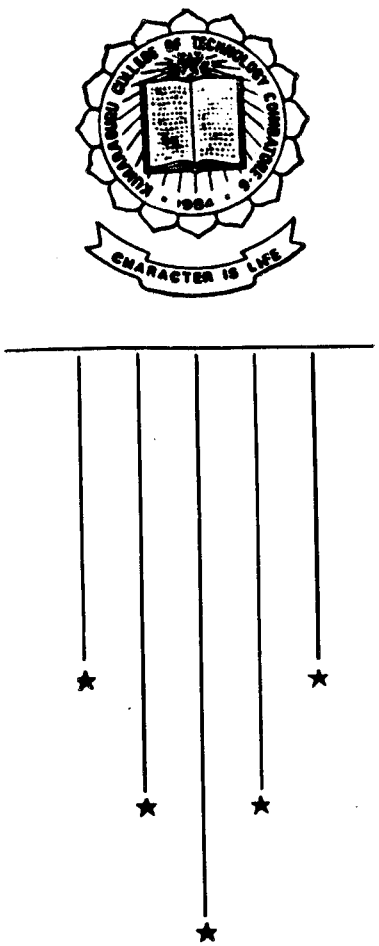
Digital Image Processing

PROJECT REPORT 1996 - 97



Submitted by

K. DHEVENDRAN

S. KARTHIKEYAN

V. PRAKASH

S. SANKAR

Guided by

Prof. K. RAMPRAKASH, M.E.,

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
BACHELOR OF ENGINEERING IN
ELECTRONICS AND COMMUNICATION ENGINEERING
OF THE BHARATHIAR UNIVERSITY, COIMBATORE

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Kumaraguru College of Technology

COIMBATORE - 641 006.

Kumaraguru College of Technology

Coimbatore - 641 006.

Department of Electronics and Communication Engineering

Certificate

This is to Certify that this Project Entitled

DIGITAL IMAGE PROCESSING

Has been submitted by

Mr. .....

in partial fulfilment of the requirements for the award of Degree of Bachelor of Engineering in the Electronics and Communication Engineering Branch of the Bharathiar University, Coimbatore-641046 during the academic year 1996-'97.

(Guide) 7/4/97

(Head of Department)

Certified that the Candidate was Examined by us in the Project Work.

Viva-Voce Examination held on _____

University Register Number _____

(Internal Examiner)

(External Examiner)

Dedicated to our Beloved parents

Acknowledgement

ACKNOWLEDGEMENT

First of all we wish to thank our respected principal **Dr.S.Subramanian** for providing us with all the facilities .

We wish to express our sincere gratitude to the Head of the department of Electronics and Communication Engineering, Prof. **M.Ramasamy** for his encouragement and facilities he has provided us.

We are thankful to our project guide Assistant Prof. **K.Ramprakash** for his guidance and support without which this project would not have been successful.

Our heartful thanks are due to Prof.**P.Shanmugam** and senior lecturer **P.Rajendran** for providing us the required computer facilities and access to their image scanner.We also thank **Mr.R.Hariharan** for his valuable suggestions on this project.

We are indebted to all the staffs of our department who motivated us by all means. Finally we thank our friends and all of those who have helped us to reach this stage.

Synopsis

SYNOPSIS

Digital Image Processing is one of the rapidly evolving field in science and engineering.Image Processing deals with the improvement of images for human perception.

In this project we have tried to extract some features in the ever growing field of Digital Image Processing. Generally Image Processing is used to

* Sharpen the image that is out of focus,

* Enlarge the image to view the picture clearly,

* View the image from different angles,

* Modify the image.

In this project effort has been made to implement the following image processing algorithms:-

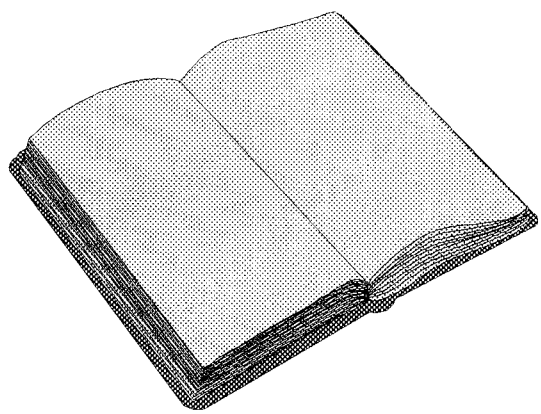
1. Negation ,colour detection, Histogram plotting,Contrast stretching in point process.

2. Edge Enhancement,Smoothening,sharpening in area process.

3. Rotation and Mirroring of images in geometric process.

4. Morphing of two dimensional diagrams in frame process.

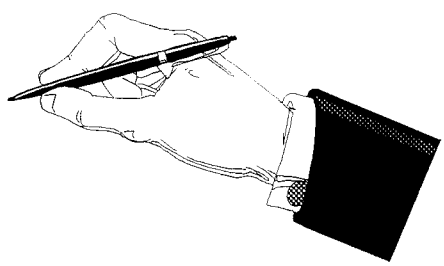
Borland c++ has been utilised for the software implementation in this project.



Contents

CONTENTS

Chapter 1



Introduction

1. INTRODUCTION

The term Digital Image Processing refers to processing of a two dimensional picture or data in a digital computer.A Digital Image is an array of real or complex numbers represented by a finite number of bits. An image given in the form of transparency, slide or photograph is first digitized and stored as a matrix of binary digits in computer memory . This digitized image is then processed and/or displayed.

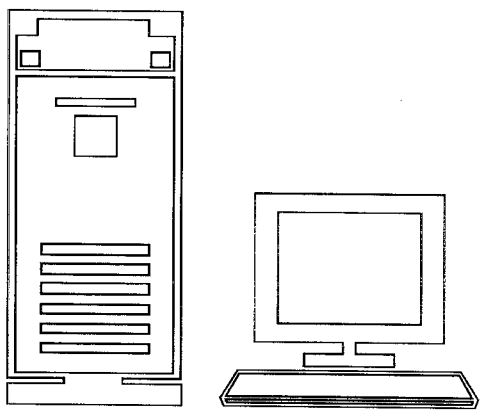
For real time applications processing should be done at a speed of 30 frames per second which requires very high speed processors and parallel processing should be implemented. Hence we are restricting our project to a single frame in which only one image is processed and output is displayed.

Image processing functions can be broadly classified as two types viz. individual pixel operations and neighbourhood pixel operations. In individual pixel operations we need not care about other pixel i.e each pixels are processed independently. In neighbourhood operations, other pixel values are taken into account.

The image can be scanned in various formats like PCX and BIT MAP. Most commonly in WINDOWS we use BIT MAP files. We will thus process only bitmap files. In bitmaps certain number of bits are allocated for each picture element or pixel.

If "n" is the number of bits allocated for each pixel, then that image can be represented in 2^n colours. In a monitors way, the image refers to a function of space f(x,y). The value of f(x,y) denotes brightness or colour of the scene.

Chapter II



Elements of Digital Image Processing System

ELEMENTS OF A DIGITAL IMAGE PROCESSING SYSTEM

The components of basic, general-purpose digital image processing system are shown in Fig 1. The operation of each block is explained briefly below. The sequence of operation is depicted in Fig 2.

2.1 Image Processors

A digital image processor is the heart of any image processing system. An image acquisition, storage, low level(fast) processing, and display. Typically, the image acquisition module has a TV signal as the input and converts this signal into digital form, both spatially and in amplitude. Most modern image processors are capable of digitizing a TV image in one frame time (i.e) 1/30 of a second). For this reason, the image acquisition module is often referred to as a frame grabber.

The storage module, often called a frame buffer, is a memory capable of storing an entire digital image. Usually, several such modules are incorporated in an image processor. The singled most distinguishing characteristic of an image storage module is that the contents of the memory can be loaded or read at TV rates(on the order of 30 images per second). This feature allows the image acquisition module to deposit a complete image into storage as fast as it is being

A DIGITAL IMAGE PROCESSING SYSTEM

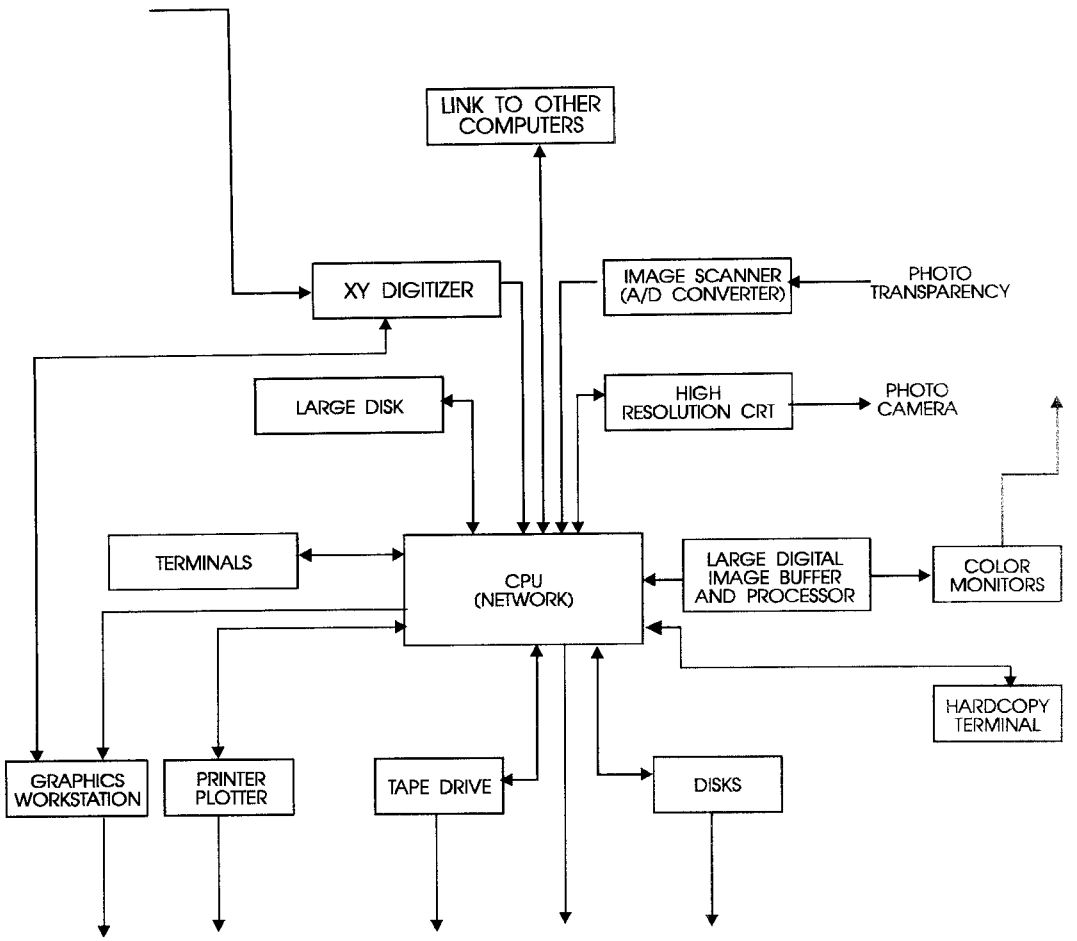


Fig.

A TYPICAL DIGITAL IMAGE PROCESSING SEQUENCE

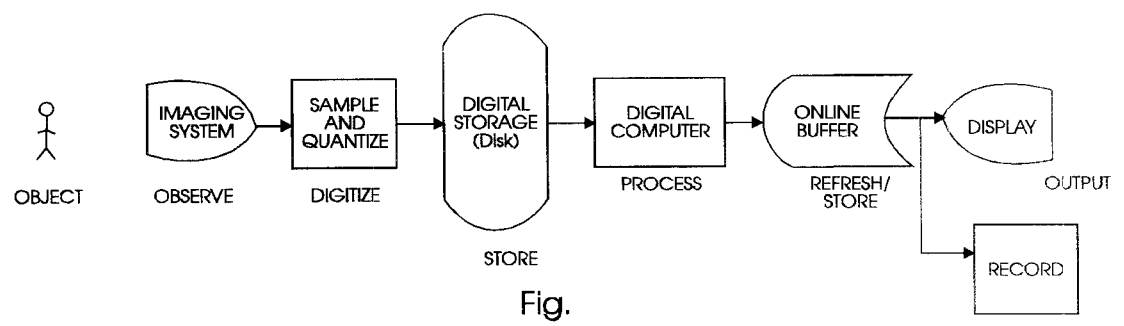


Fig.

grabbed. Conversely the memory can be addressed at TV rates by display module, which outputs the image to a TV monitor.

The processing module performs low-level functions such as arithmetic and logic operations. Thus this module is often called an Arithmetic-Logic Unit (ALU). It is a specialized hardware device designed specifically to gain speed by processing pixels in parallel. The function of the display module is to read an image memory. Convert the stored digital information into an analog video signal and output this signal to a TV monitor or other video device. Typical additional hardware display options include gray-level transformation functions and graphics, as well as alphanumeric overlays.

2.2 Digitizers

A digitizer converts an image into a numerical representation suitable for input into a digital computer. Among the most commonly used input devices are micro densitometers, flying spot scanners, image dissectors, vidicon cameras, and photosensitive solid-state arrays. The first two device require that the image to be digitized be in the form of a transparency (e.g) a film negative) or photograph. Image dissectors, vidicon cameras, and solid-state arrays can accept images

recorded in this manner, but they have the additional advantages of being able to digitize natural images that have sufficient light intensity to excite the detector.

Microdensitometer

In microdensitometers the transparency or photograph is mounted on a flat bed or wrapped around a drum. Scanning is accomplished by focusing a beam of light. In the case of transparencies the beam passes through the film. In photographs it is photodetector and the gray level at any point in the image is recorded by the detector based on the intensity of the beam. A digital image is obtained by allowing only discrete values of intensity and position in the output. Although microdensitometers are slow devices, they are capable of high degrees of position accuracy due to the essentially continuous nature of the mechanical translation used in the digitization process.

Flying Spot Scanners

It operates on the principle of focusing a transmitted or reflected source beam on a photodetector. In this case the image is stationary and the light source is a cathode ray tube(CRT) in which a beam of electrons, deflected by electromagnets, impinges on a fluorescent phosphor surface. The beam thereby produces a spot of light that moves in a scanning pattern on the face of the tube.

The fact that the beam is moved electronically allows high scanning speeds. Flying spot scanners are also ideally suited for applications in which it is desirable to control the beam scanning pattern externally (e.g. in tracing the boundaries of objects in an image.) This flexibility is afforded by the fact that the position of the electron beam is quickly and easily established by external voltage signals applied to the electromagnets.

