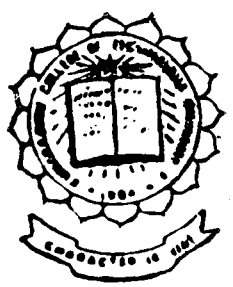
IMAGE PROCESSING IN C

PROJECT REPORT

BY

N. C. RANGANATHAN
N. GIRIDHARAN
V. KANNAN
S. ELIZABETH

1991-92



GUIDED BY

Mr. K. RAMPRAKASH, M.E

Submitted 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

Department of Electronics and Communication Engineering

Kumaraguru College of Technology

COIMBATORE-641 006.

Department of Electronics & Communication Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE - 641 006

Certificate

This is to certify that the report entitled

IMAGE PROCESSING IN 'C'

has been submitted by

Mr. .....

In partial fulfilment for the award of Bachelor of Engineering in the Electronics & Communication Engineering branch offered by the Bharathiar University, Coimbatore – 641 046 during the academic year 1991–92.

.....

Guide

.....

Head of the Department

Certified that the Candidate with University Register No. ..... was examined in Project work Viva–Voce by us on .....

.....

Internal Examiner

.....

External Examiner

CONTENTS

ACKNOWLEDGEMENTS

We are grateful to our respected principal Major. **T.S.Ramamurthy** who encouraged and motivated all of us to proceed through and compute our project work.

We are thankful to our beloved Head of the Department **Prof. K. Palaniswami** for being the source of inspiration for us and for his valuable advices during the course of our project.

We express our gratitude from the bottom of our heart to our guide **Mr. K. Ramprakash**, who guided us in this project work.

We also thank **Prof. P. Shanmugam**, Head of the Department of Computer Science and Engg for providing us with the facilities available in the computer lab. We are indebted to the staff members of the CSE department for their valuable suggestions.

We thank the Lab technicians of the Computer lab and the ECE lab for their timely services during the course of our project.

We are particularly thankful to **Mr. K.Sutharsan** Kumar Consultant Madras for his valuable and timely help inspite of his busy schedule. We are also thankfulto **Mr. Govindaraj** of Chip-Brain systems, Pollachi and Mr. **Mathuranthagan** of Vigneshwar Computer Systems Pollachi for providing the required facilities.

Last but not the least, we express our sincere thanks to all the staff members and the student friends of ECE department who gave valuable suggestions with great care at various junctures.

SYNOPSIS

In this project, Image Processing Functions which alter the picture information in some manner are delt with. Various processing algorithms , that change the individual pixel values , based on their own values or those of the neighbourhood pixels, are discussed . These include point processes, geometric processes & frame processes.

The processing is done using image 'Pseudocoloring' technique. The entire image processing support functions were developed using TurboC. The integerated graphics environment provided by Borland's. TurboC serves as an ample tool in developing these function.

CHAPTER II POINT PROCESSES

2.1 Introduction:

Point processes are fundamental image processing operations. They are simplest and yet probably the most frequently used of image processing algorithms, they are natural starting place.

Point processes are algorithms that modify a pixel's value in an image based solely upon that pixel's value and sometimes its location. No other pixels values are involved in that transformation. In gneral point processes do not modify the spacial relationships within an image. For this reason, point processes cannot modify the detail contained in an image.

The point porcesses discussed are

a. Image brightening

b. Negative images

c. Image thresholding

d. Image pseudocolouring.

2.2 Image Brightening:

The appearence of an image can be visually enhanced by adjusting its brightness. Brightening is a point process that adds or subtracts a constant value to or from pixels in an image. Expressed algebrically, a pixel with intensity value V is transformed as follows:

$V = V + b$, Where b is the brightness constant, +ve or -ve. If b is +ve, the brightness of the pixels increases and if it is -ve, the brightness decreases.

2.3 Negative Images:

Negative images, resembling photographic negatives, are easy to produce with point process. The idea is to make the portion of an image that was once light, dark and that was once dark light.Image negation is accomplished by subtracting, the value of a pixel from the maximum possible pixel value of 63. The darkest areas of an images, which has pixel values of 0 or black are then trasnformed into the brightest white pixel value of 63. Conversely the brightest white pixels are converted to black.

Negation of a full image is useful when looking for detial in bright portions of an image. The human eye is much more capable of discerning detail in a dark area of image than in a light area.

2.4 Threshold Images:

Image thresholding is a technique for converting a continous tone image into a black and white image. Pixel values below a specified threshold are all convertd to black, whereas pixel values at or above the threshold are converted to white.. This technique has application ranging from art to machine vision. For artistic application, continous tone images that are correctly thresholded produce what is referred to as line art. Line art can then be used effectively in desk top publishing application and in sign and banner production. Thresholding is also used as a crude methods of obtaining hard printout of continuous tone images on a dot matrix printer.

In the field of machine vision, images are typically thresholded before an attempt is made at edge detection. In this case, thresholding eliminates from an image extraneous information that might upset the edge detection process.It is very important to select that threshold value correctly, however to ensure that not too much information is lost during thresholding process.

2.5 Image pseudo coloring:

Image pseudo coloring involves false coloring of gray scale images. This assigns different color values to different gray scales. The resulting is a pseudo colored images. Each pixel is assigned a color value, corresponding to its gray scale value.

CHAPTER III AREA PROCESSES

3.1 Introduction:

Area processes, also refereed to as group processes, use groups of pixels to derive information about an image. This is different from point processes, which use only a singel pixels information for performing the point process. The group of pixels used in area processing is referred to as the neighbourhood. The neighbourhood is generally a two dimensional matix of pixel values with each dimension having an odd number of elements. The pixel of interest the pixel whose old value is being replaced by its new value as a result of an algorithmic computation resides at the centre of neigbbourhood. Having a cluster of pixels of interest furnishes brightness trend information in two dimensions that is utilised by most area processes. Another, more proper, term for brightness trend information is spatial frequency.

Spatial frequency is defined as the rate of change of pixel brightness or intensity divided by the distance over which the change has occured. Spatial frequency has components both in the vertical and horizontal direction in an image. An image with high spatial frequency content contains large, closely spaced changes in pixel vlaues. An image of a black and white checkboard would contain high spatial frequency component. The smaller the squares the higher the frequency content. An image with low spatial

frequency contains large area of constant or slowly chaging pixel values. Images of clouds generaly have low spatial frequency content.

Having access to the spatial frequency information allows area processes to act as filters for removing or enhancing selective frequency components found in an image. Many area processes then falls into the general category of spatial filters. Spatial filters have a firm basis in mathematics.

Spatial filtering has many applications in image processing. It can be used for example, for extraction of image features (edge enhancement and detection), for sharpening an image, for smoothing an image, and for removal of random noise present in an image. These aspects of spatial filtering will be demonstrated in this section.Images will be used to show the effect of various area processes in action.

Here three area process algorithms are provided: Convolution, median filtering, and sobél edge detection. The median filter algorithm has single specific use, where as the convolution algorithm being more general in nature has many uses. The operation of each of these algorithms is similar to that utilised for point processes. That is:

a. A single pass is made over an input image on a pixel by pixel basis.

b. Each pixel in the input image is processed via a tranformation into a new value.

c. The new values for the pixel is placed into the output image buffer at the name location as it was taken from the input image buffer.

The difference between a point process and an area process is that, a point process utilizes only the value and sometimes the location of the input pixel in generating the output pixel.

3.2 Convolution:

Convolution is a very general purpose algorithm that can be used in performing a variety of area process transformations. Complex as convolution might sound it is actually quite easy to understand and implement. Figure1 illustrates the convolution process.

Each pixel in neighbour hood is multiplied by a similarly dimensioned convolution kernal; the sum that results replaces the value of the centre pixel of interest. Each element of convolution kernel is a wieghing factor also called a convolution coefficeint. The size and arrangements of the wieghing factors contained in a convolution kernel determine the type of area tranform that will be applied to the image data changing a weighing factor within a convolution kernel influences the magnitude and possibly the sign of the overall num and therfore affects the value given to the pixel of interest. Figure shows various convolution

kernels and the transfer functions they represent. As you can see most kernels are three by three and most have odd number of rows and columns . This format of concolution coefficients within the kernel has been accepted by the industry as a standard. A larger kernel seze increases the flexibility of the convolution processes.

Unfortunately, the simple weighted sum convolution calculation has some implementation details that complicate its realization. The first and foremost has to do with the edges of the image. As we move the convolution kernel with the pixel of interest under the centre of the kernel across an image a pixel at a time, we will have problems without claculations whenever we come to the borders of the images. That is because the weighing coeficients in the kernel are no longer positioned over nine pixels of the source image. In other words, the convolution kernel is, in effect, hanging over the edge of the image buffer. This perturbance happens at the top, left, right and bottom borders of an image. Several methods may be used to cope with this situation. The two most straight forward solutions are (1) the data at the edges of the image can be ignored, or (2) image data can be replicated to synthesis additional border data. Method (1) was utilised in the code provided in the area processing function library.

3.2.1 Various convolution kernels:

The various convolution kernels used in area processing are shown in fig 2.

Under close inspection, it is easy to understand why the blurring kernel produces much large values while the other kernels listed no well behaved. All kernels except blkurring kernel the num of all weighing coefficeints is sum of the 25 pixel values in the five by five neighbourhood. If for example, each of the 25 pixels had a value equal to one half of the maximum value or 32 the sum would equal 25x32 or 800. To bring this sum back to appropriate range, a scale value of 4 or 5 would be applied. This would reduce the value of the weighted sum to 50 or 25 respectively. The final choice of scale is somewhat subjective and will depend upon

The final consideration that must be taken into account in the implementation of a convolution algorithm is the sign of the calculated pixel values . When a convolution kernel contains negative weighing coefficients as most do it is possible to produce negative pixel intensity values. Even though negative intensity values are interesting, we cannot display them. For this reason, our convolution implementation sets negative pixel - intensity values to 0. Other methods could be used to deal with negative intensity values . For

instance, the abolute value of the intensity would be used instead of setting the value to 0.

3.2.2 Low pass spatial filters:

Low pass spatial filters leave the low frequency content of an image intact while attentuating the high frequency content. Low pas filters are good at reducing the visual noise contained in a image. They are also used to remove the high frequency content of an image so that the low frequency contents can be examined more closely. With the high frequencies gone, more subtle low frequency changes can be identified. The cut off of a low pass filter is determined by the size of the kernel and the kernel coefficients. Three different low pass frequency kernels are given in fig 2.The sum of the kernel values for all of the low pass filters is 1. This fact is important for understanding how low pass filter operate.

Consider a portion of an image without high frequency content. This means that the pixel values are of constant value or that they are changing slowly. As a low pass kernel is passed over this protion of the image, the new value for the pixel of interest the pixel centered under the kernel is calculated as the sum of the kernel coefficients times the neighbourhood pixel values. If all the neighbourhood pixel values are the same constant, the new pixel values is the same as the old value. This is the reason the sum of the coefficients is chosen to be 1. Low frequency content has

been preserved. As the kernel is moved over a portion of the image with high frequency content any rapid changes in intensity get averaged out with the remaining pixels in the neighbourhood there by lowering the high frequency content. The visual result of low pass filtering is a slight blur of image. This blur result because any sharp pixel transitions are averaged with their surroundings as the high frequency content is attenuated.

Contrary as it sounds, low pass filtering can be used to sharpen the appearene of an image. If a low pass filtered image is subtracted from the original image, the result is in relative increase in high frequency informational content without an increase in image noise.Subjectively the resultant image appears sharper than the original.

3.2.3 High-pass spatial filters

High-pass filters attenuate the high-frequency details of an image while leaving the low-frequency content intact. Relative to the high - frequnency content the low-frequency content, is attenuated. High pass filter is used whenever objects with high spatial frequency content need to be examined. The higher-frequency portions of an image will be highlighted (become brighter), while the lower-frequency portions become black. Image sharpness is sometimes enhanced with high-pass filtering at the expense of attenutated image noise. Edge enhancement of an image in also possible with high-pass filtering. Fig 2: shows three high-pass filter kernels.

