of grey levels
                      used
   contract()         compress a given number of grey level bins in the
                      histogram
   find_limit()       find which grey levels bins to compress
   count_bins()       find the number of occupied grey level bins in the
                      histogram below a given value
   maximum()          find the maximum value of the two parameters

*/

/* Structure containing information about the histogram */

typedef struct hist_info {
   int start;         /* starting position of histogram */
   int end;           /* end position of histogram */
   int no_levels;     /* number of occupied levels in the histogram */
};

#define SPACE 10      /* contains the number of bins to compress */
#define OFFSET 50     /* contains the offset of the monitor.  In this case
                         it is for the GDM-16404-40 */
#define INC 1         /* increment to be added to the lut value */
#define E 3           /* increment to be added to bins close to the
                         background*/

/* External definitions */

int lut[256];         /* lut[i] separation of adjacent bins from the
                         background grey level i */

int alut[256][256];      /* alut[i][j] separation of adjacent bin from j
                            against a background grey level i */

/* External declarations: these are defined elsewhere */

extern int histogram_of_image[256];   /* image histogram */
```

global lut.code **global_lut.code**

```
extern int temp_histogram[256];       /* histogram of the background area */


int *global_lut_proc()
{
   int *new_level, i, nb, na, available, sp, start;
   int *left_side(), *right_side(), back;
   struct hist_info *li, *gi, *histogram_info();

   set_lut()   /* sets up the look up tables for the different monitors */

   /* Initialise new levels storing grey scale transformation */

   new_level = (int *) malloc(256*(sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Find information about the background selected */

   li = (struct hist_info *) histogram_info(temp_histogram);

   /* Find information about the histogram of the image */

   gi = (struct hist_info *) histogram_info(histogram_of_image);

   /* Calculate the number of grey levels availabel for stretching */

   available = 256 - OFFSET;

   /* Find the number of occupied grey levels in the image between the
      start of occupied grey scale and the start of the background */

   nb = find_nl(histogram_of_image, gi->start, li->start);

   /* Find the number of occupied grey levels in the image between the
      end of the background and the end of the occupied grey scale */

   na = find_nl(histogram_of_image, li->end, gi->end);

   /* If the number of grey levels occupied plus the space for stretching
      is larger than the available grey scale then compress the histogram */

   if (na + nb + (li->end-li->start+1) + SPACE > available) {

      contract(na+nb+(li->end-li->start+1)-available+space,new_level,gi,li);
      return;
   }
```

global lut.code **global_lut.code**

```
   /* Skewed position of the background based on the number of proportion
      of occupied grey levels before and after the background */

   sp =  OFFSET + (nb*available)/(na+nb+(li->end-li->start)+1);

   /* Move the background to the skewed position */

   for (i = li->start; i <= li->end; i++) new_level[i] = sp+i-li->start;

   /* Separate the bin immediately to the left of the background */

   new_level[li->start - 1] = sp - lut[sp] - INC - E;
   start = li->start - 2;

   /* Set the background to be midway its range */

   back = new_level[(li->start + li->end)/2];

   /* Calculate the number of occupied grey levels before the starting
      point */

   nb = find_nl(histogram_of_image, gi->start, start+1);

   /* Separate the grey level bins to the left */

   if (nb > 0)
      new_level=left_side(start,gi,nb-1,new_level,histogram_of_image,back);

   /* Separate the bin immediately to the right of the background */

   new_level[li->end + 1] = sp + (li->end - li->start) + 1 + lut[sp]
                              + INC + E;
   start = li->end + 2;

   /* Calculate the number of occupied grey levels after the background
      and the end of the occupied grey scale range of the image */

   na = find_nl(histogram_of_image, start, gi->end+1);

   /* Separate the grey level bins to the right of the background */

   if (na > 0)
      new_level=right_side(start,gi,na-1,new_level,histogram_of_image,back);

   free(li);
   free(gi);
   return(new_level);
}

/* Separate grey levels of the background */
```

global lut.code **global_lut.code**

```
int *left_side(start, gi, nb_bins, new_level, histogram, back)
int start, nb_bins, *new_level, *histogram, back;
struct hist_info *gi;
{
   int flag = 0, end, i, point, counter = 0;

   point = new_level[start+1];

   for (i = start; i >= gi->start; i--) {

      if (histogram[i] == 0)   /* Remove empty bins */
         new_level[i] = point;

      else if (nb_bins <= new_level[i+1] - OFFSET) {  /* Check space left */

         if (counter < 4) {
            new_level[i] = new_level[i+1] - alut[back][new_level[i+1]];
            new_level[i] = new_level[i]-INC-maximum(0,E-1-(start-i));
         } else {
            new_level[i] = new_level[i+1] - lut[new_level[i+1]];
            new_level[i] = new_level[i]-INC-maximum(0,E-1-(start-i));
         }

         ++counter;

         if (new_level[i] < 0) new_level[i] = 0;

         point = new_level[i];
         --nb_bins;

      } else {
         ++nb_bins;
         flag = 1;
         end = i;
         i = gi->start-1;
      }
   }

   if (flag == 1) {   /* Fill in the remaining bins */
      new_level[end + 1] = OFFSET + nb_bins;
      point = new_level[end+1];

      for (i = end; i >= gi->start; i--) {

         if (histogram[i] == 0)
            new_level[i] = point;

         else {
            new_level[i] = new_level[i+1] - 1;
         point = new_level[i];
      }
    }
  }
  return(new_level);
}

/* Do the right side (same as left side) */

int *right_side(start, gi, nb_bins, new_level, histogram, back)
int start, nb_bins, *new_level, *histogram, back;
struct hist_info *gi;
{
   int flag = 0, end, i, point, counter = 0;

   point = new_level[start-1];

   for (i = start; i <= gi->end; i++) {

      if (histogram[i] == 0)
         new_level[i] = point;

      else if (nb_bins <= 255 - new_level[i-1]) {

         if (counter < 4) {
            new_level[i] = new_level[i-1] + alut[back][new_level[i-1]];
            new_level[i] += INC + maximum(0, E - 1 - (i-start));
         } else {
            new_level[i] = new_level[i-1] + lut[new_level[i-1]];
            new_level[i] += INC + maximum(0, E - 1 - (i-start));
         }

         if (new_level[i] > 255) new_level[i] = 255;
         ++counter;

         point = new_level[i];
         --nb_bins;

      } else {
         ++nb_bins;
         flag = 1;
         end = i;
         i = gi->end+1;
      }
   }

   if (flag == 1) {

      new_level[end - 1] = 255 - nb_bins;
      point = new_level[end-1];
```

```
      for (i = end; i <= gi->end; i++) {

         if (histogram[i] == 0)
            new_level[i] = point;

         else {
            new_level[i] = new_level[i-1] + 1;
            point = new_level[i];
         }
      }
   }
   return(new_level);
}

/* Find the number of occupied bins between the limits start and end */
/* end should be one more than the end point required */

find_nl(histogram, start, end)
int *histogram, start, end;
{
   int i, count = 0;

   for (i = start; i < end; i++)
      if (histogram[i] != 0)
         ++count;

   return(count);
}

/* This procedure returns the starting and end positions of the histogram */

struct hist_info *histogram_info(histogram)
int *histogram;
{
   register int i;
   int total = 0, cdf = 0, count = 0;
   struct hist_info *hi;

   hi = (struct hist_info *) malloc(sizeof(struct hist_info));

   for (i = 0; i < 256; i++)
      total += histogram[i];

   for (i = 0; i < 256; i++) {

      if (histogram[i] != 0)
         count++;

      if ((cdf == 0) && (histogram[i] > 0)) {
```

```
      hi->start = i;
    }

    cdf += histogram[i];

    if ((cdf == total) && (histogram[i] > 0))
      hi->end = i;
  }
  hi->no_levels = count;
  return(hi);
}

/* Contract the histogram.  Remove limit bins.  Limit is never exact */
/* it is always an overestimate.  */

contract(limit, new_level, gi, li)
int limit, *new_level;
struct hist_info *gi, *li;
{
  int threshold, i, count = 0, point, *more_levels, *newer_histogram;
  int nb, na, available, sp, start, nb_bins, back;
  struct hist_info *ni, *histogram_info();

  /* Find the value below which all bins should be compressed */

  threshold = find_limit(limit, li);

  /* Initialise newer histogram to contain the recalculated distribution
     and more levels to store the grey scale transformation */

  newer_histogram = (int *) malloc(256 * (sizeof(int)));
  more_levels = (int *) malloc(256 * (sizeof(int)));

  for (i = 0; i < 256; i++) {
    more_levels[i] = i;
    newer_histogram[i] = 0;
  }

  /* Compress the bins below the value of threshold */

  point = 0;
  for (i = gi->start; i <= gi->end; i++) {

    if (i == li->end + 1) point = li->end;

    if (i >= li->start && i <= li->end) {
      new_level[i] = i;
    } else {
      if (histogram_of_image[i] != 0) {
        if (histogram_of_image[i] > threshold) {
```

```
          new_level[i] = i;
          point = i;
        } else {
          new_level[i] = point;
        }
      } else {
        new_level[i] = i;
      }
    }

    /* Calculate the new histogram */

    newer_histogram[new_level[i]] += histogram_of_image[i];
  }

  /* Separate the adjacent bins (as before) */

  ni = (struct hist_info *) histogram_info(newer_histogram);
  available = 256 - OFFSET;
  nb = find_nl(newer_histogram, 0, li->start);
  na = find_nl(newer_histogram, li->end+1, 256);
  sp =  OFFSET + (nb*available)/(na+nb+(li->end-li->start));

  for (i = li->start; i <= li->end; i++) more_levels[i] = sp+i-li->start;

  back = more_levels[(li->start+li->end)/2];
  more_levels[li->start - 1] = sp - lut[sp];
  start = li->start - 2;
  nb = find_nl(newer_histogram, 0, start+1);
  more_levels = left_side(start,ni,nb,more_levels,newer_histogram,back);

  more_levels[li->end + 1] = sp + (li->end - li->start) + 1 + lut[sp];
  start = li->end + 2;
  na = find_nl(newer_histogram, li->end+2, 256);
  more_levels = right_side(start,ni,na,more_levels,newer_histogram,back);

  free(more_levels); free(newer_histogram);
  free(ni); free(gi); free(li);
  return(new_levels);
}

/* Find a mark in the histogram below which there are so many bins */
/* Telephone search */

find_limit(limit, li)
int limit;
struct hist_info *li;
{
  extern struct rasterfile rh;
  int average, bottom, bottom_bins, top, flag = 0, value, old_value;
```

```
   average = rh.ras_length/256;
   value = average/2;
   bottom = 0;
   bottom_bins = count_bins(value, li);

   while (flag == 0) {
      old_value = value;

      /* Within five bins is close enough */

      if (bottom_bins - limit > 0 && bottom_bins - limit < 5)
         flag = 1;

      else if (bottom_bins - limit < 0) { /* Less than - add to value */
         bottom = value;
         value = (top+bottom)/2;

      } else { /* More than - subtract from value */
         top = value;
         value = (top + bottom)/2;
      }

      bottom_bins = count_bins(value, li);

      if (old_value == value) { /* Exit if same value twice */
         flag = 1;
         ++value;
      }
   }
   bottom_bins = count_bins(value, li);
   return(value);
}

/* Count the number of bins below value and not in the background region */

count_bins(value, li)
int value;
struct hist_info *li;
{
   int i, count = 0;

   for (i = 0; i < li->start; i++)
      if (histogram_of_image[i] > 0 && histogram_of_image[i] <= value)
         ++count;

   for (i = li->end+1; i < 256; i++)
      if (histogram_of_image[i] > 0 && histogram_of_image[i] <= value)
         ++count;
```

```
   return(count);
}

/* Return the maximum value of a and b */

maximum(a, b)
int a, b;
{
   if (a >= b)
      return(a);
   else
      return(b);
}

/* Set_lut()

   Called : by [ simple_lut_stretch_with_parameter() ] above.

   Function : sets up the lut values for the monitor.
*/

set_lut()
{
   static void set_lut_gdm(), set_lut_hitachiLR(), set_lut_hitachiHM();

   switch(OFFSET) {

      case 10: set_lut_hitachiLR();
               break;

      case 60: set_lut_gdmA();
               break;

      case 50: set_lut_gdm();
               break;

      default: break;
   }
}

/*
   Set_lut_gdm()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the GDM-1604-40 monitor.

*/

set_lut_gdm()
```

```
{
   int i, j;

   for (i = 0; i < 40; i++)  lut[i] = 46 - i;
   for (i = 40; i < 45; i++) lut[i] = 9;
   for (i = 45; i < 50; i++) lut[i] = 5;
   for (i = 50; i < 55; i++) lut[i] = 4;
   for (i = 55; i < 65; i++) lut[i] = 3;
   for (i = 65; i < 85; i++) lut[i] = 2
   for (i = 85; i < 185; i++) lut[i] = 1
   for (i = 185; i < 255; i++) lut[i] = 2


    /* Ensures that for lut values less than 55 the transformation goes
       off the scale */

   for (i = 0; i < 50; i++)
      for (j = 0; j < i; j++)
         alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 50; i < 70; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 70; i < 210; i++)
      for (j = 50; j < 60; j++)
         alut[i][j] = 2;

   for (i = 210; i < 255; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;
}

/*
   Set_lut_hitachiLR()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the Hitachi-LR47156 monitor.

*/

set_lut_hitachiLR()
{
   int i;

   for (i = 0; i < 5; i++)
```

```
      lut[i] = 6;
   for (i = 5; i < 10; i++)
      lut[i] = 5;
   for (i = 10; i < 15; i++)
      lut[i] = 4;
   for (i = 15; i < 30; i++)
      lut[i] = 3;
   for (i = 30; i < 240; i++)
      lut[i] = 2;
   for (i = 240; i < 255; i++)
      lut[i] = 3;

    /* Ensures that for lut values less than 15 the transformation goes
       off the scale */

   for (i = 0; i < 10; i++)
      for (j = 0; j < i; j++)
         alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 10; i < 30; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 4;

   for (i = 30; i < 75; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 75; i < 160; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 2;

   for (i = 160; i < 256; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;
}

/*
   Set_lut_gdmA()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the GDM-1604A40 monitor.