Image Dissectors And Vidicon Cameras

In image dissectors and vidicon cameras the image is focused directly on the surface of a photosensitive tube whose response is proportional to the incident light pattern. Dissector operation is based on the principle of electronic emission, where the image incident on the photosensitive surface produces an electron beam whose cross section is roughly the same as the geometry of the tube surface. Image pickup is accomplished by using electromagnets to deflect the entire beam past a pinhole located on the back of the dissector tube. The pinhole lets through only a small cross section of the beam and thus "looks" at one point in the image at a time. Since photo emissive materials are very inefficient, the time that the pinhole has to look at the point source in order to collect enough electrons tends to make image dissectors rather digitizers. Most devices integrate the emission of each input point over a specified time interval before yielding a signal that is

proportional to the brightness of a point. This integration capability is beneficial noise brightness of the point. This integration capability is beneficial in terms of noise reduction, thus making image dissectors attractive in applications where high signal to noise ratios are required. As in flying spot scanners, control of the scanning pattern in image dissectors is easily varied by external voltage signal applied to the electromagnets.

The operation of vidicon cameras is based on the principle of photo conductivity . An image focused on the tube surface produces a pattern of varying conductivity as image. An independent, finely focused electrons beam scans the rear surface of the photo conductive target and, by charge neutralization, this creates a potential difference that produces on a collector a signal proportional to the input brightness pattern, position of the scanning beam.

Solid-state arrays are composed of discrete silicon imaging elements, called photosites, that have a voltage output proportional to the intensity of the incident light. Solid state arrays are organized in one of two principal geometrical arrangements, Line Scan Sensors and Area Sensors. A line scan sensors consist of a row of scene and the detector. An area sensor is composed of photosites and is therefore capable of capturing an image in the same manner as, say a vidicon tube.

Vidicon and area sensors are typically packaged as TV cameras. Image digitization is achieved by feeding the output of camera into the image acquisition module as discussed in the previous section. Although TV cameras are in general less accurate than the system discussed above, they have numerous advantages that in many applications outweigh their relative lack of precision. Vidicon systems, for example, are among the most inexpensive digitizers in the market. They also have the distinct monitor. This capability being digitized can be viewed in its entirety on a TV monitor. This capability, not available in any of the system discussed above, is ideal for general purpose applications.

Digital Computers

An image processor may be equipped with internal processing capabilities, the level of this processing is rather low in sophistication. Thus one usually finds the image processors are interfaced to a general-purpose computer, which provides versatility as well as case of programming. Computer systems used for image processing range from microprocessor devices to large computer systems capable of performing computationally intensive functions to large image arrays. The principal parameters influencing the structure of a computer for image processing are the intended application and the required data throughout. For dedicated applications which normally dictate low cost, a well equipped microcomputer or

minicomputer will often be sufficient. If the application involves extensive program then minicomputer will often be sufficient. If the application involves extensive program development or is characterized by significant data throughput, a mainframe computer or minicomputer will often be sufficient. If the application involves extensive program development or is characterized by significant data throughput, a mainframe computer would most likely be required. In this case, a computer with virtual addressing makes disk peripheral storage available to the user as if it were main memory. This feature which is transparent to the user, is of crucial importance because digital images utilize large amounts of memory during processing. The alternative to virtual addressing is a set of user-supplied complex routines whose only function is to swap image segments is and out of peripheral storage during processing.

Storage Devices

A digital image consisting of 512 x 512 pixels, each of which is quantized into eight bits, requires 0.25 megabytes of storage. Thus providing adequate bulk storage facilities is one of the most important aspects in the design of a general purpose image processing system. The three principal storage media used in this type of work are magnetic disks, magnetic tapes, and optical disks. Magnetic disks with a capacity of 700 megabytes or more are common. A 700 Megabytes disk

would hold on the order of 2800 images of the size mentioned above. High-density magnetic tapes (6400 bytes per inch) can store one such image in approximately four feet of tape. Optic disks, which are based on laser writing and reading technology, have recently become commercially available. The storage capacity of a single optical disk platter can approach 4 gigabytes, which translates into approximately 16,000 images per disk.

Image Representation And Modeling

In image representation one is concerned with characterization of the quantity that each picture-element (also called pixel or (pel) represents. Am image could represent luminance characteristics of a body tissue(X-ray imaging), the radar cross section of a target (radar imaging), the temperature profile of a region (infrared imaging), or the gravitational field in an area(in geophysical imaging). In general, any two dimensional function that bears information can be considered an image. Image models give a logical or quantitative description of the properties of this function.

An important consideration in image representation is the fidelity or intelligibility criteria for measuring the quality of an image or the performance of a processing technique. Specification of such measures requires models of

perception of contrast, spatial frequencies, color, and so on. Knowledge of a fidelity criterion helps in designing the image sensor, because it tells us the variable that should be measured most accuracy.

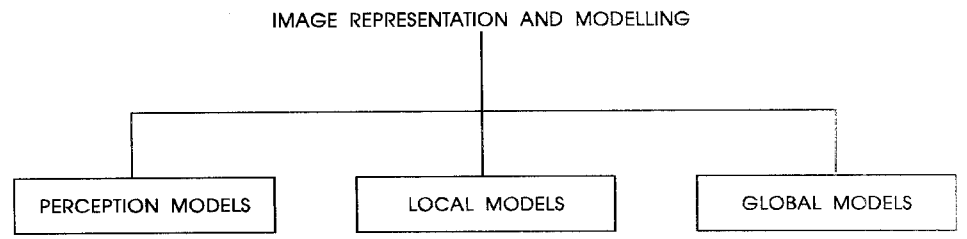
The fundamental requirement of digital processing is that image must be sampled and quantized. The sampling rate has to be large enough to preserve the useful information in an image, it is determined by the bandwidth or raster scanned common television signal is about 4 MHz. From the sampling theorem, this requires a minimum sampling rate of 8 MHz. At 30 frames/s this means each frame should contain approximately 266,000 pixels. Thus for a 512 line raster, the means each image frame contains approximately 512 x 512 pixels. Image quantization is the analog to digital conversion of a samples image to a finite number of gray levels.

A classical method of signal representation is by an orthogonal series expansion, such as the Fourier series. For image, analogous representation is possible via two-dimensional orthogonal functions called basis images. For sampled images, the basis images can be determined from unitary matrices called image transforms. Any given image can be expressed as a weighted sum of the basis images. Several characteristics of image, such as their spatial frequency

content, bandwidth, power spectrum, and application in filter design, feature content, and so on , can be studied via such expansions.

Statistical models describe an image as a member of an ensemble, often characterized by its mean and covariance functions. This permits development of algorithms that are useful for an entire class or an ensemble of images rather than for a single image. often the ensemble is assumed to be stationary so that the mean and covariance functions can easily be estimated. Stationary models are useful in data compression problems such as transform coding, restoration problems such as wiener filtering, and in other applications where global properties of the ensemble are sufficient. A more effective use of these models in image processing is to consider them to be spatially varying or piece wise spatially invariant.

To characterize short-term or local properties of the pixels, one alternative is to characterize each pixel by a relationship with its neighbourhood pixels. For example,a linear system characterized by a (low-order) difference equation and forced by white noise or some other random field with known power spectrum density is a useful approach for representing the ensemble. Figure shows three types of stochastic models where an image pixel is characterized in terms of its neighbouring pixels. If the image were scanned top to bottom and then left to right,



- VISUAL PERCEPTION OF CONTRAST, SPATIAL FREQUENCIES, AND COLOUR
- IMAGE FIDELITY MODELS
- TEMPORAL PERCEPTION
- SCENE PERCEPTION

- SAMPLING AND RECONSTRUCTION
- IMAGE QUANTIZATION
- DETERMINATION/UNITARY
- SERIES EXPANSIONS/UNITARY TRANSFORMS
- STATISTICAL MODELS

- SCENE ANALYSIS/ARTIFICIAL INTELLIGENCE MODELS
- SEQUENTIAL AND CLUSTERING
- IMAGE UNDERSTANDING MODELS

Fig.

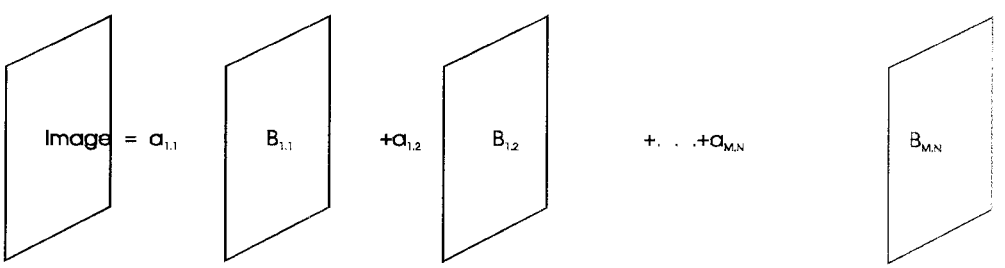


Fig.

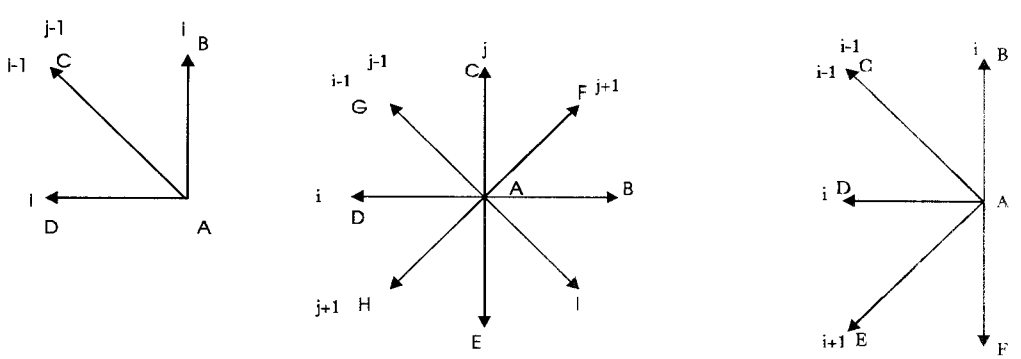


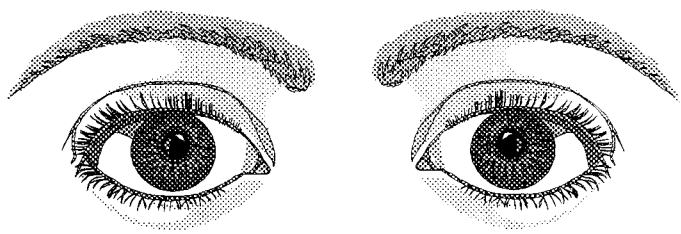
Fig.

the model of fig.3 would be called as causal model. This is because the pixel A is characterized by pixels that lie in the "past". Extending this idea , the model of fig 4 is a noncasual model because the neighbours of A lie in the past as well as the "future" in both the directions. In fig 5 we have a semicausal model because the neighbours of A are in the past in the j-direction and are in the past as well as future in the i-direction.

Such models are useful in developing algorithms that have different hardware realizations. For example, causal models can realize recursive filters, which require small memory while yielding an infinite impulse response (IIR). On the other hand, noncausal models can be used to design fast transform-based finite impulse response (FIR) filters. Semicausal models can yield two-dimensional algorithms, which are recursive in one dimension and none recursive in the other. Some of the these stochastic models can be thought of a generalization of one dimensional random processes represented by auto regressive (AR) and auto regressive moving average (ARMA) models.

In global modeling, an image is considered as a composition of several objects. Various objects in the scene are detected (for example, by segmentation techniques), and the model gives the rules for defining the relationship among various objects. Such representations fall under the category of image understanding models.

Chapter III



Processing Techniques

PROCESSING TECHNIQUES

We will first categorize the different image processing algorithms possible.

* If an algorithm changes a pixel value based on only on that pixel value is called a as "point process".

* If an algorithm changes a pixel value based on the values of neigbhouring pixels it is called as "area process".

* If an algorithm changes the position or arrangement of pixels, it is called as "geometric process".

* Algorithm that change pixel value based on comparing two or more images are called "frame process".

3.1 Point process

In point process we do not modify the spatial relationships within an image. Hence these processes donot change the position of the image. The various point processes on which we are concentrating are

1. Image negation
2. Colour detection
3. Histogram
4. Contrast stretching

Here while operating on a pixel, the other pixels are not taken into account.

Point operations are zero memory operations where a given gray level μ[0,l] is mapped into a gray level v[0,l) according to a transformation.

$$v=f(u)$$

3.1.1. Image Negation

A negative image can be obtained by reverse scaling of the gray levels according to the transformation v=f(u).

$$v= L-u$$

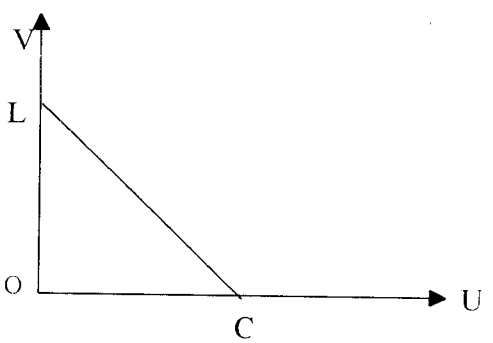


Fig.3.1 Digital Negative Transformation

Negative images are produced by inverting each bit of the image i.e. one's are replaced by zeros and zeros by one's. Negation is used to notice minor details of brighter portions of the image. When dark light falls on pupil of eye, it enlarges allowing more light to fall into it, negation is necessary in such situations.

$$\text{If Image} = \sum_{i=0}^{h} \sum_{j=0}^{w} I(i,j)$$

Where I(i,j) = Intensity of image at the point (i,j)

h = height of image.

w = width of image.

$$\text{Then negative image} = \sum_{i=0}^{h} \sum_{j=0}^{w} (255 - I(i,j))$$

3.1.2 Colour detection

The study of color is important in the design and development of color video systems. Use of colour in image displays is not only more pleasing, but it also enables us to receive more visual information. While we can perceive only a few dozen gray levels, we have the ability to distinguish between thousands of colours. The Perceptual attributes arecolour brightness, hue and saturation. Brightness represents the perceived luminance as mentioned before. The hue of a colour refers to redness, greenness, and so on. For monochromatic light source, differences in hues are manifested by the differences in wavelengths. Saturation is that aspect of perception that varies most strongly as more and more white light is added to a monochromatic light. These definitions are somewhat imprecise because hue, saturation, and brightness all change when either the wavelength, the intensity, the hue, or the amount of white light in a color is changed.

Each wavelength in this range is perceived by the eye as a certain tint or hue. All such hues are present in white sunlight, along with energies at other wavelengths beyond this range- namely, infrared and ultraviolet. The color of an object is a function of the unabsorbed wavelengths of light reflected from it. That is why an object will have different colours depending on the light under which it is viewed. An object viewed under sunlight will radiate a different color when viewed under electric light.

