A Novel Approach to Non-Halftone Binary Image Transformations, Digital Halftoning and Color Halftone Proofing

BY

PRADEEP KUNDU

B.PRINT.E. (HONS)

Thesis Submitted in Fulfillment of the Requirement for the Degree of Doctor of Philosophy (Engineering)

JADAVPUR UNIVERSITY

KOLKATA 700 032

INDIA

2013

JADAVPUR UNIVERSITY
Kolkata 700 032

1. Title of the thesis: A Novel Approach to Non-Halftone Binary Image Transformations, Digital Halftoning and Color Halftone Proofing

2. Name, Designation & Institution of the Sepervisor/s: Prof. Dr. Arun Kiran Pal, Professor, Department of Printing Engineering, Jadavpur University

3. List of publication:

    1) Kundu, Pradeep, Pal, Arun Kiran; "Some Methods of Digital Halftoning", TAGA Conference, March 15-18, 2009, New Orleans, Louisiana, USA.
    2) Kundu, Pradeep, Pal, Arun Kiran; "A Color Prediction Model for Additive and Subtractive Mixing of Colors", TAGA Conference, March 15-18, 2009, New Orleans, Louisiana, USA.
    3) Kundu, Pradeep, Pal, Arun Kiran; "Some Methods of Non-halftone Binary Image Transformations", International Journal of Intelligent Information Processing, ISSN: 09-3892, Vol. 4 Number 2, July-December 2010, Pages: 167-172
    4) Kundu, Pradeep, Pal, Arun Kiran; "A Novel Versatile Method Of Generating Soft Halftone Proofs", TAGA Conference, March 12-21, 2012, Jacksonville, Florida, USA.
    5) Kundu, Pradeep, Pal, Arun Kiran; "Non-halftone Binary Image Transformations by Processing the Image in an Arbitrary Path", TAGA Journal of Graphic Technology (to be communicated).
    6) Kundu, Pradeep, Pal, Arun Kiran; "Non-halftone Binary Image Transformations by Processing the Image Row-wise", International Journal of Intelligent Information Processing, ISSN: 09-3892 (to be communicated).

4. List of Patents: NIL

5. List of Presentations in National/ International: NIL

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled "**A Novel Approach to Non-Halftone Binary Image Transformations, Digital Halftoning and Color Halftone Proofing**" submitted by **Pradeep Kundu**, who got his name registered on **29th November, 2006** for the award of Ph.D (Engg) degree of Jadavpur University is absolutely based upon his own work under the supervision of **Prof. Dr. Arun Kiran Pal**, Professor, Department of Printing Engineering, Jadavpur University and that neither his thesis nor any part of the thesis has been submitted for any degree or any other academic award anywhere before.

[illegible] 21/10/13

Signature of the Supervisor
and date with Office Seal

DR. ARUN KIRAN PAL
PROFESSOR
Department of Printing Engineering
Jadavpur University
Salt Lake Campus
Kolkata- 700 098

Acknowledgement

The author takes this privilege to acknowledge his deep gratitude and admiration for Prof. Dr. Arun Kiran Pal, Professor, Department of Printing Engineering, Jadavpur University, under whose keen supervision this research has been carried out. The author is deeply indebted to him for his valuable suggestions, inspiring guidance and help which made possible the completion of this research.

The author wishes to take this opportunity to express his gratitude and record his thanks to Dr. Mark Bohan, TAGA, and to Mr. Vijay Kumar of Serial Publications, for their appreciation in this research work.

Thanks are due to Dr. Swati Bandyapadhyay, Associate Professor, Department of Printing Engineering, for her interest, kind help and appreciation of the work.

The author express his gratitude to the all teaching and non-teaching staff of the Department of Printing Engineering for their necessary help and assistance to complete this work.

During the tenure of the works, love and inspiration received by the author from Mrs. N. Kundu, Mr. M.M. Kundu, Mrs. G. Kundu, Mr.S. Kundu and Mr. D. Kundu deserve special mentioning. As a close blood relation they gave me essential support and encouragement in various ways during the long period of this study.

PRADEEP KUNDU

Contents

Chapter 1

Introduction

Chapter 1

Introduction

1.1 Introduction

Transforming gray images to binary ones through non-halftone binarization (algorithm driven digital line conversion methods) and novel digital halftoning techniques and building multicolor soft halftone proof would always find their general and special application in the field of printing and imaging science and engineering. This present research work concentrates on such type of approaches to the design of image transformation methods for converting gray scale images to non-halftone binary and halftone images and generation of multicolor soft halftone proofs.

Chapter 1 gives an introduction to the background works and related theory on non-halftone binarization, digital halftoning techniques and soft halftone proof.

Chapter 2 gives a review of the early works prior to the present research work.

Chapter 3 introduces various novel methods of non-halftone binary image transformations. This chapter also compares the novel techniques with the earlier analog techniques.

Chapter 4 describes two novel halftoning techniques. This chapter also compares the new techniques with the earlier techniques.

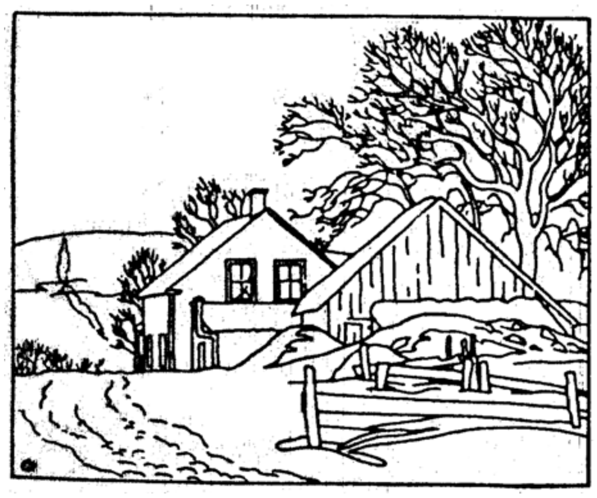
Chapter 5 introduces one novel methods of soft halftone proofing. This chapter also compares this novel technique with the other earlier techniques.

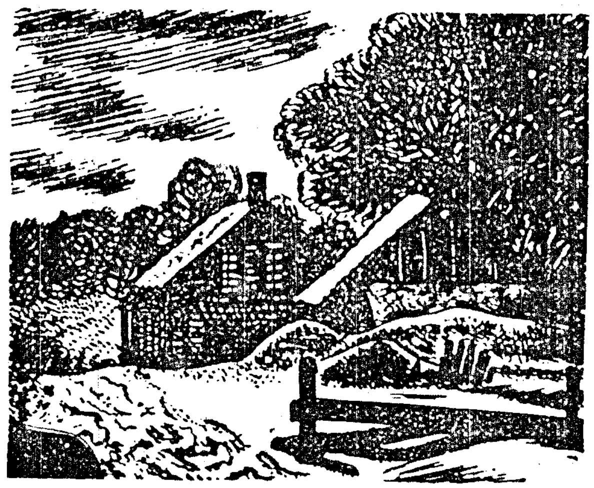
Chapter 6 draws a conclusion and describes the scope for future investigation.

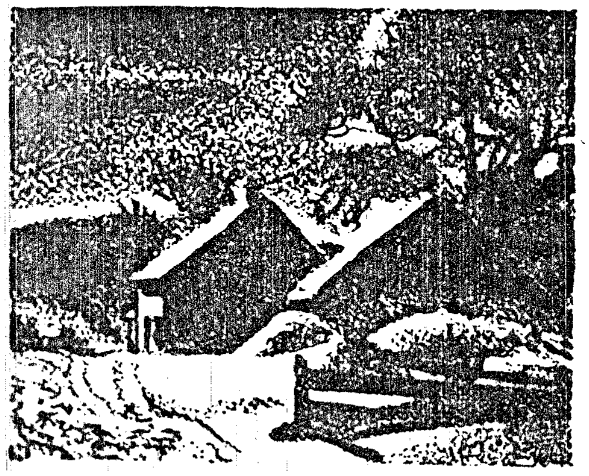
1.2 Originals for Reproduction

Copies are divided into two general classes, line and continuous tone, and two subdivisions viz. combination and color. Combination copies represent both line and continuous tone originals as single unit, while color copies are restricted to full-color originals, although multicolor effects can be produced from monochrome (black and white) copies by various means and processes.

Fig. 1.1 shows some line drawing techniques like Outline pen drawing, Brush drawing, Charcoal drawing and Scratchboard drawing


(a) Outline pen drawing


(b) Brush drawing


(c) Charcoal drawing

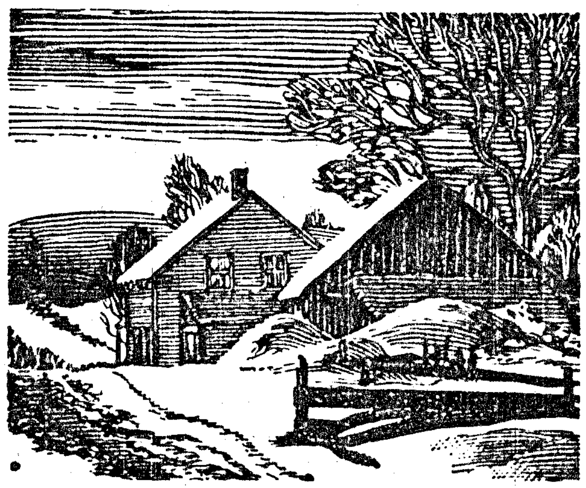
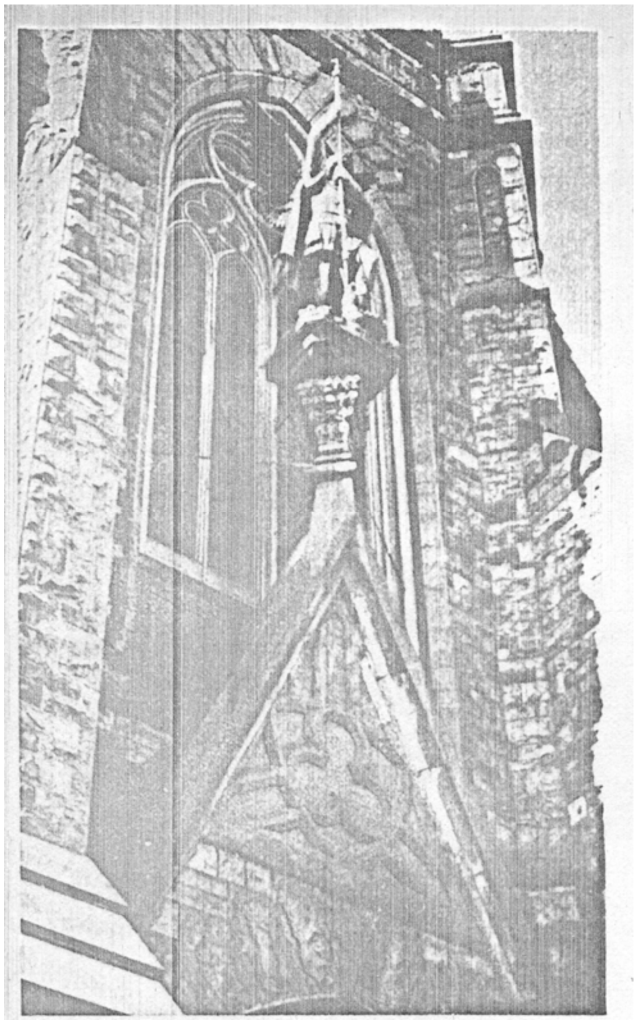