*/

set_lut_gdmA()
{
```

global lut.code   global_lut.code

```
   int i, j;

   for (i = 0; i < 50; i++)  lut[i] = 53-i;
   for (i = 50; i < 54; i++) lut[i] = 4;
   for (i = 55; i < 65; i++) lut[i] = 3;
   for (i = 65; i < 75; i++) lut[i] = 2;
   for (i = 75; i < 230; i++) lut[i] = 1;
   for (i = 230; i ≤ 255; i++) lut[i] = 2;

    /* Ensures that for lut values less than 65 the transformation goes
       off the scale */

   for (i = 0; i < 60; i++)
      for (j = 0; j < i; j++)
          alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 60; i < 70; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 70; i < 256; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 2;
}
```

Appendix C

The code for "multiple lut stretch with parameter"

mult lut **mult_lut.code**

```
/* @(#) Multiple_lut_stretch */

/* The functions in the file are listed below

   set_lut()             set the lut scale for the appropriate monitor
   set_lut_hitachiLR()   sets the lut scale for the Hitachi LR47156
   set_lut_gdm()         sets the lut scale for the GDM-1604-40
   set_lut_gdmA()        sets the lut scale for the GDM-1604A40
   get_parameters()      reads in the parameters for the algorithm from a
                         file "pixy_parmaeters"
   find_limit()          find which grey levels bins to compress
   count_bins()          find the number of occupied grey level bins in the
                         histogram below a given value
   find_nl()             find the number of occupied grey levels between the
                         grey levels A and B
   multiple_contract()   removes the required number of bins from the
                         histogram
   find_largest_pos()    finds the largest bin iin a given histogram
                         between two limits
   skewness()            finds the position to map a bin according to
                         the number of occupied grey level bins before
                         and after it
   spread_peaks()        spreads the bins on either side of a background
                         peak
   first_levels()        spreads the bins around the first peak
   last_levels()         spreads the bins around the last peak
   histogram_info()      find the lowest grey level value in the image, the
                         highest value, and the total number of grey levels
                         used
   maximum()          find the maximum value of the two parameters

*/

/* Structure containing the parameters to be used in the algorithm */

typedef struct read_in_parameters {
    float *pars;       /* stores the parameter values */
    int number;        /* the number of parameters */
};

/* Structure containing information about the histogram */

typedef struct hist_info {
   int start;         /* starting position of histogram */
   int end;           /* end position of histogram */
   int no_levels;     /* number of occupied levels in the histogram */
};


#define OFFSET 50     /* contains the offset of the monitor.  I this case
```

mult lut **mult_lut.code**

```
                         it is for the GDM-16404-40 */
#define TRUE 1          /* Boolean definitions */
#define FALSE 0

/* External definitions */

int lut[256];           /* lut[i] is the separation of adjacent bins from the
                           background grey level i */

int alut[256][256]      /* alut[i][j] is the separation of adjacent bin j from
                           a background grey level i */

/* External declarations : these are defined elsewhere */

extern int temp_histogram[256];         /* image histogram */
extern int image_length;                /* length of the image */


int *multiple_lut_stretch_with_parameter_proc()
{
   int point, *new_level, *multiple_contract();
   struct read_in_parameters *params, *get_parameters();

   /* Set the lut values for the monitor used */

   set_lut();

   /* Get the parameters for the algorithm : these are the number of
      bins to compress and the bounds of the regions selected */

   params = (struct read_in_parameters *) get_parameters();

   if ((point = find_limit(params)) != -1)
      new_level = (int *) multiple_contract(point, params);
   else
      fprintf(stderr, "\n Error : too few bins available to compress");

   free(params->pars);
   free(params);
   return(new_level);
}

/* Find a mark in the histogram below which there are so many bins */
/* Telephone search */

find_limit(params)
struct read_in_parameters *params;
{
   struct hist_info *hist_info(), *info;
   int average, bottom, bottom_bins, top, flag = 0, value, old_value;
```

mult lut **mult_lut.code**

```
int count = 0, i, j, number, start, end, limit, *minus_histogram;

minus_histogram = (int *) malloc(256*(sizeof(int)));

for (i = 0; i < 256; i++) minus_histogram[i] = temp_histogram[i];

/* The number of bins to compress */

limit = (int) params->pars[0];

/* The number of regions selected */

number = ((int) params->number - 1)/2;

/* Remove the regions of interest from the histogram */

for (i = 0; i < number; i++) {

   for (j = params->pars[2*i+1]; j <= params->pars[2*i+2]; j++) {

      minus_histogram[j] = 0;

      if (temp_histogram[j] != 0)
         ++count;
   }
}

/* Find the start, end and number of occupied bins in the image
   histogram */

info = (struct hist_info *) histogram_info(temp_histogram);
count = info->no_levels - count;

/* Must be enough bins to compress */

if (count < limit) {
   start = (start+end)/2 - 1;
   end = (start+end)/2 + 1;
} else
   return(-1);

/* Find below which bin size to contract */

average = image_length/256;
value = average/2;
bottom = 0;
bottom_bins = count_bins(value, minus_histogram);

while (flag == 0) {
   old_value = value;
```

mult lut **mult_lut.code**

```
      if (bottom_bins - limit > 0 && bottom_bins - limit < 5)
         flag = 1;
      else if (bottom_bins - limit < 0) { /* Less than - add to value */
         bottom = value;
         value = (top+bottom)/2;
      } else { /* More than - subtract from value */
         top = value;
         value = (top + bottom)/2;
      }
      bottom_bins = count_bins(value, minus_histogram);
      if (old_value == value) { /* Exit if same value twice */
         flag = 1;
         ++value;
      }
   }
   bottom_bins = count_bins(value, minus_histogram);
   free(minus_histogram);
   return(value);
}


/* Count the number of bins below value and not in the background regions */

count_bins(value, histogram)
int value, *histogram;
{
   int i, count = 0;

   for (i = 0; i < 256; i++)
      if (histogram[i] > 0 && histogram[i] <= value)
         ++count;

   return(count);
}

/* Find the number of occupied bins between the limits start and end */
/* end should be one more than the end point required */

find_nl(histogram, start, end)
int *histogram, start, end;
{
   int i, count = 0;

   for (i = start; i < end; i++)
      if (histogram[i] != 0)
         ++count;

   return(count);
}
```

```
/* Reads in the parameters for the algorithm from the file "pixy parameters".
   The first parameter contains the number of bins to compress, the rest are
   the paired upper and lower bounds of the regions identified by the user
   (the pairs are sorted into increasing grey level order; they are also
   non-overlapping) */


struct read_in_parameters *get_parameters()
{
   FILE *input, *fopen();
   int i;
   struct read_in_parameters *params;

   input = fopen("pixy_parameters", "r");

   /* Get the number of parameters */

   fscanf(input, "%d", &params.number);

   params = (struct read_in_parameters *)
            malloc(sizeof(struct read_in_parameters));

   params->pars = (float *) malloc (params.number * (sizeof(float)));

   /* Read in the parameters */

   for (i = 0; i < params->number; i++) {
      fscanf(input, "%f", &params->pars[i]);
   }

   fclose(input);
   return(params);
}


/*
   Deletes small bins below threshold not contained in areas of
   interest, then spreads out the in the areas of interest around
   the background level of the area.
*/

int *multiple_contract(threshold, params)
int threshold;
struct read_in_parameters *params;
{
   int i, j, number, *new_level, *new_histogram, start, end, point;
   int *skew, position, *newer_histogram, *newer_level;
   struct hist_info info;

   new_level = (int *) malloc(256*(sizeof(int)));
```

```
   newer_level = (int *) malloc(256*(sizeof(int)));
   new_histogram = (int *) malloc(256*(sizeof(int)));
   newer_histogram = (int *) malloc(256*(sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i; new_histogram[i] = 0;
      newer_level[i] = i; newer_histogram[i] = 0;
   }

   number = ((int)params.number - 1)/2;

   point = 0;

   for (i = 0; i < number + 1; i++) {

      if (i == 0) {
         start = 0;
         end = (int) params->pars[1];
      } else if (i == number) {
         start = (int) params->pars[2*i] + 1;
         end = 256;
         point = start - 1;
      } else {
         start = (int) params->pars[2*i] + 1;
         end = (int) params->pars[2*i+1];
         point = start - 1;
      }

      for (j = start; j < end; j++) {

         if (temp_histogram[j] != 0) {

            if (temp_histogram[j] > threshold) {

               new_level[j] = j;
               point = j;

            } else {
               new_level[j] = point;
            }
         } else {
            new_level[j] = j;
         }
      }
   }

   for (i = 0; i < 256; i++)
      new_histogram[new_level[i]] += temp_histogram[i];

   skew = (int *) malloc(params->number * (sizeof(int)));
   for (i = 1; i < params->number; i+=2) {

      position = find_largest_pos(new_histogram,(int)params->pars[i],
                                  (int)params->pars[i+1]);
      params->pars[i/2] = (int) position;
      skew[i/2] = skewness(new_histogram, position);
   }

   params->number = (params->number - 1)/2;

   info = (hist_info *) hist_info(new_histogram);
   newer_level = (int *) spread_peaks(new_histogram, params, skew, info);

   for (i = 0; i < 256; i++) {
      newer_histogram[newer_level[i]] += new_histogram[i];
      new_level[i] = newer_level[new_level[i]];
   }

   free(newer_level); free(new_histogram);
   return(new_level);
}

/* Finds the position of the bin [ peak ] according to the number
   of occupied bins befor and after it */

skewness(histogram, peak)
int histogram[256], peak;
{
   struct hist_info *info;
   int left = 0, right = 0, i;

   info = (struct hist_info *) histogram_info(histogram);

   for (i = info->start; i < peak; i++) {
      if (histogram[i] > 0) {
         ++left;
      }
   }

   left = peak - info->start;

   for (i = peak; i < info->end; i++) {
      if (histogram[i] > 0) {
         ++right;
      }
   }

   right = info->end - peak;
   left = (255-OFFSET)*left/(left+right);
```

```
   return(left);
}


/* This procedure returns the starting and end positions of the histogram */

struct hist_info *histogram_info(histogram)
int *histogram;
{
   register int i;
   int total = 0, cdf = 0, count = 0;
   struct hist_info *hi;

   hi = (struct hist_info *) malloc(sizeof(struct hist_info));

   for (i = 0; i < 256; i++)
      total += histogram[i];

   for (i = 0; i < 256; i++) {

      if (histogram[i] != 0)
         count++;

      if ((cdf == 0) && (histogram[i] > 0)) {
         hi->start = i;
      }

      cdf += histogram[i];

      if ((cdf == total) && (histogram[i] > 0))
         hi->end = i;
   }
   hi->no_levels = count;
   return(hi);
}

/* Spreads the bins on either side around the peak */

int *spread_peaks(histogram, params, skew, info)
int histogram[256], *skew;
struct read_in_parameters *params;
struct hist_info *info;
{
   int i, j, k, left_position, right_position, marker;
   int left_counter, right_counter;
   int *new_level, start, end, gap = 0, last, first, *first_levels();
   void last_levels();

   new_level = first_levels(histogram, info->start,
                            (int) params->pars[0], skew[0]);
```

```
for (j = 0; j < params->number - 1; j++) {

   left_counter = 0;
   right_counter = 0;
   marker = FALSE;
   gap = find_nl(histogram, (int)params->pars[j], (int)params->pars[j+1]);

   new_level[(int)params->pars[j]] = skew[j];
   left_position = skew[j];

   new_level[(int)params->pars[j+1]] = skew[j+1];
   right_position = skew[j+1];

   start = (int)params->pars[j] + 1;
   end = (int)params->pars[j+1] - 1;

   i = start;

   while (i < end + 1) {

      if (histogram[i] == 0)
         new_level[i] = left_position;

      else {

         if (left_counter < 4) {
            if (left_counter == 0)
               new_level[i] = new_level[i-1] + lut[new_level[i-1]]
                                             + INC + E;
            else {
               new_level[i]=new_level[i-1]+alut[skew[j]][new_level[i-1]];
               new_level[i]=new_level[i]+INC+maximum(0,E-1-(i-start));
            }
         } else {
            new_level[i] = new_level[i-1] + lut[new_level[i-1]];
            new_level[i] = new_level[i] + INC + maximum(0,E-1-(i-start));
         }

         ++left_counter;
         left_position = new_level[i];
         --gap;

         if (gap > right_position - left_position) {

            new_level[i] = right_position - gap;
            left_position = new_level[i];

            for (k = i + 1; k <= end - (i - start); k++) {
```

```
               if (histogram[k] == 0)
                  new_level[k] = left_position;

               else {
                  new_level[k] = new_level[k-1] + 1;
                  left_position = new_level[k];
               }
            }

            i = end + 1;
            marker = TRUE;
         }
      }

      if (marker == FALSE) {

         if (histogram[end - (i - start)] == 0)
            new_level[end - (i - start)] = right_position;

         else {

            if (right_counter < 4) {
               if (right_counter == 0) {
                  new_level[end-(i-start)]=new_level[end-(i-start)+1]
                                 - lut[new_level[end - (i - start) + 1]];
                  new_level[end-(i-start)] = new_level[end-(i-start)] -
                                                      INC - E;
               } else {
                  new_level[end-(i-start)]=new_level[end-(i-start)+1]
                        - alut[skew[j+1]][new_level[end-(i-start)+1]];
                  new_level[end-(i-start)] = new_level[end-(i-start)] -
                                             INC-maximum(0,E-1-(i-start));
               }
            } else {
               new_level[end-(i-start)]=new_level[end-(i-start)+1]
                                 - lut[new_level[end - (i - start) + 1]];
               new_level[end-(i-start)] = new_level[end-(i-start)] -
                                       INC - maximum(0,E-1-(i-start));
            }
            ++right_counter;
            right_position = new_level[end - (i - start)];
            --gap;
         }

         if (gap > right_position - left_position) {

            new_level[end - (i - start)] = left_position + gap;
            right_position = new_level[end - (i - start)];

            for (k = end - (i - start) - 1; k >= i; k--) {
                if (histogram[k] == 0)
                   new_level[k] = right_position;

                else  {
                   new_level[k] = new_level[k+1] - 1;
                   right_position = new_level[k];
                }

             }

             i = end + 1;
          }
       }
       i++;
    }
  }

  last_levels(histogram, new_level, info.end,
     (int)params->pars[params->number-1], skew[params->number-1]);

  return(new_level);
}

/* Spreads out the bins to the left of the first peak, providing
   there is enough space.
*/


int *first_levels(histogram, start, peak, map_peak)
int histogram[256], start, peak, map_peak;
{
   int end, *new_level, gap, i, position, counter = 0;

   new_level = (int*) malloc(256*(sizeof(int)));
   for (i = 0; i < 256; i++) new_level[i] = i;

   new_level[peak] = map_peak;
   position = map_peak;

   i = peak - 1;
   gap = find_nl(histogram, start, peak);
   end = start - 1;

   if (gap == 0) return(new_level);

   while (i >= start) {

      if (histogram[i] == 0)
         new_level[i] = position;
```

```
      else {

         if (counter < 4) {
            if (counter == 0)
               new_level[i] = new_level[i+1] - lut[new_level[i+1]] - INC - E;
            else {
               new_level[i]=new_level[i+1]-alut[map_peak][new_level[i+1]];
               new_level[i] = new_level[i]-INC-maximum(0, E-1-(peak-1-i));
            }
         } else {
            new_level[i] = new_level[i+1] - lut[new_level[i+1]];
            new_level[i] = new_level[i] - INC - maximum(0, E-1-(peak-1-i));
         }