The Red, Green and Blue (RGB) primaries are physical primaries that can be used in actual experiments. These primaries are not unique and other primaries can be selected instead. The main advantage of these is that luminance signal is provided directly by only one of the primaries (Y). Y will provide a gray-level image from the color image. These primaries are derived from the R,G,B physical primaries through a linear transformation given by:

$$X = 2.7690\,R + 1.7518\,G + 1.300\,B$$

$$Y = 1.0000\,R + 4.5907\,G + 0.0601\,B$$

$$Z = 0.0000\,R + 0.0565\,G + 5.5943\,B$$

The user is given an option to choose a particular colour and in the image only pixels with that colors are displayed and other are set dark. This process is

helpful in biological sciences for viewing microbes. In biomedical technology it is used to trace for tracing foreign particles inside the body.

3.1.3 Luminance Signal

The luminance of a color image is basically the black-and-white version of that same image. The luminance signal is given by

$$Y = R + 4.5907\ B$$

To illustrate that equation actually does produce a B&W we will develop program to change the colour image to a luminance image. To correct for the aspect ratio we will reduce the width of the luminance image to 256 x 256. The luminance image will also be scaled to have levels from 0 to 255 (8 bits). The scale factor multiplying Y is

$$255.00/((1+4.5907+0.0601) \times 32.0)$$

This transforms the levels to lie between 0 and 255. An alternative colour television signals to black and white is given by

$$Y = 0.030\ R + 0.59\ G + 0.11\ B$$

This equation was derived by taking into account the sensitivity of the eye to the three primaries.

3.1.4 Histogram

The histogram of gray levels in an image is defined by

$$h(i) = n(i)/n$$

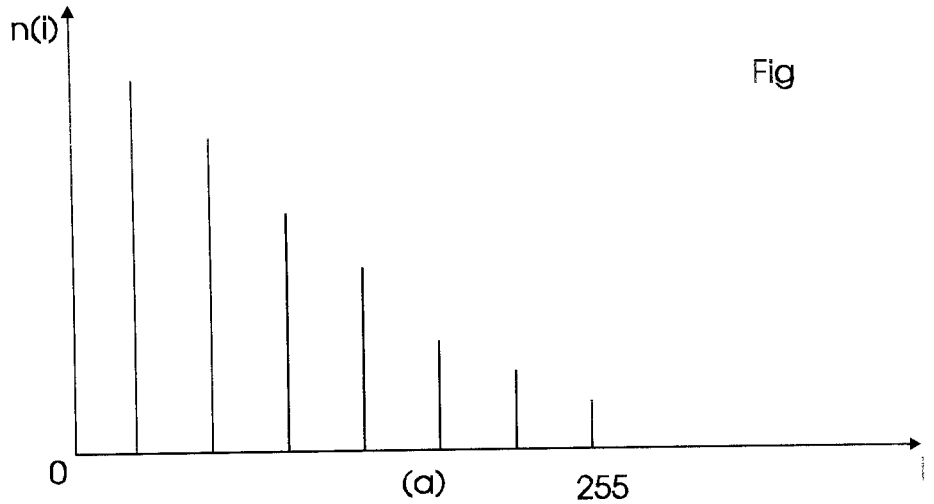
where n(i)= sum of gray levels in the image having the value i and n = total number of gray levels in the image.

The distribution p(i) or n(i) can provide information about the appearance of the image. An image that has a gray-level distribution similar to that shown in fig 6a has predominantly dark tones. On the other hand, fig 6b indicate an image with predominantly light tones. Therefore, it should be clear that one should be able to enhance the appearance of an image by modifying the distribution n(i) to provide some balance between the various gray-level tones in the image.

Let the variable 'r' represent the gray level of the pixels in the image to be enhanced. For simplicity, it will be assumed in the following discussion that the pixel values have been normalized so that they lie in the range

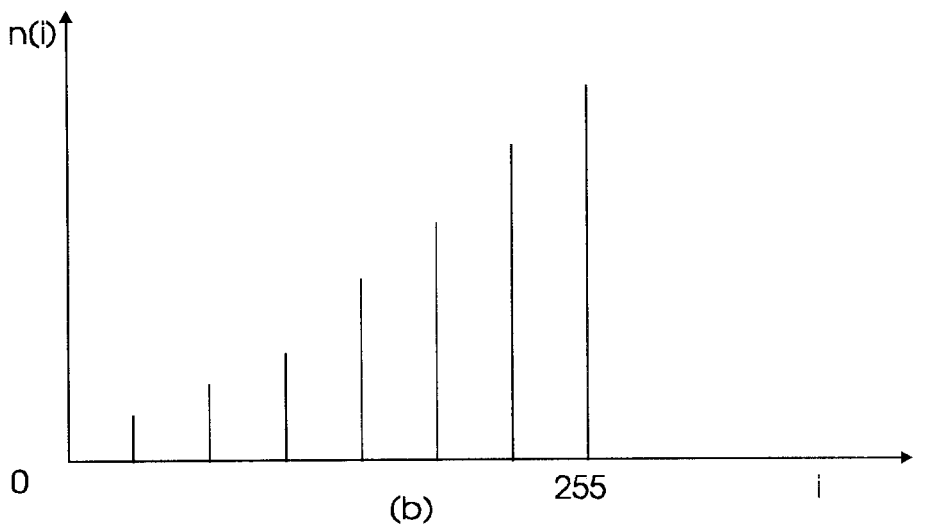
$$0 \leq r \leq 1,$$

with r=0 representing black and r=1 representing white in the gray scale.
For any r in the interval [0,1], attention will be focussed on transformations of the form

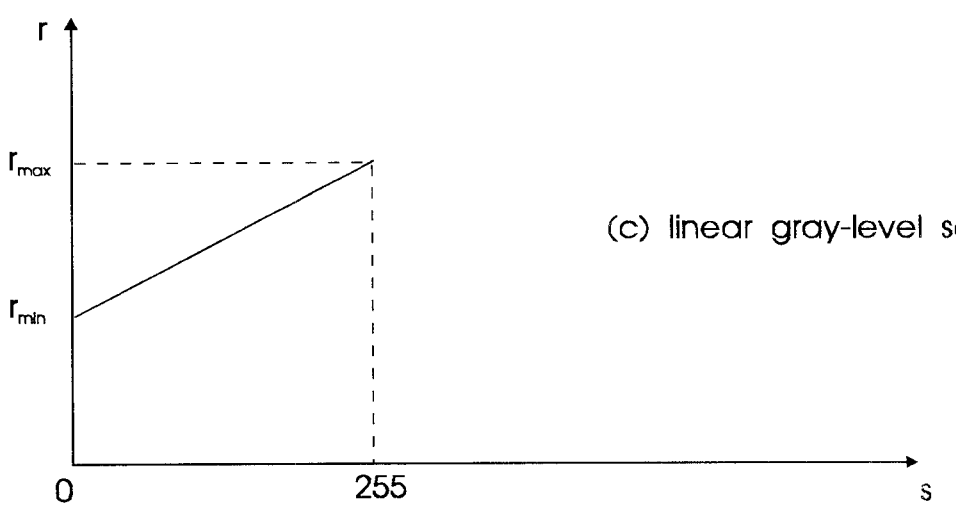


Fig

(a) Histogram of an image with predominantly dark tones



(b) Histogram of an image with predominantly light tones



(c) linear gray-level scaling

$$s = T(r),$$

which produce a level for every pixel value r in the original image. It is assumed that the transformation function given in equation satisfies the conditions:

(a) T(r) is single-valued and monotonically increasing in the interval $0 \leq r \leq 1$, and

(b) $0 \leq T(r) \leq 1$ for $0 \leq s \leq 1$

Condition (a) preserves the order from black and white in the gray scale, while condition (b) guarantees a mapping that is consistent with the allowed range of pixel values.

It is helpful in detecting the overall brightness of that image from the histogram plot one can extract lots of information.

3.1.5 Contrast Stretching

Contrast is the difference between the lowest pixel value and the highest pixel value. At low contrast, image cannot be distinguished clearly, it is either too light or too dark. At high contrast we can have darkness along with the light. An upper threshold value and a lower threshold value is fixed for stretching. If (x,y) exceeds upper threshold value, then that value is fixed as maximum. If f(x,y) is

below lower threshold, then it is set to zero. Pixel values between two threshold values are scaled properly.

Linear gray stretching can be achieved by mapping the gray levels of the original image through the linear mapping function shown in Fig 6c..

That is, $s = ((r-r_{min}) / (r_{max} - r_{min})) \times 255$

Where r denotes a gray level in the original image and s the mapped gray level. The mapped image will have gray levels that are stretched between and 255. This operation can provide some improvement on the appearance of the image.

Low contrast images occur often due to poor or nonuniform lighting conditions or due to nonlinearity or small dynamic range of the imaging sensor. Figure shows a typical contrast stretching transformation, which can be expresses as

$$v = \begin{cases} u, & 0 \le u < a \\ \beta(u-a) + v_a, & a \le u < b \\ (u-b) + v_b, & b \le u < l \end{cases}$$

The slope of the transformation is chosen greater than unity in the region of stretch,

The parameters a and b can be obtained by examining the histogram of the image, For example, the gray scale intervals where pixels occur most frequently would be stretched most to improve the overall visibility of a scene.

3.2 Area Process

We use a group of pixels for performing area process. usually we do the operation with the neighbourhood pixels. We use a two dimensional matrix containing pixels values with each dimension as an odd number. This matrix is termed as neighbourhood matrix. The pixel of interest is at the centre of this neighbourhood matrix. A Mask (a matrix which has same dimension as neighbourhood) is moved through each pixels and calculated value replaces that Pixel. In this process, we sharpen, smoothen and remove random noise from the image. Edge detection is also done in this process.

3.2.1 Image Smoothening

Many image enhancement techniques are based on spatial performed on local neighbourhoods of input pixels. often, the image is convolved with a finite impulse response filter called spatial mask.

Here each pixel is replaced by a weighted average of its neighbourhood pixels, that is

$$V(m,n) = \sum_{(k,l)\varepsilon W} \sum a(k,l)y(m-k,n-l$$

Where y(m,n) and v(m,n) are the input and output images, respectively, W is a suitably chosen window, and a (k,l) are the filter weights, giving

$$V(m,n) = \frac{1}{N} \sum_{w(k,l)\varepsilon W} \sum y\,(m-k,\ n-l)$$

Where a(k,l)=1/Nw and Nw is the number of pixels in the window W. Another spatial filter used often is given by

$$v(m,n) = \tfrac{1}{2}[y(m,n) + \tfrac{1}{4}\{y(m-1,n)+y(m+1,n) + y(m,n-1)+y(m,n-1)\}]$$

That is, each pixel is replaced by its average with the average of its nearest four pixels. Figure shows some spatial averaging masks.

Spatial averaging is used for noise smoothing, low-pass filtering, and sub-sampling of images. Suppose the observed image is given as

$$y(m,n) = u(m,n) + n(m,n)$$

Where n(m,n) is white noise with zero mean and variance $^{2}_{n}$. Then the spatial average of yields.

$$v(m,n) = \frac{1}{N} \sum_{w(k,l)\varepsilon W} \sum u(m-k,n-l) + n(m,n)$$

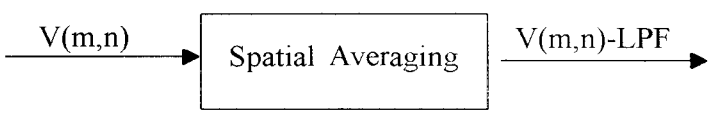
Where n(m,n) is the spatial average of n(m,n). it is a simple to show that n(m,n) has zero mean and variance ^{2}n/Nw, That is, the noise power is reduced by a factor equal to the number of pixels in the window W. If the noiseless image u(m,n) is constant over the window W, then spatial averaging results in the improvement in the output signal-to-noise ratio by a factor of Nw. in practice the size of the window W is limited due to the fact that u(m,n) is not really a constant, so that spatial averaging introduces a distortion i the form of blurring. Figure shows examples of spatial averaging of an image containing Gaussian noise.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1/10	1/10	1/10
1/10	1/5	1/10
1/10	1/10	1/10

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

$$\Sigma a(k,l)=1$$



SPATIAL LOW PASS FILTER

3.2.2 Image Sharpening

The unsharp masking technique is used commonly in the printing industry for crispening the edges. A signal proportional to the unsharp, or low-pass filtered, version of the image is subtracted from the image. This is equivalent to adding the gradient, or a high-pass signal, to the image. In general the unsharp masking operation can be represented by

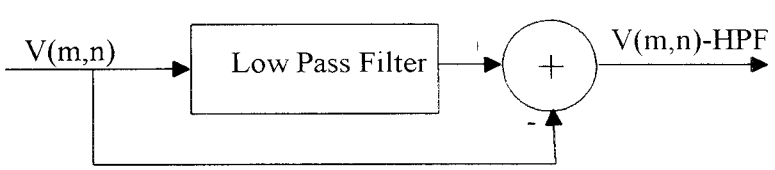
$$v(m,n) = u(m,n) + g(m,n)$$

where $u(m,n) > 0$ *and* $g(m,n)$ *is a suitably defined gradient at* (m,n).

These accenuate high frequency details of an image. While leaving the low frequency contents intact. In this process high frequency is alone highlighted. Masks applied are

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Difference between centre element and all the other elements are high. hence, when it is applied to images, it produces a high pass effect. Sum of all the elements in the matrix is unity.



SPATIAL HIGH PASS FILTER

3.2.4 Image segmentation

Segmentation is the process that subdivides an image into constituent parts or objects. Segmentation is one of the most important elements in automated image analysis because it is at this stage that objects or other entities of interest are extracted from an image for subsequent processing such as description of isolated points, detection of lines and edges of the images.

w1	w2	w3	x1	x2	x3
w4	w5	w6	x4	x5	x6
w7	w8	w9	x7	x8	x9

Let w1,w2,w3.....,w9 represent the coefficients of a 3x3 mask shown below in first fig 3.2.4a. Let x1,x2,x3,.....x9 represents the gray levels of the pixels under the mask.

w1	w2	w3	x1	x2	x3
w4	w5	w6	x4	x5	x6
w7	w8	w9	x7	x8	x9
3 x 3 mask			3 x 3 image region with grey level		

$x = w1x1 + w2x2 + \ldots\ldots\ldots\ldots + w9x9$

Image is transformed using mask and X is calculated.

a) Point Detection

The problem of detecting and then segmenting isolated points in an image applies in noise removal and particle analysis. A basic mask is used for detecting isolated points in an image is shown in Figure. The centre of this mask is moved

from pixel to pixel in an image. At each mask region we compute the sum of product given by the equation.,

$$x = -x1 -x2 -x3-x4+8x5-x6-x7-x8-x9$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

This mask is used for detecting isolated points different from constant background. In practice when one is interested only in strong responses we say that an isolated point whose intensity is significantly different from the background has been detected if

$$|X| > T,$$

Where T is a nonnegative threshold.

b) Line Detection

Consider the mask shown in fig 3.2.4b below. if the first mask is moved around an image, it will respond more strongly to line (one pixel thick) oriented horizontally with constant background, the maximum response will result when the line is passed through the middle row of the mask

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

The second mask responds best to lines oriented at 45 degrees; the third mask to vertical lines and the fourth mask to lines in the -45 degree direction.