(d) Scratchboard drawing

Figure 1.1: Line drawing Techniques

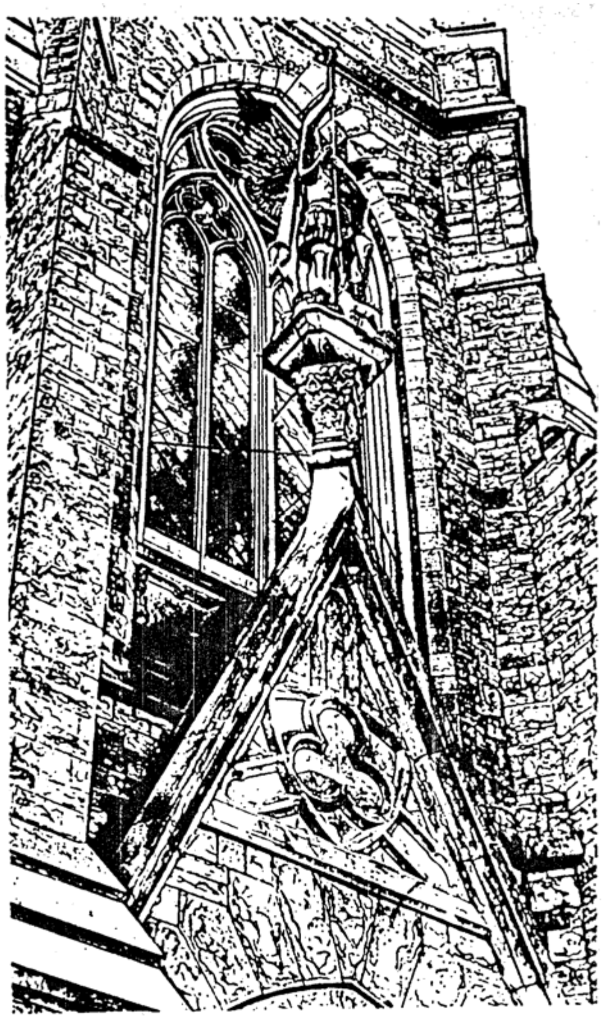
1.3 Line Reproduction from Continuous Tone Originals

1.3.1 Tone-Line Process

The Eastman Kodak Company introduced in 1953, a photographic method for making line reproductions from continuous tone originals as the Tone-Line Process [35]. In this procedure a continuous tone negative is masked with a positive image of nearly equal contrast and the assembly exposed on a sheet of Kodalith film by rapidly spinning it (in a printing frame) under a fixed light, or by rotating a movable light above a stationary vacuum printing frame. Either method allows some light to creep around the edges of the masked negative and produce the line effect displayed in fig.1.2, this shows the continuous tone original and the result obtained therefrom by the Tone-Line process.



(a) Tone-Line Reproduction: Original Image



(b) Tone-Line Reproduction: Reproduced Image

Figure 1.2 : Tone-Line Reproduction

1.4 Halftoning

Halftoning is the process of breaking the contone (continuous tone) images into dots to create an illusion of tonal gradation by physical (screens) or logical (algorithmic tools) means to circumvent the constraints of printing process or device to produce contone.

1.4.1 AM and FM Halftoning

The halftoning methods can mainly be divided into two main types, namely AM (Amplitude Modulated) and FM (Frequency Modulated). In the AM methods the sizes of the halftone dots vary, while their spatial frequency is constant. This means that the size of the halftone dot becomes bigger as the tone gets darker. In the FM methods, on the other hand, the dot size is constant while the frequency (the number of micro dots) varies. Something worth observing here is that the terms AM and FM halftoning are sometimes incorrectly replaced by conventional and stochastic halftoning, respectively.

1.4.2 Dithering Techniques

The term dithering is used in various contexts [19]. Primarily, it refers to techniques for approximating halftones without reducing resolution, as pixel-grid patterns do. But the term is also applied to halftone-approximation methods using pixel grids, and sometimes it is used to refer to color halftone approximations only.

Main classes of dithering

1. Ordered dither
   1.1. Clustered dot dither
   1.2. Dispersed dot dither
2. Error Diffusion
   2.1. Floyd and Steinberg
   2.2. Jarvis filter weight
   2.3. Stucki’s filter weight
   2.4. Shiau’s & Fan’s filter weight
   2.5. Knuth’s Dot Diffusion

1.4.2.1 Ordered dither

The ordered dithering techniques are divided into two parts, clustered dot and dispersed dot. In the clustered dot dithering the threshold matrices are arranged in a way that the final halftone dot is a cluster of black microdots. In the dispersed dot dithering the black microdots are dispersed. Fig. 1.3 is a clustered dot dither array and fig. 1.4 is the enlarged image of the same array. Fig.1.5 is a dispersed dot dither array (Bayer dither) and fig.1.6 is the enlarged image of the same array. Clustered dot dither is AM (Amplitude Modulated) and Dispersed dot dither is FM (Frequency Modulated). Error diffusion dithers are FM (Frequency Modulated).

3	9	17	27	25	15	7	1
11	29	38	46	44	36	23	5
19	40	52	58	56	50	34	13
31	48	60	64	62	54	42	21
30	47	59	63	61	53	41	20
18	39	51	57	55	49	33	12
10	28	37	45	43	35	22	4
2	8	16	26	24	14	6	0

Figure 1.3: An 8x8 dither array (clustered dot)

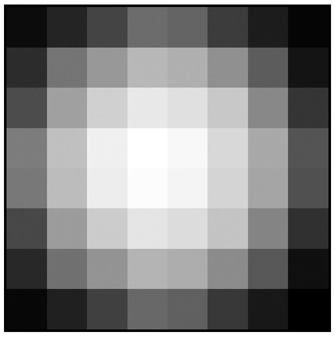


Figure 1.4: Image of a clustered dot dither array

0	58	14	54	3	57	13	53
32	16	46	30	35	19	45	29
8	48	4	62	11	51	7	61
40	24	36	20	43	27	39	23
2	56	12	52	1	59	15	55
34	18	44	28	33	17	47	31
10	50	6	60	9	49	5	63
42	26	38	22	41	25	37	21

Figure 1.5: An 8x8 Bayer dither array (dispersed dot)

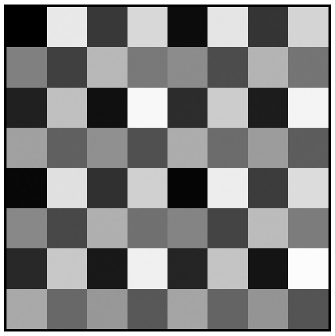


Figure 1.6: Image of a dispersed dot dither array

1.4.2.2 Error Diffusion

For a far better approach to FM halftoning, Floyd and Steinberg [10] proposed the revolutionary error diffusion algorithm, an adaptive technique that quantized each pixel according to a statistical analysis of an input pixel and its neighbors (neighborhood process), leading to a stochastic arrangement of printed dots. Floyd's and Steinberg's proposed error filter is shown in fig.1.7. Fig. 1.8 shows Floyd and Steinberg algorithm applied to a grayscale ramp. Fig.1.9 shows same algorithm applied to an image.

	●	7/16
3/16	5/16	1/16

Figure 1.7 : Floyd's and Steinberg's
Proposed error filter

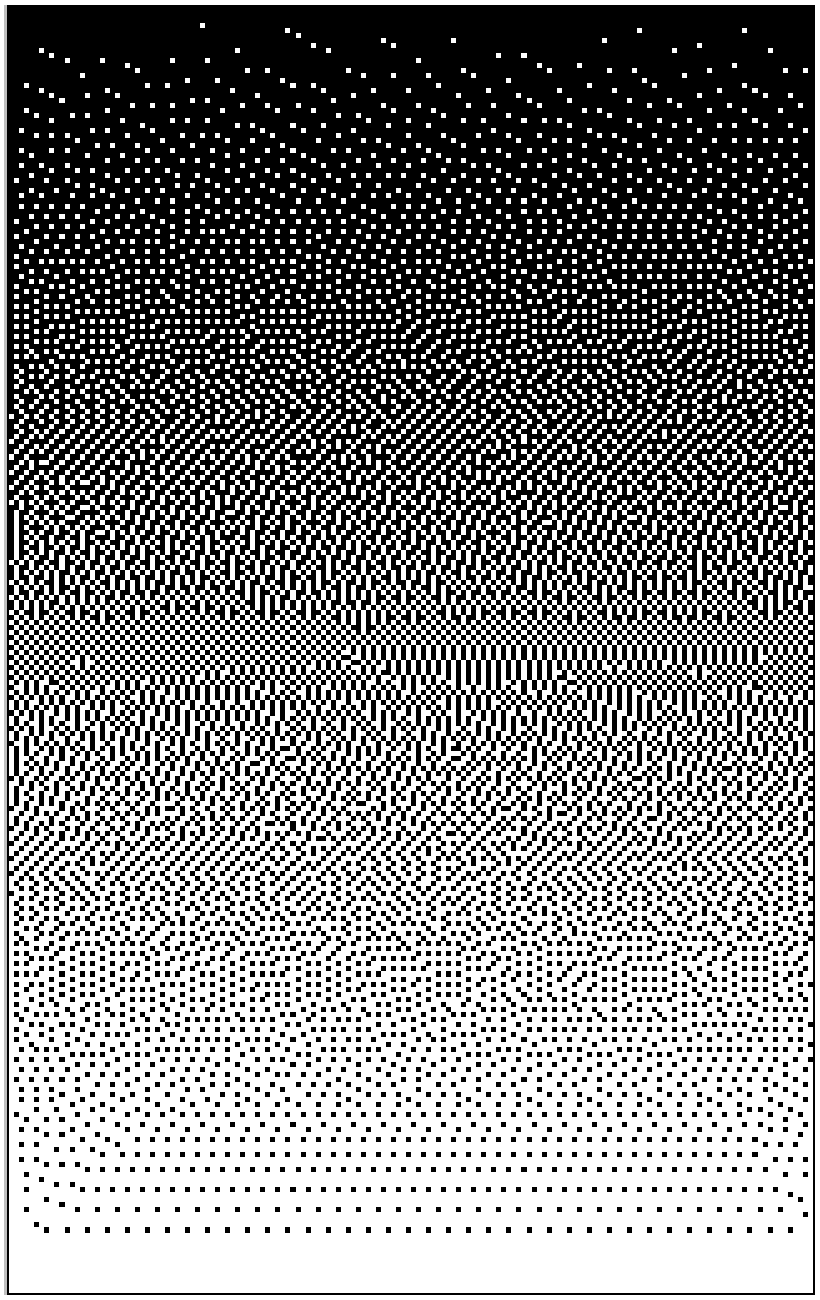


Figure 1.8: Error diffusion (Floyd and Steinberg)

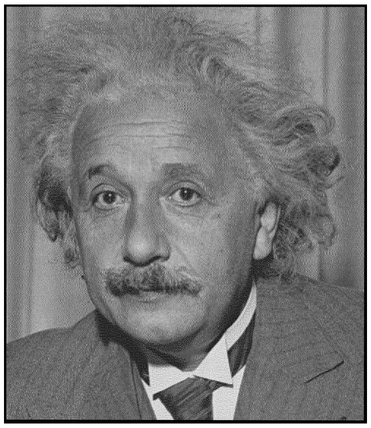


Figure 1.9: Image formed by Floyd and Steinberg error diffusion

1.4.3 Parameters of AM Halftoning

1.4.3.1 Screen Frequency

The screen frequency is the number of lines or rows of clustered-dots per inch of the resulting halftone. This is illustrated in fig. 1.10.

$$\text{No of unique gray levels [3]} = \left(\frac{\text{Resolution (DPI)}}{\text{Screen Frequency (LPI)}} \right)^2 + 1 \qquad 1.1$$

Equation 1.1 shows number of gray levels that can be determined from resolution and screen frequency of a halftone image.

1.4.3.2 Dot Shape

Dot shape refers to the specific arrangement of thresholds within the dither array. This is illustrated in fig. 1.10.