The large center kernel co-efficients holds the key to the operation of high-pass filters. As the large centre co-efficient moves across a portion of an image with the high spatial frequency content (meaning a large-step change in fixed intensity) the new value of the pixel of interest is multiplied many times in value. The smaller negative co-efficients in the kernel clustered around the large center intensity are intensified, while areas of constant pixel(areas of low spatial frequencies) are not affected by this transformation.

3.3 Edge-Enhancement

Another area process that can be performed using convolution is edge enhancement. Edge enhancement is used as a preliminary step in image feature extraction and is image content is reduced and in many cases completely eliminated. For this reason the processed image may not closely resemble the original image. The brightness of an edge after enhancement is proportional to the change in brightness surrounding the edge in the original image.

Although edge enhancement is used mainly for machine vision, it does have other uses as well. For example, the edge information provided by an edge-enhancement process can

be added back into the original image to sharpen it. Edge enhancement can also be used as an artistic tool to produce striking outlined images. These images can be touched up with a point programme to produce real works of art.

In this section we will present three different edge-enhancement methods and one method of edge detection. All of the brightness trends in a pixel's neighbourhood are used to find and accent the edges contained in an image. By its very definition, an edge is a large change in intensity. It should be noted that all edge-enhancement algorithms that use convolution are linear. That is, they are made up of the sum of first-degree products. The Sobel edge-detection algorithm, however uses first derivatives to detect edges. The non-linear method does a better job of detecting edges feeling for greater accuracy that non linear processes can provide.

3.3.1 Laplacian Edge Enhancement

Lalplacian Edge Enhancement differs from the other enhancement methods to be discussed in being omni-directional. That is, it highlights edges regardless of direction. It is called Laplacian enhancement because this transformation approximates the Laplacian operator utilized

throughout mathematics and electronics. Laplacian edge enhancement generates sharper edge definition than do most other enhancement operations. Additionally, its highlights edges having both positive and negative brightness slopes. For these reasons, Laplacian edge enhancement finds use in many machine vision applications.

For the more mathematically oriented reader, the Laplacian of an function f(x, y) is

$$L(f(x, y)) = d\,f/dx + d\,f/dy$$

Where d f/dx is second partial derivative of f with respect to x and d f/dy is the second partial derivative of f with respect to y. For discrete functions the second partial derivatives can be approximated by

$$d\,f/dx = f(x+1)-2.f(x)+f(x-1) \quad \text{and}$$

$$d\,f/dy = f(y+1)-2.f(y)+f(y-1)$$

The Laplacian can therefore be approximated by

$$L(f(x,y)) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4.f(x,y)$$

This can be expressed as a convolution kernel that is convolved with f(x,y). The kernel becomes:

$$P(x, y) = \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

The Kernel is given as LAP 1.

All edge-enhancement operations, Laplacian included, attenuate, the low spatial frequencies of an image. Regions

of constant intensity or linearly increasing intensity become black as a result of these transformations, whereas regions of rapidly changing intensity values are highlighted. Convolution Kernels that attenuate low frequencies have coefficients that sum to 0.

3.3.2 The shift and difference edge enhancement

As the name implies, this algorithm enhances image edges by shifting an image by a pixel and then subtracting the shifted image from the original. The result of the subtraction is the measure of the slope of the brightness trend. In the area of constant pixel intensity, the subtraction will yield a slope of zero.Zero slope results in black pixel valued. In an area with large changes in intensity, an edge, for example, the subtraction will yield a large value for the slope, which will become light colored pixel. The larger the difference in intensities, the "whiter" the resultant pixel. Care must be taken when implementing this technique, as negative slope values will occur with transitions from white to black . An absolute value function should be employed so the shift and difference algorithm can detect both black to white and white to black pixel transition edges.

When this approach is used to enhance vertical edges, an image is shifted left one pixel and then subtracted from the original. To enhance horizontal edges, an image is shifted up one pixel and subtracted. To enhance both

vertical and horizontal edges the image is first shifted left one pixel and then shifted up one pixel before the subtraction is performed.

Simple as this approach sounds,its implementation is somewhat complex. For this reason, instead of actually shifting the images around, a convolution will be used to obtain the same effect. The convolution kernels that provide shift and difference edge enhancements are shown in fig 2. These kernels resemble visually the shift and difference algorithm. Take for example, vertical edge enhancement. We said earlier that to enhance vertical edges the image is shifted to the left one pixel and subtracted. The vertical Kernel performs the same process. It compares two horizontally adjacent pixels in an image in an attempt to find the slope of the brightness. If it passes over constant-intensity areas, the result ofthe convolution will be 0, because

$$-1 * \text{Intensity} + 1 * \text{Intensity} = 0$$

If there is a large intensity change, however, the result will be either a large positive number (for a black to white transition) or a large negative number (for a white to black transition). The intensity of the resultant pixel will be directly proportional to the intensity slope. Then, the white to black and black to white transitions will be enhanced.

3.3.3 Gradient Directional Edge Enhancement

The shift and difference edge enhancement detailed earlier showed how vertical and horizontal edges in an image can be enhanced. In actual practical application, most edge-enhancement algorithms utilizing only a 3x3 Kernel will enhance more than just completely vertical and horizontal lines. A larger Kernel may be used to enforce the vertical and horizontal edge requirements.

Sometimes, it is necessary to highlight edge in an image other than strictly vertical or horizontal edges. Diagonal edges of parts during a machine inspection operation may also be important. Selectively hightlighting edges in different directions can be used to give a computer an overall idea of what it is looking at. The gradient directional edge-enhancement method can be used just for this purpose. It provides eight different convolution kernels to highlight edges in eight different directions. The directions are called out as points on a compass "North", "NorthEast", "East", "SouthEast", "South", "SouthWest", "West" and "NorthWest". These kernels are shown in fig 2.

If a positive slope in the direction of the kernel exists, a light-colored pixel will be placed in the output image. The intensity of the output pixel will depend upon the slope of the brightness. The larger the slope, brighter the pixel. For example,the "East" gradient kernel will

enhance edges that transition from black to white from left to right.

We now know that since the summation of the kernel coefficients equals 0, regions of constant brightness (low spatial frequency) will be attenuated. In otherwords, areas of constant brightness will result in black pixels being output.

Many special convolution kernels exist for enhancing and detecting image edges. Two such kernels, referred to "matched filter kernels" are shown. They are called matched filters because, they resemble the attributes of the edges they are designed to enhance. They are examples of large convolution kernels, which guarantee more accurate edge detection at a price of decreased performance. Still larger kernels are possible. These contain 'templates' or shape definitions of the object(s) being searched for in an image. When a shape matches that of the template, the edges are highlighted and all other portions of the image become black.

3.4 Image Blurring

Intentionally, blurring an image might seem contrary to the philosophy of image enhancement with image processing. True, image blurring does not bring to mind any industrial applications, although there may be some. However, as an artistic tool, blurring can be put to good use. It can be used to provide a blurred background over which a foreground

object is to be placed. The contrast of sharness and blur can have a pleasing, eye-catching effect. Blurring can be produced using convolution. A Kernal used to blur an image is shown in figure. It is a 5X5 kernel containing all 1's. The larger the kernel dimensions. the more blurred the image. In essence, convolution with the blur kernel averages all pixel values in the pixel neighbourhood Averaging causes the details of an image to be reduced, resulting in the blurring effect.

3.5 Edge detection with Sobel's Algorithm

Sobel's alghorithm is the only non-linear edge enhancement/detection method discussed here, because of its wide use and efficiency.

Actually there are two distinct methods to implement Sobel's algorithm. The first method calculates two different convolutions at the pixel of interest.

x using, kernel	-1	0	1
	-2	0	-2
	-1	0	1
y using, kernel	1	2	1
	0	0	0
	-1	-2	-1

and from these convolutions, calculates the magnitude direction of the detected edges from,

$$\text{magnitude} = \text{SQRT}\ (x^2 + y^2)$$

$$\text{Direction} = \tan^{-1}\ (y/X)$$

This is computationally very expensive process to perform each pixel's operation. For this reason, a different method is used to implement Sobel's algorithm. First, let us consider a 3 X3 pixel neighbouhood as shown below.

a	b	c
d	e	f
g	h	i

Pixel 'e' is the pixel of interest. Exactly four unique lines can be drawn through this neighbourhood to pass through pixel 'e'. These lines are:

Line 1 : a - e - i

Line 2 : b - e - h

Line 3 : c - e - h

Line 4 : d - e - f

Each line drawn through the pixel subdivides the pixel space into two three-pixel neighbourhoods. e.g., line 3 subdivides the pixel space into a , b, d and h, i f. For each of the four lines,the absolute value of the differences in the averages of the two subneighbourhoods is calculated. Thus, for total calculations are performed. The pixel of interest, here 'e' is given a value of the largest of the four absolute differences.

Value 1 = abs ((a+b+d)/3 - (f+i+h)/3)

Value 2 = abs((a+d+g)/3 - (c+f+i)/3)

Value 3 = abs((b+c+f)/3 - (g+h+I)/3)

Value 4 = abs((a+b+c)/3 - (g+h+i)/3)

e = The greatest of these values.

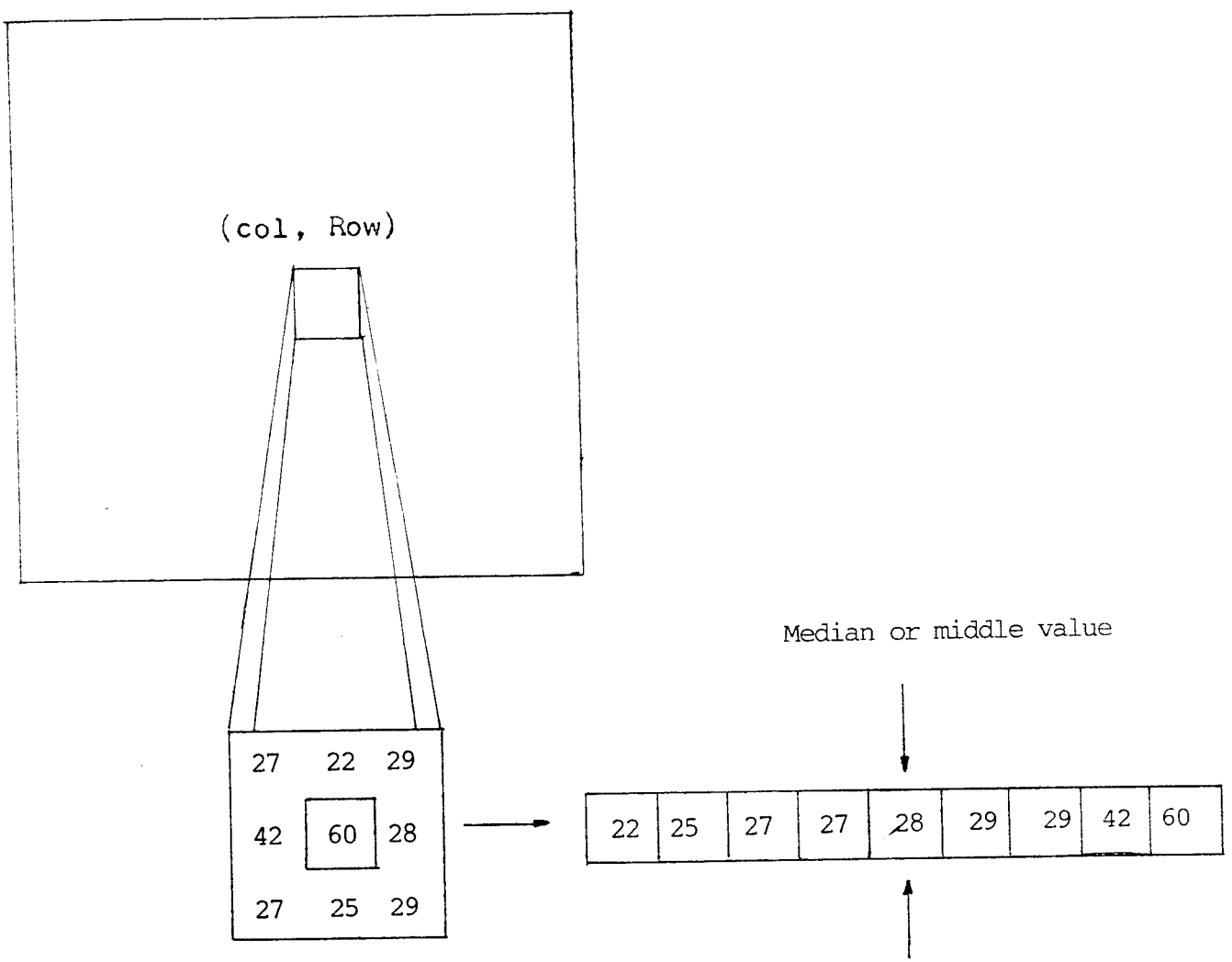
After the appliciation of Sobel's algorithm to each pixel in an image, the output image is usually subjected to a thresholding point process operation. The net result is that the output is a black and white image that contains none of the original image information except edge information.