         ++counter;
         position = new_level[i];
         --gap;

         if (new_level[i] - gap < OFFSET) {

            new_level[i] = gap + OFFSET;
            position = new_level[i];
            end = i - 1;
            i = start - 1;
         }
      }
      i--;
   }

   for (i = end; i >= start; i--) {

      if (histogram[i] == 0)
         new_level[i] = position;
      else {
         new_level[i] = new_level[i+1] - 1;
         position = new_level[i];
      }
   }
   return(new_level);
}

/* Spreads out the bins to the right of the last peak, providing
   there is enough space.
*/

last_levels(histogram, new_level, finish, peak, map_peak)
int histogram[256], *new_level, finish, peak, map_peak;
{
   int end, i, gap, position, counter = 0;
   position = map_peak;

   end = finish + 1;
   gap = find_nl(histogram, peak+1, finish);
   i = peak + 1;

   if (gap == 0) return;

   while (i <= finish) {

      if (histogram[i] == 0)
         new_level[i] = position;

      else {

         if (counter < 4) {
            if (counter == 0)
               new_level[i] = new_level[i-1] + lut[new_level[i-1]] + INC + E;
            else {
               new_level[i]=new_level[i-1]+alut[map_peak][new_level[i-1]];
               new_level[i]=new_level[i]+INC+maximum(0, E-1-(i-(peak+1)));
       }
         } else {
            new_level[i] = new_level[i-1] + lut[new_level[i-1]];
            new_level[i] = new_level[i] + INC + maximum(0, E-1-(i-(peak+1)));
         }
         ++counter;
         position = new_level[i];
         --gap;

         if (new_level[i] + gap > 255) {

            new_level[i] = 255 - gap - 1;
            position = new_level[i];
            end = i + 1;
            i = finish + 1;
         }
      }
      i++;
   }

   for (i = end; i <= finish; i++) {

      if (histogram[i] == 0)
         new_level[i] = position;

      else {
         new_level[i] = new_level[i-1] + 1;
         position = new_level[i];
```

```
         }
      }
}

/* Returns the position of the laregst bin in the [ histogram ]
   between [ start ] and [ end ].  */

find_largest_pos(histogram, start, end)
int *histogram, start, end;
{
   int i, largest = 0, position;

   for (i = start; i <= end; i++) {
      if (histogram[i] > largest) {
         largest = histogram[i];
         position = i;
      }
   }
   return(position);
}

/* Return the maximum value of a and b */

maximum(a, b)
int a, b;
{
   if (a >= b)
      return(a);
   else
      return(b);
}

/* Set_lut()

   Called : by [ simple_lut_stretch_with_parameter() ] above.

   Function : sets up the lut values for the monitor.
*/

static void set_lut()
{
   static void set_lut_gdm(), set_lut_hitachiLR(), set_lut_hitachiHM();

   switch(OFFSET) {

      case 10: set_lut_hitachiLR();
               break;

      case 60: set_lut_gdmA();
               break;
      case 50: set_lut_gdm();
               break;

      default: break;
   }
}

/*
   Set_lut_gdm()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the GDM-1604-40 monitor.

*/

static void set_lut_gdm()
{
   int i, j;

   for (i = 0; i < 40; i++)  lut[i] = 46 - i;
   for (i = 40; i < 45; i++) lut[i] = 9;
   for (i = 45; i < 50; i++) lut[i] = 5;
   for (i = 50; i < 55; i++) lut[i] = 4;
   for (i = 55; i < 65; i++) lut[i] = 3;
   for (i = 65; i < 85; i++) lut[i] = 2
   for (i = 85; i < 185; i++) lut[i] = 1
   for (i = 185; i < 255; i++) lut[i] = 2


    /* Ensures that for lut values less than 55 the transformation goes
       off the scale */

   for (i = 0; i < 55; i++)
      for (j = 0; j < i; j++)
          alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 55; i < 70; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 70; i < 210; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 2;

   for (i = 210; i < 255; i++)
```

```
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;
}

/*
   Set_lut_hitachiLR()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the Hitachi-LR47156 monitor.

*/

static void set_lut_hitachiLR()
{
   int i, j;

   for (i = 0; i < 5; i++)
      lut[i] = 6;
   for (i = 5; i < 10; i++)
      lut[i] = 5;
   for (i = 10; i < 15; i++)
      lut[i] = 4;
   for (i = 15; i < 30; i++)
      lut[i] = 3;
   for (i = 30; i < 240; i++)
      lut[i] = 2;
   for (i = 240; i < 255; i++)
      lut[i] = 3;

    /* Ensures that for lut values less than 15 the transformation goes
       off the scale */

   for (i = 0; i < 10; i++)
      for (j = 0; j < i; j++)
          alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 10; i < 30; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 4;

   for (i = 30; i < 75; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 75; i < 160; i++)
      for (j = 0; j < 256; j++)
```

```
        alut[i][j] = 2;

   for (i = 160; i < 256; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;
}

/*
   Set_lut_gdmA()

   Called : from [ set_lut() ] above.

   Function : sets the JND values for the GDM-1604A40 monitor.

*/

static void set_lut_gdmA()
{
   int i, j;

   for (i = 0; i < 50; i++)  lut[i] = 53-i;
   for (i = 50; i < 54; i++) lut[i] = 4;
   for (i = 55; i < 65; i++) lut[i] = 3;
   for (i = 65; i < 75; i++) lut[i] = 2;
   for (i = 75; i < 230; i++) lut[i] = 1;
   for (i = 230; i < 255; i++) lut[i] = 2;

    /* Ensures that for lut values less than 65 the transformation goes
       off the scale */

   for (i = 0; i < 60; i++)
      for (j = 0; j < i; j++)
         alut[i][j] = j + 1;

   /* Sets the whole scale, though only those close to the actual background
      are used */

   for (i = 60; i < 70; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 3;

   for (i = 70; i < 256; i++)
      for (j = 0; j < 256; j++)
         alut[i][j] = 2;

}
```

Appendix D

The code for "histogram equalisation"

eql code **eql.code**

```
/* @(#) Histogram_equalisation_algorithm */


/* This procedure carries out histogram equalisation */


#define OFFSET 50     /* Contains the offset of the monitor.  In this case
                         it is for the GDM-16404-40 */

extern int image_size   /* The image size in pixels: defined elsewhere */

extern int histogram_of_image[256] /* Contains the histogram of the image */


int *histogram_equalisation_proc()
{
   int i, *new_level;
   float total = 0, amount;

   /* Initialise new levels storing grey scale transformation */

   new_level = (int *) malloc(256 * (sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Calculate the transfromation: stretch according to height */
   /* Place the stretched bin in the middle of the range allocated to it */

   for (i = 0; i < 256; i++) {

      if (histogram_of_image[i] != 0) {

         amount = (float) (255 - OFFSET)*hist[i]/image_size;
         new_level[i] = (int) offset + total + amount/2.0;
         total += (float) amount;

      }
   }
   return(new_level);
}
```

Appendix E

The code for "smoothed equidistributing regrading"

```
/* @(#) Smoothed_regrading_algorithm */

/*

  This procedure implements the padding procedure in "Smoothed Histogram
  Modification for Image Processing", by Kautsky et al CVGIP 26, 271 - 291.
  The resulting histogram is a smoothed histogram lying between the extremes
  of a linear stretch (the identity if the ranges are the same, and the
  equidistributing transform).  The smoothing is governed by a parameter,
  lambda (0 <= lambda <= 0), giving the equidistributing regrading and
  linear respectively.  This value is stored in LAMBDA

*/

/* Each function and its purpose is listed below:

   find_peak()          finds the largest bin in the histogram between two
                        limits
   interpolate()        matches the cummulative distribution functions of the
                        input and equalised histogram to calculate the
                        transformation

*/

#define LAMBDA 0.5

extern int image_size;   /* Contains the size of the image: defined
                            elsewhere */
extern int histogram_of_image[256]  /* Contains the histogram of the image */


int *smoothed_regrading()
{
   float eqlz_cdf[257], input_cdf[257];
   int padded_input[256], padded_eqlz[256];
   int input_cdftotal = 0, *interpolate(), *levels, peak, i;
   int eqlz_cdftotal = 0, lam1, lam2, run_total1, run_total2;

   eqlz_cdf[0] = input_cdf[0] = 0.0;

   /* Initialise and calculate total number of pixels in the image */

   for (i = 0; i < 256; i++) {
      padded_histogram[i] = 0;
      padded_eqlz[i] = image_size/256;
   }

   /* Find the maximum value of the original histogram */

   peak = find_peak(histogram_of_image, 0, 255);
```

```
   lam1 = (float) LAMBDA*((histogram_of_image[peak]*LAMBDA)
                                        + (2*(1 - LAMBDA)*image_size)/256.0);

   lam2 = (float) LAMBDA*((LAMBDA*image_size/256.0)
                                        + (2*(1 - LAMBDA)*image_size)/256.0);

   /* Calculate the padded histogram, and the c.d.f. of the equidis his*/

   for (i = 0; i < 256; i++) {
      if (histogram_of_image[i] < lam1) {
         padded_input[i] = lam1;
      } else {
         padded_input[i] = temp_histogram[i];
      }
      if (padded_eqlz[i] < lam2) {
         padded_eqlz[i] = lam2;
      }
      input_cdftotal += padded_input[i];
      eqlz_cdftotal += padded_eqlz[i];
   }

   /*Calculate the c.d.f. of the padded histogram */

   run_total1 = run_total2 = 0;

   for (i = 0; i < 256; i++) {
      run_total1 += padded_input[i];
      run_total2 += padded_eqlz[i];
      input_cdf[i+1] = (float) run_total1/input_cdftotal;
      eqlz_cdf[i+1] = (float) run_total2/eqlz_cdftotal;
   }

   /* Calculate the transform using Kautsky p. 276 */

   levels = interpolate(input_cdf, eqlz_cdf);
}

/* Generates the transform values between the input and equidistributed
   histograms */

int *interpolate(original, reference)
float original[257], reference[257];
{
   register int i, k, l, m;
   int start, j = 1;
   float gradient, inter, temp = 0;
   int *new_level;
   struct hist_info *info;
```

smoothed regrading **smoothed_regrading.code**

```
/* Initialise the storage for the grey level transformation */

new_level = (int *) malloc(256*(sizeof(int)));

for (i = 0; i < 256; i++) {
   new_level[i] = i;
}

/* Get the histogram information (see the code for the "global lut
   stretch" */

info = (struct hist_info *) histogram_info(reference);

m = 0;

for (l = info->start; l < 257; l++) {

   /* Start from the point where the last search ended (cdf increasing) */

   i = m;

   /* Find the level at which the reference c.d.f exceeds the original */

   while (reference[l] > original[i]) {
      i++;
   }
      i--;
   m = i;

   /* Find the gradient between the levels in which the reference falls
      in the original */

   gradient = (float) original[i+1] - original[i];

   /* Find how far along the x axis the reference is */

   inter = (float) (reference[l] - original[i])/gradient;

   k = j;

   /* Find the values k that fall between the bounds */

   while (k - 0.5 <= inter + i && k - 0.5 > temp) {
      new_level[k-1] = l-1;
      k++;
   }

   /* Store upper bound as it is next iterations upper bound */

   temp = inter + i;
```

smoothed regrading **smoothed_regrading.code**

```
      /* Store the last level mapped so that the next iteration begins
         here */

      j = k;
   }

   return(new_level);
}
```

Appendix F

The code for “peak smoothing”

peak smoothing **peak_smoothing.code**

```
/* @(#) peak_smoothing */

/*

   This procedure carries out the peak smoothing algorithm.  There is one
   parameter ABOVE, a % of the pixel population in the largest bin.  All
   bins with a larger pixel population are stretched in proportion to their
   size above the parameter, while all bins below are preserved.

*/

/* Each function and its purpose is listed below:

   find_peak()         finds the largest bin in the histogram between two
                       limits
   a_b_info()          gets information about the histogram
                       (See the code for basic inverse equalisation)

*/

#define ABOVE 0.5      /* The parameter for the algorithm.  If a bin has a
                          pixel population larger than this it is stretched,
                          according to its height, otherwise it is preserved */

#define OFFSET 50      /* Contains the offset of the monitor.  In this case
                          it is for the GDM-16404-40 */


/* External declarations: these are defined elswewhere */

extern int histogram_of_image[256];   /* Histogram of the image */


int *peak_smoothing()
{
   int average, *new_level, total = 0, i, peak;
   float over, current_position = 0, stretch, map;
   struct above_and_below *infer, *a_b_info();

   /* Initialise new levels storing the grey scale transformation */

   new_level = (int*) malloc(256*(sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Find the position of the laregest bin in the histogram */

   peak = find_peak(histogram_of_image, 0, 255);
```

peak smoothing **peak_smoothing.code**

```
   map = (float) ABOVE*histogram_of_image[peak];
   average = (int) map;

   infer = (struct above_and_below *) a_b_info(average, 0, 0);
   total = 255 - OFFSET - infer->no_under_levels;

   /* If there is not enough space for stretrching */

   if (total < infer->no_over_levels) return(NULL);

   current_position = (float) OFFSET;
   i = 0;
   while (i < 256) {

      if (histogram_of_image[i] != 0) {

        / * Stretch peaks greater than average */

        if (histogram_of_image[i] >= average) {

           over = (float) (histogram_of_image[i]/average);
           stretch = (float) (over/infer->over_total)*total;
           stretch = (float) stretch/2.0;
           map = (float) current_position + stretch;
           new_level[i] = (int) map;
           current_position += stretch*2.0;

        } else {
           if (i != OFFSET) ++current_position;
           new_level[i] =  (int) current_position;
        }
      }
      i++;
   }

  free(infer);
  return(new_level);
}
```

Appendix G

The code for "inverse histogram equalisation" (and variants)

inverse code **inverse.code**

```
/* @(#) inverse_histogram_equalisation */

/*

   This procedure carries out inverse equalisation.  There is one parameter
   ABOVE, a % of the pixel population in the largest bin.  All peaks with
   a larger pixel population than ABOVE*largest_size are compressed, while
   the bins below are stretched in proportion to their size

*/

/* Each function and its purpose is listed below:

   find_peak()         finds the largest bin in the histogram between two
                 limits
   a_b_info()          gets information about the histogram

*/

/* Structure containing information about the histogram */

typedef struct above_and_below {
   int no_over_levels;
   int no_under_levels;
   int lowest_levels;
   float under_total;
   float over_total;
};

#define ABOVE 0.5      /* The parameter for the algorithm.  If a bin has a
                          pixel population larger than this it is compressed,
                          otherwise it is stretched according to its height
                       */