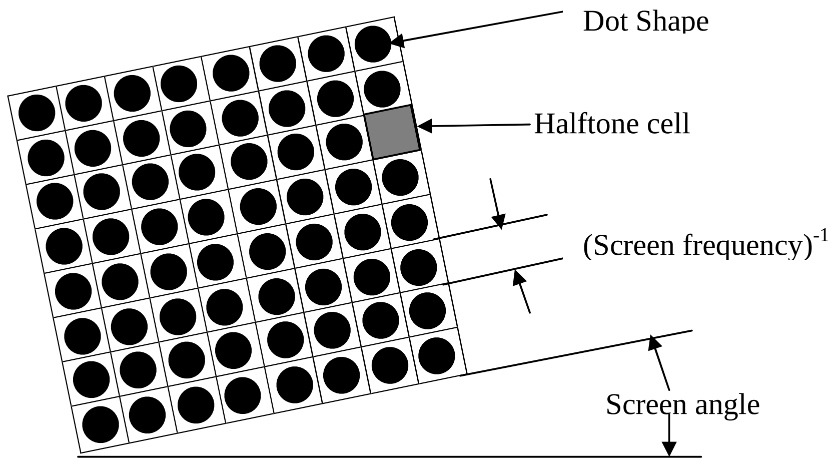


Figure 1.10: The screen frequency, dot shape, and screen angle for an analog halftone screen

1.4.3.3 Screen Angle

It is the orientation of screen lines relative to the horizontal axis. This parameter is a function of human visual system where directional artifacts are their least noticeable when oriented along the 45 degree diagonal. It follows that for monochrome printing, this screen angle should be also 45 degree. The screen angle plays a fundamental role in elimination of moiré, the interference pattern produced by super imposing two or more regular patterns. Fig. 1.12 shows the moiré patterns associated with superimposed just two regular grids. Fig. 1.13 shows moiré pattern in four process color halftoning due to incorrect use of screen angle. While this interference cannot be altogether eliminated, it is through the use correct screen angle i.e. 75º, 105º, 90º and 45º for cyan, magenta, yellow and black respectively, that moiré is minimized, creating pleasant rosette pattern of fig.1.14 – fig.1.16.

Fig.1.11 shows a vignetted dot within an analog halftone screen cell which is responsible for variable dot size produced by the conventional halftone screen.

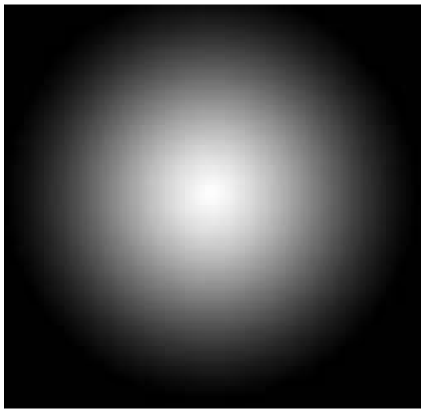
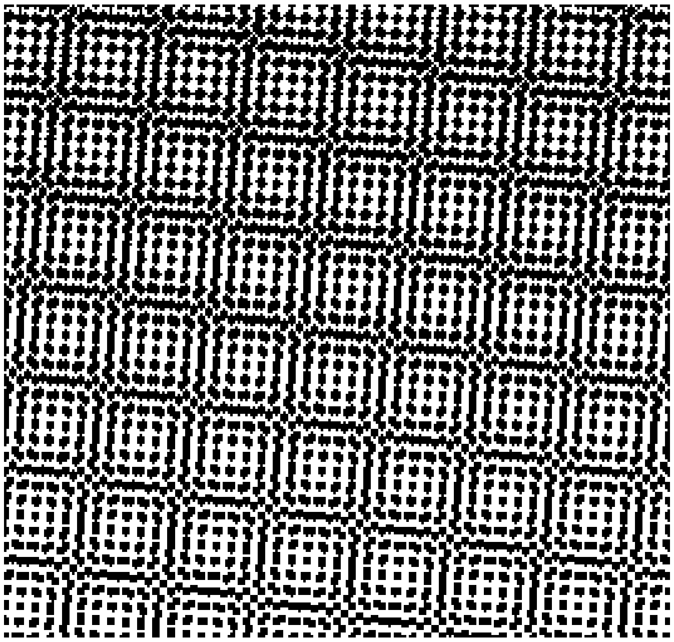
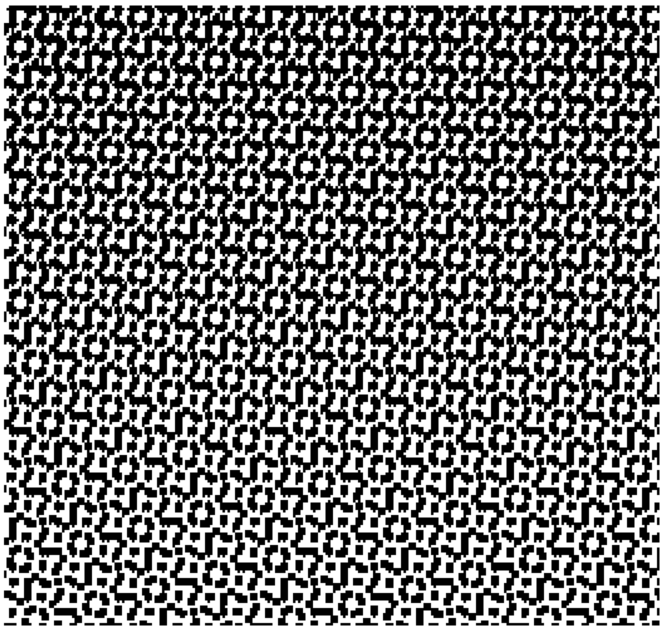


Figure 1.11: A vignetted dot within an analog halftone screen cell which is responsible for variable dot size


(a) 10 degree



(b) 30 degree

Figure 1.12: The Moiré patterns created by offsetting two AM halftone patterns by (a) 10 degree (b) 30 degree

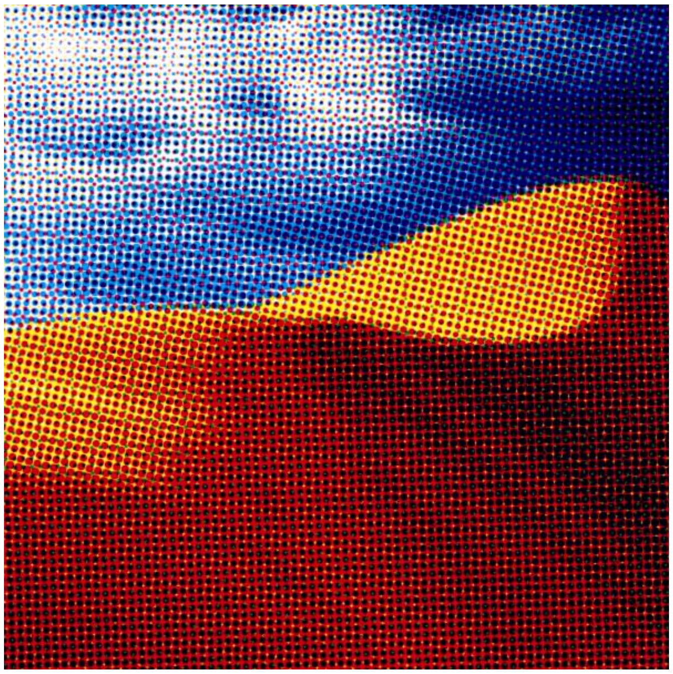


Figure 1.13: A CMYK AM halftone proof with incorrect screen angles and Moiré patterns appear. Screen angles are C 92º, M 75º, Y 0º and K 45º

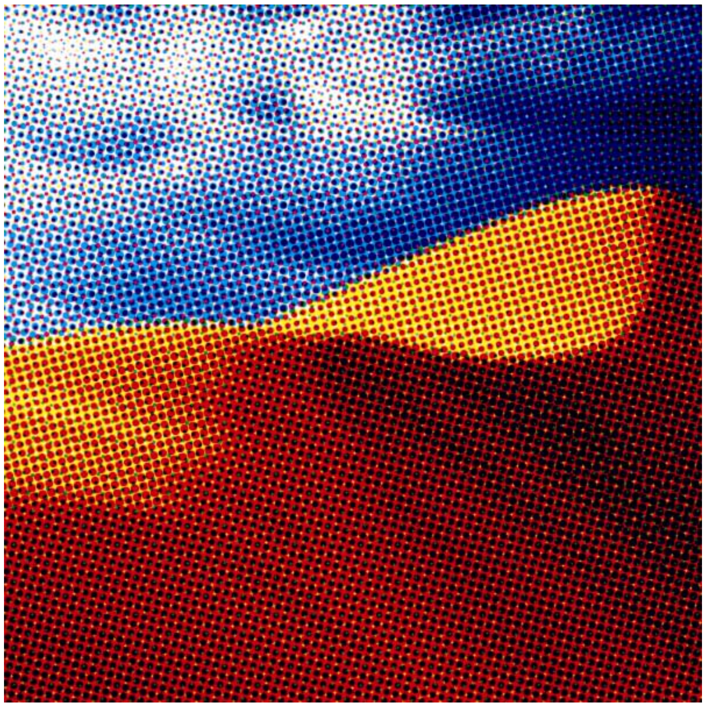


Figure 1.14: A CMYK AM halftone proof with correct screen angles, no Moiré pattern but only Rosette. Screen angles are C 75º, M 105º, Y 90º and K 45º

1.4.4 Moiré and Rosette in AM Halftoning

If the job required 120 lpi, place the yellow on 133 lpi. The ideal frequency ratio of yellow to other colors is about 1.11 higher.

When the colors are correctly angled, a rosette pattern will be visible in highlight and middletone areas where all the colors are present. The frequency of rosette pattern is such that it occurs at one half that of the screen ruling; i.e., the rosettes appear at 75 per inch frequency for a 150 lpi process color halftone.

There are two types of rosette pattern [9] : those with a clear center (fig.1.16) and those with a dot center (fig.1.15). Dot centered rosettes are less noticeable but must be printed to tighter register tolerances in order to minimize color variations in certain tonal values (this is the reason why dot-on-dot same angle printing has generally been unsuccessful, despite its superior resolution when compared to multi-angle printing).

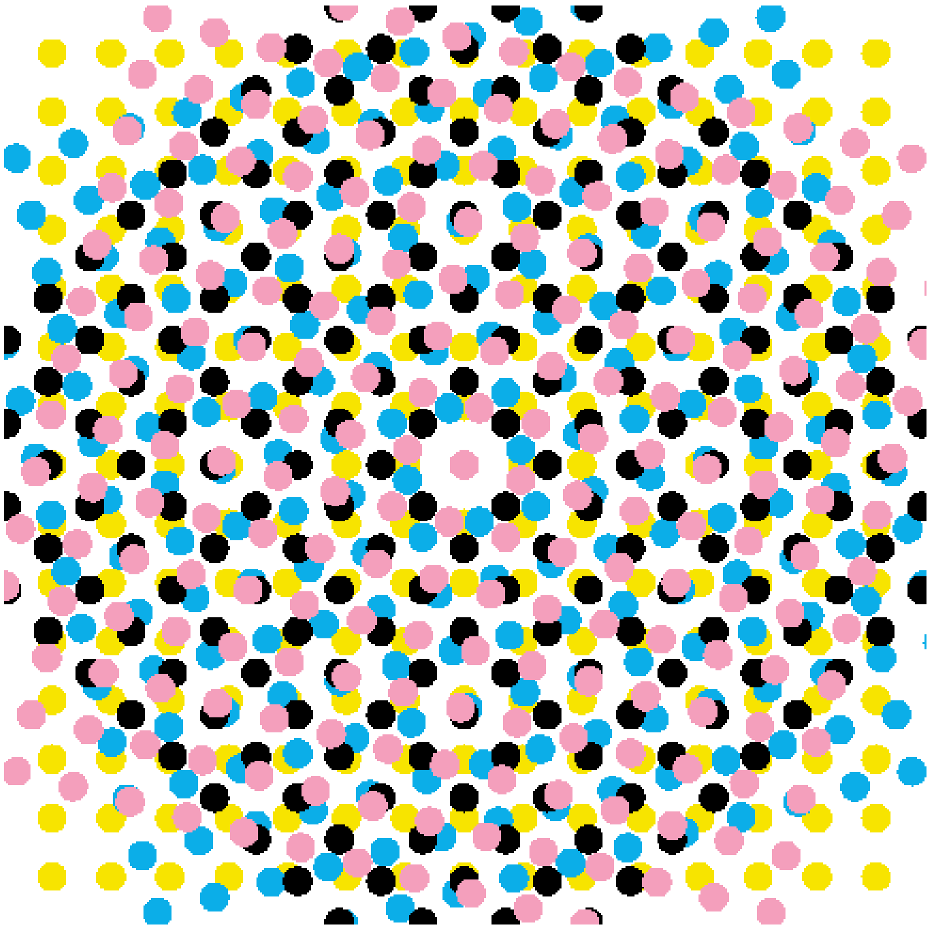


Figure 1.15: Rosette Pattern (dot centered or closed)