These directions can also be established by noting that the preferred direction of each mask is weighed with a larger coefficient other than the possible directions.

Although point and line detection certainly are elements of any discussion on segmentation, edge detection is by far the most common approach for detecting meaningful discontinuities in gray level. The reason for this is that isolated points and thin lines are not frequency occurrences in most applications of practical interest.

We define edge as a boundary between two regions with relatively distinct gray level properties. We assume that the regions in question are sufficiently homogeneous so that the transition between two regions can be described on the basis of gray level discontinuities alone.

Basically the idea underlying most edge detection techniques is the computation. An edge (transitions from dark to light) is modeled as ramp rather than as an abrupt change of gray level.

This model is representation of the fact that edges in digital images are generally slightly blurred as a result of sampling. The magnitude of the first derivative can be used to detect the presence of an edge.

The gradient of an image f(x,y) at location (x,y) is defined as the two dimensional vector.

$$G[f(x,y)] = \begin{bmatrix} G_c \\ G_y \end{bmatrix} = \begin{bmatrix} \delta f/\delta x \\ \delta f/\delta y \end{bmatrix}$$

For an edge detection we are interested in the magnitude of this vector, generally simply refered as gradient and denoted by $G[f(x,y)]$

where $G[F(x,y)] = \sqrt{[Gx * Gx + Gy * Gy]}$

This quantity is equal to the maximum rate of increase of f(x,y) per unit distance in the direction of G. it is common practice to approximate the gradient by absolute values:

$$G[f(x,y)] = |Gx| + |Gy|$$

Computation of gradient is based on obtaining the practical derivaties $\delta f/\delta x$ and $\delta f/\delta y$ at every pixel location. Pixels in 3x3 neighbourhood location about point (x,y) can be formulated as follows. Consider the subimage area shown in figure, where X5 represents the gray levels of eight neighbourhoods of (x,y). We define the component of gradient in x-direction as

$$Gx = (X7+2X8+X9) - (X1-2X2+X3)$$

and in y-direction $Gy = (X3+2X6+X9) - (X1-2X4+X7)$

X1	X2	X3
X4	X5	X6
X7	X8	X9

(a)

−1	−2	−1
0	0	0
1	2	1

(b)

−1	0	1
−2	0	2
−1	0	1

(c)

(a) 3x3 Image Region

(b) Mask used to compute at the center point of 3x3 region.

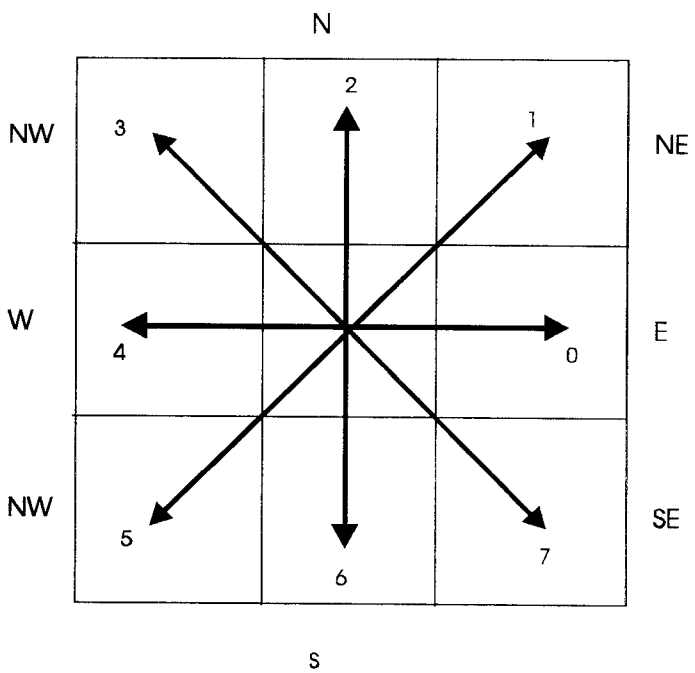
(c) mask used to compute Gy at the center point of 3x3 region.

The responses of the two sobel operators at any point (x,y) are combined using equations written previously, to obtain gradient at that point. Convolving these masks with an image f(x,y) yields the gradient at all points in the image. the result is referred to as gradient image(edge detected image).

Compass gradient masks

These were developed by taking into consideration all the possible orientations of an edge in a discrete image. Thus instead of utilizing just two masks as in the previous two approach, eight masks are used, each providing an edge strength along one of the eight possible directions of the compass as shown in

COMPASS DIRECTIONS



Fig

Fig 7. Four different types of masks of this type are presented next. These were developed on the basis of some underlying data model for the edges in the image.

Prewitt Operators

Two types of compass operators according to prewitt:

Type 1:

$$W_0 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{bmatrix} \quad W_2 = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix} \quad W_3 = \begin{bmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad W_5 = \begin{bmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad W_6 = \begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad W_7 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix}$$

Type 2:

$$W_0 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix} \quad W_2 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad W_3 = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad W_5 = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad W_6 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad W_7 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

With type 2 you will only need to use the first maks because of the symmetry between them and the last four.

Sobel Compass Operator

This is given by the following eight masks:

$$W_0=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}\quad W_1=\begin{bmatrix}2 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -2\end{bmatrix}\quad W_2=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}\quad W_3=\begin{bmatrix}0 & -1 & -2\\ 1 & 0 & -1\\ 2 & 1 & 0\end{bmatrix}$$

$$W_4=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}\quad W_5=\begin{bmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{bmatrix}\quad W_6=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\quad W_7=\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & 1 & 0\end{bmatrix}$$

and again because of the symmetry we need to use only the first four masks.

KIRSH COMPASS OPERATOR

This was proposed as homogeneity operator. it is sensitive to small changes in gradients and generally tends to the superior in comparison trails to the methods previously described, The eight masks are as follows:

$$W_0=\begin{bmatrix}-5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}\quad W_1=\begin{bmatrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}\quad W_2=\begin{bmatrix}5 & -3 & -3\\ 5 & 0 & 3\\ 5 & -3 & -3\end{bmatrix}\quad W_3=\begin{bmatrix}-3 & -3 & -3\\ 5 & 0 & -3\\ 5 & 5 & -3\end{bmatrix}$$

$$W_4=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}\quad W_5=\begin{bmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{bmatrix}\quad W_6=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\quad W_7=\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & -1 & 0\end{bmatrix}$$

For the compass operators the edge-strength image can be calculated from

y(i,j) = max {|Yo(i,j)|,|,.....|Yr(i,j)|}Where Yo, Y1, etc., are the cross correlations of masks Wo, W1, etc, with the image.

3.3 Geometric Process

In this process positions and arrangements of pixels are manipulated. Applying geometric process to image sometimes results in loosing the one to one pixel correspondence between source and destination. In geometric process, one to one mapping of source to destination pixel mapping is further complicated because it must generally be performed from the destination image perspective instead of from the source image perspective.

3.3.1 Rotation

The rotation geometric process allows an image to be rotated about its center point through any arbitrary angle specified. A complete image is rotated by separately rotating each pixel that makes up the image. The equations which govern the transformation of the location of the pixel of the source image ('x old, y old") into its new rotated location in the destination image ("s new, U new:) are as follows:

$$X_{new} = X_{old} * \text{Cos (Angle)} + Y_{old} * \text{Sin (Angle)}$$

$$Y_{new} = Y_{old} * \text{Cos (Angle)} - X_{old} * \text{Sin (Angle)}$$

3.3.2 Image Mirroring

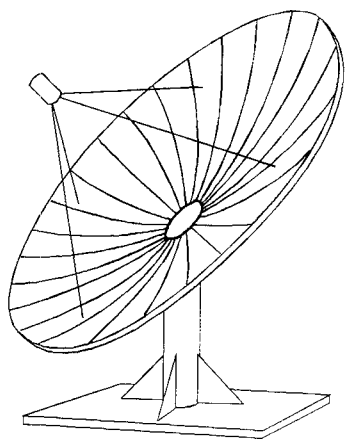
It is the rearrangement of pixels so that image appears as if it is viewed through certain angle. Vertical mirroring generates mirror in vertical direction so that image looks from bottom to top instead of top of bottom. Horizontal mirroring generates mirror in horizontal direction so that image is viewed from right to left instead of viewing from left to right. No complications are encountered in application of the mirror function because of the maintenance of the one-for-one mapping between source and destination pixels. For this reason neither aspect ratio nor interpolation issues need to be considered.

3.4 Frame Process

In the process two different images are considered. This process is used for comparing two images. For instance if two images are subtracted and the result is zero, then the images are identical. Attempt has been made to implement morphing in is type of process

In this process one image is made to disappear slowly and simultaneously another image is made to appear. This process is done in two-dimensional diagrams like ellipse, circle etc..

Chapter IV



Applications

4. APPLICATIONS

Digital image processing has a broad spectrum application is, such as

* Remote sensing via satellites and other space crafts,

* Image transmission and storage for business applications,

* Medical processing,

* Radar and Sonar,

* Forensic science

* Acoustic image processing

* Robotic and automated inspection of industrial process. Images acquired by satellites are useful in

Tracking earth resources, Geographical mapping, Prediction of agricultural crops, urban growth, Weather, Flood and fire control and many other environmental applications.

Space image include recognition and analysis of objects contained in images obtained from deep space probe missions.

Image transmission and storage applications occur in broadcast television, teleconferencing, transmission of facsimile over computer networks, closed circuit television based secutiry monitoring system, and in military communications.

In medical applications one is more concerned with processing of chest X-rays, cineangiograms, projection images of transaxial tomography, and other medical images that occur in radiology, nuclear magnetic resonance (NMR), and ultrasonic scanning. These images may be used for patient screening and monitoring or for detection of turmours or other disease in patient.

Radar and sonar images are used for detection and recognition of various types of targets are in guidance and maneuvering of aircraft or missile systems.

In forensic science is used in finger print analysis, Analysis of blood marks, Character recognition, Analysis of culprit profiles, reconstruction of crime scenes, etc.

There are may other applications ranging from robot vision for industrial automation to image synthesis for cartoon masking or fashion design, In other 8words, whenever a human or a machine or any other entity receives data of two or more dimensions, an image is processed.

Chapter V

Software Implementation

5. SOFTWARE IMPLEMENTATION

VIDEO MODE

Video Graphic Adapter (VGA)

The VGA specification (Table 8) includes all video modes available with MDA, CGA and VGA. When displaying in the 200-line modes, VGA double scans each line. This makes each pixel twice as tall as it would be in CGA or low scanning makes the image sharper, but also changes the aspect ratio. Drawings created on a CGA screen will appear distorted on a VGA system operating in exactly the same video mode.

As for performance, a key new feature in VGA is its connection between the controller card and the monitor. CGA and EGA displays have a digital connection. VGA has an analog connection allowing it to offer for more colors.

The frame buffer contains pointers to colors stores in a RAM look up table in a converters (DACs) to convert the digital information (the color code into an analog signal for the monitor.

Higher the resolution (and the more frame memory used for pixel data, the fewer colors that are possible. In its 640 x 480 mode, an 18-bit VGA used four bits per pixel for color information, allowing it to address (and display 16 colors (2^4). In the 320 x 480 mode, however, the color information for each pixel is eight bits deep, allowing 256 (2^8) simultaneous colors.

VGA COLOURS

C3	C2	C1	C0	COLOUR
0	0	0	0	Black
0	0	0	1	Blue
0	0	1	0	Green
0	0	1	1	Cyan(Blue Green)
0	1	0	0	Red
0	1	0	1	Brown
0	1	1	1	White
1	0	0	0	Dark Gray
1	0	0	1	Light Blue
1	0	1	0	Light Green
1	1	0	0	Light Red
1	1	0	1	Light Magneta
1	1	1	0	Yellow
1	1	1	1	Intensified White

Note: Bit planes Co, C1, C2 and C3 are blue, green, red, and intensified pixels, respectively.

VGA CHARACTERISTICS

BIOS MODE	RESOLUTION	COLOURS	TEXT/ GRAPHICS	CGA	EGA
0	40 x 25	16	8 x 8 text	X	X
0*	40 x 43	16	8 x 14 text		X
0*	40 x 50	16	9 x 16 text	X	
1	40 x 25	16	8 x 8 text		X
1*	40 x 43	16	8 x 14 text		X
1*	40 x 50	16	9 x 16 text		
1*	80 x 25	16	8 x 8 text	X	X
2	80 x 43	16	8 x 14 text		X
2*	80 x 50	16	8 x 16 text		
3*	80 x 50	16	9 x 16 text		
3	80 x 25	16	8 x 8 text	X	X
3*	80 x 43	16	8 x 16 text		X
4	320 x 200	4	graphics	X	X
5	320 x 200	2	graphics	X	X
6	640 x 200	2	graphics	X	X
7	80 x 25	0	9 x 14 text		X
7*	80 x 25	0	9 x 16 text		
D	320 x 200	16	graphics		X
E	640 x 200	4	graphics		X
F	640 x 350	2	graphics		X
10	640 x 350	16	graphics		X
11	640 x 480	2	graphics		
12	640 x 480	16	graphics		
13	320 x 200	256	graphics		

* Text mode with 350 vertical lines

* Text mode with 400 vertical lines

ALGORITHMS

1. Negation

1. Initialise graphics
2. Read input image array
3. For y=0 to height
4. For x= 0 to width, begin
5. Get image (x,y)
6. Subtract it from 255
7. Plot image
8. End graphics

This function alters all the bits in the image i.e., all `1's are converted into `0's and all `0's are converted into `1'.

2. Color Detection

1. Intialise graphics
2. Get color value
3. For i=0 to height
4. For j=0 to width, begin
5. Get image value
6. It image value=color, plot at location (i,j)

7. If image value is not equal to color, blacken that location (i,j)

8. Close graphics

This algorithm detects a particular color, but this does not filter the percentage context of that color in all the other colors.

3. Histogram

1. Intialise graphics

2. Reset all the elements of array of size 255 ie[255]=0

3. For i=0 to height

4. For j= 0 to width, begin

5. Get value of image at location (i,j), say x

6. Increment value at x^{th} position in the array H[255]

7. Plot a graph with 0 to 255 in x axis and the value of H[x] s y axis

8. Close graphics

Histogram is used in analyzing the density of colors is that particular image, it gives a brief idea about contrast of that image.

4. Smoothening By Averaging

1. Intialise graphics and sum=0

2. Read input image array

3. For y=0 to height

4. For x=0 to width, begin

5. Get image (x,y)

6. For i=-1 to +1

7. For j=-1 to +1, begin

8. Sum = Sum +image(i,j)

9. Plot sum at (x,y)

10. End graphics

The function smoothes the image so that image will have uniform variations. Non-uniformities are reduced.

5. Median

1. Intialise graphics

2. For i=0 to height

3. For j=0 to width

4. For K=0 TO 8

5. For m=j-1 to j+1

6. For m=i-1 to i+1, begin

7. Sort image (m,n) and store it in array of size a ie a[9]

8. Plot a[5] at location (i,j)

9. End graphics.

This function removes very low and very high noises. This algorithm, when applied to image repeatedly it gives a pleasent appearance to the image.