#define OFFSET 50      /* Contains the offset of the monitor.  In this case
                          it is for the GDM-16404-40 */


/* External declarations: these are defined elswewhere */

extern int histogram_of_image[256];   /* Histogram of the image */


int *st_inverse_proc()
{
   int *new_level, total = 0, flag = 0, i = 0, peak;
   float under, current_position, stretch, map;
   struct above_and_below *infer, *a_b_info();

   /* Initialise new levels storing the grey scale transformation */
```

inverse code **inverse.code**

```
   new_level = (int*) malloc(256*(sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Find the position of the laregest bin in the histogram */

   peak = find_peak(histogram_of_image, 0, 255);

   /* The parameter for the algorithm, calculated as a % of the largest bin:
      in this case 50 % of the largest bin.  Bins above this are compressed,
      bins below stretched according to their size */

   map = (float) ABOVE*histogram_of_image[peak];
   average = (int) map;   /* Integer conversion of the parameter */

   /* Find the number of bins that have a larger pixel population than the
      parameter, and the number of those that are smaller */

   infer = (struct above_and_below *) a_b_info(average, 0, 0);

   /* Space available for stretching in the histogram - each peak above the
      parameter is allocated one bin */

   total = 255 - OFFSET - infer->no_over_levels;

   current_position = (float) OFFSET;
   i = 0;
   while (i < 256) {

      if (histogram_of_image[i] != 0) {   /* Skip over empty bins */

         /* Compress peaks larger than "average" */

         while (histogram_of_image[i] >= average && i < 256) {

            if (flag == 0 && i != 0) ++current_position;

            flag = 1;
            new_level[i] = (int) current_position;
            ++i;
         }

         if (flag == 1) {
            --i;
            flag = 0;
         }
```

inverse code **inverse.code**

```
        /* Stretch bins less than "average" according to their size */

        if (histogram_of_image[i] < average) {

           under = (float) (histogram_of_image[i])/average;
           stretch = (float) (under/(infer->under_total))*total;
           stretch = (float) stretch/2.0;
           map = (float) current_position + stretch;
           new_level[i] =  map;
           current_position += stretch*2.0;
        }
     }
     i++;
  }

  free(infer);
  return(new_levels);
}

/* Calculate information about the histogram */

static struct above_and_below *a_b_info(average, low, flag)
int average, low, flag;
{
   int i = 0, count_over = 0;
   float over, under;
   struct above_and_below *infer;

   /* Allocate memory for the structure, and initilaise */

   infer = (struct above_and_below *) malloc(sizeof(struct above_and_below));
   infer->no_over_levels = infer->no_under_levels = 0;
   infer->lowest_levels = 0; infer->under_total = 0.0;

   while (i < 256) {

      /* Count peaks larger than "average" */

      while (i < 256 && histogram_of_image[i] >= average){

         over = (float) histogram_of_image[i]/average;
         infer->over_total += over;
         count_over++;
         i++;
      }
      if (count_over > 0) {
         --i;
         ++infer->no_over_levels;
         count_over = 0;
      }
```

inverse code **inverse.code**

```
      /* Calculate the how big each bin > low and < average is as a
         proportion of average - sum this */

      if (histogram_of_image[i] > low && temp_histogram[i] < average) {
         if (flag == 0)
            under = (float) histogram_of_image[i]/average;
         else
            under = (float) average/histogram_of_image[i];

         infer->under_total += under;

      /* Calculate the number of bins > low and < average */

         ++infer->no_under_levels;
      }

      /* Calculate the number of bins below low */

      if (histogram_of_image[i] != 0 && temp_histogram[i] <= low) {
         ++infer->lowest_levels;
      }
      ++i;
   }
   return(infer);
}

/* Find the peak in the histogram between start and start+count */

find_peak(histogram, start, count)
int *histogram, start, count;
{
   int i, peak, value;

   peak = histogram[start];
   value = start;

   for (i = start; i < (start + count); i++) {
      if (peak < histogram[i+1]) {
         peak = histogram[i+1];
         value = i+1;
      }
   }
   return(value);
}
```

```
/* @(#) smoothed_inverse_histogram_equalisation version 1 */

/*

   This procedure carries out smooothed inverse equalisation version 1.
   There are two parameters:

      ABOVE, a % of the pixel population in the largest bin.
      BELOW, also a % of the pixel population in the largest bin.

   All bins with a larger pixel population than ABOVE*largest_size are
   compressed, while all bins below BELOW*largest_size are retained.  The
   remaining bins are stretched in proportion to their size.

*/

/* Each function and its purpose is listed below:

   find_peak()         finds the largest bin in the histogram between two
                       limits
   a_b_info()          gets information about the histogram
                       (see code for basic inverse equalisation)

*/

#define ABOVE 0.7      /* If a bin has a larger pixel population than this it
                          is compressed */

#define BELOW 0.3      /* If a bin has a smaller pixel population than this it
                          is retained */

#define OFFSET 50      /* Contains the offset of the monitor.  In this case
                          it is for the GDM-16404-40 */

/* External declarations: these are defined elswewhere */

extern int histogram_of_image[256];   /* Histogram of the image */


int *smoothed_inverse_eql_v1()
{
   int average, smooth;
   int *new_level, total = 0, i, flag = 0, peak;
   float under, under_total = 0, current_position = 0, stretch, map;
   struct above_and_below *infer, *a_b_info();

   /* Initialise new levels storing the grey scale transformation */

   new_level = (int*) malloc(256*(sizeof(int)));
```

```
   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Find the position of the laregest bin in the histogram */

   peak = find_peak(histogram_of_image, 0, 255);

   /* The two parameters for the algorithm, calculated as a % of the largest
      bin: in this case 70 % and 30 % of the largest bin respectively.  */

   map = (float) ABOVE*histogram_of_image[peak];
   average = (int) map;

   map = (float) BELOW*histogram_of_image[peak];
   smooth = (int) map;

   /* Find the number of bins that have a larger pixel population than the
      parameter, and the number of those that are smaller.  See the code
      for basic inverse equalisation */

   infer = (struct above_and_below *) a_b_info(average, smooth, 0);

   /* Space available for stretching in the histogram - each peak above the
      parameter is allocated one bin, those below "smooth" are retained and
      at least for bins between "smooth" and "average" */

   total = 255 - OFFSET - infer->no_over_levels - infer->lowest_levels
            - infer->no_under_levels;

   /* It is possible that there is not enough room for stretching if the
      number of levels to be preserved is too large compared to the space
      available for stretching */

   if (total < 0) return(NULL);

   current_position = (float) OFFSET;
   i = 0;
   while (i < 256) {

      if (histogram_of_image[i] != 0) {

         /* Compress peaks larger than "average" */

         while (i < 256 && histogram_of_image[i] >= average) {

            if (flag == 0 && i != OFFSET) ++current_position;

            flag = 1;
```

inverse v1 code **inverse_v1.code**

```
/* @(#) smoothed_inverse_histogram_equalisation version 1 */

/*

   This procedure carries out smooothed inverse equalisation version 1.
   There are two parameters:

      ABOVE, a % of the pixel population in the largest bin.
      BELOW, also a % of the pixel population in the largest bin.

   All bins with a larger pixel population than ABOVE*largest_size are
   compressed, while all bins below BELOW*largest_size are retained.  The
   remaining bins are stretched in proportion to their size.

*/

/* Each function and its purpose is listed below:

   find_peak()         finds the largest bin in the histogram between two
                       limits
   a_b_info()          gets information about the histogram
                       (see code for basic inverse equalisation)

*/

#define ABOVE 0.7     /* If a bin has a larger pixel population than this it
                         is compressed */

#define BELOW 0.3     /* If a bin has a smaller pixel population than this it
                         is retained */

#define OFFSET 50     /* Contains the offset of the monitor.  In this case
                         it is for the GDM-16404-40 */

/* External declarations: these are defined elswewhere */

extern int histogram_of_image[256];   /* Histogram of the image */


int *smoothed_inverse_eql_v1()
{
   int average, smooth;
   int *new_level, total = 0, i, flag = 0, peak;
   float under, under_total = 0, current_position = 0, stretch, map;
   struct above_and_below *infer, *a_b_info();

   /* Initialise new levels storing the grey scale transformation */

   new_level = (int*) malloc(256*(sizeof(int)));
```

inverse v1 code **inverse_v1.code**

```
   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }

   /* Find the position of the laregest bin in the histogram */

   peak = find_peak(histogram_of_image, 0, 255);

   /* The two parameters for the algorithm, calculated as a % of the largest
      bin: in this case 70 % and 30 % of the largest bin respectively.  */

   map = (float) ABOVE*histogram_of_image[peak];
   average = (int) map;

   map = (float) BELOW*histogram_of_image[peak];
   smooth = (int) map;

   /* Find the number of bins that have a larger pixel population than the
      parameter, and the number of those that are smaller.  See the code
      for basic inverse equalisation */

   infer = (struct above_and_below *) a_b_info(average, smooth, 0);

   /* Space available for stretching in the histogram - each peak above the
      parameter is allocated one bin, those below "smooth" are retained and
      at least for bins between "smooth" and "average" */

   total = 255 - OFFSET - infer->no_over_levels - infer->lowest_levels
               - infer->no_under_levels;

   /* It is possible that there is not enough room for stretching if the
      number of levels to be preserved is too large compared to the space
      available for stretching */

   if (total < 0) return(NULL);

   current_position = (float) OFFSET;
   i = 0;
   while (i < 256) {

      if (histogram_of_image[i] != 0) {

         /* Compress peaks larger than "average" */

         while (i < 256 && histogram_of_image[i] >= average) {

            if (flag == 0 && i != OFFSET) ++current_position;

            flag = 1;
```

inverse v1 code **inverse_v1.code**

```
            new_level[i] = (int) current_position;
            ++i;
        }

        if (flag == 1) {
           --i;
           flag = 0;
        }

        /* Stretch bins between "average" and "smooth" according to size */

        if (histogram_of_image[i] < average && histogram_of_image[i] >
              smooth) {

           under = (float) (histogram_of_image[i])/average;
           stretch = (float) (under/infer->under_total)*total;
           stretch = (float) stretch/2.0;
           map = (float) current_position + stretch;
           new_level[i] =  (int) map;
           current_position += stretch*2.0;
        }

        /* Preserve the remaining bins */

        if (histogram_of_image[i] != 0 && histogram_of_image[i] <= smooth) {

           if (i != OFFSET) ++current_position;
           new_level[i] = (int) current_position;
        }
     }
     i++;
   }
   free(infer);
   return(new_level);
}
```

inverse v2 code **inverse_v2.code**

```
/* @(#) smoothed_inverse_histogram_equalisation version 2 */

/*

   This procedure carries out smooothed inverse equalisation version 2.
   There are two parameters:

      ABOVE, a % of the pixel population in the largest bin.
      BELOW, also a % of the pixel population in the largest bin.

   All bins with a larger pixel population than ABOVE*largest_size are
   compressed, while all bins below BELOW*largest_size are retained.  The
   remaining bins are stretched in inverse proportion to their size.

*/

/* Each function and its purpose is listed below:

   find_peak()         finds the largest bin in the histogram between two
                       limits
   a_b_info()          gets information about the histogram
                       (see code for basic inverse histogram equalisation)

*/

#define ABOVE 0.7     /* If a bin has a larger pixel population than this it
                         is compressed */

#define BELOW 0.3     /* If a bin has a smaller pixel population than this it
                         is retained */

#define OFFSET 50     /* Contains the offset of the monitor.  In this case
                         it is for the GDM-16404-40 */

/* External declarations: these are defined elswewhere */

extern int histogram_of_image[256];   /* Histogram of the image */


int *smoothed_inverse_eql_v2()
{
   int i, average, *new_level, total = 0, flag = 0, peak, smooth;
   float under, under_total = 0, current_position = 0, stretch, map;
   struct above_and_below *infer, *a_b_info();

   new_level = (int *) malloc(256 * (sizeof(int)));

   for (i = 0; i < 256; i++) {
      new_level[i] = i;
   }
```

inverse v2 code **inverse_v2.code**

```
   /* Find the position of the laregest bin in the histogram */

   peak = find_peak(histogram_of_image, 0, 255);

   /* The two parameters for the algorithm, calculated as a % of the largest
      bin: in this case 70 % an 30 % of the largest bin respectively.  */

   map = (float) ABOVE*histogram_of_image[peak];
   average = (int) map;

   map = (float) BELOW*histogram_of_image[peak];
   smooth = (int) map;

   /* Find the number of bins that have a larger pixel population than the
      parameter, and the number of those that are smaller.  See the code
      for basic inverse equalisation */

   infer = (struct above_and_below *) a_b_info(average, smooth, 1);

   total = 255 - OFFSET - infer->no_over_levels - infer->lowest_levels
           - infer->no_under_levels;

   /* It is possible that there is not enough room for stretching if the
      number of levels to be preserved is too large compared to the space
      available for stretching */

   if (total < 0) return(NULL);

   current_position = (float) OFFSET;
   i = 0;
   while (i < 256) {

      if (histogram_of_image[i] != 0) {

         /* Compress peaks larger than "average" */

         while (i < 256 && histogram_of_image[i] >= average) {

            if (flag == 0 && i != OFFSET) ++current_position;

            flag = 1;
            new_level[i] = (int) current_position;
            ++i;
         }

         if (flag == 1) {
            --i;
            flag = 0;
         }
```

inverse v2 code **inverse_v2.code**

```
        /* Stretch bins between "average" and "smooth" inversely according
           to size */

        if (histogram_of_image[i] < average && histogram_of_image[i] >
              smooth) {

           under = (float) average/histogram_of_image[i];
           stretch = (float) (under/infer->under_total)*total;
           ++stretch;
           stretch = (float) stretch/2.0;
           map = (float) current_position + stretch;
           new_level[i] =  (int) map;
           current_position += stretch*2.0;

        /* Preserve the remaining bins */

        } else if (histogram_of_image[i] <= smooth) {

           if (i != OFFSET) ++current_position;
           new_level[i] = (int) current_position;
        }
     }
     i++;
  }
  free(infer);
  free(new_level);
}
```

Appendix H

The code for "local histogram equalisation"

lhe code **lhe.code**

```
/* @(#) local_histogram_equalisation */

/*

   This procedure carries out local histogram equalisation

*/

/* Each function and its purpose is described below:

   create_image_buffer() = creates space for the storage of the processed
                           image.

   do_top() = local histogram equalisation of the top of the image.
   do_low() = local histogram equalisation of the bottom of the image.
   do_mid() = local histogram equalisation of the middle of the image.

   eql() = returns the value of the current pixel after transformation of
        the local histogram.