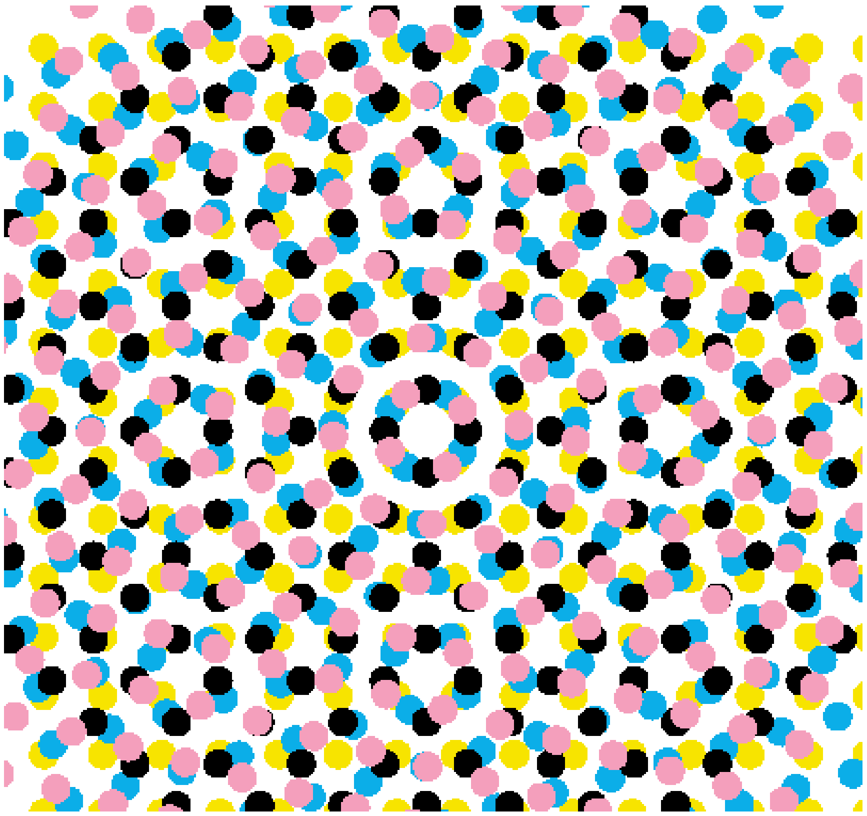


Figure 1.16: Rosette Pattern (clear centered or open)

1.5 Object of the Present Work

The present research work has been subdivided into three sections.

1. Non-Halftone Binary Image Transformations.
2. Digital Halftoning.
3. Soft Halftone Proofing.

Non-Halftone Binary Image Transformations: One earlier method of converting continuous tone image into line work is Tone Line process, developed by Eastman Kodak in the year 1953. After that, in analog field no system is found (to the best of the knowledge of the author), which is better and equivalent to Tone Line process. This part of the research work aims to develop various non-halftone binarization methods. The methods might be

1. Row-wise and column-wise processing
2. Blockwise processing,
3. Processing based on pixel location and path based on location.

3.1. Processing the image column-wise

3.2. Processing the image row-wise

3.3. Processing the image in arbitrary path

3.4. Processing the image diagonally

- 3.4.1 Starting point of the diagonal is Left Top or Right Bottom (LTRB)
  - 3.4.1.1 LTDAVFTC (Left top diagonal alternate vector spiral with first turn clockwise)
  - 3.4.1.2 LTDAVFTAC (Left diagonal alternate vector spiral with first turn anti-clockwise)
  - 3.4.1.3 LTDVLB2RT (Left top diagonal vectors, left bottom to right top)
  - 3.4.1.4 LTDVRT2LB (Left top diagonal vectors, right top to left bottom)
  - 3.4.1.5 RBDAVFTAC (Right bottom (starting point) diagonal alternate vector spiral with first turn anti-clockwise. 5 is reverse of 1)
  - 3.4.1.6 RBDAVFTC (Right bottom diagonal alternate vector spiral with first turn clockwise. 6 is reverse of 2)
  - 3.4.1.7 RBDVRT2LB (Right bottom diagonal vectors, right top to left bottom. 7 is reverse of 3)
  - 3.4.1.8 RBDVLB2RT (Right bottom diagonal vectors, left bottom to right top. 8 is reverse of 4)
- 3.4.2 Starting point of the diagonal is Right Top or Left Bottom (RTLB)
  - 3.4.2.1 RTDAVFTC (Right top diagonal alternate vector spiral with first turn clockwise)
  - 3.4.2.2 RTDAVFTAC (Right top diagonal alternate vector spiral with first turn anticlockwise)
  - 3.4.2.3 RTDVRB2LT (Right top diagonal vectors, right bottom to left top)
  - 3.4.2.4 RTDVLT2RB (Right top diagonal vectors, left top to right bottom)
  - 3.4.2.5 LBDAVFTAC (Left bottom (starting point) diagonal alternate vector spiral with first turn anti-clockwise. 5 is reverse of 1)
  - 3.4.2.6 LBDAVFTC (Left bottom diagonal alternate vector spiral with first turn clockwise. 6 is reverse of 2)

3.4.2.7 LBDVLT2RB (Left bottom diagonal vectors, left top to right bottom. 7 is reverse of 3).

3.4.2.8 LBDVRB2LT (Left bottom diagonal vectors, right bottom to left top. 8 is reverse of 4)

Digital Halftoning: Digital halftoning has replaced conventional halftoning long back. Various authors have contributed in this field (Refer Section 2.3). This part of the research aims to contribute to novel halftoning methods specially in ordered dithering. The methods might be

1. Halftoning by pre-embedding the pattern
2. Halftoning by simulating character-writing pattern

Soft Halftone Proofing: This part of the research work aims to develop novel technique in soft halftone proofing which could be used directly to produce amplitude modulated and frequency modulated halftone proof and display it on the monitor screen in comparison to earlier techniques (Refer Section 2.4).

A detailed review has been done on the three sections.

1.6 Scope of the Present Work

Non-halftone binary image transformations are implemented through various algorithms to convert gray images into non-halftone binary images. These methods of non-halftone binarization could be used as imaging tools for special applications and sometimes as a substitute for halftone where semi-halftone nature of the image is seen.

In the digital halftoning section, two novel techniques are presented to generate halftones. The first method titled “Halftoning by pre-embedding the pattern” is a technique which could be used for general purpose digital halftoning. The second method titled “Halftoning by simulating character-writing pattern” could be used in special halftoning (i.e. to show a change in halftone dot percentage with the progressive writing of strokes, to complete a character) application.

The method of generating soft halftone proof as presented in Chapter 5 is a unique, novel and versatile method with wider application than the existing methods. It could be used to produce both amplitude and frequency modulated onscreen soft halftone proofs prior to taking hardcopy proofs through printers like color laser printer, color ink jet printer or other suitable output systems.

Chapter 2

Review of Early Works

Chapter 2

Review of Early Works

2.1 Introduction

In this chapter the author has presented some earlier photomechanical and digital imaging and imageprocessing methods. These are the existing techniques that might serve as a background for the present research work.

2.2 Non-Halftone Binary Image Transformations

In the analog photomechanical field one method e.g. Tone Line Process [35], has been discussed in Chapter 1 (Section 1.3.1) which is equivalent to non-halftone binary image conversion. To the best of the knowledge of the author there is no digital non-halftone binarization method (algorithm driven digital line conversion methods) that could serve as a background for the present work.

2.3 Digital Halftoning

Before digital halftoning came into the picture, electronic halftoning were used in electronic drum scanners like Crossfield Magnascan, which made use of electronic dot generation (EDG) through built in analog computing systems. Before electronic halftoning conventional halftoning using glass crossline screen and contact screen were made. After electronic halftoning many digital halftoning methods have been discovered which are outlined in the following paragraphs.

In 1973, Bayer [4] popularized the most widely used patterns for dispersed dot ordered dither. Fig. 1.5 and fig. 1.6 shows and 8x8 Bayer dither array and image of the same array respectively.

In 1976, Floyd and Steinberg [10] discovered revolutionary error diffusion algorithm. Fig. 1.7 shows Floyd's and Steinberg's proposed error filter.

In 1976, Jarvis et al. [22] discovered further improvement in error diffusion. In an effort to break up worm patterns in error diffusion, they introduced 12-element error filters.

In 1981, Stucki [44], modified error diffusion with different filter weight. Similarly in an effort to break up worm patterns in error diffusion, they also introduced 12-element error filters.

In 1987, Knuth [24] introduced algorithm for dot diffusion. This algorithm leads to strong periodic patterns, where pixels are processed in the order of clustered dot ordered dither.

In 1988, Ulichney [48], Blue noise (error diffusion). Blue noise halftoning is characterized by a distribution of binary pixels where the minority pixels are spread as homogeneously as possible.

In 1993, Ulichney [50], Void and clustered dither: Among the modifications of ordered dither, a prominent place belongs to Ulichney's void and cluster method, which is also dispersed dot ordered dither.

In 1996, Shiau's and Fan's [42] error diffusion with different filter weight. Here, the filter is restricted to just two rows but extraordinarily long tail.

In 1999, Mese and Vaidhyanathan [36] introduced improved dot diffusion and showed that order can optimized for 16 x 16 blocks to results of comparable visual quality to error diffusion.

The author has proposed two new methods in digital halftoning (ordered dither) that are presented in Chapter 4.

2.4 Soft Halftone Proofing

Images almost equivalent to halftone proofs have been generated through existing image editing softwares like Adobe Photoshop 7 and Corel PhotoPaint 11 in fig. 2.1 and fig. 2.5 respectively. These halftone proofs are mainly meant for amplitude modulated halftoning. In the table: 2.1 the original image “flcmyk.tif” (fig.5.3) is at 144 dpi, and screen angles and max dot radius are also given to generate the proofs. Individual dots in each color channel of the proofs are antialiased, which must not be there for any halftone image. Fig. 2.3 and fig. 2.7 are the part of the enlarged black channels containing antialiased dots for Photoshop 7 and Corel PhotoPaint 11 respectively. Fig. 2.4 and fig. 2.8 are the histograms of the fig.2.3 and fig.2.7 respectively, which show tonal variation because of having antialiased dot structure. Fig.2.2 and fig.2.6 shows histograms of the four channels of the fig.2.1 and fig.2.5 respectively. These histograms of fig.2.2 and fig.2.6 show that there is tonal variation within the individual channels and colors used are not pure. The author has proposed a more versatile method i.e. with wider applications, using Photoshop 7 as described in the Chapter 5.