6. Edge Detection Using Gradient Operators

1. Intialise graphics
2. Intialise two masks one for x direction and another for y direction.
3. for i = 0 to height
4. for j = 0 to width, begin
5. Move mask over the image elements and calculate the sum of all products of mask element and corresponding image element as sum.
6. Move mask over the image of mask element and corresponding image element as sum 2.
7. Value = $\sqrt{Sum1^2 + Sum2^2}$
8. Plot value at (i,j)
9. End graphics

This algorithm is based on differentiation i.e, by differentiating, constant terms are eliminated and hence only the points having higher variations are highlighted well.

7. Edge Detection Using Compass Operator

1. Intialise graphics

2. Initialise eight masks so that they are oriented in eight different directions.

3. For i=0 to height

4. for j=0 to width, begin

5. Apply each mask separately at the location (i,j).

6. A value is calculated by summing all the products of mask element and corresponding image element.

7. Find the maximum value obtained from applying all masks.

8. Close graphics

This operator deals with eight directions and the direction is which edge is very narrow is chosen to plot. Instead of 8 masks if only one mask is applied then the above funciton acts as a **Filter.** According to the mask applied filter may be high pass filter or low pass filter.

8. Rotation

1. Intialise graphics

2. Get the angle of rotation as

3. For i=0 to height

4. For j=0 to width

5. Get pixel value as p

6. Find x= i Cos + j sin

7. Find y= y cos + i sin

8. Put value p at location (x,y)

9. End graphics

In this process whole frame is moved through given angle. it gives different views of an image buy at two dimensional level.

9. Mirror

1. Initialise graphics

2. Read the image array

3. Read the orientation of mirror

4. If orientation is horizontal do steps 6 through 8

5. If orientation is vertical do steps 9 through 11

6. For i=0 to height

7. For j= width to 0, begin

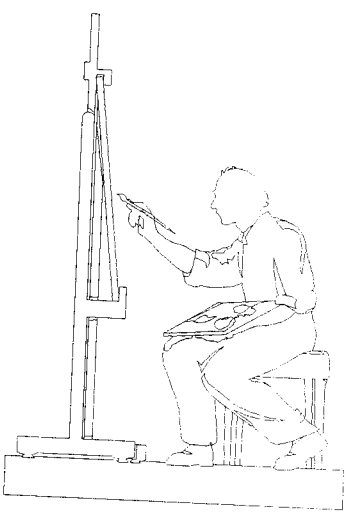
8. Plot the image array, end graphics

9. For i= height to 0

10. For j=0 to width, begin

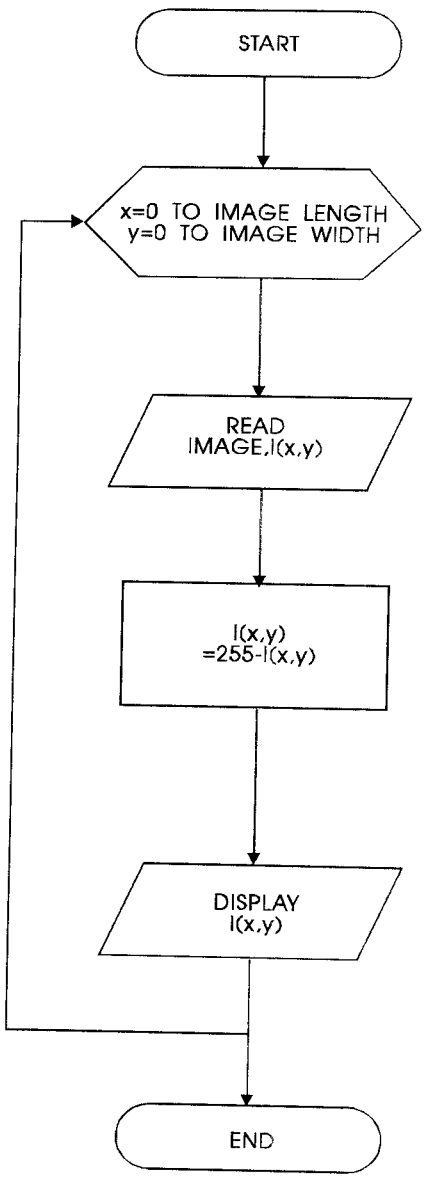
11. Plot the image array, end graphics

This function places a mirror in two images appear as if they are mirror images to each other.