3.6 Median filtering

Median filtering is an area process that does not fall under the category of convolution. Median filtering, can also be thought of as a point process by the way in which it works.

Median filtering uses the values of the pixels contained in the pixel neighbourhood to determine the new value given tothe pixel of interest. But, it does not algorithmically calculate the new pixel value from those of the neighbourhood in ascending order and picks the middle or median pixel value as the new value for the pixel of interest. The median filter algorithm is illustrated in figure 3.

The result of median filtering is that any random noise contained in the image will be effectively eliminated. This is because, any random, adrupt transition in pixel intensity within a pixel neighbourhood will be workedout. That is it will be placed at either top or the bottom of the sorted neighbourhood values and will be ignored because of the median value is always picked for the new pixel value. Multiple applications of median filtering to an image can result in rather pleasing visual effects.

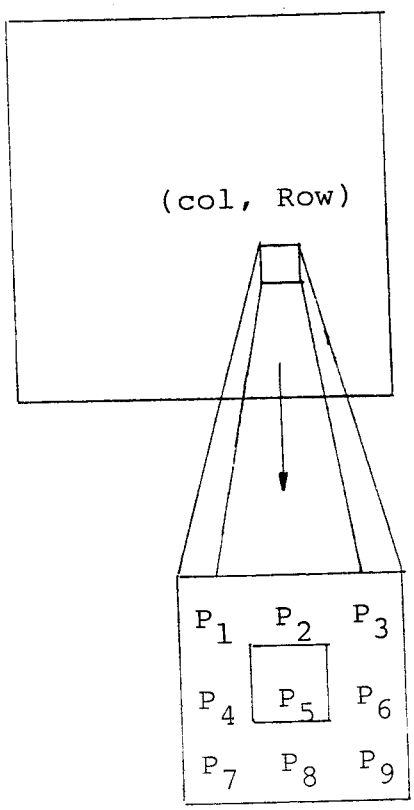


3 X 3 Pixel neighour hood

Centre pixel with value of 60 is under process.

A Value of 28 replaces the value of 60 in the output image.

Fig. 1 (Contd)



X

K_1	K_2	K_3
K_4	K_5	K_6
K_7	K_8	K_9

3 X 3 pixel neighbourhood

P_5 is pixel being computed.

3 X 3 convolution kernel.

Weighted sum calculation

$(K_1 . P_1) +$

$(K_2 . P_2) +$

$(K_3 . P_3) +$

$(K_4 . P_4) +$

$(K_5 . P_5) +$

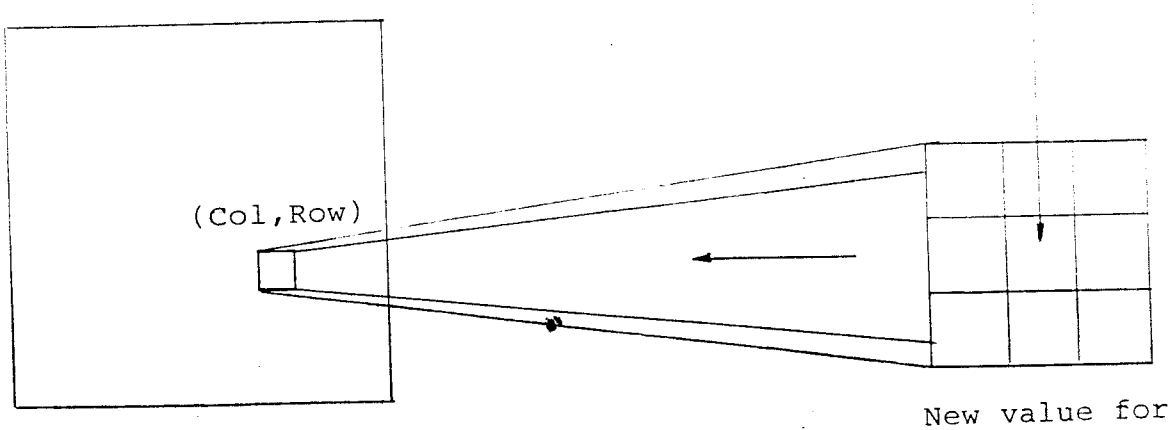
$(K_6 . P_6) +$

$(K_7 . P_7) +$

$(K_8 . P_8) +$

$(K_9 . P_9) +$

New value of P_5



New value for P_5 placed in output image.

Fig: 1 Contribution illustrated.

1.Low Pass Filters

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

LP1

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

LP2

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

LP3

sum = 1

2. High - Pass Spatial Filters

-1	-1	-1
-1	9	-1
-1	-1	-1

HP1

0	-1	0
-1	5	-1
0	-1	0

HP2

1	-2	1
-2	5	-2
1	-2	1

HP3

sum = 1

3. Shift and Difference Edge Enhancements

0	0	0
-1	1	0
0	0	0

Vertical edges

0	-1	0
0	1	0
0	0	0

Horizantal edges

-1	0	0
0	1	0
0	0	0

Horizantal and Vertical edges

sum = 0

4. Matched Filter Edge Enhancements

-1	0	1
-1	0	1
-1	0	1
-1	0	1
-1	0	1

Vertical edges

-1	-1	-1	-1	-1
0	0	0	0	0
1	1	1	1	1

Horizantal edges

sum = 0

5. Gradient Directional Edge Enhancements

1	1	1
1	-2	1
-1	-1	-1

North

1	1	1
-1	-2	1
-1	-1	1

Northeast

-1	1	1
-1	-2	1
-1	1	1

East

-1	-1	1
-1	-2	1
1	1	1

Southeast

Fig. 2 Horizontal Masks

-1	-1	-1		1	-1	-1		1	1	-1		1	1	1
1	-2	1		1	-2	-1		-1	-2	1		1	-2	-1
1	1	1		1	1	1		1	1	-1		-1	-1	-1
South				SouthWest				West				NorthWest		

sum = 0

6. Blurring Kernel

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

sum = 25

7. Laplace Edge Enhancements

0	1	0		-1	-1	-1		-1	-1	-1		1	-2	1
1	-4	1		-1	8	-1		-1	9	-1		-2	4	-2
0	1	0		-1	-1	-1		-1	-1	-1		1	-2	1
LAP1				LAP2				LAP3				LAP4		

sum normally = 0 except LAP3

Notes:

1. All Kernels from Baxex except from Dawson
2. Equivelent to adding original image to output of convolution using kernel LAP2.
3. Proof can be found in Gonzalez and Wintz.

Fig 2. Various Convolution kernels.

CHAPTER IV

FRAME PROCESSES

4.1 Introduction

Frame processes use information from two (or more) images together with a combination function to produce a brand-new image. This new image depends not only on the content of each input image but also upon the type of function used to combine them. These processes are called frame processes because a complete video image is called a frame. They work on two complete video images.

4.2 Application of Frame Processes

Frame processes have many practical applications. They are being used in industry today for security, quality control and image-qualty-enhancement applications. They have artistic uses as well, as we shall see. We shall investigate some of these applications in the discussion that follows.

For security applications, frame processing can be used to detect motion and therefore introduces. Assume a video camera is positioned within the lobby of a building in such a manner that its field of view convers the complete lobby If each frame from the video camera is digitized and compared to the previous frame, any movement within the field of view of the camera can be detected. The comparision of video images is actually the substraction of the images followed by a thresholding process and finally a pixel

difference tally. If the number of pixels that differ in two sequential video images crosses a pre-determined threshold, it is time to set off the alarms. Not only is motion detected, but the direction and speed of the motion can also be measured. This is possible as long as three things are known:

1. The time between the sequential video frames.

2. The distance from the camera to the moving object.

3. The total displacement of the object in pixels.

Armed with this information and some trignometry, speed and direction can be approximated.

Ths code provided in the frame process provides each of the following contribution functions: "And", "Or", "Xor", "Add", "Subtract", "MULTIPLY", "Divide", "Average", " Maximum value", "Minimum value", and "Overlay". Some of these continuation functions make infinitive sense while others (multiplying images, for example) make very little sense.

Notes:

1."S Data" is data fetched from the source image. "D data" is data fetched from the destination images and eventually stored back into the destination image.

2. The destination images is modified by all these combine operations.

3. Overby performs a copy operation from the source to the destination image.

Truth table for basic logic functions.

This table gives the rules for the basic combination functions (and then some). With this table it is possible to predict the result of combining overlapping image data using the simple logic functions.

Figure 4 shows image combination functions Provided by the frame process

4.3 Various frame processes

The bitwise "AND" continuation function is used mainly to mark out portions of an image. First, a mask image is produced that contains a 1 in every pixel location that is to be retained and a 0 in each pixel location to be marked out (changed to black). When an image and the mark image are combined using the "AND" function, the output image will be the same as the source image wherever the mark image contained 1's and will be black every where the mark contained 0's.

The use of the "AND" function with a mark having values other than 0 or 1 is of questionable value. The effect would be selectively to remove bits from each pixel's value, thereby changing its displayed color. A practical application of this technique is unknown, although artistic uses can probably be imagined. Also, using the "AND" function to combine actual images may have interesting visual impact but otherwise is of dubious value. The same is true when combining image with any logical operations: "AND", "OR" and "XOR".

The "OR" function can be used under certain conditions to combine two images quite effectively. If two images are available that have bright foreground objects on dark (black) backgrounds and that do not overlap, the "OR" function can be used to merge both foreground objects into a single image. The foreground object of one image overlays the other image's block background and vice versa. The "OR" function will cause the bits that are set in the foreground objects to be animilated into output image, because any bit set in the foreground object "OR" ed with the 0 bit of the back-ground will become set. "OR"ing follows exactly the same rules for image combination as it does in logic gates in electronics.

Problems arise, however if the foreground objects of the two images being combined overlap. The "OR"ing of the two nonzero pixel values in the foreground objects will result in a new pixel value that in the combination of the two. The color used for the display (gray scale tone0 of the overlapping pixels will then be unreleased to either image. The results are not very pleasing to look at and there is no known practical use.

The exclusive "OR", "XOR", is another combination function available. From elementary logic, with an "XOR" operation any bits that are the same become 0 and bits that are different become 1's. With this combining funcition it is possible to detect all pixels of an image with a specified value. To do this, an image buffer is prepared by clearing

it to specified pixel value, say 32. If this image is then "XOR" ed with a real image, every pixel in the real image that was exactly equal to 32 will be set to black, with all other pixels becoming someother non black color (gray tone). To make the effect more noticeable, a point process can follow the "XOR" operation that sets all black pixels to white and all other pixels to black.