*/

typedef IMAGE **unsigned char;

#define WIDTH 256 /* The width of the image */
#define HEIGHT 256   /* The height of the image */

#define OFFSET 50 /* The offset for the monitor - here the GDM-1640-40 */

#define WINDOW_SIZE 65    /* The window size used: it must be odd */

/* External definitions: these are defined elsewhere */

extern IMAGE image;   /* Array containing the image pixels */


IMAGE local_histogram_equalisation()
{
   int *local_histogram, margin, i;
   IMAGE pro_image, create_image_buffer();

   pro_image = create_image_buffer(WIDTH, HEIGHT);
   local_histogram = (int *) malloc (256*(sizeof(int)));

   margin = WINDOW_SIZE/2;

   do_top(pro_image, margin, local_histogram);
   do_low(pro_image, margin, local_histogram);
   do_mid(pro_image, margin, local_histogram);
```

lhe code **lhe.code**

```
   free(local_histogram);
   return((IMAGE) pro_image);
}

/*
   Apply the histogram equalisation algorithm to the top part of the
   image, from i (y axis) = 0 .. margin-1, and j (x axis) = 0 .. WIDTH.
*/

static do_top(pro_image, margin, histogram)
int margin, *histogram;
IMAGE pro_image;
{
   int i, j, k, l, m;

   for (i = 0; i < margin; i++) {

      for (j = 0; j < WIDTH; j++) {

         if (j == 0) {
            for (m = 0; m < 256; m++)
               histogram[m] = 0;

            for (k = 0; k < WINDOW_SIZE; k++) {
               for (l = 0; l < WINDOW_SIZE; l++) {
                  histogram[image[l][k]]++;
               }
            }

         } else if (j > margin && j < WIDTH - margin) {
            for (m = 0; m < WINDOW_SIZE; m++) {
               histogram[image[j-margin-1][m]]--;
               histogram[image[j+margin][m]]++;
            }
         }
         pro_image[j][i] = (unsigned char) eql(histogram, WINDOW_SIZE,
                                                       image[j][i]);
      }
   }
}

/*
   Apply the histogram equalisation algorithm to the bottom part of the
   image i (y axis)  = HEIGHT - margin .. HEIGHT, j (x axis) = 0 .. WIDTH.
*/

static do_low(pro_image, margin, histogram)
int margin, *histogram;
IMAGE pro_image;
{
   int i, j, k, l, m;

   for (i = HEIGHT - margin; i < HEIGHT; i++) {

      for (j = 0; j < WIDTH; j++) {

         if (j == 0) {
            for (m = 0; m < 256; m++)
               histogram[m] = 0;

            for (k = 0; k < WINDOW_SIZE; k++) {
               for (l = 0; l < WINDOW_SIZE; l++) {
                  histogram[image[l][HEIGHT-k-1]]++;
               }
            }

         } else if (j > margin && j < WIDTH - margin) {

            for (m = 0; m < WINDOW_SIZE; m++) {
               histogram[image[j-margin-1][HEIGHT-m-1]]--;
               histogram[image[j+margin][HEIGHT-m-1]]++;
            }
         }
         pro_image[j][i] = (unsigned char) eql(histogram, WINDOW_SIZE,
                                               image[j][i]);
      }
   }
}

/*
   Apply the local histogram equalisation algorithm to the middle part of
   the image: i (y axis) = margin .. HEIGHT-margin-1, j = (x axis)
   0 .. WIDTH.
*/

static do_mid(pro_image, margin, histogram)
int margin, *histogram;
IMAGE pro_image;
{
   int i, j, k, l, m;

   for (i = margin; i < HEIGHT - margin; i++) {

      for (j = 0; j < WIDTH; j++) {

         if (j == 0) {
            for (m = 0; m < 256; m++)
               histogram[m] = 0;

            for (k = 0; k < WINDOW_SIZE; k++) {
```

```
               for (l = 0; l < WINDOW_SIZE; l++) {
                  histogram[image[l][i-margin+k]]++;
               }
            }

         } else if (j > margin && j < WIDTH - margin) {

            for (m = 0; m < WINDOW_SIZE; m++) {
               histogram[image[j-margin-1][i-margin+m]]--;
               histogram[image[j+margin][i-margin+m]]++;
            }
         }
         pro_image[j][i] = (unsigned char) eql(histogram, WINDOW_SIZE,
                                               image[j][i]);
      }
   }
}

/* Create the space for the processed image to be stored in */

static IMAGE Create_Image_Buffer(width, height)
int width, height;
{
   IMAGE image;
   int i;

   image = (unsigned char**) calloc(width, (sizeof(unsigned char*)));

   if (image != NULL) {

      for ( i = 0; i < width; i++ ) {
         image[i]=(unsigned char*)calloc(height,(sizeof(unsigned char)));

         if (image[i] == NULL) {

            fprintf(stderr, "\nUnable to allocate memory for image\n");
            exit(1);
         }
      }

   } else {

      fprintf(stderr, "\nUnable to allocate memory for image\n");
      exit(1);
   }

   return(image);
}

/* Finds the equalised value of the pixel */
```

lhe_code **lhe.code**

```
static eql(histogram, size, pixel_level)
int *histogram, size, pixel_level;
{
   int i;
   float total = 0, amount;

   for (i = 0; i < pixel_level; i++) {

      if (histogram[i] != 0) {
         amount = (float)  (255-OFFSET)*histogram[i]/size;
         total += (float) amount;
      }
   }

   amount = (float)  (255-margin)*histogram[pixel_level]/size;
   return((int) margin + total + amount/2.0);
}
```

Appendix I

The behaviour of points in models 1 and 2 (section 5.3.1)

In Chapter 5 the behaviour of two points in region A of models 1 and 2 are discussed. In this appendix the behaviour of the other points in region A of model 1 and model 2 is given. For both regions the initial window is of size M, i.e. $l > 1$.

I.1 Points in model 1

For Figure 5.4 when $l \approx> 2$ all points in region A of grey level μ are stretched by the amount given in equation 5.2, because the window, when placed at any point in the region, will cover the whole region. On the other hand, when $1 < l \leq 2$, the window may only partially cover region A, depending on the location of the pixel in the region.

In Figure 5.4 consider the case when $1 < l \leq 2$. Let (x, y) be the central point of the square. Consider points to the right of (x, y) i.e. (x', y') with

$$\begin{aligned} x' &= x + k \quad k = 1 \ldots \frac{M-1}{2} \\ y' &= y + m \quad m = 1 \ldots \frac{M-1}{2} \end{aligned}$$

The model is symmetric, so only points in the bottom right corner are considered. If an $lM \times lM$ window is placed over the area, only points where

$$\begin{aligned} x + k - \frac{lM-1}{2} &> x - \frac{M-1}{2} \\ \Rightarrow k &> \frac{M}{2}(l-1) \end{aligned}$$

or

$$\begin{aligned} y + m - \frac{lM-1}{2} &> y - \frac{M-1}{2} \\ \Rightarrow m &> \frac{M}{2}(l-1) \end{aligned}$$

or both are of interest. The points that satisfy these conditions are shown in regions I, J and K in Figure I.1. When the window is placed over such a point, it only partially covers region A. When the window is placed over the other points in the corner, it completely covers the region, and the stretch is given by equation 5.2

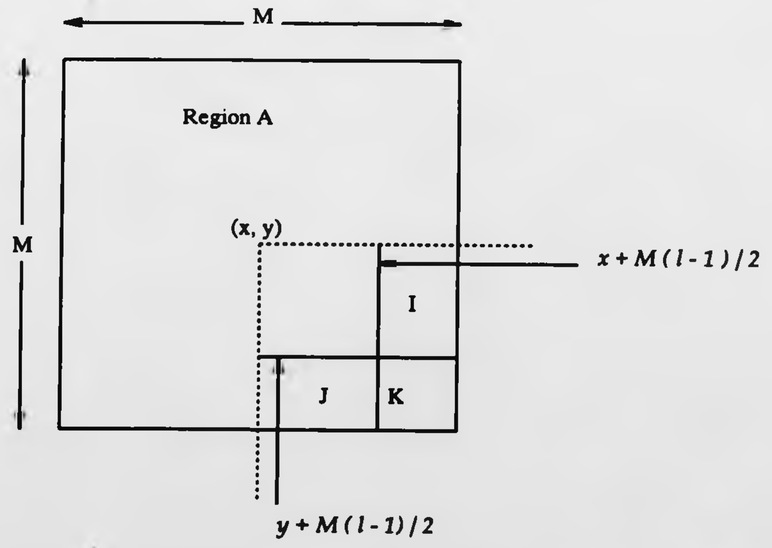


Figure I.1: Points of interest for an $lM \times lM$ window $1 < l \leq 2$

If the point (x', y') lies in region I in Figure I.1 then the window covers region A

in the vertical direction, but fails to cover some of the region along the horizontal axis (see Figure I.2 (a)).

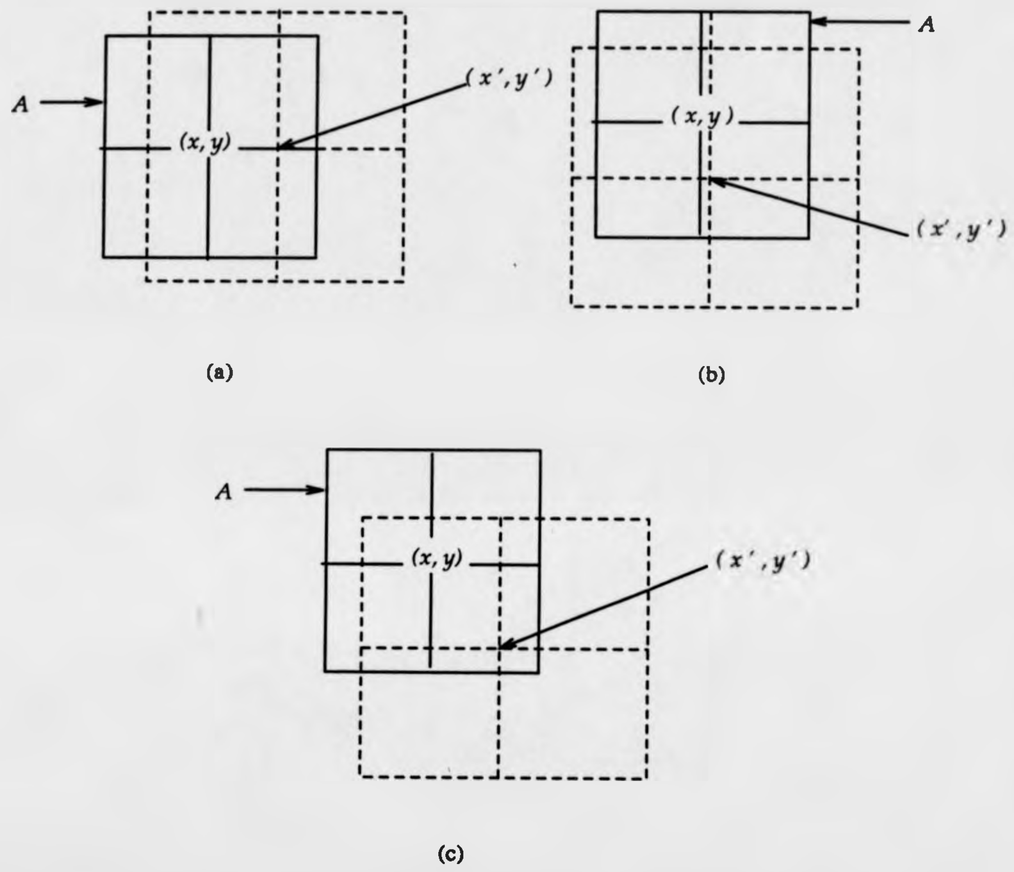


Figure I.2: Overlap along (a) the horizontal axis (b) the vertical axis and (c) on both axes

In this case the stretch given to the mode is (see Figure I.3) (a)

$$\left[\left(\frac{lM-1}{2}\right)+1+\frac{M-1}{2}-k\right]\frac{M}{l^2M^2}p_w(\mu)(K-1)$$
$$=-\left(\frac{p_w(\mu)(K-1)}{l^2M}\right)k+\frac{1}{l^2}(l+1)p_w\mu(K-1)$$

where $\frac{M}{2}(l-1) < k \leq \frac{M-1}{2}$. For fixed l this is a straight line of gradient $-\frac{p_w(\mu)(K-1)}{l^2M}$. Thus the stretching will decrease towards the righthand edge of region I. If (x, y) lies in region J in Figure I.1, then the window covers region

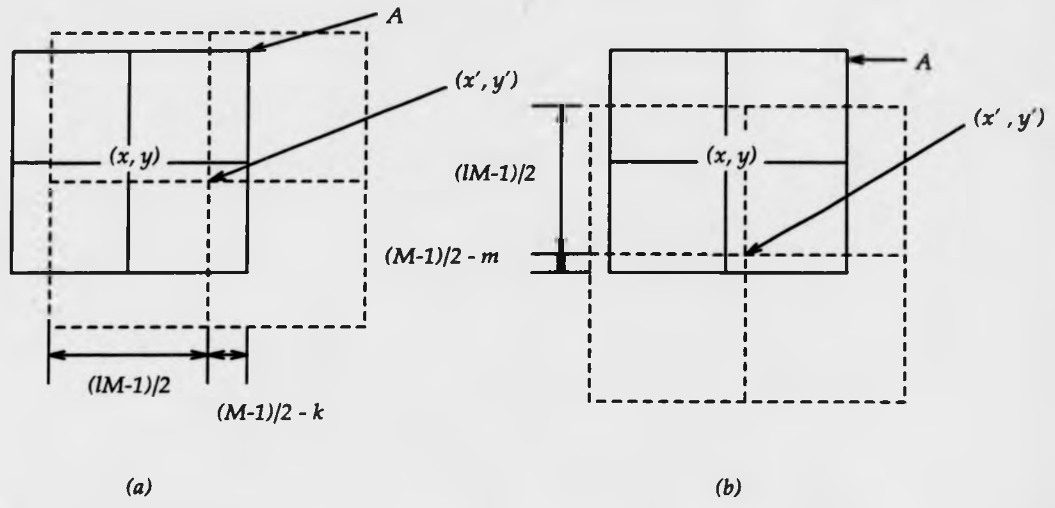


Figure I.3: (a) Calculating the mode size when the pixel is in region I, and (b) region J.

A in the horizontal direction, but fails to cover some of the region along the vertical axis (see Figure I.2 (b)). The stretch given to the mode is the same as equation I.1, with k replaced by m, i.e.

$$\left[\left(\frac{lM-1}{2}\right)+1+\frac{M-1}{2}-m\right]\frac{M}{l^2M^2}p_w(\mu)(K-1)$$
$$=-\left(\frac{p_w(\mu)(K-1)}{l^2M}\right)m+\frac{1}{l^2}(1+l)p_w(\mu)(K-1)$$

for $\frac{M}{2}(l-1) < m \leq \frac{M-1}{2}$

Now consider points (x, y) in region K in Figure I.1. The window fails to cover region A in both the horizontal and vertical directions (see Figure I.2 (c)). In this case the stretch is given by

$$\left[\frac{M}{2}(l+1)-k\right]\left[\frac{M}{2}(l+1)-m\right]\frac{p_\mu(K-1)}{l^2M^2}$$

where $\frac{M}{2}(l-1) < k \leq \frac{M-1}{2}$ and $\frac{M}{2}(l-1) < m \leq \frac{M-1}{2}$. If $C = \frac{p_\mu(K-1)}{l^2M^2}$ and

$D = \frac{M}{2}(l+1)$ this equation becomes

$$kmC - CD(k+m) + CD^2$$

This is the equation of a plane. For $p_w(\mu) = 0.071$, $k = 256$, $M = 25$ and $l = 1.5(lM = 37)$, $C = 0.013$, and $D = 39.1$, the plane is shown in Figure I.4.

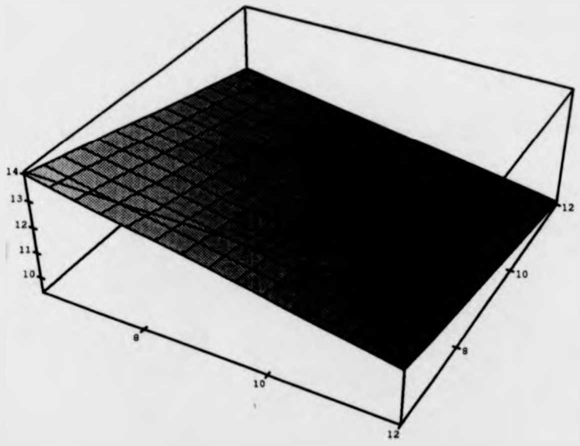


Figure I.4: Plane surface showing the variation in stretching for k and m

I.2 Points in model 2

For the model in Figure 5.5 consider the case where $1 < l \leq 2$. Let the point (x, y) be half way along the width of region A. Consider points $x + k$, $k = 1 \ldots \frac{M-1}{2}$, to the left of (x, y). The points of interest have

$$k > \frac{M}{2}(l-1)$$

These are the points where the $lM \times lM$ window no longer covers the region along the horizontal axis. In this case the stretching is given by

$$-\left(\frac{p_\mu(K-1)}{lM}\right)k + \frac{1}{2}\left(1+\frac{1}{l}\right)p_w(\mu)(K-1)$$

As before, the stretching decreases as the window moves to the right.

Appendix J

Increasing the window size (section 5.4.2)

In Chapter 5, section 5.4.2 describes how the window over each pixel is varied according to the region the pixel belongs to. The square window surrounding the pixel is increased in size until the appropriate degree of smoothness is obtained. Consider the case when the window size is already of dimension $w \times w$, and it is to be increased to dimension $w + 2 \times w + 2$. One of the nine cases in Figure J.1. Figure J.1 (i) shows the most common case: here it is possible to increase the

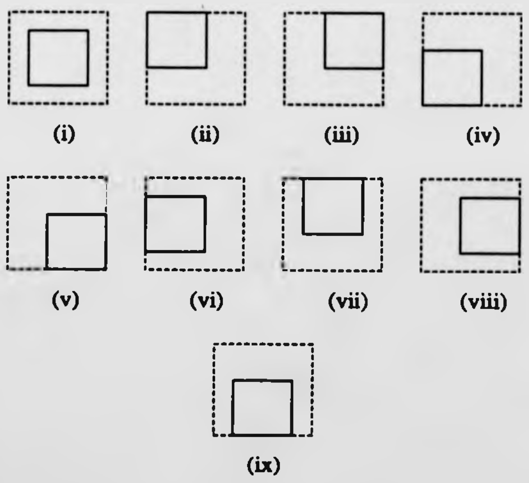


Figure J.1: Increasing the window size

window size without exceeding the boundaries of the image. Figure J.1 (ii) - (ix) show the cases where it is impossible to increase the window size without exceeding the window size in at least one direction. Figure J.1 (ii) shows the case when increasing the window size would result in exceeding the dimensions of the image both to the left and above. This is compensated for by adding extras pixels in the two remaining directions. Similarly for Figure J.1 (iii) - (ix).

Appendix K

The code for the "smoothed local histogram equalisation" (section 5.4) algorithm