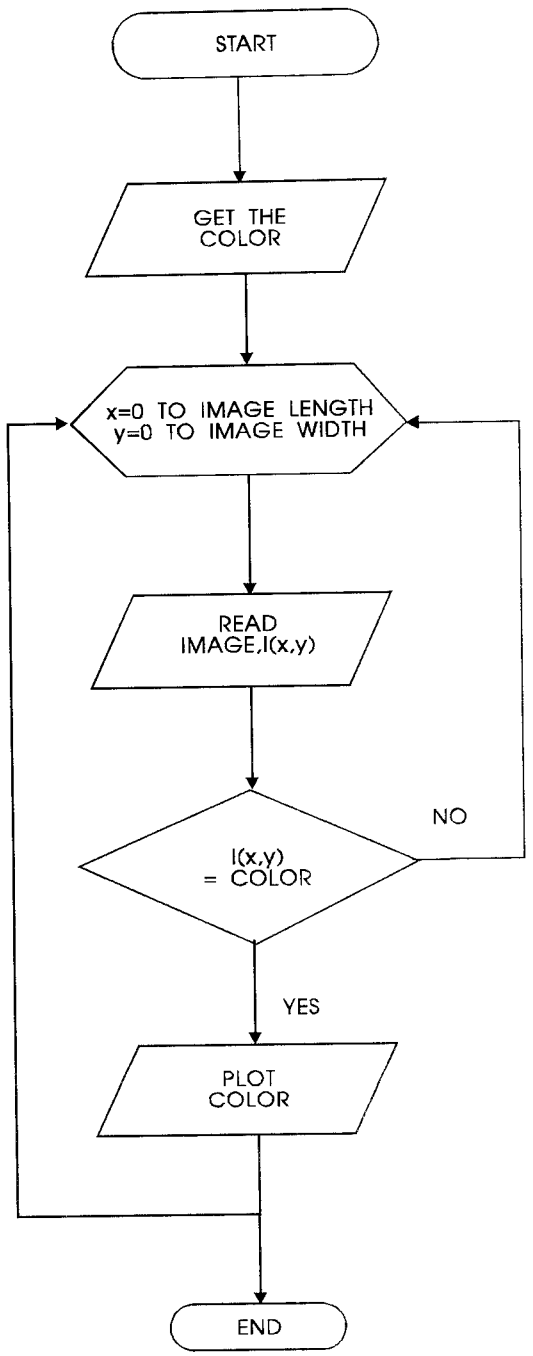


Flow Chart

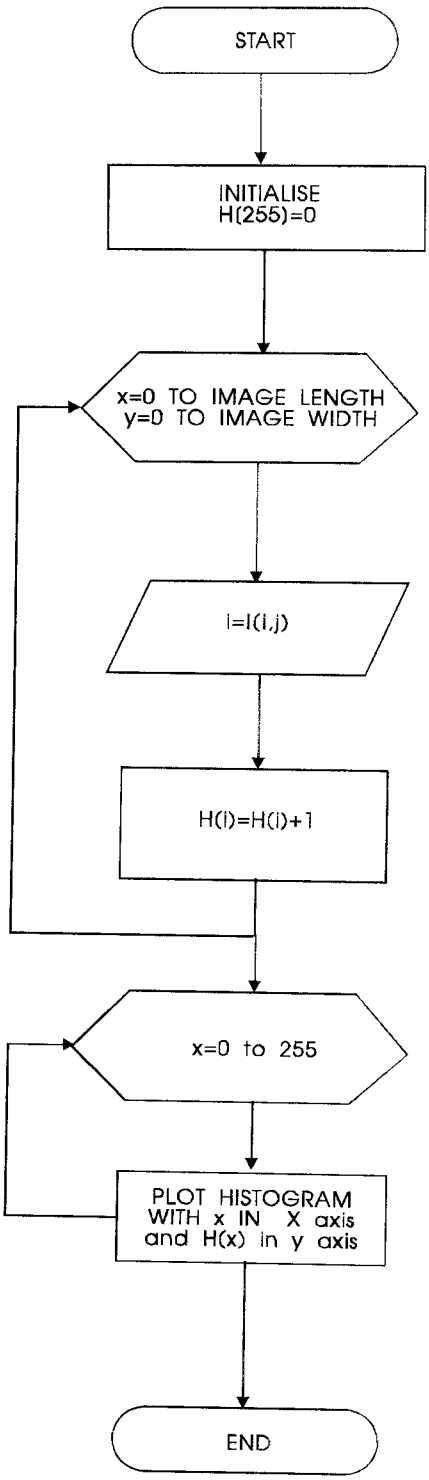
NEGATION


START
x=0 TO IMAGE LENGTH
y=0 TO IMAGE WIDTH
READ
IMAGE,I(x,y)
I(x,y)
=255-I(x,y)
DISPLAY
I(x,y)
END

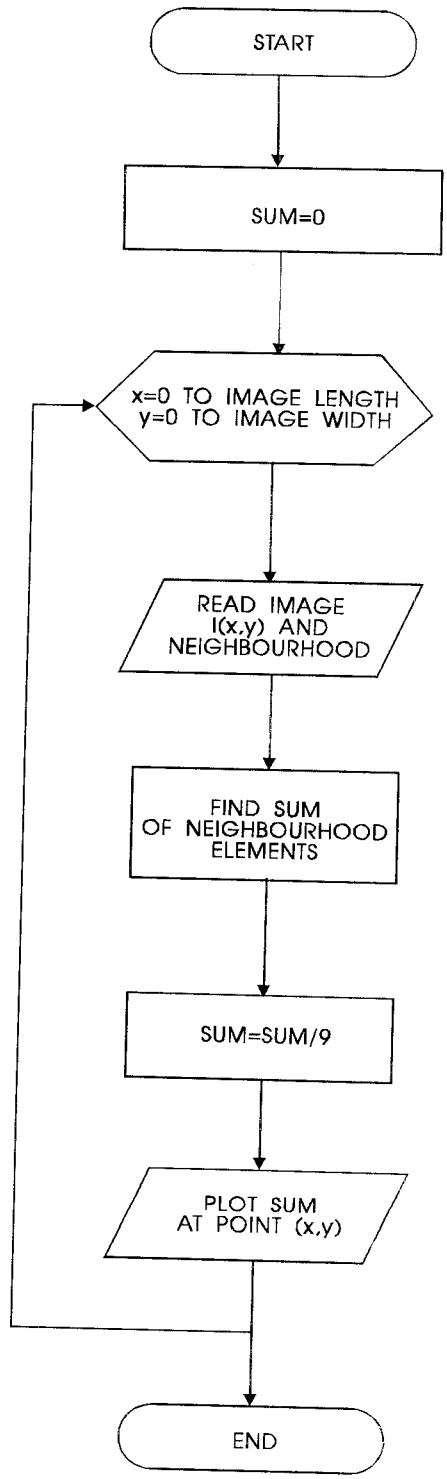
COLOUR DETECTION

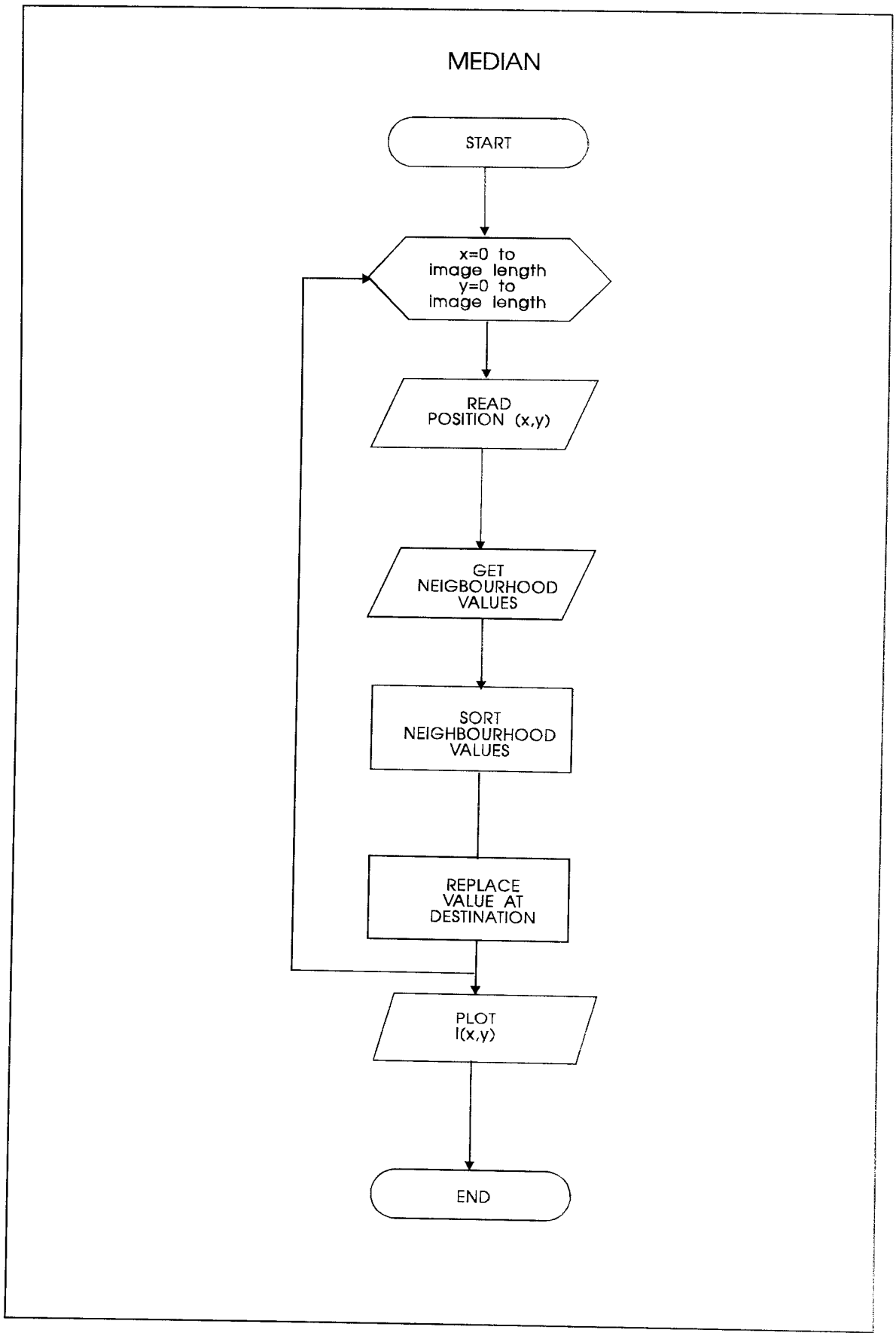


HISTOGRAM

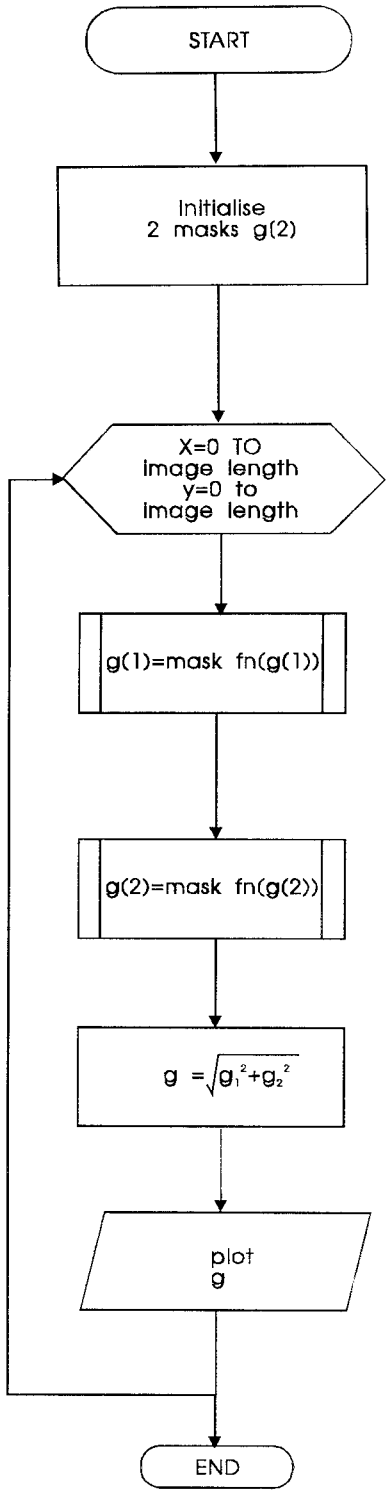

START
INITIALISE
H(255)=0
x=0 TO IMAGE LENGTH
y=0 TO IMAGE WIDTH
i=I(i,j)
H(i)=H(i)+1
x=0 to 255
PLOT HISTOGRAM
WITH x IN X axis
and H(x) in y axis
END

SMOOTHENING BY AVERAGING

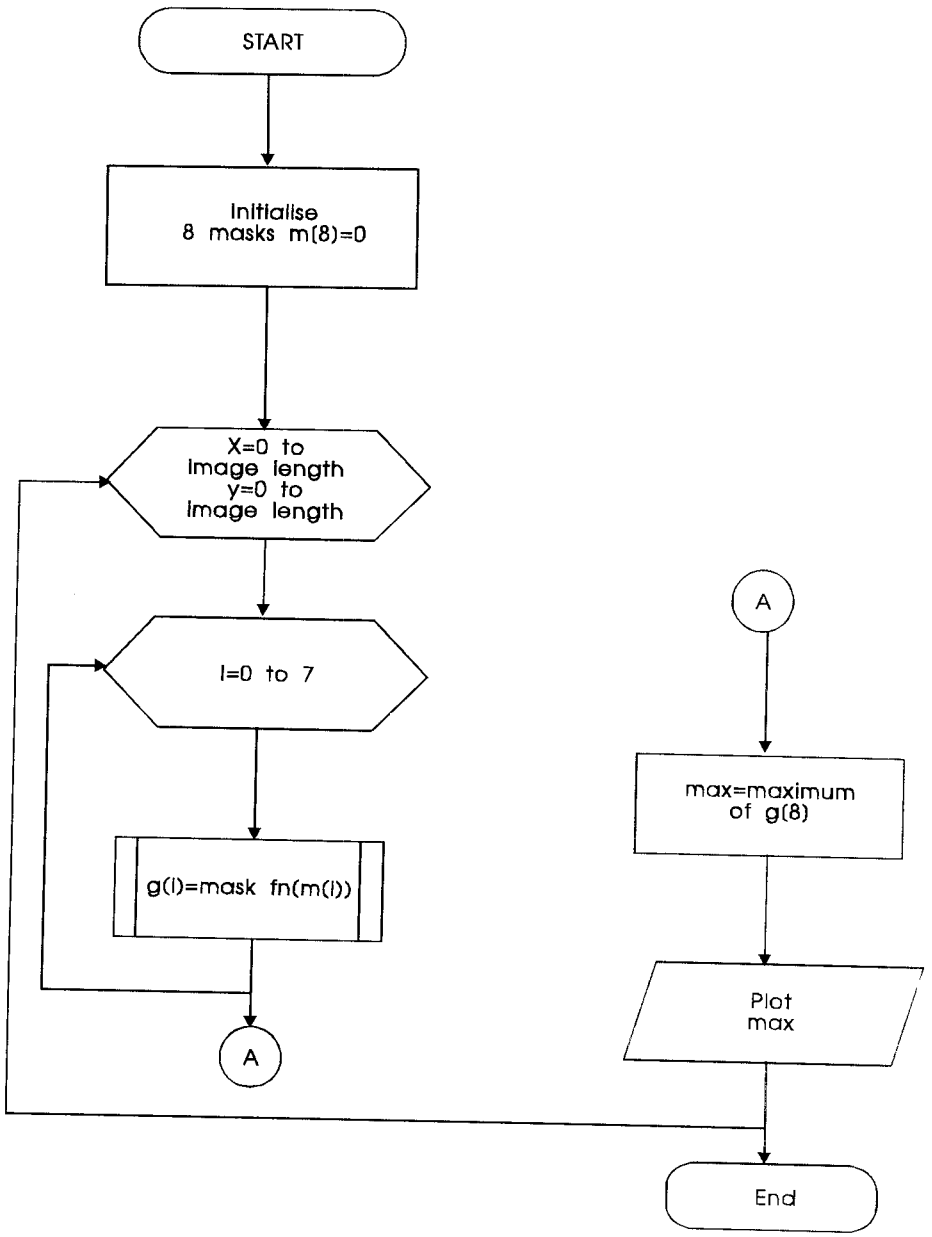



MEDIAN
START
x=0 to
image length
y=0 to
image length
READ
POSITION (x,y)
GET
NEIGBOURHOOD
VALUES
SORT
NEIGHBOURHOOD
VALUES
REPLACE
VALUE AT
DESTINATION
PLOT
I(x,y)
END

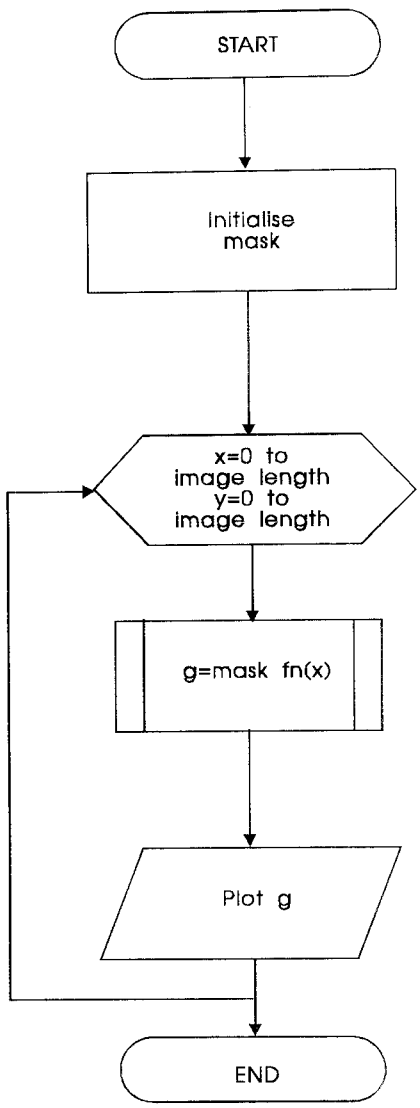
GRADIENT OPERATOR



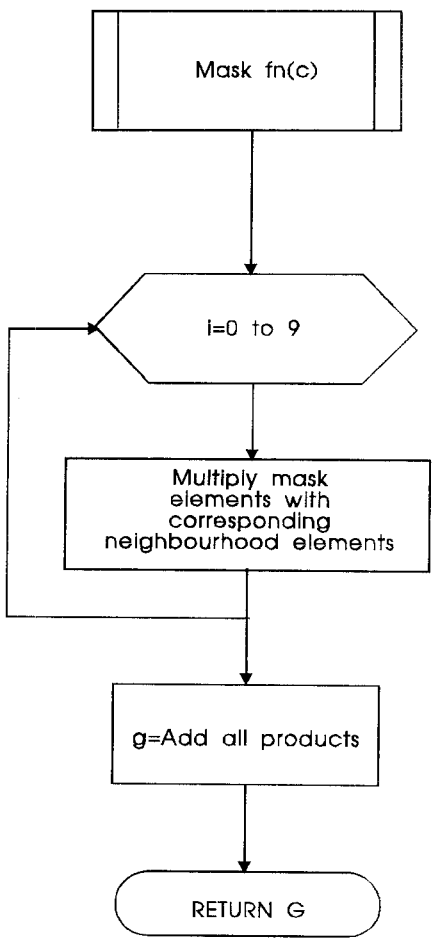
COMPASS OPERATOR



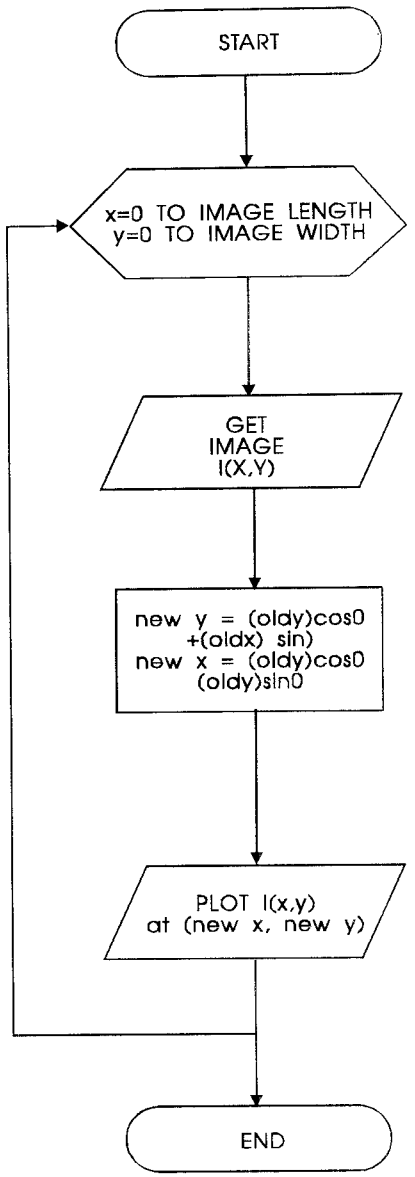
FILTERS


START
Initialise mask
x=0 to image length
y=0 to image length
g=mask fn(x)
Plot g
END

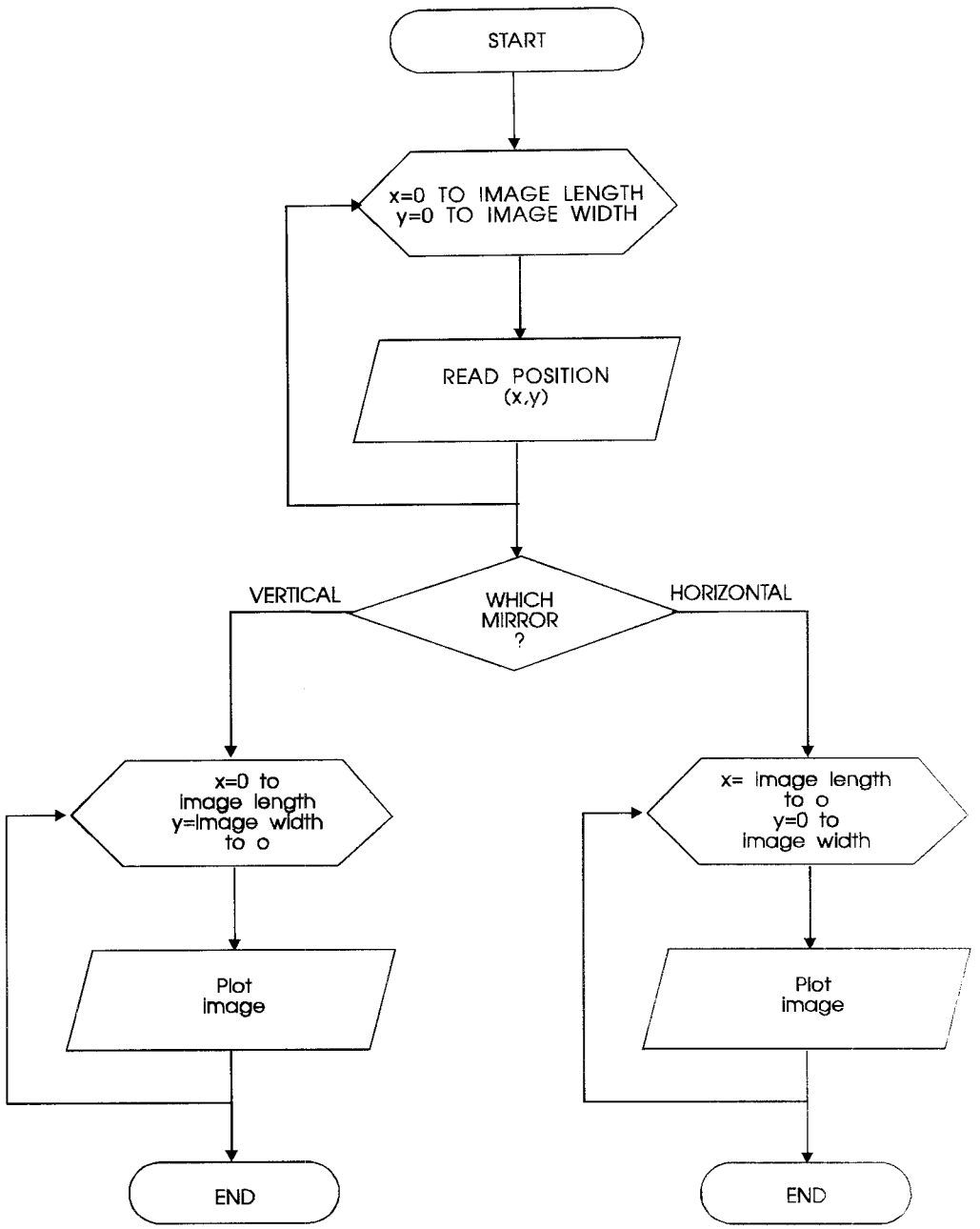
MASK SUBROUTINE

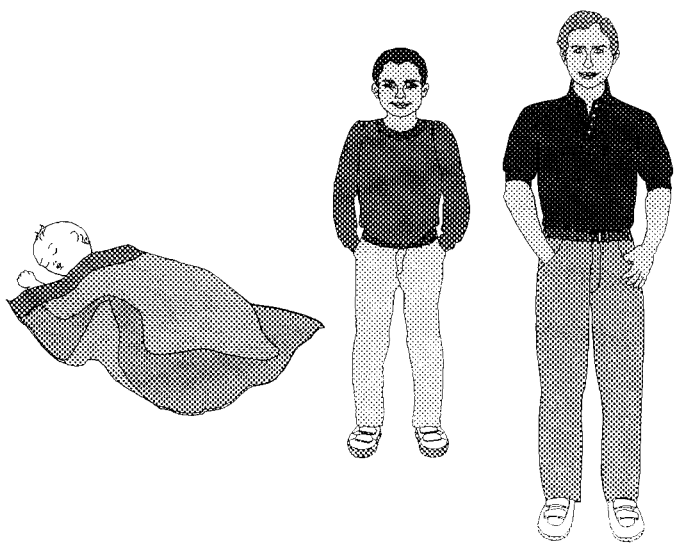


ROTATION


START
x=0 TO IMAGE LENGTH
y=0 TO IMAGE WIDTH
GET
IMAGE
I(X,Y)
new y = (oldy)cos0
+(oldx) sin)
new x = (oldy)cos0
(oldy)sin0
PLOT I(x,y)
at (new x, new y)
END