5.2 Image scaling

The geometric process of scaling allows an image or portion of an image to be changed in size. The resultant image may be a magnified or reduced version of the original image. A limitation of the scaling functions is that, the scaled image must reside completely within an image buffer. This,of course, is a problem only when images are magnified in size, and not when they are reduced. So, here we discuss only image reduction and not image magnification.

The geometric process of scaling can change the spatial organization of image data to such a degree that the original image data is not recoverable. As an image is reduced in size, one pixel of the reduced image is derived from four (for a reduction factor of 2) pixels of the source image. Obviously, some information is lost in this process.

5.3 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 a pixel of the source image (i old, j old) into its new rotated location in the destination image ("i new " , "j new ") are as follows:

i new = i old * cos(O) + j old * sin(O) and

j new = j old * cos(O) - i old * sin(O)

Where, O is the Angle of rotation.

5.4 Translation

Translation is a geometric transformation that allows an image or portion of an image to be changed in position. In case of image processing and computer graphics, translation means movement.

This transformation is performed from the source image's perspective instead of from that of the destination image. The trasformation is a one-for-one mapping between source and destination pixels. This makes the translation transformations quick compared to image scaling/rotation.

5.5 Mirror Images

Mirror imaging, simply rearranges the pixels of a source image to generate a mirror image of the source, as the destination image. The resulting image trasformation appears as if a mirror were used to produce it.

The types of mirror images are discussed here, viz., horizontal mirroring, which generates a mirror of the source in the horizontal direction, and vertical mirroring, which works in the vertical direction. With horizontal mirroring source image left becomes destination image right and source image right becomes destination image left. With vertical mirroring, source image top becomes destination image bottom and vice versa.

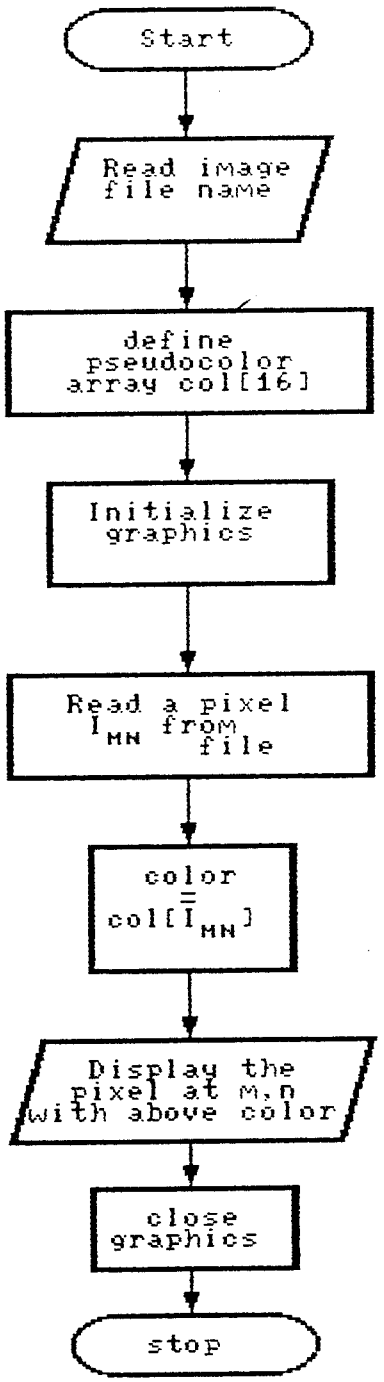
CONCLUSION

The image processing functions provided here all deal with pseudo coloring of a gray scale image, due to non-availability of system. The processing with actual gray colors, can be obtained by properly adjusting the Red, Green and Blue color registers of a VGA (Video Graphics Adapter)in its video mode 13x (which is a 256-color, non-standard mode, not supported by version 2.0 of Borland's turboC) color image processing involves the same processing techniques as for grayscale images.

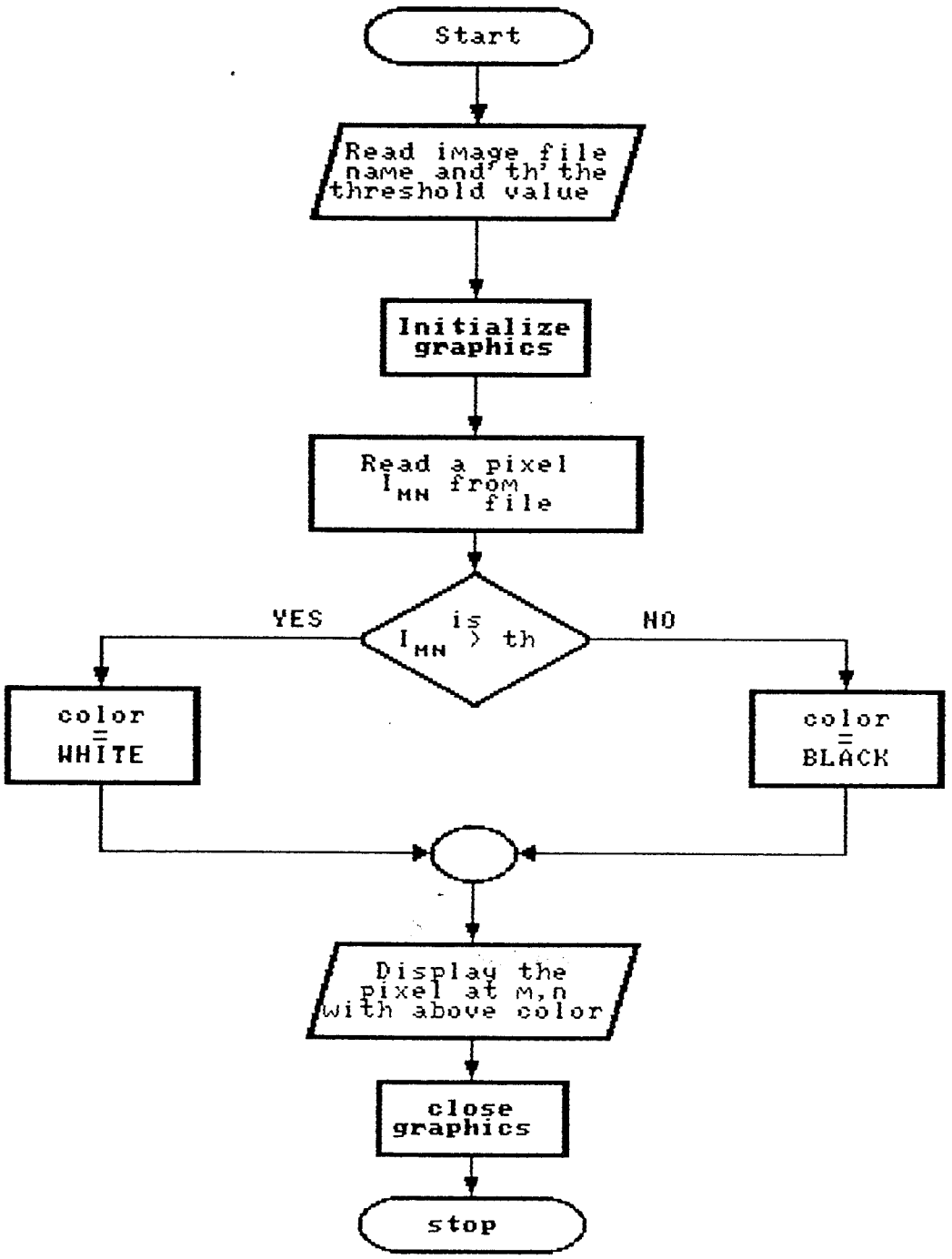
The techniques of image processing can be applied to data even if they are not in a visible form. The manipulation of visible image data is just one of the many uses of image processing. This can be done to produce a visible image of purely numeric data enhanced in some manner to highlight some aspect of the data. Examples of this kind can be found in magnetic resonant medical imaging equipment, sonar, radar, ultrasound equipment, heatsensing equipment, fractals, and so on.

One final word, applying an image processing algorithm to an image is not always done with the appearance of the image in mind. Actually, the result might not be pleasing to look at Aesthetics are not the only criterion by which to judge the effectiveness of the applied transformation. If

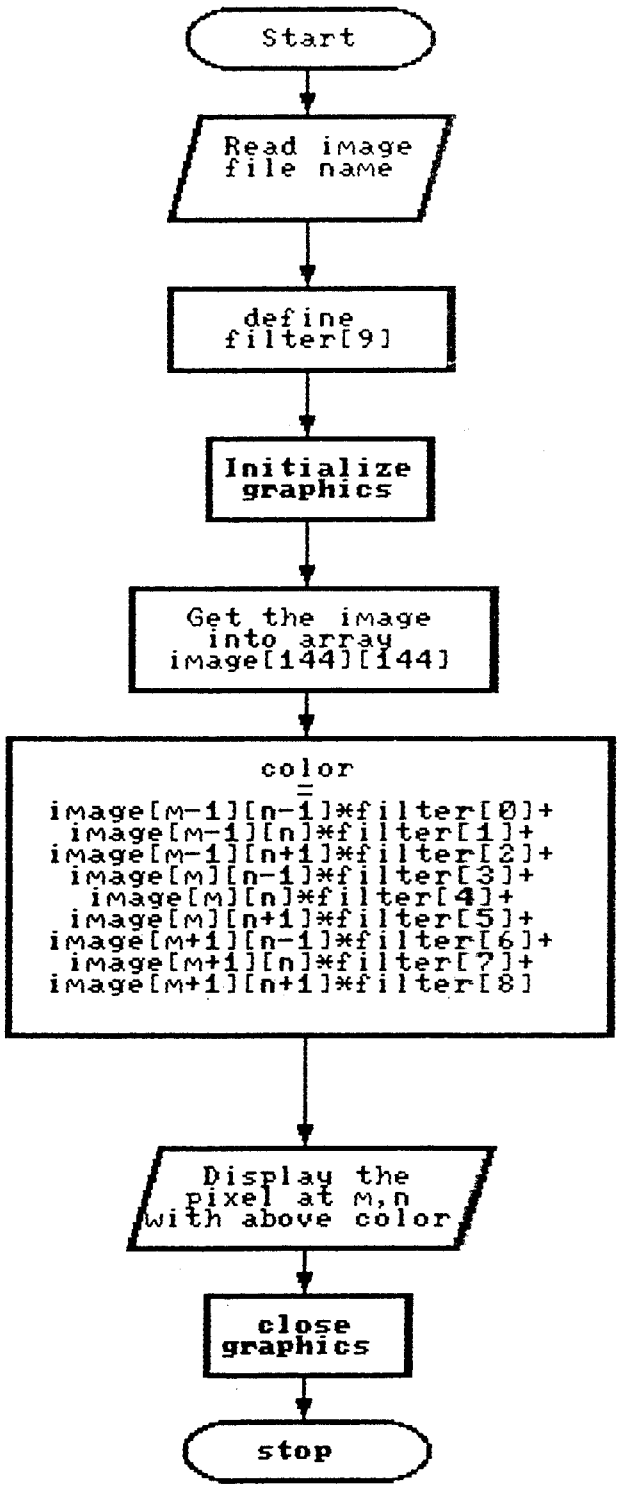
the transformation is designed to bring out additional information and / or details not visible in the original image, the result can be considered successful even if it is not pleasing to look at.