```
/* @(#) smoothed_lhe.c */

/* This file contains an algorithm to do smoothed local histogram
   equalisation; adapting the window size to the region.

   It produces to output images - the processed image, and a grey image
   (called "outfile.grey"). which shows how the window size varies over
   the pixels.  The pixel is dark when the window size is large, and white
   when it is small.

   The expected usage is:

      smoothed_lhe [ infile ] [ outfile ] -m [0-3] -v [1-2]

   The input file is assumed to be a rasterfile.
   The output file is a rasterfile

   The -m option is to specify the monitor used: this in turn specifies
   the grey level separations used by the algorithm.  The options are:

   0: for GDM-1640-40
   1: for GDM-1604A40
   2: for Hitachi LR47156

   The -v option is used to specify which version of the smoothing algorithm
   to use.  These are:

   1: to smooth by using the average stretch given to the whole region
   2: to smooth using the mode and the LUT of values

   Compile using :

      cc -o slhe smoothed_lhe.c -lpixrect -lcurses -ltermcap -lm

   Functions used:

   print_use() : prints out error message
   create_image_buffer() : allocates memory for the image
   read_image() : reads the image into an array and calculates the histogram
   write_image() : writes the processed image as a rasterfile
   process() : iterates over the image and do the processing
   prelim_neighbour() : calculates the histogram of the
   find_added_pixels() : add pixels to the original image to smooth the regio
   find_lut_pixels() : as above except (version 2) using the LUT to determine
                       the smoothness
   find_peak() : find the largest bin in the histogram
   calculate_hist() : calculate the histogram in a square around the pixel
   histogram_info() : calculate where the histogram starts ends and how many
                      bins it contains
   calculate_histogram() : calculate the histogram of the intitial area aroun
```

```
                         a pixel
   initial_histograms() : according to the position of the pixel in the image
                          choose the initial region
   main_image() : if the histogram is not in the edge of the image find
                  the region it belongs to - the 3 by 3 window with the
                  smallest variance
   recalculate_min_hist() : if the initial region estimate is larger than 3
                            by 3 increase the chosen window in this direction
   calculate_mean_and_std() : calulate the mean and standard deviation of the
                              histogram
   calculate_added_hist() : add new pixels according to the windows position
   calculate_middle() : add pixels when the window can be expanded on all
                        sides
   calculate_lcor() : add pixels when the window is in the top left hand
                      corner of the image
   calculate_mtop() : add pixels when the window is in the top middle part
                      of the image
   calculate_mbottom() : add pixels when the window is in the bottom middle
                         part of the image
   calculate_mleft() : add pixels when the window is in the middle left of th
                       image
   calculate_mright() : add pixels when the window is in the middle right of
                        the image
   calculate_blcor() : add pixels when the window is in the bottom left hand
                       corner of the image
   calculate_brcor() : add pixels when the window is in the bottom right hand
                       corner of the image
   eql() : equalise the histogram and transform the pixel
   find_position() : find the postion the pixel is mapped to in the grey
                     level scale
   set_lut() : set the LUT to the values of the monitor
   set_gdm() : LUT values for the GDM-1604-40 monitor
   set_hitachiLR() : LUT values for the Hitachi LR monitor
   set_gdmA() : LUT values for the GDM-1604A40 monitor

*/

#include <stdio.h>
#include <malloc.h>
#include <curses.h>
#include <math.h>
#include <sys/time.h>
#include <rasterfile.h>
#include <pixrect/pixrect_hs.h>

#define R_SIZE 2      /* Region size */

/* Structure containing histogram statisics */

typedef struct stats {
   int mean;          /* Histogram mean */
   float stdv;        /* Histogram standard deviation */
};

/* Structure containing histogram information */

typedef struct hinf {
   int start;        /* Where the histogram starts */
   int end;          /* Where the histogram ends */
   int no_levels;    /* The number of occupied grey levels */
};

/* 2-D array for storing the images */

typedef unsigned char **IMAGE;


int kit, kat;    /* Positions to move the pixel to in order for the window
                    selected to be centered */
int offset;     /* Monitor offset */
int region_size;   /* Size of region selected */

/* Histogram for the image, and LUT of values for the monitor */

int histogram_of_image[256], lut[256];

float rs = 3.0;   /* The value to smooth the image to */

/* Various raster functions for SUNVIEW */

extern unsigned char red[256], green[256], blue[256];
unsigned char red[],green[],blue[];
colormap_t colormap = {RMT_EQUAL_RGB, 256, red, green, blue};

main(argc, argv)
int argc;
char *argv[];
{
   FILE *input, *output, *grey_output;
   IMAGE image, pro_image, grey_image, create_image_buffer();
   struct rasterfile rh;
   void read_image(), write_image(), process(), print_use();
   int i, monitor = 0, version = 1;
   char buf[15];

   if (argc < 3)      /* Need at least three arguments */
      print_use();

   for (i = argc-1; i > 0; --i) { /* Check for options -m: default 0 (GDM) */
      if (argv[i][0] == '-') {
```

```
         if (argv[i][1] == 'm') {
            if (i+1 > argc-1) {
               print_use();
            } else {
               monitor = atoi(argv[i+1]);
               if (monitor > 3 | monitor < 0) print_use();
            }
         }
         if (argv[i][1] == 'v') {   /* Check for -v option : default 1 */
            if (i+1 > argc-1) {
               print_use();
            } else {
               version = atoi(argv[i+1]);
               if (version > 2 | version < 1) print_use();
            }
         }
      }
   }

   if ((input = fopen(argv[1], "rb")) == NULL) {      /* Open input file */

      printf("\n Error opening input file %s\n", argv[1]);
      print_use();
   }

   if ((output = fopen(argv[2], "w")) == NULL) {     /* Open output file */

      printf("\n Error opening output file %s\n", argv[2]);
      print_use();
   }

   if (pr_load_header(input, &rh)) {     /* Load rasterfile header */

      printf("\n Error reading header: file must be a raster image\n");
      print_use();
   }

   if (pr_load_colormap(input, rh, colormap)) {    /* Load colormap */

      printf("\n Error reading header\n");
      print_use();
   }

   sprintf(buf, "%s%s", argv[2], ".grey");  /* To store the grey image */

   if ((grey_output = fopen(buf, "w")) == NULL) {

      printf("\n Error opening grey output file %s\n", buf);
      exit(1);
   }
```

smoothed lhe **smoothed_lhe.c**

```
   /* Create space for image and processed image */

   image = (IMAGE) create_image_buffer(rh.ras_width, rh.ras_height);
   pro_image = (IMAGE) create_image_buffer(rh.ras_width, rh.ras_height);
   grey_image = (IMAGE) create_image_buffer(rh.ras_width, rh.ras_height);

   /* Read in the image */

   read_image(image, input, rh.ras_width, rh.ras_height);

   fclose(input);
   set_lut(monitor);   /* Set up the LUT for the monitor */

   initscr();
   clear();

   process(image, pro_image, grey_image, rh.ras_width, rh.ras_height, version);

   write_image(pro_image, rh, output);
   write_image(grey_image, rh, grey_output);

   printf("\n\n   Finished Pocessing.\n\n");
}

/* Usage message */

void print_use()
{
   printf("\n Usage: lut [ input file ] [ output file ] -m [ monitor ] ");
   printf(" -v [ version ]\n");
   printf("\n -m 0 for GDM-1604-40");
   printf("\n    1 for GDM-1604A40");
   printf("\n    2 for Hitachi LR47156\n\n");
   printf("\n -v 1 for average stretch of the region");
   printf("\n    2 for mode stretch according to the look up table\n\n");
   exit(1);
}

/* Create space for the image */

IMAGE create_image_buffer(width, height)
int width, height;
{
   IMAGE image;
   register int i;

   image = (unsigned char**) calloc(width, (sizeof(unsigned char*)));

   if (image != NULL) {
```

smoothed lhe **smoothed_lhe.c**

```
      for (i = 0; i < width; i++) {

         image[i] = (unsigned char*) calloc(height,(sizeof(unsigned char)));

         if (image[i] == NULL) {

            printf("\n Unable to allocate memory for image\n");
            exit(1);
         }
      }
   } else {

      printf("\n Unable to allocate memory for image\n");
      exit(1);
   }

   return(image);
}

/* Read in the image from the file */

void read_image(image, fp, width, height)
IMAGE image;
FILE *fp;
int width, height;
{
   register int x, y, i;

   for (i = 0; i < 256; i++)      /* Initialise histogram of the image */
      histogram_of_image[i] = 0;

   for (y = 0; y < height; y++) {

      for (x = 0; x < width; x++) {

         image[x][y] = getc(fp);
         ++histogram_of_image[image[x][y]];    /* Calculate the histogram */
      }
   }
}

/* Write the image to a file */

void write_image(pro_image, rh, output)
IMAGE pro_image;
struct rasterfile rh;
FILE *output;
{
   int i, j;
   fwrite(&rh, sizeof(struct rasterfile), 1, output);

   fwrite(red, sizeof(unsigned char), 256, output);
   fwrite(green, sizeof(unsigned char), 256, output);
   fwrite(blue, sizeof(unsigned char), 256, output);

   for (i = 0; i < rh.ras_height; i++)
      for (j = 0; j < rh.ras_width; j++)
         fputc(pro_image[j][i], output);

   fclose(output);
}

/* Process the image */

void process(image, pro_image, grey_image, width, height, version)
IMAGE grey_image, pro_image, image;
int width, height, version;
{
   int i, j, k, *new_levels, *histogram, *find_lut(),total,*prelim_neighbour();
   struct stats *hs, *calculate_mean_and_std();
   int stotal;

   move(1, 3); printw("Starting processing ...");
   move(3, 3); printw("Processing row:      column:   ");
   refresh();

   for (i = 0; i < 256; i++) {

      move(3, 19); printw("%d  ", i);

      for (j = 0; j < 256; j++) {

         if (j % 2 == 0) {
            move(3, 32); printw("%d  ", j);
            refresh();
         }

         /* Calculate region histogram */

         histogram  = prelim_neighbour(i, j, image, width, version);

         /* Calculate the grey image: small windows white, large black */

         grey_image[j][i]=(int)offset+((255-region_size)*(255.0-offset))/255.0;

         /* Number of pixels used */

         total = region_size*region_size;
```

```
         /* Equalise */

         pro_image[j][i] = eql(histogram, total, image[j][i]);

         free(histogram);
      }
   }
}

/* Calculate the region histogram */

int *prelim_neighbour(i, j, image, width, version)
int i, j, width, version;
IMAGE image;
{
   int *min_hist, size = 1, total = 0, diff, left_sd, right_sd;
   int inc = 1, *find_region_size(), *calculate_histogram();
   struct hinf *histogram_info(), *info;
   struct stats *hs;