MIRROR


START
x=0 TO IMAGE LENGTH
y=0 TO IMAGE WIDTH
READ POSITION
(x,y)
WHICH
MIRROR
?
VERTICAL
HORIZONTAL
x=0 to
image length
y=image width
to o
Plot
image
END
x= image length
to o
y=0 to
image width
Plot
image
END



Source Code

GENERATING GRAY PALLETTE

```
/* Program for generating 256
 gray-levels that can be used
 as a palette for displaying
 8-bit gray-level images on the
 VGA screen. */

#include <stdio.h>
#include <conio.h>

main()
   {
   int i,color,r,g,b;
   float y,scale;
   FILE *fptr,*fptr1;

   clrscr();
   fptr=fopen("BWPAL.DAT","wb");
   scale=255.0/63.0;
   for(i=0;i<256;i++)
     {
    for(r=0;r<64;r++)
        for(g=r-2;g<(r+2);g++)
          for(b=r-2;b<(r+2);b++)
            {
            if((g<0)||(b<0)) continue;
            y=0.59*(float)g+0.30*(float)r+0.11*(float)b;
            color=(int)(y*scale+0.5);
             if(color==i)
               {
               putc(r,fptr);
               putc(g,fptr);
               putc(b,fptr);
               if(color==255) break;
               goto st;
               }
            }
          st: ;
           }
        fclose(fptr);
        fclose(fptr1);
        getch();
 }
```

KIRSH'S EDGE DETECTION

```
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <alloc.h>

# define x1 image[i-1][j+1]
# define x2 image[i][j+1]
# define x3 image[i+1][j+1]
# define x4 image[i-1][j]
# define x5 image[i+1][j-1]
# define x6 image[i+1][j]
# define x7 image[i-1][j-1]
# define x8 image[i][j-1]
# define x9 image[i+1][j+1]
# define x 275
# define y 375

unsigned char **image;
int i,j;


long int process(int M[8][9])
{
 long int g;
 unsigned max;
 int b[9];
 for(int k=0;k<8;k++)
 {      b[k]=M[k][0]*x1 + M[k][1]*x2 + M[k][3]*x3 + M[k][4]*x4 + M[k][5]*x5 +
                                M[k][6]*x6 + M[k][7]*x7 + M[k][8]*x8 + M[k][8]*x9;
                b[k]=abs(b[k]);
 }
 max=b[0];
        for(int n=0;n<9;n++)
        {               if(b[n]>max) max=b[n];
        }
 return(max);
}
main()
{
```

```
  long int g;
  int gd=DETECT,gm;
  FILE *fp;
  int  M[8][9]={                { 5, 5, 5,-3, 0,-3,-3,-3,-3},
                                { 5, 5,-3, 5, 0,-3,-3,-3,-3},
                                { 5,-3,-3, 5, 0,-3, 5,-3,-3},
                                {-3,-3,-3, 5, 0,-3, 5, 5,-3},
                                {-3,-3,-3,-3, 0,-3, 5, 5, 5},
                                {-3,-3,-3,-3, 0, 5,-3,-3, 5},
                                {-3,-3, 5,-3, 0, 5,-3,-3, 5},
                                {-3, 5, 5,-3, 0, 5,-3,-3,-3}
                             };
 initgraph(&gd,&gm,"c:\\bc4\\bgi");
 fp=fopen("c:\\image.dat","r");
 // ALLOCATE BUFFER
        image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
         if(image==0)
                { printf("Unable to allocate image\n");
                  exit(0);
                }
         for(i=0; i<y;i++)
                { image[i]=(unsigned char  *) farmalloc(x
                  * sizeof (unsigned char));
                  if(image[i]==0)

                        { printf("\nUnable to allocate image[%d]",i);
                          exit(0);
                        }
                        }
 for(i=375;i>0;--i)
  for(j=0;j<=275;++j)
  {       image[i][j]=getc(fp);
           putpixel(j,i,image[i][j]);
           printf("%d %d\n",i,j);
         }
for(i=375;i>0;--i)
 for(j=0;j<=275;++j)
         {  g=process(M);
                putpixel(j+250,i,g);
         }
 getch();
 closegraph();
}
```

FILTER

```
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# define X 100
# define Y 100

# define lim 10
# define x1 image[i-1][j+1]
# define x2 image[i][j+1]
# define x3 image[i+1][j+1]
# define x4 image[i-1][j]
# define x5 image[i][j]
# define x6 image[i+1][j]
# define x7 image[i-1][j-1]
# define x8 image[i][j-1]
# define x9 image[i+1][j-1]

#define m1 m[0]
#define m2 m[1]
#define m3 m[2]
#define m4 m[3]
#define m5 m[4]
#define m6 m[5]
#define m7 m[6]
#define m8 m[7]
#define m9 m[8]
#define x 640
#define y 480
unsigned char **image;
int i,j;
main()
{
        int gd = DETECT, gm;
        long int process(int*);
        initgraph(&gd, &gm, "c:\\tc\\bgi");
        image = (unsigned char  ** )farmalloc( y*
                                        sizeof(unsigned char  *));
        if(image==0)
                { printf("Unable to allocate image\n");
```

```
                    exit(0);
                   }

          for(i=0; i<y;i++)
                   { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                          { printf("\nUnable to allocate image[%d]",i);
                           exit(0);
                          }
                   }

  int m[9]={-1,-1,-1 ,-1,9,-1,-1,-1,-1};/*HPF1*/

                   for(i=0;i<=X-1;j++)
          {         for(j=0;j<=Y-1;j++)
                                  a[i][j]=getpixel(i,j);
          }
                  long int g,gx,gy;
                  for(i=0;i<=375;i++)
          {         for(int j=0;j<=275;j++)
                    {
                          g=process(m);
                          putpixel(i,j,g);
                    }
          }
          closegraph();
 }

 long int process(int *m)
 {                long int g,gx,gy;
                  g=m1*x1 + m2*x2 + m3*x3 + m4*x4 +
                         m5*x5 + m6*x6 +m7*x7 +m8*x8 + m9*x9;
                  putpixel(i,j,g);
                  getch();
 }
```

NEGATION

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
void main()
{
 int i,j,gd=DETECT,gm;
 unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }
          for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                          { printf("\nUnable to allocate image[%d]",i);
                            exit(0);
                          }
                           }
 char infile[14];
 FILE *fp;
 initgraph(&gd,&gm,"c:\\bc4\\bin");
 fp=fopen("c:\\image.dat","r");
 for(i=375;i>0;--i)
  for(j=0;j<=275;++j)
  {       image[i][j]=fgetc(fp);
                  putpixel(j,i,image[i][j]);
          }
          fclose(fp);
          for(i=375;i>0;--i)
           for(j=0;j<=275;++j)
           {
            putpixel(j+300,i,~(image[i][j]));
           }
 closegraph();
 }
```

COLOUR DETECTION

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
#include <process.h>
# include <alloc.h>
# define x 640
# define y 480
void main()
{
 int i,j,gd=DETECT,gm;
 unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }

          for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                            { printf("\nUnable to allocate image[%d]",i);
                              exit(0);
                            }
                             }
int ch;
char infile[14];
FILE *fp;
initgraph(&gd,&gm,"c:\\bc4\\bin");
fp=fopen("c:\\image.dat","r");
for(i=375;i>0;--i)
 for(j=0;j<=275;++j)
 {      image[i][j]=fgetc(fp);
        putpixel(j,i,image[i][j]);
        }
        fclose(fp);
 outtextxy(350,180,"Enter your choice :");
 gotoxy(40,1);
```

```
 printf("BLACK=0     BLUE =1        GREEN=2");
 gotoxy(40,2);
 printf("CYAN =3     RED  =4        MAGENTA=5");
 gotoxy(40,3);
 printf("BROWN=6     LIGHTGRAY=7    DARKGRAY=8");
 gotoxy(40,4);
 printf("LIGHTBLUE=9 LIGHTGREEN=10  LIGHTCYAN=11 ");
 gotoxy(40,5);
 printf("LIGHTRED=12 LIGHTMAGENTA=13 YELLOW=14");
 gotoxy(40,6);
 printf("WHITE=15 ");
 gotoxy(65,12);
 scanf("%d",&ch);
 for(i=375;i>0;--i)
        for(j=0;j<=275;++j)
        {  if((image[i][j]%16)==ch)  putpixel(j,i,ch);

                else                              putpixel(j,i,0);
        }
 outtextxy(50,400,"after processing");
getch();
closegraph();
}
```

HISTOGRAM

```
# include <stdio.h>
# include<iostream.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
void hist(void)
{
 int i,j,gd=DETECT,gm;
 int hist[255],k,r,data;
  char infile[14];
 FILE *fp;
  unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
           if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }

           for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                           { printf("\nUnable to allocate image[%d]",i);
                             exit(0);
                           }
                           }
initgraph(&gd,&gm,"c:\\bc4\\bin");
fp=fopen("c:\\image.dat","r");
for(i=375;i>0;--i)
 for(j=0;j<=275;++j)
 {       image[i][j]=fgetc(fp);
         putpixel(j,i,image[i][j]);
         }
         fclose(fp);
         outtextxy(300,200,"Press any key to plot histogram - plot");
```

```
        getch();
        cleardevice();
        fp=fopen("c:\\display.dat","r");
  outtextxy(200,460,"Histogram plot");
        for(k=0;k<=255;k++)
  hist[k]=0;
       for(i=0;i<=375;++i)
        for(j=0;j<=275;++j)
        { data=image[i][j];
              hist[data]++;
        }
       for(k=0;k<=255;k++)
        {
              r=hist[k];
              for(i=r;i>=1;--i)
              bar(100+k,440-i,102+k,440);
        }
}

void main()
{
hist();
getch();
}
```

SMOOTHENING

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
# define x1 image[i-1][j+1]
# define x2 image[i][j+1]
# define x3 image[i+1][j+1]
# define x4 image[i-1][j]
# define x5 image[i+1][j-1]
# define x6 image[i+1][j]
# define x7 image[i-1][j-1]
# define x8 image[i][j-1]
# define x9 image[i+1][j+1]
void main()
{
 int i,j,gd=DETECT,gm;
 unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }

          for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                            { printf("\nUnable to allocate image[%d]",i);
                              exit(0);
                            }
                             }
 int g;
 char infile[14];
 FILE *fp;
 initgraph(&gd,&gm,"c:\\bc4\\bin");
```

```
fp=fopen("c:\\image.dat","r");
for(i=375;i>0;--i)
 for(j=0;j<=275;++j)
 {       image[i][j]=fgetc(fp);
         putpixel(j,i,image[i][j]);
         }
         fclose(fp);
         for(i=375;i>0;--i)
          for(j=0;j<=275;++j)
          { g=(x1+x2+x3+x4+x5+x6+x7+x8+x9)/9 ;
                          putpixel(j+300,i,g);
           }
getch();
closegraph();
}
```

MIRRORING

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
#include <process.h>
# include <alloc.h>
# define x 640
# define y 480
void main()
{
int i,j,gd=DETECT,gm;
unsigned char **image;
 image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                { printf("Unable to allocate image\n");
                  exit(0);
                }
          for(i=0; i<y;i++)
                { image[i]=(unsigned char  *) farmalloc(x
                  * sizeof (unsigned char));
                  if(image[i]==0)
                        { printf("\nUnable to allocate image[%d]",i);
                          exit(0);
                        }
                        }
  char infile[14];
  FILE *fp;
  initgraph(&gd,&gm,"c:\\bc4\\bin");
  fp=fopen("c:\\image.dat","r");
  for(i=375;i>0;--i)
  for(j=0;j<=275;++j)
  {       image[i][j]=fgetc(fp);
                putpixel(j,i,image[i][j]);
        }
        fclose(fp);
        for(i=375;i>0;--i)
         for(j=0;j<275;++j)
         {
          putpixel(550-j,i,image[i][j]);
         }
 getch();
 closegraph();
}
```

ROTATION

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
void main()
{
 int i,j,gd=DETECT,gm,cal;
 unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }

          for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                              { printf("\nUnable to allocate image[%d]",i);
                                exit(0);
                              }
                               }
 float ang,in,jn;
 char infile[14];
 FILE *fp;
 initgraph(&gd,&gm,"c:\\bc4\\bin");
 fp=fopen("c:\\image.dat","r");
 for(i=375;i>0;--i)
  for(j=0;j<=275;++j)
  {       image[i][j]=fgetc(fp);
          putpixel(j,i,image[i][j]);
          }
 outtextxy(300,390,"Enter angle");
 gotoxy(50,25);
 scanf("%f",&ang);
 ang*=(3.14/180.0);
          cleardevice();
          rewind(fp);
```