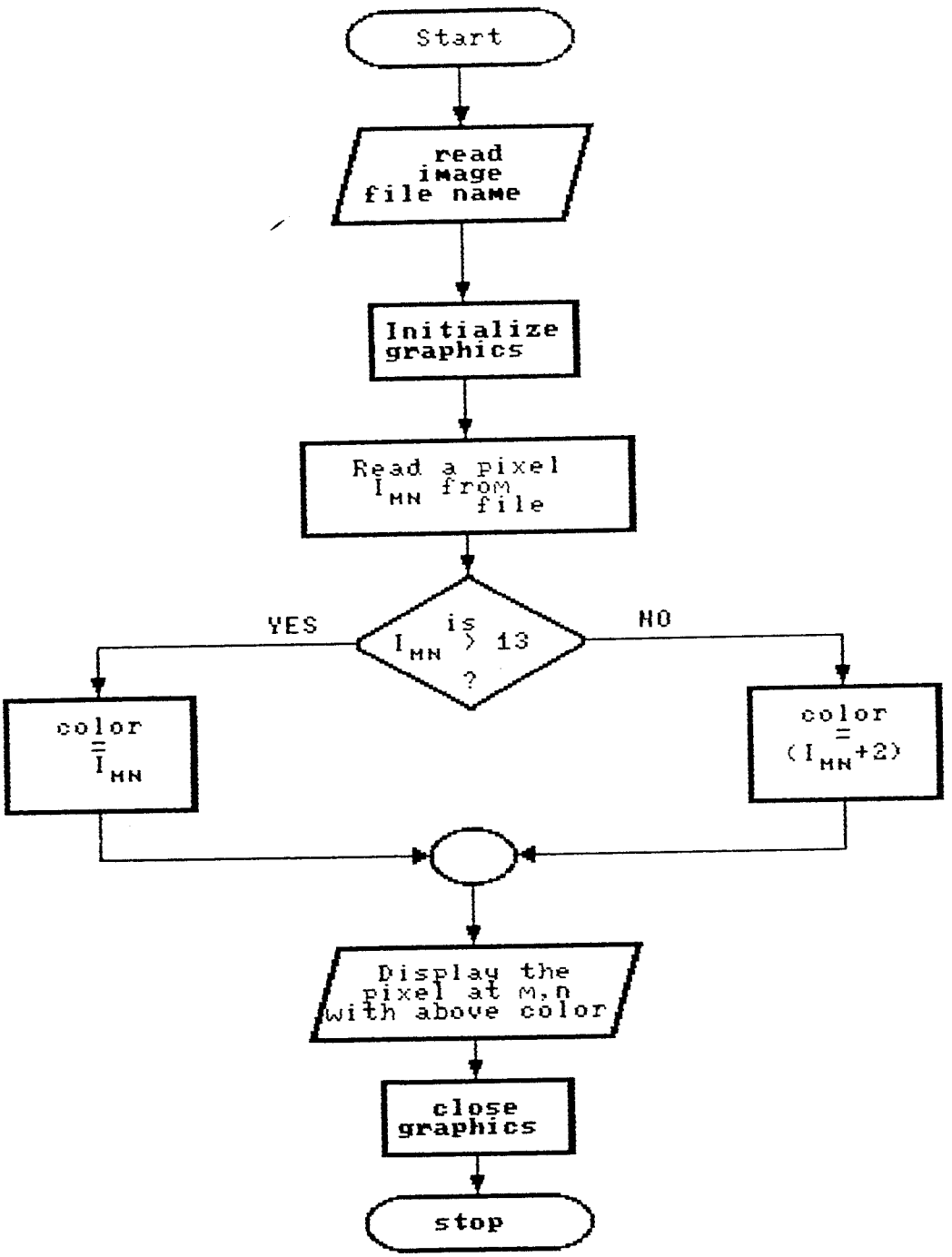
FLOW CHART FOR PSEUDOCOLORING



FLOWCHART FOR THRESHOLDING



FLOWCHART FOR FILTERING



FLOWCHART FOR BRIGHTENING

```
struct menu{ void (*fn)();
                        char *name ; }
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include<process.h>
int g_d = EGA, g_m = EGAHI ;
#define PI 3.14156798
#include"point.h"
#include"area.h"
#include"geometri.h"

                        /* MAIN PROGRAM */

main()
    { int display();
          void point();
          void area();
          void frame();
          void geometric();
          void histgm();
          int n = 6,ret;
          static struct menu main[6] =  {
                                          { point ,"Point process"},
                                          {area ,"Area process"},
                                          {frame ,"Frame process"},
                                          {geometric ,"Geometric process"},
                                          {histgm,"histogram"},
                                          {exit ,"Exit"}
                                          };
      char *mainmenu = "MAIN MENU";
      ret = display(mainmenu,main,n);
      window(1,1,80,25);
      clrscr();
      textcolor(YELLOW);
      textbackground(BLUE);
      clrscr();
      gotoxy(13,10);
      if(ret != 5)
           (*main[ret].fn)();
      window(1,1,80,25);
      clrscr();
      exit(1);
         }
```

```
                /* PROGRAM TO DISPLAY A MENU */
int display(subn,disp,n)
char *subn;
struct menu disp[10];
int n;
   {  int x,y,i,k,j,z,choice;
      window(1,1,80,25);
      textcolor(YELLOW);
      textbackground(BLUE);
      clrscr();
      x = 20;
      i = y = 3;
      gotoxy(x,i++);printf("IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;");
      gotoxy(x,i++);printf(":                                        :");
      gotoxy(x,i++);printf("LMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM9");
      while(i < y+2*n)
          {
           gotoxy(x,i++);printf(":                                        :");
           gotoxy(x,i++);printf(":DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD6");
          };
        gotoxy(x,i++);printf(":                                        :");
        gotoxy(x,i);  printf("HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<");
        j = y+1;
        window(21,j,41,j);
        gotoxy(10,1);
        printf("%s",subn);
        j += 2;
        k = 0;
        while(j < y+2*(n+1))
           {
              window(21,j,41,j);
              gotoxy(10,1);
                  printf("%s",disp[k++].name);
              j += 2;
           }
         z = 0;
         window(21,6+z*2,54,6+z*2);
         textcolor(WHITE);
         textbackground(RED);
         clrscr();
         gotoxy(10,1);
         printf("%s",disp[z].name);
         gotoxy(1,1);
         {
                scan: choice = getch() ;
                      switch(choice)
                        {
                            case 80: window(21,6+z*2,54,6+z*2);
                                     textcolor(YELLOW);
                                     textbackground(BLUE);
                                     clrscr();
                                     gotoxy(10,1);
                                     printf("%s",disp[z].name);
                                     z++;
```

```
                    if(z == n) z = 0;
                    window(21,6+z*2,54,6+z*2);
                    textcolor(WHITE);
                    textbackground(RED);
                    clrscr();
                    gotoxy(10,1);
                    printf("%s",disp[z].name);
                    gotoxy(1,1);

                    break;
            case 72: window(21,6+z*2,54,6+z*2);
                    textcolor(YELLOW);
                    textbackground(BLUE);
                     clrscr();
                    gotoxy(10,1);
                    printf("%s",disp[z].name);
                    z--;
                    if (z < 0) z = n-1;
                    window(21,6+z*2,54,6+z*2);
                    textcolor(WHITE);
                    textbackground(RED);
                    clrscr();
                    gotoxy(10,1);
                    printf("%s",disp[z].name);
                    gotoxy(1,1);
                    break;
             case 13: return z;
             default: break;
          } goto scan;
}
```

```
                /* PROGRAM TO DISPLAY CHOICES */
int chart(subn,list,n)
char *subn;
char *list[];
int n;
    {  int x,y,i,k,j,z,choice;
       window(1,1,80,25);
       textcolor(YELLOW);
       textbackground(BLUE);
       clrscr();
       x = 20;
       i = y = 3;
       gotoxy(x,i++);printf("IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;");
       gotoxy(x,i++);printf(":                                    :");
       gotoxy(x,i++);printf("LMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM9");
       while(i < y+2*n)
          {
           gotoxy(x,i++);printf(":                                    :");
           gotoxy(x,i++);printf(":DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD6");
          };
          gotoxy(x,i++);printf(":                                    :");
          gotoxy(x,i);  printf("HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<");
          j = y+1;
          window(21,j,41,j);
          gotoxy(10,1);
          printf("%s",subn);
          j += 2;
          k = 0;
          while(j < y+2*(n+1))
             {
                window(21,j,41,j);
                gotoxy(10,1);
                    printf("%s",list[k++]);
                j += 2;
             }
           z = 0;
           window(21,6+z*2,54,6+z*2);
           textcolor(WHITE);
           textbackground(RED);
           clrscr();
           gotoxy(10,1);
           printf("%s",list[z]);
           gotoxy(1,1);
           {
                  scan: choice = getch() ;
                        switch(choice)
                          {
                              case 80: window(21,6+z*2,54,6+z*2 ;
                                       textcolor(YELLOW);
                                       textbackground(BLUE);
                                       clrscr();
                                       gotoxy(10,1);
                                           printf("%s",list[z]
                                       z++;
```

```
                        if(z == n) z = 0;
                        window(21,6+z*2,54,6+z*2);
                        textcolor(WHITE);
                        textbackground(RED);
                        clrscr();
                        gotoxy(10,1);
                            printf("%s",list[z]);
                        gotoxy(1,1);
                        break;
                case 72: window(21,6+z*2,54,6+z*2);
                        textcolor(YELLOW);
                        textbackground(BLUE);
                        clrscr();
                        gotoxy(10,1);
                            printf("%s",list[z]);
                        z--;
                        if (z < 0) z = n-1;
                        window(21,6+z*2,54,6+z*2);
                        textcolor(WHITE);
                        textbackground(RED);
                        clrscr();
                        gotoxy(10,1);
                            printf("%s",list[z]);
                        gotoxy(1,1);
                        break;
                    case 13: return z;
                default: break;
            }  goto scan;
        }
}
```

```
            /* MENU PROGRAM FOR GEOMETRIC PROCESSES */
void geometric()
   {
      void rotate();
      void scale();
      void mirror();
      void translate();
          static struct menu geom[5] = { {rotate,"Rotation"},
                                         {scale , "Scaling"},
                                         {mirror,"Mirror"},
                                         {translate,"Translation"} ,
                                         {main,"Exit to main menu"} };
      int ret;
      char *sbmenu = "GEOMETRIC PROCESS";
          ret = display(sbmenu,geom,5);
      window(1,1,80,25);
      clrscr();
          (*geom[ret].fn)();
          (*geom[4].fn)();
        }
```

```
                    /* PROGRAM FOR FRAME PROCESSES */
void frame()
         {return;          }
         char *fram[5] = { "AND",
                           "OR",
                           "XOR",
                           "OVERLAY" ,
                           "EXIT TO MAIN MENU" };
       int ret;
           char *sbmenu = "FRAME PROCESS";
void far *bitmap;
unsigned int i,j,value[144][144];
unsigned long nb;
FILE *f_p;
FILE *f_p1
f_p = fopen("h1.img","rb");
f_p1 = fopen("hsc.img","rb");
           ret = chart(sbmenu,fram,5);
           window(1,1,80,25);
        clrscr();
initgraph(&g_d,&g_m,"");
setgraphmode(g_m);
for(i = 0;i<144;i++)
 for (j = 0;j < 144; j++)
  {
    value[i][j] = getc(f_p);
    putpixel(j,i,value[i][j]/16);
  }
for(i = 0;i<72;i++)
 for (j = 0;j < 72; j++)
  {
    value[i][j] = getc(f_p1);
    putpixel(j+150,i,value[i][j]/16);
  }
nb = imagesize(150,0,221,71);
bitmap = farmalloc(nb);
getimage(150,0,221,71,bitmap);
putimage(150,0,bitmap,XOR_PUT);
putimage(35,35,bitmap,fram[ret]);
farfree(bitmap);
getch();
           }
```

```
            /* PROGRAM FOR CONVOLUTION FILTERING  */
convolute()
{
    char source_file[20] ;
    unsigned int image[144][144] ;
    unsigned int graylvl ;
    int filt_sum = 0;
    FILE *fp ;
    int i,j,ret;
    int conv_choice,filter_choice ;
    int filter[6][9] = {    1, 1, 1, 1, 1, 1, 1, 1, 1,
                            1, 1, 1, 1, 2, 1, 1, 1, 1,
                            1, 2, 1, 2, 4, 2, 1, 2, 1,
                           -1,-1,-1,-1, 9,-1,-1,-1,-1,
                            0,-1, 0,-1, 5,-1, 0,-1, 0,
                            1,-2, 1,-2, 5,-2, 1,-2, 1
                                                  } ;
    char name[20] = "CONVOLUTION KERNAL";
    char *ch[7] = {"lp1","lp2","lp3","hp1","hp2","Hp3","exit"};
    clrscr();
    filter_choice = chart(name,ch,7);
    while (filter_choice != 6)
          {
                        window(43,21,45,21);
                        textbackground(RED) ;
                        textcolor(YELLOW) ;
                        clrscr() ;
                        printf("Enter Source Image file : ") ;
                        scanf("%s",&source_file) ;
                        clrscr() ;
                        fp = fopen(source_file,"rb");
                        initgraph(&g_d,&g_m,"") ;
                        setgraphmode(g_m) ;
                        for(i = 0 ; i < 144 ; i++)
                        for (j = 0 ; j < 144 ; j++)
                                {
                                        image[i][j] = fgetc(fp) ;
                                        graylvl = image[i][j]/16 ;
                                        putpixel(j,i,graylvl) ;

                                }
```

```
    for(i = 0 ; i < 144 ; i++)
    for(j = 0 ; j < 144 ; j++)
    {
    if((i == 0 || j == 0) && (i == 143 || j == 143))
    {
    graylvl = 4 ;
    }
    else
    {
    graylvl = image[i-1][j-1] * filter[filter_choice][0] +
              image[i-1][j]   * filter[filter_choice][1] +
              image[i-1][j+1] * filter[filter_choice][2] +
              image[i][j-1]   * filter[filter_choice][3] +
              image[i][j]     * filter[filter_choice][4] +
              image[i][j+1]   * filter[filter_choice][5] +
              image[i+1][j-1] * filter[filter_choice][6] +
              image[i+1][j]   * filter[filter_choice][7] +
              image[i+1][j+1] * filter[filter_choice][8];

        for(i=0;i<9;filt_sum += filter[filter_choice][i++]);
        graylvl /= filt_sum;
        putpixel(j+150,i,graylvl/16) ;
            } ;
      }
            settextstyle(4,HORIZ_DIR,1) ;
            moveto(10,180) ;
            outtext("ORIGINAL IMAGE") ;
            moveto(160,180) ;
            outtext("PROCESSED IMAGE") ;
            getch() ;
            closegraph() ;
            window(1,1,80,25) ;
            textcolor(YELLOW) ;
            textbackground(BLUE) ;
            clrscr() ;
            return;
      }
  return;
}
```

```
                /*  PROGRAM FOR EDGE DETECTION  */
dge()
 {  void shift_diff();
        void matched_filter();
        void grad_directional();
        void laplace();
        struct menu edg[5] = { {shift_diff,"Shift & difference"},
                               {matched_filter,"Matched filter"},
                               {grad_directional,"Grad_directional"},
                               {laplace,"Laplace"},
                               { exit , "exit to main"}};
        int enhance_choice ;
        char *title = "EDGE ENHANCEMENTS";
        enhance_choice = display(title,edg,5);
        if(enhance_choice != 4)
          {
               (*edg[enhance_choice].fn)();
               window(1,1,80,25) ;
               textcolor(YELLOW) ;
               textbackground(BLUE) ;
               return;
          }
        return;


lurr()

    int value,i,j;
    unsigned int image[144][144];
    int level;
    char infile[20];
    FILE *fp;
    window(43,21,45,21);
    textbackground(RED) ;
    textcolor(YELLOW) ;
    clrscr() ;
    printf("Enter Source Image file : ") ;
    scanf("%s",&infile) ;
    clrscr() ;
    initgraph(&g_d,&g_m,"");
    setgraphmode(g_m);
    fp = fopen(infile,"rb");
    for(i=0;i<=143;++i)
        for(j=0;j<=143;++j)
         {
         image[i][j] = fgetc(fp);
         level = image[i][j]/16;
         putpixel(j,i,level);
         }
```

```
        for(i=1;i<143;++i)
        for(j=1;j<=143;++j)
        {
        level   = image[i-2][j-2]+image[i-2][j-1]+image[i-2][j]+
                  image[i-2][j+1]+image[i-2][j+2]+image[i-1][j-2]+
                  image[i-1][j-1]+image[i-1][j]+image[i-1][j+1]+
                  image[i-1][j+2]+image[i][j-2]+image[i][j-1]+
                  image[i][j]+image[i][j+1]+image[i][j+2]+
                  image[i+1][j-2]+image[i+1][j-1]+image[i+1][j]+
                  image[i+1][j+1]+image[i+1][j+2]+image[i+2][j-2]+
                  image[i+2][j-1]+image[i+2][j]+image[i+2][j+1]+
                  image[i+2][j+2];
          level /= 500;
          putpixel(j+150,i,level);
          }
          getch();
          closegraph();

    }

sobel()
{
        char source_file[20] ;
        unsigned int image[144][144] ;
        unsigned int graylvl ;
        unsigned int sobel[4],sobel1[9] ;
        FILE *fp ;
        FILE *fp1;
        void threshol(FILE *F);
        int i,j,k ;
        textcolor(WHITE) ;
        textbackground(BLACK) ;
        clrscr() ;
        window(10,10,70,15) ;
        textcolor(YELLOW) ;
        textbackground(RED) ;
        clrscr() ;
        gotoxy(10,3) ;
        printf("Enter Source Image file : ") ;
        scanf("%s",&source_file) ;
        clrscr() ;
        fp = fopen(source_file,"rb");
        fp1 = fopen("c:outimage.img","wb+");
        initgraph(&g_d,&g_m,"") ;
        setgraphmode(g_m) ;
        for(i = 0 ; i < 144 ; i++)
        for (j = 0 ; j < 144 ; j++)
                {
                        image[i][j] = fgetc(fp) ;
                        graylvl = image[i][j]/16 ;
                        putpixel(j,i,graylvl) ;
                }
```

```
    for(i = 0; i < 144 ; i++)
    for (j = 0; j < 144 ; j++)
            {
              if((i == 0 || j == 0) && (i == 143 || j == 143))
                                        graylvl = 4 ;
              else
                    {       sobel[0] = image[i-1][j-1] ;
                            sobel[1] = image[i-1][j] ;
                            sobel[2] = image[i-1][j+1] ;