Table 2.1: Filename, channels and other data for proof generation through Photoshop 7 and Corel PhotoPaint 11

Original Image	Resolution (DPI)	Color	Screen Angle (Degree)	Max Radius of a Dot (Pixels)
flcmyk.tif (Fig.5.3)	144	Cyan	108	4
		Magenta	162	
		Yellow	90	
		Black	45	

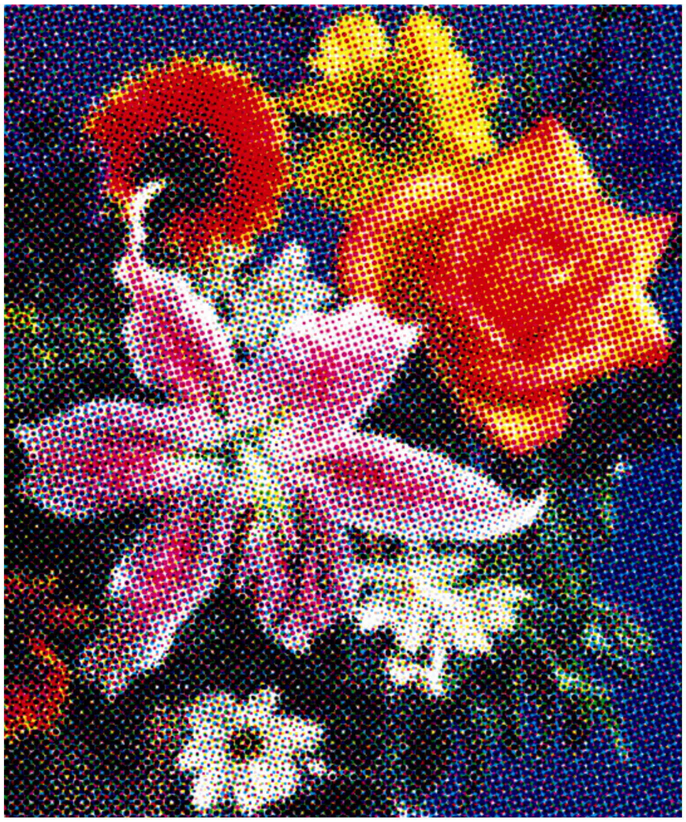


Figure 2.1 : Halftone Proof (Photoshop 7)

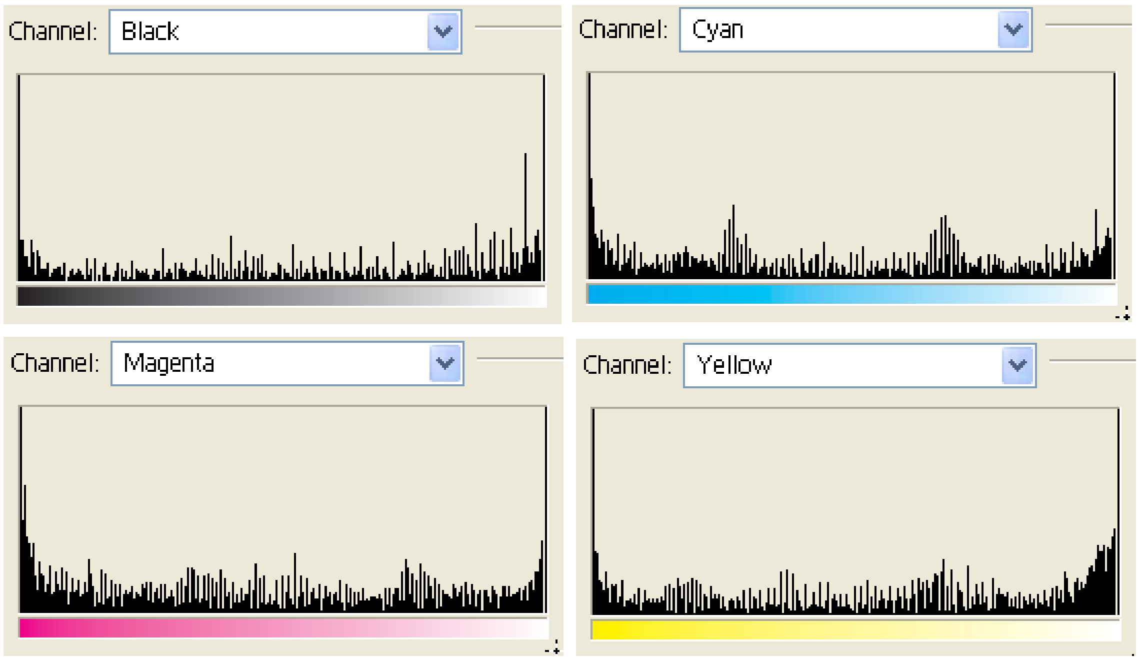


Figure 2.2 : Histograms of the four channels of the figure 2.1

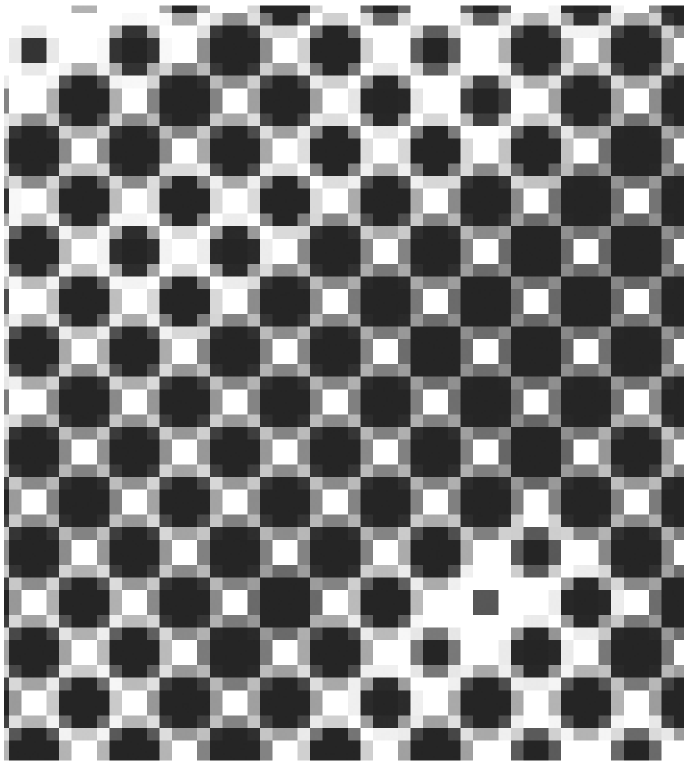


Figure 2.3 : Antialiased Dots (Photoshop 7)

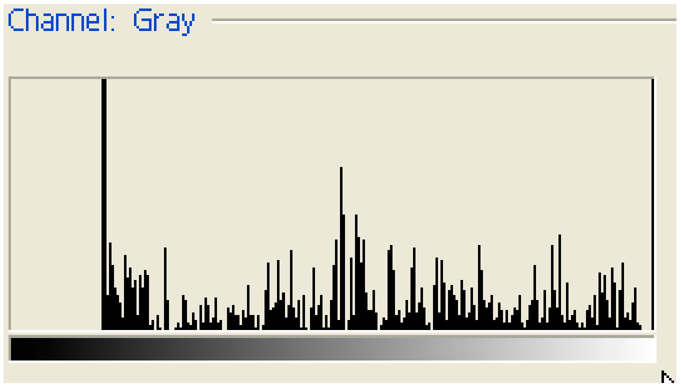


Figure 2.4 : Histogram of the figure 2.3

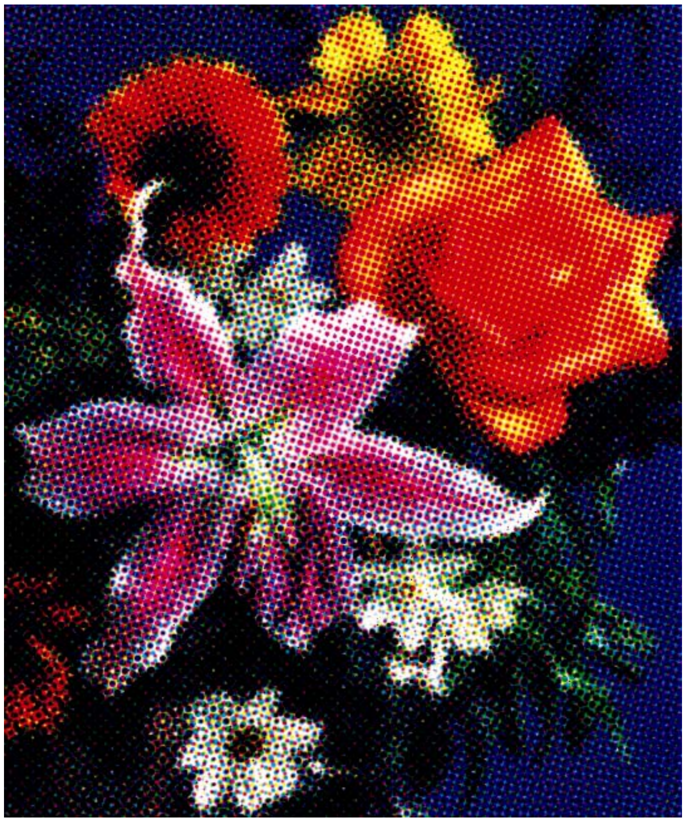


Figure 2.5 : Halftone Proof (Corel Photo-Paint 11)

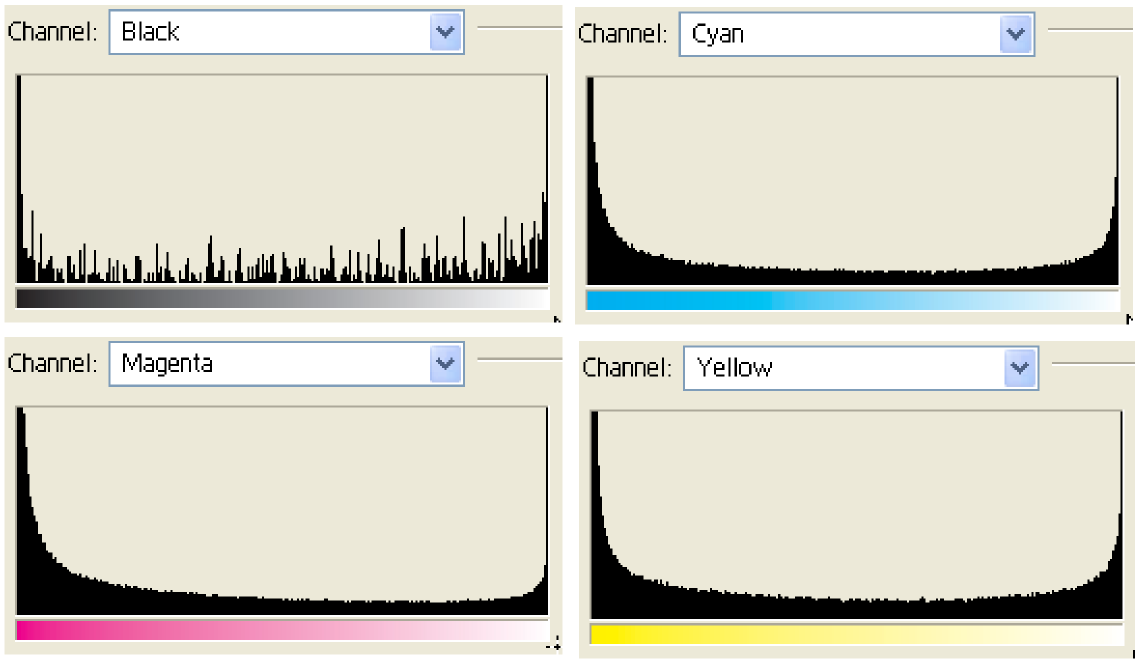


Figure 2.6 : : Histograms of the four channels of the figure 2.5

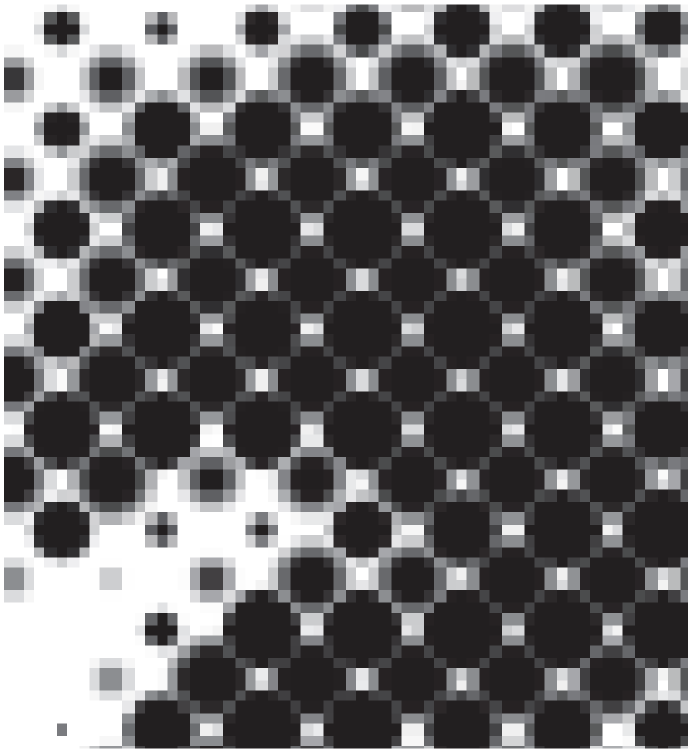


Figure 2.7 : Antialiased Dots (Corel Photo-Paint 11)

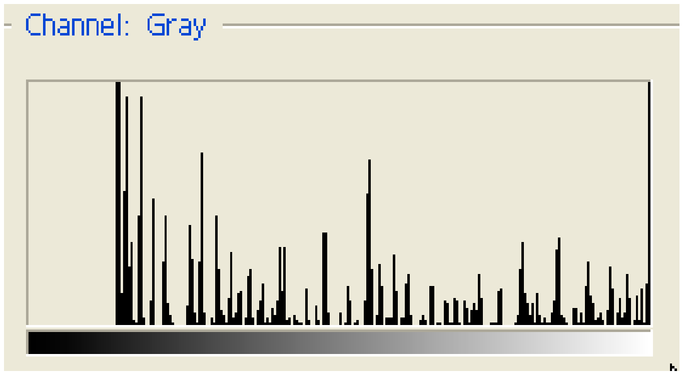


Figure 2.8: Histogram of the figure 2.7

Chapter: 3

Non-Halftone Binary Image Transformations

Chapter: 3

Non-Halftone Binary Image Transformations

3.1 Introduction

Reproducing a continuous tone image is always a challenging job. In the pre halftone era, when analog halftoning techniques were not known, non-halftone binary images were preferred to reproduce contone images, where people used to apply their own hand drawn techniques. In analog systems, particularly, in photomechanics, Eastman Kodak Company introduced Toneline process in 1953; it is a photographic method for making line reproduction from continuous tone original. This chapter describes a novel approach towards non-halftone processes of converting a gray image into binary images using different algorithm driven programs.