```
        for(i=375;i>0;--i)
         for(j=0;j<=275;++j)
          {
                in=i*cos(ang)+j*sin(ang);
                jn=j*cos(ang)-i*sin(ang);
                putpixel(jn+450,in,image[i][j]);
          }
 outtextxy(300,420,"After rotating");
getch();
closegraph();
}
```

DISPLAYING BLACK &WHITE MAGE

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#include <io.h>
#include <ctype.h>
#include <graphics.h>

#ifndef _DAC256_
#define _DAC256_
typedef unsigned char DacPalette256[256][3];
#endif

extern int far _Cdecl Svga256_fdriver[];

/* These are the currently supported modes */
#ifndef SVGA320x200x256
#define SVGA320x200x256          0      /* 320x200x256 Standard VGA */
#define SVGA640x400x256          1      /* 640x400x256 Svga/VESA */
#define SVGA640x480x256          2      /* 640x480x256 Svga/VESA */
#define SVGA800x600x256          3      /* 800x600x256 Svga/VESA */
#define SVGA1024x768x256 4               /* 1024x768x256 Svga/VESA */
#define SVGA640x350x256          5      /* 640x350x256 Svga */
#define SVGA1280x1024x256        6      /* 1280x1024x256 VESA */
#endif

#ifndef XNOR_PUT
#define XNOR_PUT  5
#define NOR_PUT           6
#define NAND_PUT  7
#define TRANS_COPY_PUT8
#endif

void display(void);
int prompt(void);
void clear_scr(void);
void del_chr(int x,int y);
FILE *fptrc;
int huge DetectVGA256(void);
void getvgapalette256(DacPalette256 *);
```

```
void setvgapalette256(DacPalette256 *);
int x,y,image_length,image_width,xstart,ystart;
int i,j,d,th,tw,W;
unsigned char ch;
unsigned char color;
char file_name[80];
float nsq;
FILE *fptri;

void main()
 {
 int graphdrv;
 int graphmode;
 int Gd=DETECT,Gm;
 int ch;
 int xasp,yasp;
 DacPalette256 a,b;

clrscr();
installuserdriver("Svga256",DetectVGA256);
initgraph(&Gd,&Gm,"");
getvgapalette256(&b);

fptrc=fopen("bwpal.dat","rb");
 if(fptrc==NULL)
   {
   printf("\n Unable to locate the gray level pallette.");
   printf("\n Install \"bwpal.dat\" in the same directory as");
   printf("\n this program.");
   exit(1);
   }
 for(i=0;i<256;i++)
    {
   a[i][0]=(char)getc(fptrc);
   a[i][1]=(char)getc(fptrc);
   a[i][2]=(char)getc(fptrc);
   }
  fclose(fptrc);
setvgapalette256(&a);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
setcolor(200);
th=textheight("k");
tw=textwidth("k");
W=getmaxx();
```

```
  getaspectratio(&xasp,&yasp);
  setaspectratio(xasp,yasp);
     /*  Displaying image. */
      while(1)
       {
      i=prompt();

      if(i==1)
          {
           clear_scr();
           display();
           fclose(fptri);
           outtextxy(1,1,"Press any key to continue. ESC to exit.");
           ch=getch();
           if(ch==27) break;
           }
       else
           {
           outtextxy(1,1.3*th,"Press any key to continue. ESC to exit.");
           ch=getch();
           clear_scr();
           if(ch==27) break;
            }
          clear_scr();
      }

   fclose(fptri);
   setvgapalette256(&b);
   graphdefaults();
   closegraph();

    }

void clear_scr()
 {
 int poly[8];
 poly[0]=0;
 poly[1]=0;
 poly[2]=W;
 poly[3]=0;
 poly[4]=W;
 poly[5]=2.5*th;
 poly[6]=0;
 poly[7]=2.5*th;
```

```
 setcolor(0);
 setfillstyle(SOLID_FILL,0);
 fillpoly(4,poly);
 setcolor(200);
 }

void display(void)
 {
   clear_scr();
    for(y=0; y<image_length; y++)
        for(x=0; x<image_width; x++)
          {
           color=getc(fptri);
           putpixel(x+xstart,y+ystart,~color);
           }
  }
int prompt(void)
  {
  char f[2];
  int k,i,x,y;

  outtextxy(1,1,"Enter name of file -->");
  k=23*tw;
  i=0;
  y=1;
  x=k;
  while(1)
   {
   f[0]=(char)getch();
   if(f[0]==27) return(0);
   f[1]='\0';
   if(f[0]==13) break;
   file_name[i]=f[0];
   if((f[0]==0)||(f[0]==8))
    {
        if(f[0]==0)
          if(getch()!=75) continue;
        i--;
        k-=tw;
        x=k;
        del_chr(x,y);
        continue;
        }
  i++;
```

```
    outtextxy(k,1,f);
    k+=tw;
     }
    file_name[i]='\0';
   fptri=fopen(file_name,"rb");
   clear_scr();
   if(fptri==NULL)
     {
     clear_scr();
     outtextxy(1,1,"file does not exist.");
     return 0;
     }
  nsq=(float)filelength(fileno(fptri));
  clear_scr();
  outtextxy(1,1,"Is this a square image?");
  while(((ch=tolower(getch()))!='y')&&(ch!='n'));
  f[0]=ch;
  f[1]='\0';
  outtextxy(25*tw,1,f);
  switch(ch)
    {
   case 'y':
         image_width=image_length=(int)sqrt((double)nsq);
         break;
   case 'n':
         clear_scr();
         outtextxy(1,1,"Enter image_width in pixels --> ");
x=32*tw;
image_width=0;
while(1)
 {
 f[0]=(char)getch();
 f[1]='\0';
 if(f[0]==13) break;
 if((f[0]==0)||(f[0]==8))
   {
       if(f[0]==0)
         if(getch()!=75) continue;
       image_width=image_width/10;
       x-=tw;
       del_chr(x,y);
       if(x<(22*tw)) x=22*tw;
       continue;
       }
```

```
  if((f[0]<48)||(f[0]>57))
   {
        del_chr(x,y);
        continue;
        }
  image_width=image_width*10+(f[0]-48);
  outtextxy(x,y,f);
  x+=tw;
   }

        image_length=(int)(nsq/image_width);
  }
 clear_scr();
 outtextxy(1,1,"Enter x position --->");
 x=22*tw;
 xstart=0;
 while(1)
  {
  f[0]=(char)getch();
  f[1]='\0';
  if(f[0]==13) break;
  if((f[0]==0)||(f[0]==8))
    {
        if(f[0]==0)
          if(getch()!=75) continue;
        xstart=xstart/10;
        x-=tw;
        del_chr(x,y);
        if(x<(22*tw)) x=22*tw;
        continue;
        }
  if((f[0]<48)||(f[0]>57))
   {
        del_chr(x,y);
        continue;
        }
  xstart=xstart*10+(f[0]-48);
  outtextxy(x,y,f);
  x+=tw;
   }
 clear_scr();
 outtextxy(1,1,"Enter y position --->");
 x=22*tw;
 ystart=0;
```

```
 while(1)
  {
  f[0]=(char)getch();
  f[1]='\0';
  if(f[0]==13) break;
        if((f[0]==0)||(f[0]==8))
    {
         if(f[0]==0)
          if(getch()!=75) continue;
         ystart=ystart/10;
         x-=tw;
         del_chr(x,y);
         if(x<(22*tw)) x=22*tw;
         continue;
         }
   if((f[0]<48)||(f[0]>57))
     {
         del_chr(x,y);
         continue;
         }
   ystart=ystart*10+(f[0]-48);
   outtextxy(x,y,f);
   x+=tw;
    }
  return 1;
   }

void del_chr(int x,int y)
  {
  int poly[8];
  poly[0]=x;
  poly[1]=y;
  poly[2]=x+tw;
  poly[3]=y;
  poly[4]=x+tw;
  poly[5]=y+th;
  poly[6]=x;
  poly[7]=y+th;
  setcolor(0);
  setfillstyle(SOLID_FILL,0);
  fillpoly(4,poly);
  setcolor(200);
  }
int huge DetectVGA256()
```

```
{
 int Vid;

 printf("Which video mode would you like to use? \n");
 printf("  0) 320x200x256\n");
 printf("  1) 640x400x256\n");
 printf("  2) 640x480x256\n");
 printf("  3) 800x600x256\n");
 printf("  4) 1024x768x256\n\n>");
 scanf("%d",&Vid);
 return Vid;
}


void getvgapalette256(DacPalette256 *PalBuf)
{
 struct REGPACK reg;

 reg.r_ax = 0x1017;
 reg.r_bx = 0;
 reg.r_cx = 256;
 reg.r_es = FP_SEG(PalBuf);
 reg.r_dx = FP_OFF(PalBuf);
 intr(0x10,&reg);
}

void setvgapalette256(DacPalette256 *PalBuf)
{
 struct REGPACK reg;

 reg.r_ax = 0x1012;
 reg.r_bx = 0;
 reg.r_cx = 256;
 reg.r_es = FP_SEG(PalBuf);
 reg.r_dx = FP_OFF(PalBuf);
 intr(0x10,&reg);
}
```

MEDIAN FILTER

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
# define x1 image[i-1][j+1]
# define x2 image[i][j+1]
# define x3 image[i+1][j+1]
# define x4 image[i-1][j]
# define x5 image[i+1][j-1]
# define x6 image[i+1][j]
# define x7 image[i-1][j-1]
# define x8 image[i][j-1]
# define x9 image[i+1][j+1]
int i,j;
unsigned char **image;
void main()
{
 unsigned process();
 int gd=DETECT,gm,cal;
 image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                { printf("Unable to allocate image\n");
                  exit(0);
                }

          for(i=0; i<y;i++)
                { image[i]=(unsigned char  *) farmalloc(x
                  * sizeof (unsigned char));
                  if(image[i]==0)

                        { printf("\nUnable to allocate image[%d]",i);
                          exit(0);
                        }
                        }
unsigned long int gx,gy,g;
 char infile[14];
 FILE *fp;
```

```
 initgraph(&gd,&gm,"c:\\bc4\\bin");
 fp=fopen("c:\\image.dat","r");
 for(i=375;i>0;--i)
  for(j=0;j<=275;++j)
  {       image[i][j]=getc(fp);
          putpixel(j,i,image[i][j]);
          }
          outtextxy(50,400,"Before processing");
          outtextxy(350,400,"After processing");
          fclose(fp);
          for(i=375;i>0;--i)
           for(j=0;j<=275;++j)
            {
                  g=process();
                   putpixel(j+300,i,g);
            }
 getch();
 closegraph();
}
unsigned process()
{  int b[9];
   int t;
         b[0]=x1;    b[1]=x2;    b[2]=x3;    b[3]=x4;
         b[4]=x5;    b[5]=x6;    b[6]=x7;    b[7]=x8;
         b[8]=x9;
    for(int m=0;m<9;m++)
  {       for(int n=0;n<9;n++)
                        if(b[n-1]<b[n])
                         {     t=b[n];
                                       b[n]=b[n-1];
                                       b[n-1]=t;
                         }
         }

         return(b[5]);
}
```

READING BITMAP

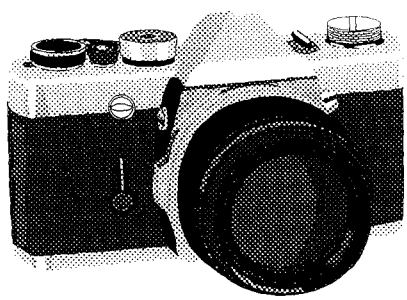
```
#include<dos.h>
#include<graphics.h>
#include<stdio.h>
#include<string.h>
#include<io.h>
#include<process.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>

struct head
{
        char arr1;
        char arr2;
        long int size;
        long int u1;
        long int u2;
        long int u3;
        long int width;
        long int height;
        int u4;
        int u5;
        };
        struct head h1;

main()
{
int gd=DETECT,gm;
int i,j;
detectgraph(&gd,&gm);
initgraph(&gd,&gm," ");
        FILE *fp;
        char bname[13];
        settextstyle(0,0,3);
        outtextxy(100,200,"Enter the file");
        scanf("%s",bname);
        if(fopen(bname,"r")==NULL)
        {
        printf("no such file..");
         exit(0);
         }
```

```
        fp=fopen(bname,"r");
        fread(&h1,sizeof(struct head),1,fp);
        cleardevice();
        fseek(fp,1079,0);
        int h,w;
        unsigned char val1;
        for(h=h1.height;h>0;h--)
        {
                for(w=0;w<h1.width;w++)
                {
                        val1=fgetc(fp);
                        putpixel(w,h,val1);
                }
        }
        fclose(fp);
        sleep(2);
        closegraph();
}
```

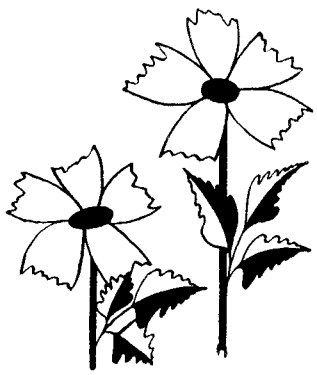
EDGE DETECTION

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
# include <process.h>
# include <alloc.h>
# define x 640
# define y 480
# define x1 image[i-1][j+1]
# define x2 image[i][j+1]
# define x3 image[i+1][j+1]
# define x4 image[i-1][j]
# define x5 image[i+1][j-1]
# define x6 image[i+1][j]
# define x7 image[i-1][j-1]
# define x8 image[i][j-1]
# define x9 image[i+1][j+1]
void main()
{
 int i,j,gd=DETECT,gm,cal;
 unsigned char **image;
  image = (unsigned char  ** )farmalloc( y*sizeof(unsigned char  *));
          if(image==0)
                  { printf("Unable to allocate image\n");
                    exit(0);
                  }

          for(i=0; i<y;i++)
                  { image[i]=(unsigned char  *) farmalloc(x
                    * sizeof (unsigned char));
                    if(image[i]==0)

                              { printf("\nUnable to allocate image[%d]",i);
                                exit(0);
                              }
                              }
unsigned long int gx,gy,g;
 char infile[14];
 FILE *fp;
 initgraph(&gd,&gm,"c:\\bc4\\bin");
 fp=fopen("c:\\display.dat","r");
```

```
for(i=0;i<325;++i)
 for(j=0;j<=275;++j)
 {       image[i][j]=fgetc(fp);
         putpixel(j,i,image[i][j]);
         }
         fclose(fp);
         cleardevice();
         for(i=0;i<=325;++i)
          for(j=0;j<=275;++j)
           {  gx=(x7 + 2*x8 + x4) - (x1 + 2*x2 + x3);
                   gy=(x3 + 2*x6 + x4) - (x1 + 2*x4 + x7);
                   g=sqrt(gx*gx + gy*gy);
                   putpixel(i,j,g);
           }
 getch();
}
```



Results...



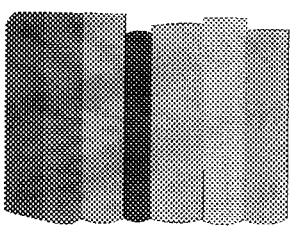
Conclusion

CONCLUSION

We have hereby successfully explored certain topics in this vast and ever growing field of image processing. Algorithms relating to various image processing techniques have been developed, studied and software has been implemented in C language.

This is a topic of fundamental importance in image communications and storage, thus project has paved us the way to enter into this field.

One restriction of our project is that it can be implemented only for 256 color bit map. In future these algorithms can be implemented in real time applications if parallel processing is used.



Bibliography

BIBLIOGRAPHY

1. Fundamentals of Digital Image Processing - *Anil K.Jain*

2. Image Processing - *M.A.Sid-Ahmed*

3. Digital Image Processing - **Pratt**

4. Exploring C - *Yashwant Khanetkar*

5. C++ and C.-Graphics - *Vijay Mukhi*

6. C. Trilogy - *Bloom*