                            sobel[3] = image[i][j-1] ;
                            sobel[4] = image[i][j] ;
                            sobel[5] = image[i][j+1] ;
                            sobel[6] = image[i+1][j-1] ;
                            sobel[7] = image[i+1][j] ;
                            sobel[8] = image[i+1][j+1] ;
sobel1[0] = abs((sobel[3] + sobel[6] + sobel[7] -
            sobel[1] - sobel[2] - sobel[5])/3) ;
sobel1[1] = abs((sobel[0] + sobel[3] + sobel[6] -
            sobel[2] - sobel[5] - sobel[8])/3) ;
sobel1[2] = abs((sobel[0] + sobel[1] + sobel[3] -
            sobel[5] - sobel[8] - sobel[7])/3) ;
sobel1[3] = abs((sobel[0] + sobel[1] + sobel[2] -
            sobel[6] - sobel[7] - sobel[8])/3) ;
graylvl = sobel1[0] ;
for(k = 1 ; k <= 3 ; k++)
    {
    if(sobel1[k] > graylvl) graylvl = sobel1[k] ;
    }
              }
    fputc(graylvl,fp1);
    }
    threshol(fp1);
    settextstyle(4,HORIZ_DIR,1) ;
    moveto(10,180) ;
    outtext("ORIGINAL IMAGE") ;
    moveto(160,180) ;
    outtext("PROCESSED IMAGE") ;
    getch() ;
    closegraph() ;
    window(1,1,80,25) ;
    textcolor(YELLOW) ;
    textbackground(BLUE) ;
    return;
}
```

```
      /* PROGRAM FOR EDGE DETECTION BY SHIFT&DIFFERENCE  */
void shift_diff()
{
        char source_file[20] ;
        unsigned int image[144][144] ;
        unsigned int graylvl ;
        FILE *fp ;
        FILE *fp1;
        int i,j,z ;
        int shift_choice,filter_choice ;
        int shift_filter[3][9] =
                {
                        0, 0, 0,-1, 1, 0, 0, 0, 0,
                        0,-1, 0, 0, 1, 0, 0, 0, 0,
                       -1, 0, 0, 1, 0, 0, 0, 0, 0
                } ;
        char *title = "SHIFT & DIFFERENCE EDGE ENHANCEMENT";
        char *list[4] = { "Vertival edges", "horizantal edges",
                                        "Vert & Horz edges" ,"exit");
        filter_choice = chart(title,list,4);
        if(filter_choice != 3)  {
        window(10,10,70,15) ;
        textcolor(YELLOW) ;
        textbackground(RED) ;
        clrscr() ;
        gotoxy(10,3) ;
        printf("Enter Source Image file : ") ;
        scanf("%s",&source_file) ;
        clrscr() ;
        fp = fopen(source_file,"rb");
        fp1 = fopen("c:outimage.img","wb+");
        initgraph(&g_d,&g_m,"") ;
        setgraphmode(g_m) ;
        for(i = 0 ; i < 144 ; i++)
        for (j = 0 ; j < 144 ; j++)
        {
                image[i][j] = fgetc(fp) ;
                graylvl = image[i][j]/16 ;
                putpixel(j,i,graylvl) ;
        }
```

```
    for(i = 0 ; i < 144 ; i++)
    for(j = 0 ; j < 144 ; j++)
    {
            if((i == 0 || j == 0) && (i == 143 || j == 143))
            {
                    graylvl = 15 ;
            }
            else
            {
graylvl = image[i-1][j-1] * shift_filter[filter_choice][0] +
          image[i-1][j]   * shift_filter[filter_choice][1] +
          image[i-1][j+1] * shift_filter[filter_choice][2] +
          image[i][j-1]   * shift_filter[filter_choice][3] +
          image[i][j]     * shift_filter[filter_choice][4] +
          image[i][j+1]   * shift_filter[filter_choice][5] +
          image[i+1][j-1] * shift_filter[filter_choice][6] +
          image[i+1][j]   * shift_filter[filter_choice][7] +
          image[i+1][j+1] * shift_filter[filter_choice][8];

            }
    fputc(graylvl,fp1);
       }
    threshol(fp1);
    settextstyle(4,HORIZ_DIR,1) ;
    moveto(10,180) ;
    outtext("ORIGINAL IMAGE") ;
    moveto(160,180) ;
    outtext("PROCESSED IMAGE") ;
    getch() ;
    closegraph() ;
    window(1,1,80,25) ;
    textcolor(YELLOW) ;
    textbackground(BLUE) ;
    clrscr() ; }
    return;
}
```

```
      /*  PROGRAM FOR EDGE DECTION BY MATCHED FILTERING  */
void matched_filter()
{
        char source_file[20] ;
        unsigned int image[144][144] ;
        unsigned int graylvl ;
        FILE *fp ;
        FILE *fp1;
        int i,j,ret;
        int conv_choice,filter_choice ;
        int match_filter[2][15]
              =  {
                 -1, 0,1,-1, 0,1,-1, 0,1,-1, 0,1,-1,0,1,
                 -1,-1,-1,-1,-1, 0, 0, 0, 0,0,1,1,1,1,1
                  } ;
        char name[20] = "MATCHED FILTERS";
        char *ch[3] = {"VERTICAL EDGES","HORIZANTAL EDGES","EXIT"};
        clrscr();
        filter_choice = chart(name,ch,3);
        while (filter_choice != 2)
            {
                 window(43,21,45,21);
                 textbackground(RED) ;
                          textcolor(YELLOW) ;
                 clrscr() ;
                 printf("Enter Source Image file : ") ;
                 scanf("%s",&source_file) ;
                 clrscr() ;
                 fp = fopen(source_file,"rb");
                 fp1 = fopen("c:outimage.img","wb+");
                 initgraph(&g_d,&g_m,"") ;
                 setgraphmode(g_m) ;
                 for(i = 0 ; i < 144 ; i++)
                 for (j = 0 ; j < 144 ; j++)
                          {
                                   image[i][j] = fgetc(fp) ;
                                   graylvl = image[i][j]/16 ;
                                   putpixel(j,i,graylvl) ;
                          }
```

```
                for(i = 0 ; i < 144 ; i++)
                for(j = 0 ; j < 144 ; j++)
                 {
                        if(filter_choice == 0)
graylvl = image[i-1][j-2] * match_filter[filter_choice][0] +
          image[i][j-2]   * match_filter[filter_choice][1] +
          image[i+1][j-2] * match_filter[filter_choice][2] +
          image[i-1][j-1] * match_filter[filter_choice][3] +
          image[i][j-1]   * match_filter[filter_choice][4] +
          image[i+1][j-1] * match_filter[filter_choice][5] +
          image[i-1][j]   * match_filter[filter_choice][6] +
          image[i][j]     * match_filter[filter_choice][7] +
          image[i+1][j]   * match_filter[filter_choice][8] +
          image[i-1][j+1] * match_filter[filter_choice][9] +
          image[i][j+1]   * match_filter[filter_choice][10] +
          image[i+1][j+1] * match_filter[filter_choice][11] +
          image[i-1][j+2] * match_filter[filter_choice][12] +
          image[i][j+2]   * match_filter[filter_choice][13] +
          image[i+1][j+2] * match_filter[filter_choice][14] ;

          else
graylvl = image[i-2][j-1] * match_filter[filter_choice][0] +
          image[i-1][j-1] * match_filter[filter_choice][1] +
          image[i][j-1]   * match_filter[filter_choice][2] +
          image[i+1][j-1] * match_filter[filter_choice][3] +
          image[i+2][j-1] * match_filter[filter_choice][4] +
          image[i-2][j]   * match_filter[filter_choice][5] +
          image[i-1][j]   * match_filter[filter_choice][6] +
          image[i][j]     * match_filter[filter_choice][7] +
          image[i+1][j]   * match_filter[filter_choice][8] +
          image[i+2][j]   * match_filter[filter_choice][9] +
          image[i-2][j+1] * match_filter[filter_choice][10] +
          image[i-1][j+1] * match_filter[filter_choice][11] +
          image[i][j+1]   * match_filter[filter_choice][12] +
          image[i+1][j+1] * match_filter[filter_choice][13] +
          image[i+2][j+1] * match_filter[filter_choice][14] ;
            fputc(graylvl,fp1);}
        threshol(fp1);
        settextstyle(4,HORIZ_DIR,1) ;
        moveto(10,180) ;
        outtext("ORIGINAL IMAGE") ;
        moveto(160,180) ;
        outtext("PROCESSED IMAGE") ;
        getch() ;
        closegraph() ;
        window(1,1,80,25) ;
        textcolor(YELLOW) ;
        textbackground(BLUE) ;
        clrscr() ;
        return;
    }
    return;
}
```

```
        /* PROGRAM FOR EDGE DECTION BY GRAD DIRECTIONAL */
void grad_directional()
        {
        char source_file[20] ;
        unsigned int image[144][144] ;
        unsigned int graylvl ;
        FILE *fp ;
        FILE *fp1;
        int i,j,ret;
        int grad_choice,filter_choice ;
        int grad_filter[8][9] = { 1, 1, 1, 1,-2, 1,-1,-1,-1,
                                  1, 1, 1, 1,-1,-2, 1,-1,-1, 1,
                                 -1, 1, 1,-1,-2, 1,-1, 1, 1,
                                 -1,-1, 1,-1,-2, 1, 1, 1, 1,
                                 -1,-1,-1, 1,-2, 1, 1, 1, 1,
                                  1,-1,-1, 1,-2,-1, 1, 1, 1 ,
                                  1, 1,-1, 1,-2,-1, 1, 1,-1,
                                  1, 1, 1, 1,-2, 1,-1,-1,-1
                                } ;
        char name[20] = "GRAD_DIRECTIONAL";
        char *ch[9] = {"NORTH","NORTH EAST","EAST","SOUTH EAST",
                       "SOUTH","SOUTH WEST", "WEST","NORTH WEST",
                       "EXIT"};
        clrscr();
        filter_choice = chart(name,ch,9);
        while (filter_choice != 8)
           {
                window(43,21,45,21);
                textbackground(RED) ;
                textcolor(YELLOW) ;
                clrscr() ;
                printf("Enter Source Image file : ") ;
                scanf("%s",&source_file) ;
                clrscr() ;
                fp = fopen(source_file,"rb");
                fp1 = fopen("c:outimage.img","wb+");
                initgraph(&g_d,&g_m,"") ;
                setgraphmode(g_m) ;
                for(i = 0 ; i < 144 ; i++)
                for (j = 0 ; j < 144 ; j++)
                {
                        image[i][j] = fgetc(fp) ;
                        graylvl = image[i][j]/16 ;
                        putpixel(j,i,graylvl) ;
                }
```

```
                for(i = 0 ; i < 144 ; i++)
                for(j = 0 ; j < 144 ; j++)
                  {
                if((i == 0 || j == 0) && (i == 143 || j == 143))
                        {
                        graylvl =256;
                        }
                else
                  {
graylvl = image[i-1][j-1] * grad_filter[filter_choice][0] +
          image[i-1][j]   * grad_filter[filter_choice][1] +
          image[i-1][j+1] * grad_filter[filter_choice][2] +
          image[i][j-1]   * grad_filter[filter_choice][3] +
          image[i][j]     * grad_filter[filter_choice][4] +
          image[i][j+1]   * grad_filter[filter_choice][5] +
          image[i+1][j-1] * grad_filter[filter_choice][6] +
          image[i+1][j]   * grad_filter[filter_choice][7] +
          image[i+1][j+1] * grad_filter[filter_choice][8];
                }
                        putc(graylvl,fp1);
                }
        threshol(fp1);
        settextstyle(4,HORIZ_DIR,1) ;
        moveto(10,180) ;
        outtext("ORIGINAL IMAGE") ;
        moveto(160,180) ;
        outtext("PROCESSED IMAGE") ;
        getch() ;
        closegraph() ;
        window(1,1,80,25) ;
        textcolor(YELLOW) ;
        textbackground(BLUE) ;
        clrscr() ;
        return;
      }
      return;
}
```

```
                /* PROGRAM FOR LAPLACE OPERATIONS */
void laplace()
{
        char source_file[20] ;
        unsigned int image[144][144] ;
        unsigned int graylvl ;
        FILE *fp ;
        FILE *fp1;
        int i,j,z ;
        int lap_choice ;
        int lap_filt[4][9] = {    0, 1, 0, 1,-4, 1, 0, 1, 0,
                                 -1,-1,-1,-1, 8,-1,-1,-1,-1,
                                 -1,-1,-1,-1, 9,-1,-1,-1,-1,
                                  1,-2, 1,-2, 4,-2, 1,-2, 1
                             } ;
        char *title = "LAPLACIAN OPERATORS";
        char *list[5] = { "LAP1", "LAP2","LAP3","LAP4" ,"exit");
        lap_choice = chart(title,list,4);
        if(lap_choice != 4)
                  {     window(10,10,70,15) ;
                        textcolor(YELLOW) ;
                        textbackground(RED) ;
                        clrscr() ;
                        gotoxy(10,3) ;
                        printf("Enter Source Image file : ") ;
                        scanf("%s",&source_file) ;
                        clrscr() ;
                        fp = fopen(source_file,"rb");
                        fp1 = fopen("c:outimage.img","wb+");
                        initgraph(&g_d,&g_m,"") ;
                        setgraphmode(g_m) ;
                        for(i = 0 ; i < 144 ; i++)
                        for (j = 0 ; j < 144 ; j++)
                        {
                        image[i][j] = fgetc(fp) ;
                        graylvl = image[i][j]/16 ;
                        putpixel(j,i,graylvl) ;
                        }
```

```
for(i = 0 ; i < 144 ; i++)
for(j = 0 ; j < 144 ; j++)
{
if((i == 0 || j == 0) && (i == 143 || j == 143))
        {
        graylvl = 256 ;
        }
else
 {
 graylvl = image[i-1][j-1] * lap_filt[lap_choice][0] +
           image[i-1][j]   * lap_filt[lap_choice][1] +
           image[i-1][j+1] * lap_filt[lap_choice][2] +
           image[i][j-1]   * lap_filt[lap_choice][3] +
           image[i][j]     * lap_filt[lap_choice][4] +
           image[i][j+1]   * lap_filt[lap_choice][5] +
           image[i+1][j-1] * lap_filt[lap_choice][6] +
           image[i+1][j]   * lap_filt[lap_choice][7] +
           image[i+1][j+1] * lap_filt[lap_choice][8];
  } ;
                        putc(graylvl,fp1);
           }
        settextstyle(4,HORIZ_DIR,1) ;
        moveto(10,180) ;
        outtext("ORIGINAL IMAGE") ;
        moveto(160,180) ;
        outtext("PROCESSED IMAGE") ;
        getch() ;
        closegraph() ;
        window(1,1,80,25) ;
        textcolor(YELLOW) ;
        textbackground(BLUE) ;
        clrscr() ; }
return;
}
```

```
                    /* PROGRAM FOR THRESHOLDING */
void threshol(FILE *fp)
{
        int i,j,threshold,value;
        rewind(fp);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value = fgetc(fp);
           if ( value  > 100 )
                    value = WHITE;
           else
                    value = BLACK;
           putpixel(j+150,i,value);
          }
        getch();
        return;
}
```

```
                    /* PRUGRAM FOR BRIGHTENING */
void brighten()
  {
        int i,j;
        int value,vl;
        char infile[20];
        FILE *fp;
        initgraph(&g_d,&g_m,"");
        printf("imagefilename:");
        scanf("%s",&infile);
        fp=fopen(infile,"rb");
        for(i=0;i<144;++i)
        for(j=0;j<144;++j)
         {
           value = fgetc(fp);
           vl = value /16;
           putpixel(j,i,vl);
           vl = (vl > 9 ? vl : vl +2);
           putpixel(j+150,i,vl);
         }
   }

                  /* PROGRAM FOR PSEUDOCOLORING  */
void pseudo()
  {
   int color[16] = { 0,3,4,2,1,5,7,20,58,56,59,57,61,60,63,62 };
        int i,j;
        int value,vl;
        char infile[20];
        FILE *fp;
        initgraph(&g_d,&g_m,"");
        printf("imagefilename:");
        scanf("%s",&infile);
        fp=fopen(infile,"rb");
        for(i=0;i<144;++i)
        for(j=0;j<144;++j)
         {
           value = fgetc(fp);
           vl = value /16;
           putpixel(j,i,vl);
           putpixel(j+150,i,color[vl]);
         }
           getch();
                }
```

```
                    / * PROGRAM FOR NEGATION  *
void negate()
{
          char im_file[20];
          int i,j,value;
          FILE *f_p;
          printf("give the image file name");
          scanf("%s",&im_file);
          f_p = fopen(im_file,"rb");
          initgraph(&g_d,&g_m,"");
          setgraphmode(g_m);
          for (i = 0; i < 144; ++i)
          for (j = 0; j < 144; ++j)
            {
             value = fgetc(f_p);
             putpixel(j,i,value/16);
            }
          rewind(f_p);
          for (i = 0; i < 144; ++i)
          for (j = 0; j < 144; ++j)
            {
             value = fgetc(f_p);
             value /= 16;
             value = 16 - value;
             putpixel(j+150,i,value);
            }
          getch();
          closegraph();
          return;
}
```