3.2 Thresholding as a Binarization Tool

To generate binary images which are of non-halftone types, the first method that come into picture is converting the whole image into a binary one (0 or 1, black and white) by thresholding. It is one of the basic methods used in digital halftoning also.

The thresholding can be expressed as follows:

Let ‘thv’ be the threshold value, ‘im’ be the image pixel value

Then if im<=thv, im=0 and im>thv, im=1

3.2.1 The Image Matrix for Threshold Operation

An original image matrix m (3 by 4) that is equivalent to a gray image is given by

```
m=   [10   20   30   40
       15   35   25    5
       55   50    4   12]
```

```
md=   [0.0392   0.0784   0.1176   0.1569
       0.0588   0.1373   0.0980   0.0196
       0.2157   0.1961   0.0157   0.0471]
```

md matrix is the double datatype equivalent of the above m matrix

3.2.2 Threshold Option: 1

a1 = mean (m, 1) = 26.6667   35.0000   19.6667   19.0000 (mean of the elements of three rows)

a2 = mean (a1, 2) = 25.0833

thv = 25.0833 (final threshold value) (0.0984 for double)

Resultant thresholded matrix is:

```
mth=   [0   0   1   1
        0   1   0   0
        1   1   0   0]
```

3.2.3 Threshold Option: 2

a3 = maximum element of m, i.e. 55

a4 = minimum element of m, i.e. 4

thv = mean of a3 and a4, i.e. 29.5

Resultant thresholded matrix is:

```
mth=   [0   0   1   1
        0   1   0   0
        1   1   0   0]
```

3.2.4 Threshold Option: 3

50% threshold, i.e. thv = 128

Resultant thresholded matrix is:

$$mth= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

There will be minute difference in images with ‘uint8’ and ‘double’ datatype-based threshold. Thresh function (Program 3.1) does ‘uint8’ data type-based thresholding

Matlab 6.1’s ‘graythresh’ function uses Otsu’s method [38] to give threshold level 0.1059 for the md matrix.

Threshold option 1 and option 2 are the new methods for threshold selection, proposed by the author.

3.2.5 Threshold Operations: Input and Output Images

Various threshold operations are done on fig.3.T1 to produce the following pictures (fig.3.T2-fig.3.T7)

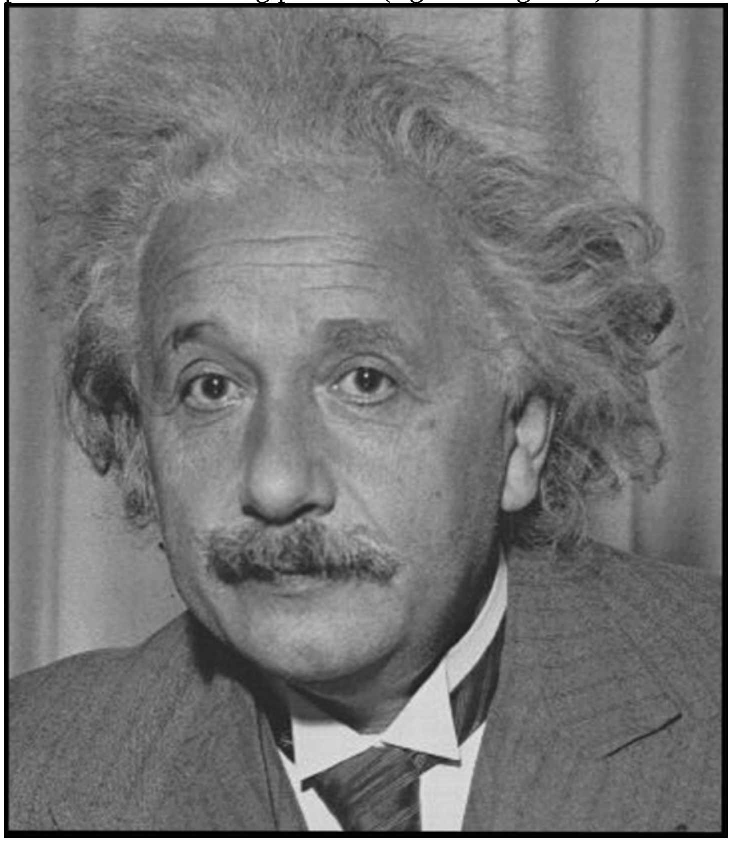


Figure 3.T1: Sample Image

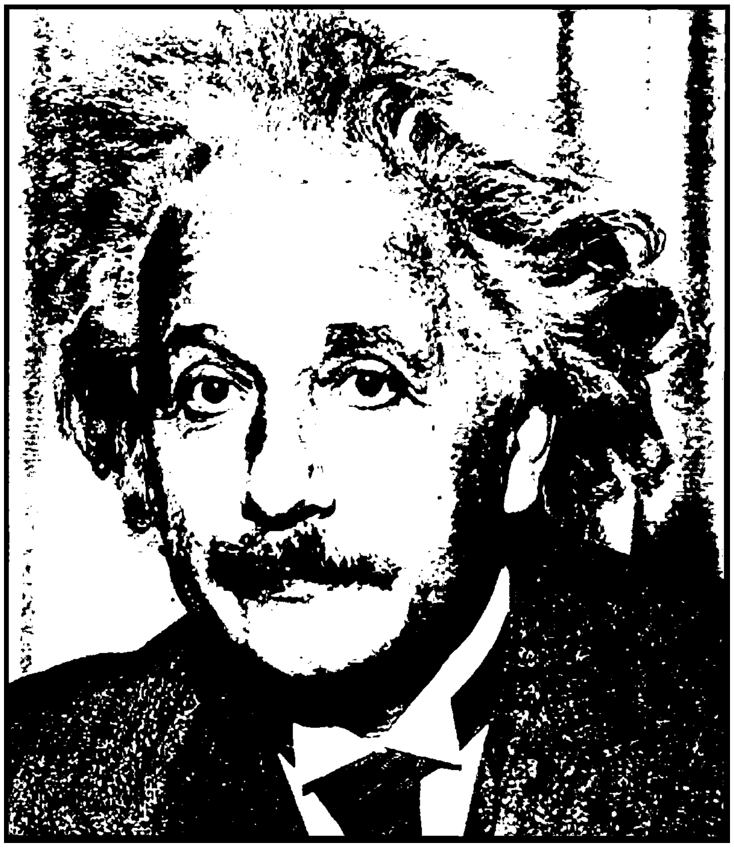


Figure 3.T2: (thopt1)

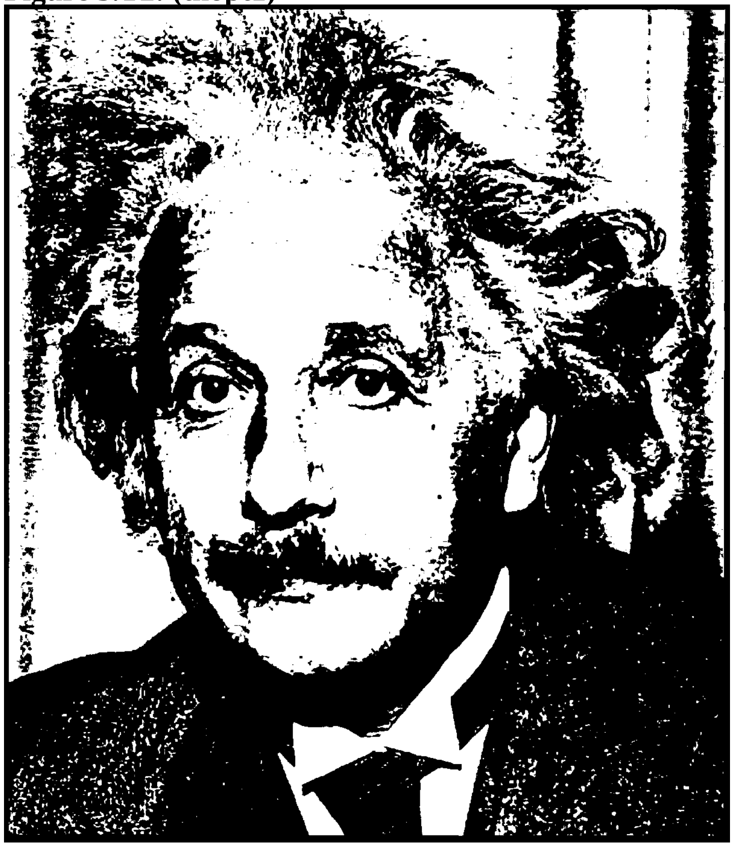


Figure 3.T3: (thopt2)

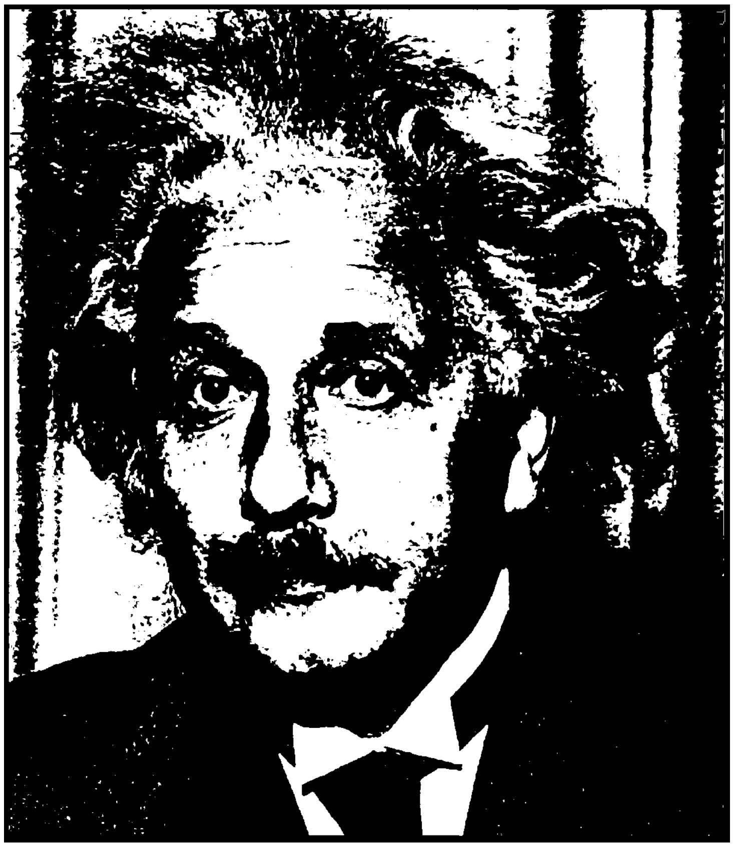


Figure 3.T4: (thopt3)