   /* Calculate the initial histogram:  compare the neighbourhoods if
      the pixel is not in a corner */

   min_hist = calculate_histogram(size, i, j, image, width);

   total = ((2*R_SIZE)+1)*((2*R_SIZE)+1);

   /* Calculate the mean and standard deviation of the chosen histogram */

   hs = (struct stats *) calculate_mean_and_std(min_hist, total);

   /* Calculate the limits (one standard deviation from the mean */

   left_sd = hs->mean - (int) hs->stdv;
   right_sd = hs->mean + (int) hs->stdv;

   /* Calculate where the region histogram starts and ends, and find out how
      many bins are occupied */

   info = (struct hinf *) histogram_info(min_hist);

   /* Reset the bounds of the histogram to one standard deviation on either
      side of the mean */

   info->start = left_sd; info->end = right_sd;

   /* Calculate the how many bins lie within the range of the sd */

   diff = info->end - info->start;
   /* If the range is less than five modify it to equal five */

   if (diff < 6) {
      info->start = (info->start + info->end)/2 - 2;
      info->end = (info->start + info->end)/2 + 2;
   }

   if (version == 1)
      find_added_pixels(min_hist, i, j, image, total, info, width);
   else
      find_lut_pixels(min_hist, i, j, image, total, info, width);

   free(hs);
   return(min_hist);
}

/* Add pixels until the region is smoothed: version 1 */

find_added_pixels(min_hist, i, j, image, total, info, length)
int *min_hist, i, j, total, length;
struct hinf *info;
IMAGE image;
{
   int k, l, peak, inc = 1;
   int *added_hist, win_size, occupied;
   float min, total_stretch, stretch;

   /* Initial check using the mode */

   peak = (int) find_peak(min_hist, info->start, info->end);
   min = (float) (min_hist[peak]*(255.0-offset))/total;

   if (min < rs) {    /* Already smooth enough with this window */
      free(info);
      region_size = 2*R_SIZE+1;
      return;
   }

   /* Increase the window size */

   for (k = R_SIZE + inc; k < length/2; k += inc) {

      win_size = (2*k+1)*(2*k+1);

      calculate_added_hist(i,j,k,k-inc,inc, image, length, min_hist);

      total_stretch = 0.0;
      occupied = 0;
```

```
      /* Average smooth of new region */

      for (l = info->start; l < info->end+1; l++) {

         stretch = (float) ((255.0 - offset)*min_hist[l])/win_size;
         total_stretch += (float) stretch;

         if (min_hist[l] != 0) ++occupied;
      }

      stretch = (float) total_stretch/occupied;

      if (stretch < rs && min > rs) {
         free(info);
         region_size = 2*k + 1;
         return;
      }

      min = (float) stretch;
   }

   region_size = length-1;
   free(info);
}

/* Smoothe the region using version 2 */

find_lut_pixels(min_hist, i, j, image, total, info, length)
int *min_hist, i, j, total, length;
struct hinf *info;
IMAGE image;
{
   int k, l, peak, inc = 1, position, largest;
   int *added_hist, win_size, point;
   float rs, min, stretch;

   peak = (int) find_peak(min_hist, info->start, info->end);
   min = (float) (min_hist[peak]*(255.0-offset))/total;

   position = find_position(peak, min_hist, total);
   rs = (float) lut[position];

   if (min < rs) {
      free(info);
      region_size = 2*R_SIZE+1;
      return;
   }

   for (k = R_SIZE + inc; k < length/2; k += inc) {
      win_size = (2*k+1)*(2*k+1);

      calculate_added_hist(i,j,k,k-inc,inc, image, length, min_hist);

      largest = 0;

      /* Find biggest bin */

      for (l = info->start; l < info->end+1; l++) {

         if (min_hist[l] > largest) {
            largest = min_hist[l];
            point = l;
         }
      }

      stretch = (float) ((255.0 - offset)*min_hist[point])/win_size;

      position = find_position(point, min_hist, win_size);
      rs = (float) lut[position];

      /* Check if the stretch is what it should be */

      if (stretch < rs) {
         region_size = 2*k+1;
         free(info);
         return;
      }
   }

   region_size = length - 1;
   free(info);
}

/* Find the largest bin */

int find_peak(hist, start, end)
int *hist, start, end;
{
   int i, largest = 0, pos;

   for (i = start; i <= end; i++) {
      if (hist[i] > largest) {
         largest = hist[i];
         pos = i;
      }
   }
   return(pos);
}
```

```
/* Calculate the histogram of a window of dimension (2*size+1)*(2*size+1)
   around the central pixel */

int *calculate_hist(size, j, i, image, width)
int size, j, i, width;
IMAGE image;
{
   int m, k, l, *hist, left, right, up, down;

   left = right = up = down = size;

   /* Redefine the window if it exceeds the bounds of the image */

   if (j - left < 0) left = j;
   if (j + right > width-1) left += right - (width - 1 - j);
   if (i - up < 0) up = i;
   if (i + down > width - 1) up += down - (width - 1 - i);

   hist = (int *) malloc (256*(sizeof(int)));
   for (m = 0; m < 256; m++) hist[m] = 0;

   for (k = 0; k < 2*size+1; k++) {
      for (l = 0; l < 2*size+1; l++) {
         hist[image[j-left+l][i-up+k]]++;
      }
   }

   return(hist);
}

/* Calculate where the histogram begins and ends, and how many occupied grey
   levels there are */

struct hinf *histogram_info(histogram)
int *histogram;
{
   int total = 0, cdf = 0, count = 0, i;
   struct hinf *info;

   /* Structure containing the information */

   info = (struct hinf *) malloc(sizeof(struct hinf));

   /* Total number of pixels in the histogram */

   for (i = 0; i < 256; i++)
      if (histogram[i] != 0)
         total += histogram[i];

   for (i = 0; i < 256; i++) {
      if (histogram[i] != 0)   /* Count the number of occupied bins */
         count++;

      if (cdf == 0 && histogram[i] != 0)   /* Histogram starts */
         info->start = i;

      cdf += histogram[i];   /* Calculates the cdf */

      if (cdf == total && histogram[i] > 0)   /* Histogram ends */
         info->end = i;
   }

   info->no_levels = count;
   return(info);
}

/* Calculates the histogram of the region */

int *calculate_histogram(size, i, j, image, width)
int size, i, j, width;
IMAGE image;
{
   int *initial_histograms(), total, *min_hist;

   /* Total number of pixels in initial region */
   total = (2*size+1)*(2*size+1);

   /* Calculate the histogram of the area with lowest variance */

   min_hist = (int *) initial_histograms(image, i, j, size, total, width);
   return(min_hist);
}

/* Calculate the histogram of the area with lowest variance */

int *initial_histograms(image, i, j, size, total, width)
IMAGE image;
int i, j, size, total, width;
{
   int *min_hist, *main_image();

   /* Calculate histograms for around the image i.e. when it is impossible
      to put a square window over the pixel without overshooting the image.
      This means the variance cannot be obtained for the surrounding regions
      so the region is assumed to be the one calculated.

      Move the pixel to the centre of the square placed over this point */

   if (i < 2*size + 1 && j < 2*size + 1) {
```

```
      min_hist = calculate_hist(R_SIZE, j+R_SIZE, i+R_SIZE, image, width);
      kit = j + size; kat = i + size;
   } else if (j > width -1 - 2*size - 1 && i < 2*size + 1) {
      min_hist = calculate_hist(R_SIZE, j-R_SIZE, i+R_SIZE, image, width);
      kit = j - size; kat = i + size;
   } else if (i > width -1 - 2*size - 1 && j < 2*size+1) {
      min_hist = calculate_hist(R_SIZE, j+R_SIZE, i-R_SIZE, image, width);
      kit = j+size; kat = i-size;
   } else if (i > width -1 - 2*size - 1 && j > width -1 - 2*size-1) {
      min_hist = calculate_hist(R_SIZE, j-R_SIZE, i-R_SIZE, image, width);
      kit = j-size; kat = i-size;
   } else if (i < 2*size+1) {
      min_hist = calculate_hist(R_SIZE, j, i+R_SIZE, image, width);
      kit = j; kat = i + size;
   } else if (i > width -1-2*size-1) {
      min_hist = calculate_hist(R_SIZE, j, i-R_SIZE, image, width);
      kit = j; kat = i - size;
   } else if (j < 2*size+1) {
      min_hist = calculate_hist(R_SIZE, j+R_SIZE, i, image, width);
      kit = j+size; kat = i;
   } else if (j > width -1-2*size-1) {
      min_hist = calculate_hist(R_SIZE, j-R_SIZE, i, image, width);
      kit = j-size; kat = i;
   } else {

      /* If the pixel is not in a corner make a comparison of regions
         surrounding it to find the region with the lowest variance */

      min_hist = (int *) main_image(image, i, j, size, total, width);
      return(min_hist);
   }

   return(min_hist);
}

/* Compare the varians in the neighbourhoods surrounding the pixel */

int *main_image(image, i, j, size, total, width)
IMAGE image;
int i, j, size, total, width;
{
   int k, l, *recalculate_min_hist(), value, smallest, *min_hist;
   int *var, **hist;
   float sig, sigma;
   struct stats *hs, *calculate_mean_and_std();

   var = (int *) malloc(9*sizeof(int));
   hist = (int **) malloc(9*(sizeof(int *)));

   /*for (k = 0; k < 9; k++)
```

```
      hist[k] = (int *) malloc(sizeof(int));
   */

   /* Calculate the histograms of the nine masks surrounding the pixel */

   hist[0] = calculate_hist(size, j-size, i-size, image, width);
   hist[1] = calculate_hist(size, j+size, i+size, image, width);
   hist[2] = calculate_hist(size, j-size, i+size, image, width);
   hist[3] = calculate_hist(size, j+size, i-size, image, width);
   hist[4] = calculate_hist(size, j+size, i, image, width);
   hist[5] = calculate_hist(size, j, i+size, image, width);
   hist[6] = calculate_hist(size, j-size, i, image, width);
   hist[7] = calculate_hist(size, j, i-size, image, width);
   hist[8] = calculate_hist(size, j, i, image, width);

   /* Calculate the variance of each distribution */

   for (k = 0; k < 9; k++) {
      sigma = sig = 0;
      for (l = 0; l < 256; l++) {
         if (hist[k][l] > 0) {
            sigma += (float) hist[k][l]*l*l;
            sig += (float) hist[k][l]*l;
         }
      }
      sig = sig*sig;
      var[k] = (int) (sigma - (sig/total))/(total-1);
   }

   /* Find the region with the smallest variance */

   value = 0;
   smallest = var[0];
   for (k = 1; k < 9; k++) {
      if (var[k] < smallest) {
         smallest = var[k];
         value = k;
      }
   }

   /* Calculate the histogram of the region chosen, but increase its
      size to R_SIZE (stored above: usually 5 by 5 region) */

   min_hist = (int *) recalculate_min_hist(image, i, j, size, value, width);
   for (k = 0; k < 9; k++) free(hist[k]);
   free(hist); free(var);
   return(min_hist);
}

/* Calculate the appropriate histogram and transfer the pixel to the
```

```
   centre of the region */

int *recalculate_min_hist(image, i, j, size, value, width)
IMAGE image;
int i, j, size, value, width;
{
   int *min_hist;

   switch(value) {

      case 0:
               kit = j-size;
               kat = i-size;
               min_hist = calculate_hist(R_SIZE,  j-R_SIZE,i-R_SIZE,image,width);
               break;
      case 1:
               kit = j+size;
               kat = i+size;
               min_hist = calculate_hist(R_SIZE,  j+R_SIZE,i+R_SIZE,image,width);
               break;
      case 2:
               kit = j-size;
               kat = i+size;
               min_hist = calculate_hist(R_SIZE,  j-R_SIZE,i+R_SIZE,image,width);
               break;
      case 3:
               kit = j+size;
               kat = i-size;
               min_hist = calculate_hist(R_SIZE,  j+R_SIZE,i-R_SIZE,image,width);
               break;
      case 4:
               kit = j+size;
               kat = i;
               min_hist = calculate_hist(R_SIZE,  j+R_SIZE,  i,image,width);
               break;
      case 5:
               kit = j;
               kat = i+size;
               min_hist = calculate_hist(R_SIZE,  j,  i+R_SIZE,image,width);
               break;
      case 6:
               kit = j-size;
               kat = i;
               min_hist = calculate_hist(R_SIZE,  j-R_SIZE,  i,  image,width);
               break;
      case 7:
               kit = j;
               kat = i-size;
               min_hist = calculate_hist(R_SIZE,  j,  i-R_SIZE,  image,width);
               break;
      case 8:
            kit = j;
            kat = i;
            min_hist = calculate_hist(R_SIZE, j, i, image, width);
            break;
      default:
            break;
   }
   return(min_hist);
}

/* Calculates the standard deviation and mean the histogram */

struct stats *calculate_mean_and_std(histogram, size)
int *histogram, size;
{
   int i;
   float sigma, sig, variant;
   struct stats *hs;

   hs = (struct stats *) malloc (sizeof(struct stats));

   sigma = sig = 0;

   for (i = 0; i < 256; i++) {
      if (histogram[i] != 0) {
         sigma += (float) histogram[i]*i*i;
         sig += (float) histogram[i]*i;
      }
   }

   hs->mean = sig/size;
   sig = (float) sig*sig;

   variant = (float) (sigma - (sig/size))/(size-1);

   hs->stdv = (float) sqrt((double)variant);
   return(hs);
}

/* Add pixels according to the window position */

calculate_added_hist(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   if (i - size < 0 && j - size < 0) {
      calculate_lcor(i, j, size, margin, inc, image, min_hist);

   } else if (i - size < 0 && j + size > length - 1) {
```

```
      calculate_rcor(i, j, size, margin, inc, image, length, min_hist);

   } else if (i + size > length - 1 && j - size < 0) {
      calculate_blcor(i, j, size, margin, inc, image,length, min_hist);

   } else if (i + size > length - 1 && j + size > length - 1) {
      calculate_brcor(i, j, size, margin, inc, image, length, min_hist);

   } else if (i - size < 0) {
      calculate_mtop(i, j, size, margin, inc, image, length, min_hist);

   } else  if (i + size > length - 1) {
      calculate_mbottom(i, j, size, margin, inc, image,length, min_hist);

   } else if (j - size < 0) {
      calculate_mleft(i, j, size, margin, inc, image, length, min_hist);

   } else if (j + size > length - 1) {
      calculate_mright(i, j, size, margin, inc, image, length, min_hist);

   } else {
      calculate_middle(i, j, size, margin, inc, image, length, min_hist);
   }
}