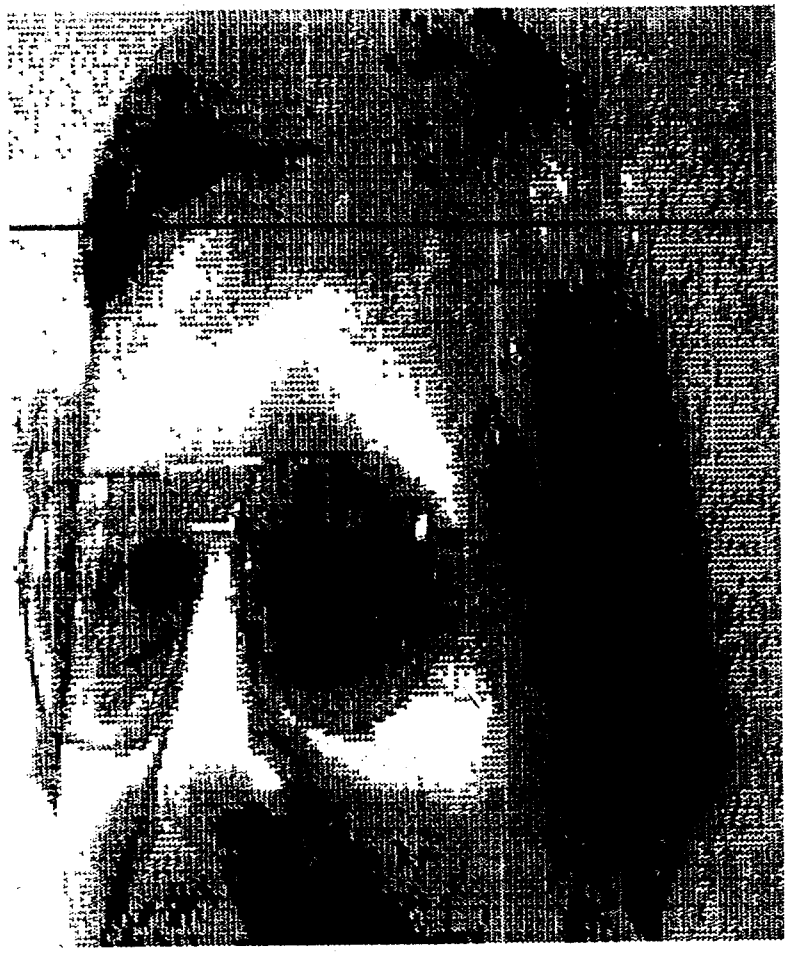
```
        /* PROGRAM FOR THRESHOLDING */
void threshold()
{
        int i,j,threshold,value;
        FILE *f_p;
        char im_file[20];
        printf("give the image file name");
        scanf("%s",&im_file);
        f_p = fopen(im_file,"rb");
        printf("GIVE THRESHOLD VALUE:");
        scanf("%d",&threshold);
        initgraph(&g_d,&g_m,"");
        setgraphmode(g_m);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value = fgetc(f_p);
           putpixel(j,i,value/16);
          }
        rewind(f_p);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value = fgetc(f_p);
           if ( value  > threshold )
                value = 3;
           else
                value = 0;
           putpixel(j+150,i,value);
          }
          getch();
          closegraph();
          return;

}
```

```
                /* PROGRAM FOR SCALING THE IMAGE */
void scale()
  {
   unsigned int i,j,value;
   int s_factor;
   unsigned int image[144][144];
   char infile[20];
   char *scale[3] = {"one","two","three"};
   char *f_name;
   FILE *f_p;
   char *title = "scaling factor";
   printf("give the source filename");
   scanf("%s",infile);
   s_factor = chart(title,scale,3)+1;
   initgraph(&g_d,&g_m,"");
   setgraphmode(g_m);
   f_p = fopen(infile,"rb");
   for ( i = 0; i < 144; ++i)
    for ( j = 0; j < 144; ++j)
      {
       image[i][j] = fgetc(f_p);
           value = image[i][j]/16;
       putpixel(j,i,value);
      }
   for ( i = 0; i < 144; i+=s_factor)
        {for ( j = 0; j < 144; j+=s_factor)
          { switch(s_factor)
               {     case 1:
                     value = image[i][j]/16;
                     break;
                     case 2:
                     value = image[i][j]+image[i+1][j]+
                             image[i][j+1]+image[i+1][j+1]/256;
                                      value /= 256;
                     break;
                     case 3:
                     value = image[i][j]+image[i+1][j]+
                             image[i+2][j]  + image[i][j+1]+
                             image[i+1][j+1]+ image[i+2][j+1]+
                             image[i][j+2]  + image[i+1][j+2]+
                             image[i+2][j+2];
                     value /= 576;
                     break;
               }
       putpixel(j/s_factor+150,i/s_factor,value);

       }
    }
    getch();
    closegraph();
  }
```

```
                /* PROGRAM FOR ROTATING THE IMAGE  */
void rotate()
{
        int i,j,in,jn,value;
        float THETA;
        char infile[20];
        FILE *f_p;
        printf("give the source filename");
        scanf("%s",infile);
        f_p = fopen(infile,"rb");
        printf("give the angle of rotation");
        scanf("%f",&THETA);
        THETA *= PI / 180;;
        initgraph(&g_d,&g_m,"");
        setgraphmode(g_m);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value = fgetc(f_p);
           putpixel(j,i,value/16);
          }
        rewind(f_p);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value = fgetc(f_p);
           in = i*cos(THETA) + j*sin(THETA);
           jn = j*cos(THETA) - i*sin(THETA);
           putpixel(jn+290,in,value/16);
          }
  getch();
  closegraph();
}
```

```
                    /*  PROGRAM FOR MIRRORING  */
void mirror()
 {
        char infile[20];
        char *submenu = "MIRRORING";
        char *mir[2] = {"HORIZANTAL","VERTICAL"};
        int i,j,value[144][144],ret;
        FILE *f_p;
        ret = chart(submenu,mir,2);
        printf("give the source filename");
        scanf("%s",infile);
        f_p = fopen(infile,"rb");
        initgraph(&g_d,&g_m,"");
        setgraphmode(g_m);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value[i][j] = fgetc(f_p);
           putpixel(j,i,value[i][j]/16);
          }
        rewind(f_p);
        switch(ret)
                { case 0:    for(i = 0; i<144 ; ++i)
                             for (j = 144; j > 0; --j)
                             {
                              putpixel(300-j,i,value[i][j]/16);
                             }
                  break;
                  case 1:    for (i = 144; i > 0; i--)
                             for (j = 0;j < 144; ++j)
                             {
                               putpixel(j+150,150-i,value[i][j]/16);
                             }
                  break;
                }
         getch();
         closegraph();

 }
```

```
    /*  PROGRAM FOR TRANSLATING THE IMAGE */
void translate()
 {
        void  far *bitmap;
        unsigned int i,j,k,y,value[144][144],re1;
        unsigned long nb;
        FILE *f_p;
        f_p= fopen("h1.img","rb");
        initgraph(&g_d,&g_m,"");
        setgraphmode(g_m);
        for (i = 0; i < 144; ++i)
        for (j = 0; j < 144; ++j)
          {
           value[i][j] = getc(f_p);
           putpixel(j,i,value[i][j]/16);
          }
        nb = imagesize(0,0,143,143);
        bitmap = farmalloc(nb);
        getimage(0,0,143,143,bitmap);
        cleardevice();
        putimage(10,10,bitmap,XOR_PUT);
        farfree(bitmap);
        getch();
         }
```