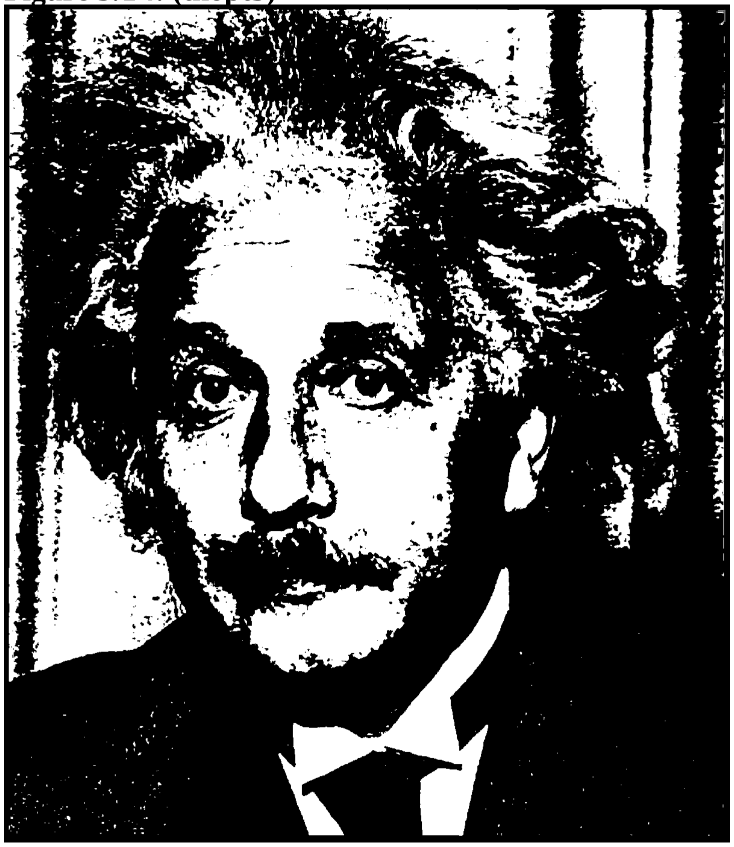


Figure 3.T5: (thps128)

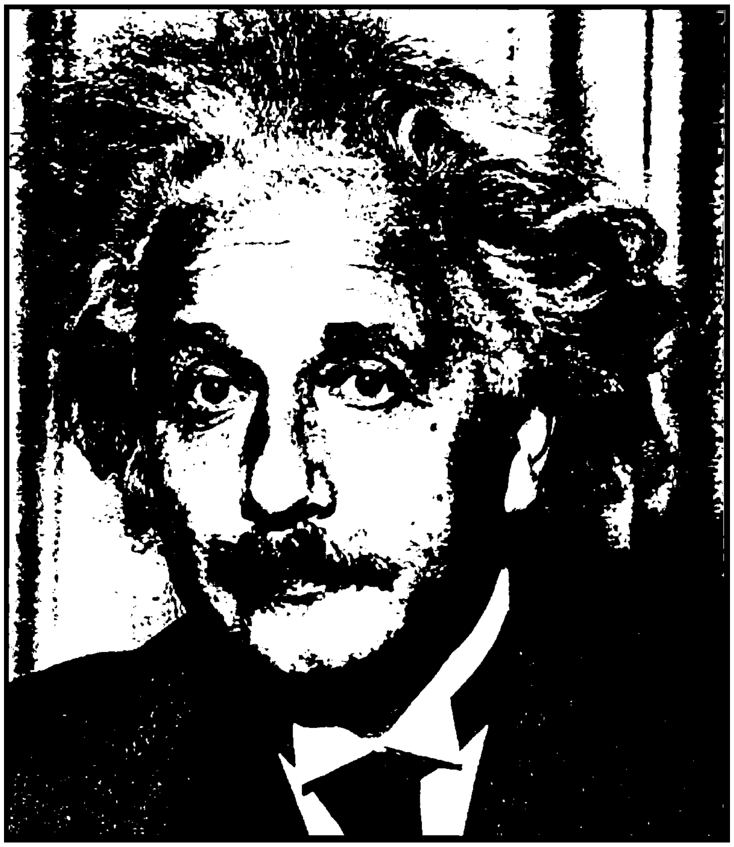


Figure 3.T6: (thdouble)

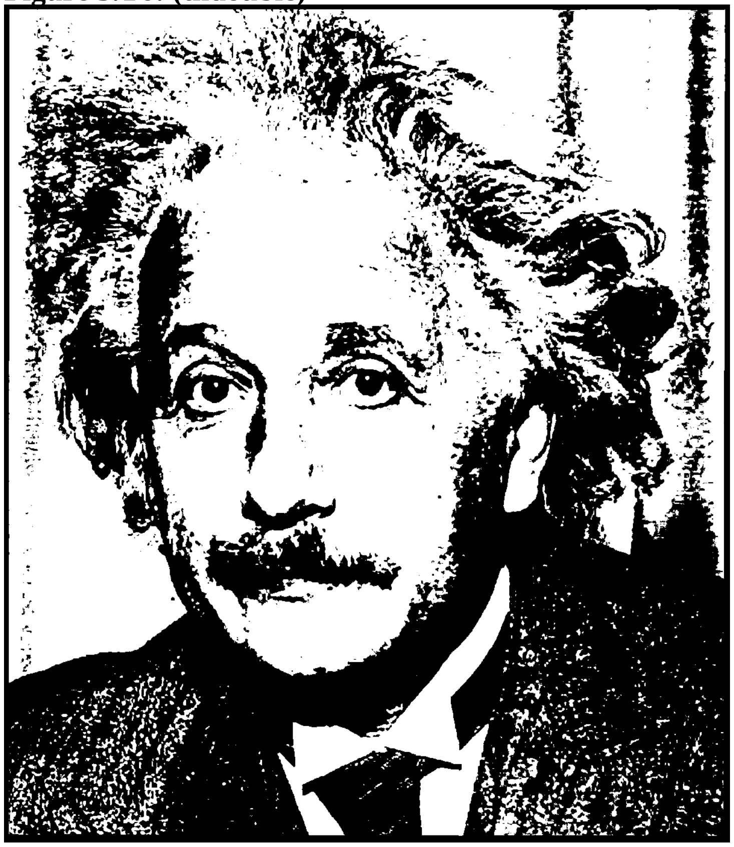


Figure 3.T7: (gthresh)

3.2.6 Discussions

The fig.3.T1 is the sample image. Filenames are given in parenthesis for the fig.3.T2-fig.3.T7.

Table 3.1

Threshold Methods and Options	**Output Figures (Results)**
Option 1 (Uint8 datatype)	Fig.3.T2
Option 2 (Uint8 datatype)	Fig.3.T3
Option 3 (Uint8 datatype)	Fig.3.T4
Photoshop 7 , threshold value 128	Fig.3.T5
Double data (Matlab 6.1) type based	Fig.3.T6
Matlab 6.1’s gthresh function	Fig.3.T7

This above table 3.1 shows results (Output figures) corresponding to different threshold methods and options.

3.2.7 Program 3.1

The following program function (in matlab 6.1) does thresholding of gray images using various options.

```
function thimg=Thresh(varargin)


%THRESH Thresholding of Gray Image
%---------------------------------------------------------
%---------------------------------------------------------
% thimg=Thresh(varargin)
% THRESH (function)
% This function does thresholding of gray image matrix
%
% INPUT ARGUMENTS:
% m=varargin{1}: gray image file or matrix or any matrix (m by n)
%
% opt=varargin{2}: options 1, 2 and 3
% 1 : Average of all the elements of the matrix
% 2 : Average of maximum and minimum values of the matrix
% 3 : Custom threshold value
% 4 : Otsu's method
%
% thmax=varargin{3}: maximum value after thresholding (thmax=<1)
% thmin=varargin{4}: minimum value after thresholding (0<=thmin)
% cthv=varargin{5}: custom threshold value (true for opt=3)
%
% OUTPUT ARGUMENT:
% thimg: thresholded image
%
% EXAMPLES:
% 1.
% thimg=Thresh('flgray.tif',1,1,0)
% 2.
% thimg=Thresh('flgray.tif',3,1,0,128)
% 3.
% m=imread('pout.tif')
% Thresh(m,1,1,0)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%----------------------------------------------------------
%----------------------------------------------------------
m=varargin{1};
opt=varargin{2};
thmax=varargin{3};
thmin=varargin{4};

if  ischar(m)
   m=imread(m);
   [r c dim]=size(m);
   if  dim==3
      m=rgb2gray(m)
   end
else
   m=m;
   [r c dim]=size(m);
end

thmax;     % thmax=1
thmin;     % thmin=0

% Option: 1
% 1. Average of all the elements of the matrix

if  opt==1
```

```
    av0=mean(m,1);
    av=mean(av0,2);
    thv=av; % threshold value
end

% Option: 2
% 2. Average of maximum and minimum values

if  opt==2
% Maximum element
    mx=max(max(m));
% Minimum element
    mn=min(min(m));

    mxmn=[mx mn];
    av=mean(mxmn)
    thv=av; % threshold value
end

% Option: 3
% 3. The custom threshold value for the operaton.
if  opt==3
    cthv=varargin{5};
    thv=cthv;
end

% Option: 4
% 4. Otsu's threshold selection method
if opt==4
   thv=graythresh(m);
   thv=thv*255;
end
```

```
% Threshold operation loop
for r1=1:r
   for c1=1:c
      if  m(r1,c1)<=thv
         thimg(r1,c1)=thmin;
      else
         thimg(r1,c1)=thmax;
      end
   end
end
thimg;
thimg=logical(thimg);
%---------------------------------------------------------
%---------------------------------------------------------
```

3.3 Non-Halftone Binary Image Transformations Methods

Different non-halftone binary image transformation methods are discussed in the following sections.

Various novel non-halftone binary image transformation methods are

1. Row-wise and column-wise processing.
2. Blockwise processing.
3. Processing based on pixel location and path based on location.
    - 3.1. Processing the image column-wise.
    - 3.2. Processing the image row-wise.
    - 3.3. Processing the image in arbitrary path.
    - 3.4. Processing the image diagonally.
        - 3.4.1 Starting point of the diagonal is Left Top or Right Bottom.
        - 3.4.2 Starting point of the diagonal is Right Top or Left Bottom.

Subdivision of the methods 3.4.1 and 3.4.2 are given in Section 3.3.3.4.2

3.3.1. Row-wise and Column-wise Processing

3.3.1.1 Threshold Selection Method

For the above-mentioned method threshold selection is done by the following method.

1. Take all the arithmetic means of all the elements of the rows of the image matrix.
2. Then take arithmetic mean of all the above means, to get the threshold value.

3.3.1.2 Algorithm for Method

1. Select the group of pixels row wise or column wise.
2. Then thresholding is done on the above group of selected pixels.

Image processing direction for row wise processing is shown in the figure 3.RC1.

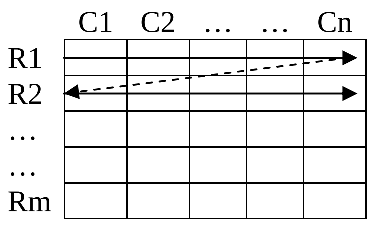


Figure 3.RC1
Row-wise

Image processing direction for column wise processing is shown in the figure 3.RC2.

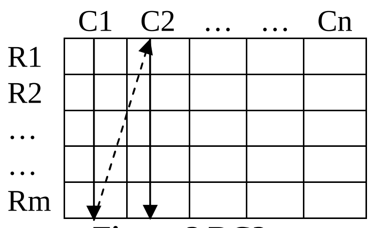


Figure 3.RC2
Column-wise

3.3.1.3 Input and Output Images

The row-wise and column-wise processing of fig.3.RC3 generates the following pictures (fig.3.RC4-fig.3.RC7) as output.