/* Add pixels when the window does not go off the edge of the image */

calculate_middle(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = i - size; k < i - margin; k++) {
      for (l = j - size; l <= j + size; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i + margin + 1; k <= i + size; k++) {
      for (l = j - size; l <= j + size; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i - margin; k <= i + margin; k++) {
      for (l = j - size; l < j - margin; l++) {
         ++min_hist[image[l][k]];
      }
   }
   for (k = i - margin; k <= i + margin; k++) {
      for (l = j + margin + 1; l <= j + size; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the top left hand corner */

calculate_lcor(i, j, size, margin, inc, image, min_hist)
int i, j, size, margin, inc, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = 0; k < 2*margin+1+2*inc; k++) {
      for (l = 2*margin + 1; l < 2*margin+1+2*inc; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = 2*margin + 1; k < 2*margin+1+2*inc; k++) {
      for (l = 0; l < 2*margin+1; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the middle top */

calculate_mtop(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = 0; k < 2*margin+1+2*inc; k++) {
      for (l = j - size; l < j - margin; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = 0; k < 2*margin+1+2*inc; k++) {
      for (l = j + margin + 1; l <= j + size; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = 2*margin + 1; k < 2*margin+1+2*inc; k++) {
```

```
      for (l = j - margin; l < j + margin+1; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the middle bottom */

calculate_mbottom(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = length-(2*margin+1)-2*inc; k < length; k++) {
      for (l = j - size; l < j - margin; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = length-(2*margin+1)-2*inc; k < length; k++) {
      for (l = j + margin + 1; l <= j + size; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = length-(2*margin+1)-2*inc; k < length-(2*margin+1); k++) {
      for (l = j - margin; l < j + margin+1; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the middle left */

calculate_mleft(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = i - size; k < i - margin; k++) {
      for (l = 0; l < 2*margin+1+2*inc; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i + margin + 1; k <= i + size; k++) {
      for (l = 0; l < 2*margin+1+2*inc; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i - margin; k < i + margin+1; k++) {
      for (l = 2*margin+1; l < 2*margin+1+2*inc; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the middle right */

calculate_mright(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = i - size; k < i - margin; k++) {
      for (l = length-(2*margin+1)-2*inc; l < length; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i + margin + 1; k <= i + size ; k++) {
      for (l = length-(2*margin+1)-2*inc; l < length; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = i - margin; k < i + margin+1; k++) {
      for (l = length-(2*margin+1)-2*inc; l < length-(2*margin+1); l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the top right hand corner */

calculate_rcor(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = 0; k < 2*margin+1+2*inc; k++) {
      for (l = length-1-(2*margin+2*inc); l <= length-1-(2*margin+1); l++) {
         ++min_hist[image[l][k]];
      }
   }
```

```
   for (k = 2*margin + 1; k < 2*margin+1+2*inc; k++) {
      for (l = length-(2*margin+1); l < length; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the bottom left corner */

calculate_blcor(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = length-(2*margin+1)-2*inc; k < length; k++) {
      for (l = 2*margin+1; l < 2*margin+1+2*inc; l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = length-(2*margin+1)-2*inc; k < length-(2*margin+1); k++) {
      for (l = 0; l < 2*margin+1; l++) {
         ++min_hist[image[l][k]];
      }
   }
}

/* Add pixels when the window is in the bottom right corner */

calculate_brcor(i, j, size, margin, inc, image, length, min_hist)
int i, j, size, margin, inc, length, *min_hist;
IMAGE image;
{
   int k, l, total = 0;

   for (k = length-(2*margin+1)-2*inc; k < length; k++) {
      for (l = length-(2*margin+1)-2*inc; l < length-(2*margin+1); l++) {
         ++min_hist[image[l][k]];
      }
   }

   for (k = length-(2*margin+1)-2*inc; k < length-(2*margin+1); k++) {
      for (l = length-(2*margin+1); l < length; l++) {
         ++min_hist[image[l][k]];
      }
   }
}
```

smoothed lhe **smoothed_lhe.c**

```
/* Equalise the histogram */

eql(histogram, size, pixel_level)
int *histogram, size, pixel_level;
{
   int i;
   float total = 0, amount;

   for (i = 0; i < pixel_level; i++) {
      if (histogram[i] != 0) {
         amount = (float) (255.0 - offset)*histogram[i]/size;
         total += (float) amount;
      }
   }

   amount = (float) (255.0 - offset)*histogram[pixel_level]/size;
   return((int) offset+total+amount/2.0);
}

/* Find where the pixel is mapped to in the grey scale */

find_position(peak, histogram, total)
int total, *histogram, peak;
{
   int i, start = 0, position;
   float amount;

   for (i = 0; i < peak; i++) {
      start += histogram[i];
   }

   amount = (float) (255.0 - offset)*start/total;
   amount += (float) 0.5*(255.0 - offset)*histogram[peak]/total;
   position = offset + (int) amount;
   return(position);
}

/* Set up the lut values */

set_lut(monitor)
int monitor;
{
   int i, j;

   if (monitor == 0)
      set_gdm();
   else if (monitor == 1)
      set_gdmA();
   else
      set_hitachiLR();
```

smoothed lhe **smoothed_lhe.c**

```
}

set_gdm()
{
   int i;

   offset = 50;

   for (i = 0; i < 35; i++)
      lut[i] = 40 - i;
   for (i = 35; i < 45; i++)
      lut[i] = 9;
   for (i = 45; i < 50; i++)
      lut[i] = 7;
   for (i = 50; i < 55; i++)
      lut[i] = 4;
   for (i = 55; i < 70; i++)
      lut[i] = 3;
   for (i = 70; i < 210; i++)
      lut[i] = 2;
   for (i = 210; i < 256; i++)
      lut[i] = 3;
}

set_gdmA()
{
   int i;

   offset = 60;

   for (i = 0; i < 50; i++)
      lut[i] = 53 - i;
   for (i = 50; i < 55; i++)
      lut[i] = 4;
   for (i = 55; i < 65; i++)
      lut[i] = 3;
   for (i = 65; i < 75; i++)
      lut[i] = 2;
   for (i = 75; i < 230; i++)
      lut[i] = 1;
   for (i = 230; i < 256; i++)
      lut[i] = 2;
}

set_hitachiLR()
{
   int i;

   offset = 10;
```

smoothed lhe **smoothed_lhe.c**

```
   for (i = 0; i < 10; i++)
      lut[i] = 7;
   for (i = 10; i < 30; i++)
      lut[i] = 4;
   for (i = 30; i < 75; i++)
      lut[i] = 3;
   for (i = 75; i < 160; i++)
      lut[i] = 2;
   for (i = 160; i < 256; i++)
      lut[i] = 3;
}
```

APPENDIX L.
203 BOOKLET.

References

H. C. Andrews, A. G. Tescher, and R. P. Kruger, Image processing by digital computer, *IEEE Spectrum*, **9**, 20-32 (1972).

H. C. Andrews, Monochrome digital image enhancement, *Applied Optics*, **15**, No. 2 (1976).

A. Beghadi, and A. Le Negrate, Contrast enhancement technique based on local detection of edges, *Computer Vision, Graphics and Image Processing*, **46**, 162-174 (1989).

A. Beghadi, *Thesis*, University of Paris-6 (1986).

P. A. Chochia, Image enhancement using sliding histograms, *Computer Vision, Graphics and Image Processing*, **44**, 211-229 (1988).

P. A. Chochia, Methods of enhancing aerospace images using fragment histograms, *Soviet Journal of Remote Sensing Vision*, **44**, 1103-1123, (1989).

M. L. Cocklin, A. R. Gourlay, P. H. Jackson, G. Kaye, I. H. Kerr, and P. Lams, Digital processing of chest radiographs, *Image and Vision Computing*, **2**, 67-78 (1983).

R. Dale-Jones, *The Elicitation of Image Processing Knowledge for an Expert System*, M.Sc thesis (1989).

R. D. Dale-Jones and T. Tjahjadi, Contrast enhancement against a constant background, *7th Scandinavian Conference on Image Analysis*, Aalborg, 894-901 (vol 2), (1991).

R. D. Dale-Jones and T. Tjahjadi, Four algorithms for enhancing images with large peaks in their historam, *Image and Vision Computing*, **10**, 495-507, (1992).

L. Dash, and B. N. Chatterji, Adaptive contrast enhancement and de-enhancement, *Pattern Recognition*, **24**, 289-302 (1991).

S. Deutsch, *Models of the Nervous System*, 201-211, Wiley, New York (1967).

R. Forsyth, (editor), *Expert Systems, principles and case studies*, Chapman and Hall, London (1984).

Franz Lisp Reference Manual, Opus 43, Franz Incorporated, Alameda, California (1986).

W. Frei, Image enhancement by histogram hyperbolisation, *Computer Graphics and Image Processing*, **6**, 286-294 (1977).

W. B. Green, *Digital Image Processing - a Systems Approach*, Van Nostrand Rheinhold, New York, (1983).

A. Goldberg, and D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesely, Reading, Mass. (1983).

R. C. Gonzalez, and P. Wintz, *Digital Image Processing*, (2nd edition), Addison-Wesely, Reading, Mass. (1987).

R. C. Gonzalez, and B. A. Fittes, Grey-level transformations for interactive image enhancement, *Mechanism and Machine Theory*, **12**, 111-122 (1977).

R. Gordon, and R. M. Rangayan, *Appied Optics*, **23**, No. 4 (1984).

E. L. Hall, R. P. Kruger, S. J. Dwyer, D. L. Hall, R. W. McLaren, and G. S. Lodwick, A survey of pre-processing and feature extraction techniques for radiographic images, *IEEE Transactions on Computers.*, **20**, 1032-1044 (1971).

E. L. Hall, *Computer Image Processing and Recognition*, Academic Press, New York, 26-30 (1979).

R. A. Hummel, Histogram modification techniques, *Computer Graphics and Image Processing*, **4**, 209-224 (1975).

R. A. Hummel, Image enhancement by histogram transformation, *Computer Graphics and Image Processing*, **6**, 184-195 (1977).

D. B. Judd and G. Wyszecki, *Color in Buisiness, Science, and Industry*, John Wiley and Sons (1975).

J. Kautsky, and N. K. Nichols, Equidistributing meshes with constraints, *SIAM Journal of Scientific and Statistical Computing*, **1**, 499-511 (1980).

J. Kautsky, D. L. B. Jupp, and N.K. Nichols, Image enhancement by smoothed histogram modification, *Proceedings of the 2nd Australasian Remote Sensing Conference LANDSAT '81*, 6.6.1-6.6.5 (1981).

J. Kautsky and N. K. Nichols, Smoothed regrading of discretised data, *SIAM Journal of Scientific and Statistical Computing*, **3**, 145-1599 (1982).

J. Kautsky, D. L. B. Jupp, and N. K. Nichols, Smoothed histogram modification for image processing, *Computer Graphics and Image Processing*, **26**, 271-291, (1984).

D. J. Ketcham, R. W. Lowe and J. W. Weber, Image enhancement techniques for cockpit displays, *TR-P74-350R Display Systems Laboratory*, Hughes Aircraft Co., Culver City, CA, USA (1974).

D. J. Ketcham, Real time image enhancement, *SPIE/OSA*, **74**, 120-124 (1976).

J. Kittler, J. Illingworth, and J. Foglein, *Computer Vision, Graphics and Image Processing*, **30**, 125-147 (1985).

F-N, Ku, The principles and methods of histogram modification adapted for visual perception, *Computer Graphics and Image Processing*, **26**, 107-117 (1984).

S. J. Lim, *Image Enhancement* in Digital Processing Techniques (ed. Ekstrom), Academic Press, 1-45 (1984).

T. Matsuyama, Expert systems for image processing: knowledge-based composition of image analysis processes, *Computer Vision, Graphics and Image Processing*, **48**, 22-49 (1989).

M. Minsky, A frame work for representing knowledge, in *The Psychology of Computer Vision* (ed. P. H. Winston), McGraw-Hill, New York, 245-162 (1975).

M. Nagao and T. Matsuyama, Edge preserving smoothing, *Computer Vision and Image Processing*, **9**, 394-407 (1979).

P. J. Nahin, A simplified derivation of Frei's histogram hyperbolisation for image enhancement, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **1**, 107-117 (1979).

L. O'Gorman and L. S. Brotman, Entropy-constant image enhancement by histogram transformation, *Computer Vision, Graphics, and Image Processing*, **26**, 107-117 (1984).

L. J. Pinson and R. S. Weiner, *An Introduction to Object-Oriented Program-*

ming and Smalltalk, Addison-Wesely, Reading, Mass. (1988).

S. M. Pizer, E. P. Amburn, D. J. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. H. Romney, J. B. Zimmerman, and K. Zuiderveld, Adaptive histogram equalisation and its variations, *Computer Vision, Graphics and Image Processing*, **39**, 355-368 (1987).

C. A. Pomalaza-Raez and C. D. McGillem, An adaptive nonlinear edge-preserving filter, *IEEE Transactions on Acoustics, Speech and Signal Processing*, **32**, 571-576 (1984).

C. Quan and P. J. Bryanston-Cross, Double-source holographic contouring using fibre optics, *Optics and Laser Technology*, **22**, 255-259 (1990).

C. E. Shannon, and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana (1949).

J. J. Sheppard, *Human Color Perception*, Elsevier, New York, (1968).

K. Sheldon, Probing space by camera, *Byte*, **12**, 143-148 (1987).

S. S. Stevens, *Handbook of Experimental Psychology*, Wiley, New York, (1951).
A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, (2 vols.), Academic Press, New York (1982).

R. A. Schowengerdt, and H. Wang, A general purpose expert system for image processing, *Photogrammetric Engineering and Remote Sensing*, **55**, 1277-1284 (1989).

M. Stroustrup, What is object-oriented programming, *IEEE Software*, **5**, No. 3, 10-20 (1988).

Sunview Programmer's Guide, Sun Microsystems Incorporated, Mountain View, California (1986).

R. Taniguchi, M. Amamiya, and E. Kawaguchi, Knowledge-based image processing system: IPSSENS-II, *IEE 3rd International Conference on Image Processing and its Application*, UK, 462-466 (1990).

F. Tomita and S. Tsuji, Extraction of multiple regions by smoothing in selected neighbourhoods, *IEEE Transactions on Systems, Man and Cybernetics* , **7**, 107-109 (1977).

T. Toriu, H. Iwase, and M. Yoshida, An expert system for image processing, *Fujitsu Science and Technology Journal*, **23**, 111-118 (1987).

D. A. Waterman, *A Guide to Expert Systems*, Addison-Wesley, Reading, Mass. (1985).

J. K. Yan and D. J. Sakrison, Encoding of images based on a two-component source model, *IEEE Transactions on Communication.* **25**, 1315-1322 (1977).