Original Image

Processed Image

Original Image Rotation by 90° Processed Image

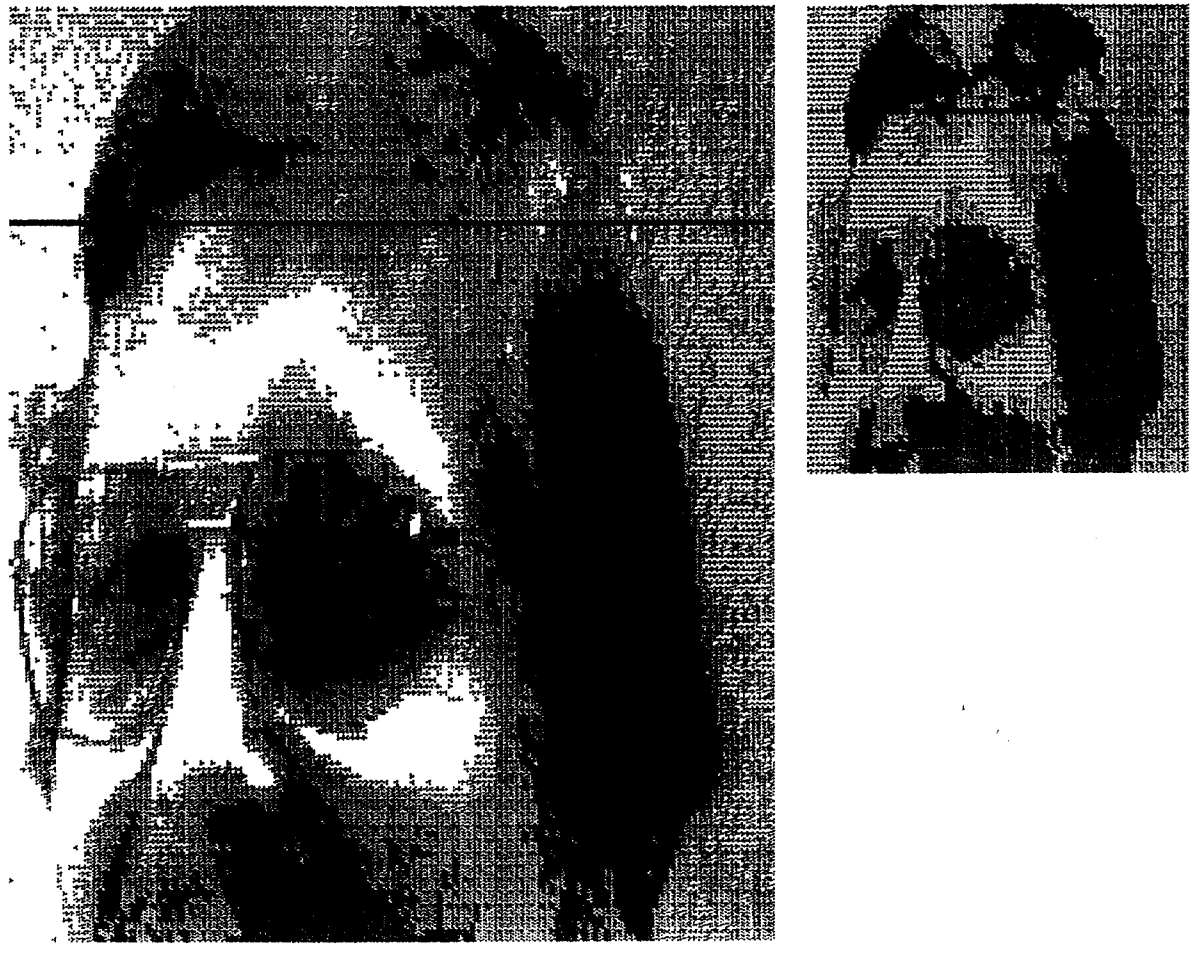


Fig. 1 Scaling by Factor 2