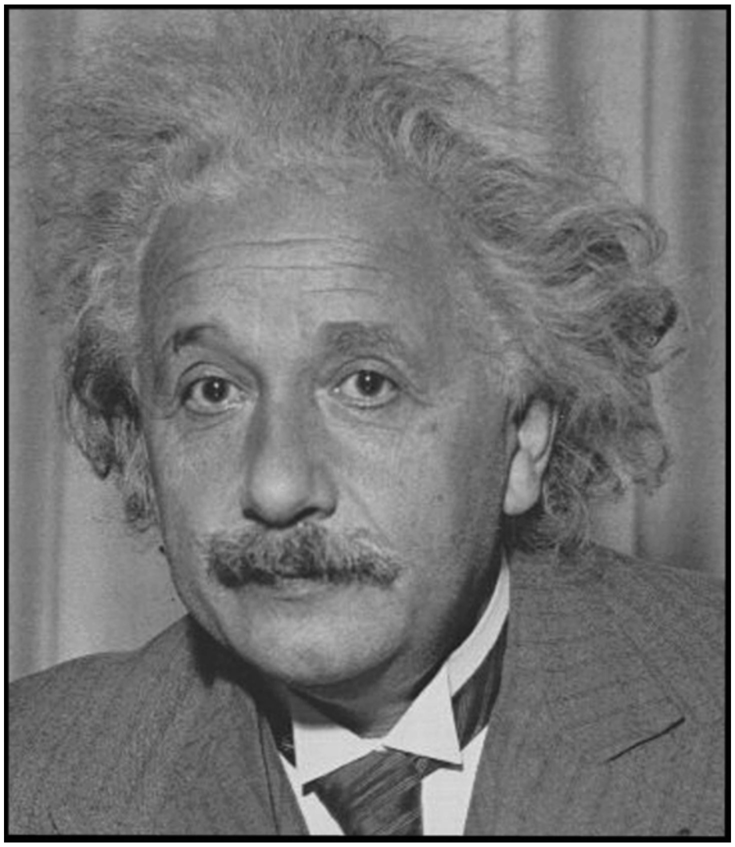


Figure 3.RC3: Sample Image (einstein300)

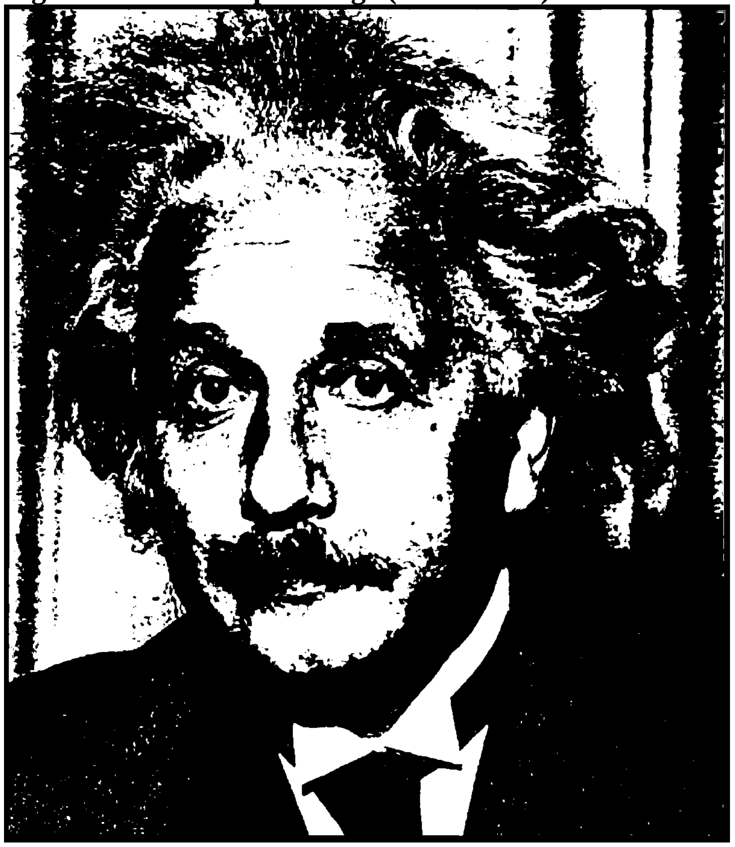


Figure 3.RC4: (einstein300_photoshop7thresh) Photoshop 7 Threshold value:128

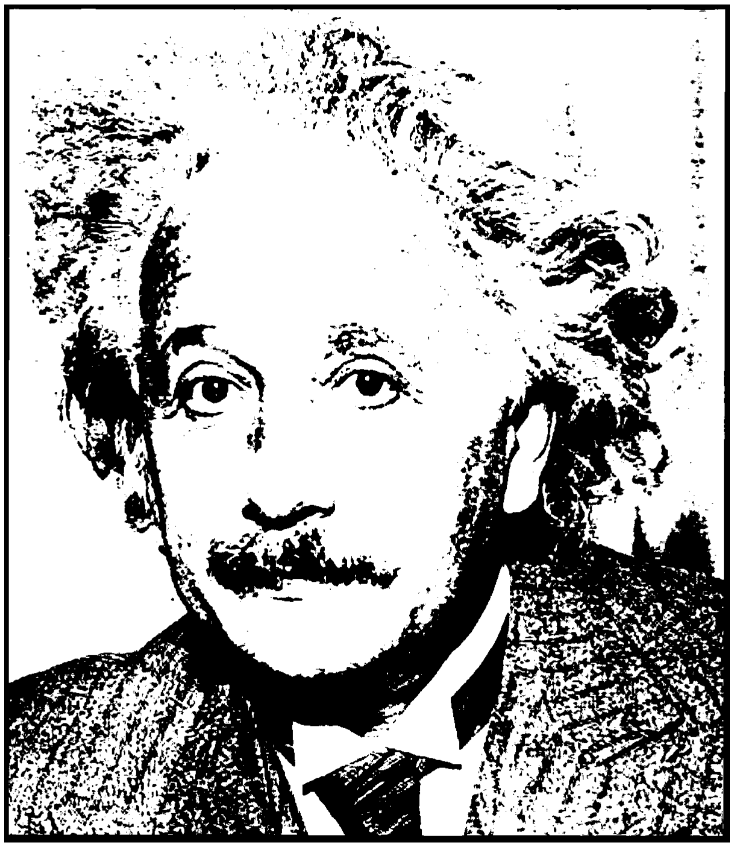


Figure 3.RC5 : (einstein300_th106) Photoshop7 Threshold value: 106

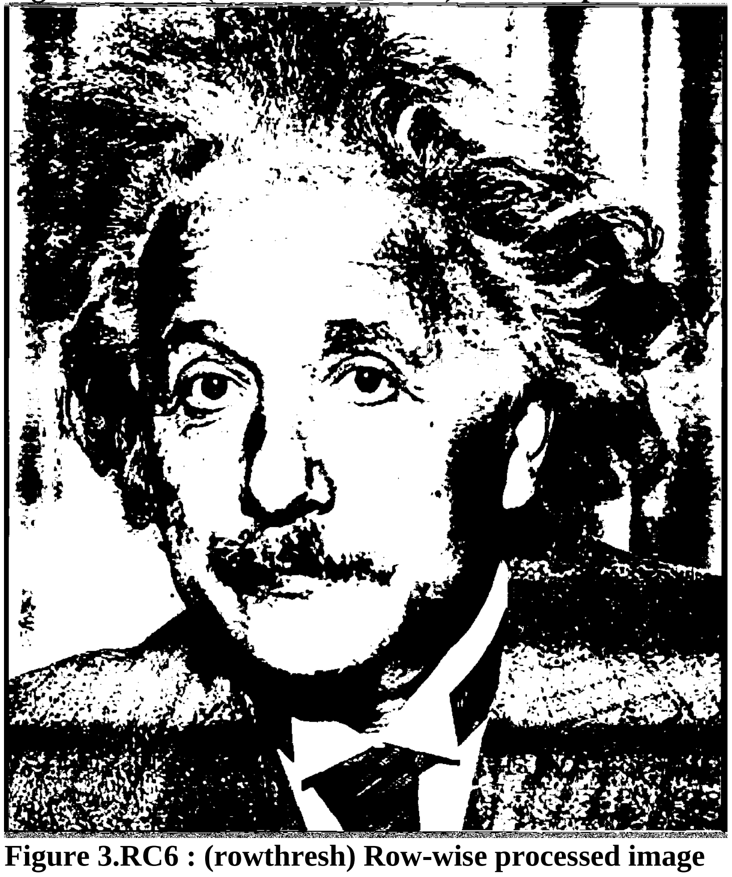


Figure 3.RC6 : (rowthresh) Row-wise processed image

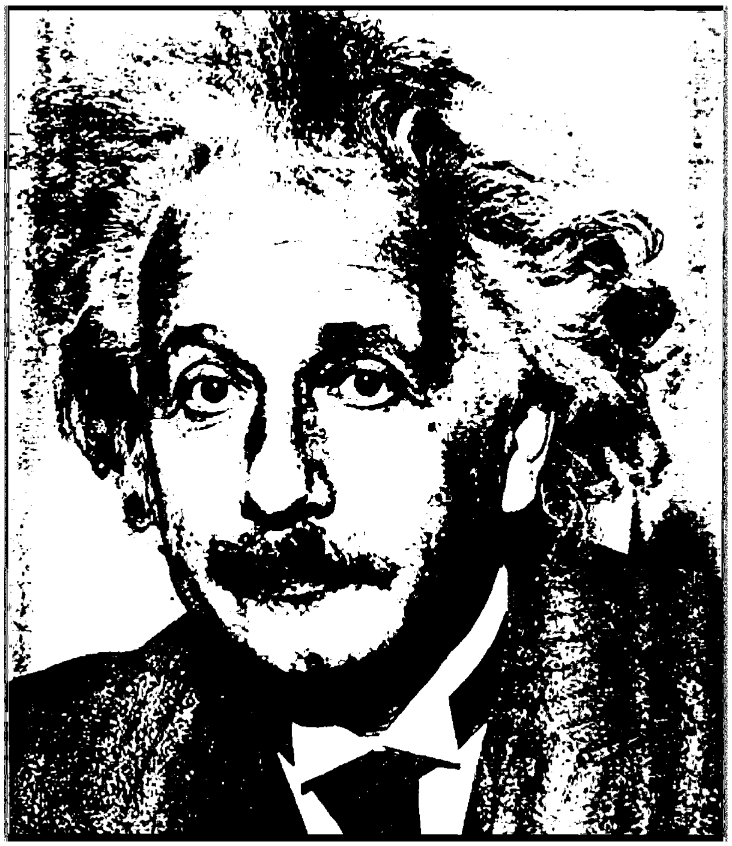


Figure 3.RC7: (colthresh) Column-wise processed image

3.3.1.4 Discussions

Fig.3.RC3 is the sample image. Filenames are given in parenthesis for the fig.3.RC3-fig.3.RC7. After close observation it can be said that the fig.3.RC6 and fig.3.RC7 carries the image details (especially coat portion of the image in Fig.3.RC3) that are absent in fig. 3.RC4, as fig.3.RC4 is a simple threshold image done through Photoshop 7 with a threshold level 128. The fig.3.RC5 which is Photoshop 7 threshold with value 106 contains the image details of the coat portion but not the face of the image. The face of the person in fig.3.RC7 is more prominent and where as background of the image in fig.3.RC6 is little bit prominent than in the fig.3.RC7.

3.3.1.5 Program 3.2

The following program function (in matlab 6.1) does row wise and column wise thresholding of gray images and generate non-halftone binary images.

```
function RCThresh(im)
%----------------------------------------------------------
%----------------------------------------------------------
% RCThresh(im)
% RCTHRESH (function)
% This function thresholds an image or matrix by selecting
% each row and column
%
% INPUT ARGUMENT:
% im: gray or rgb image file or matrix;
%
% EXAMPLE:
% RCThresh('pout.tif')
% and
% im=[..]
% RCThresh(im)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%----------------------------------------------------------
%----------------------------------------------------------
if  ischar(im)
   im=imread(im);
   [r c dim]=size(im)
   if  dim==3
      im=rgb2gray(im);
   end
else
   im=im;
end
```

```
pix=numel(im);

% Selecting each row and thresholding
htr=[];
for i=1:r
   im2=im(i,:);
   ht=Thresh(im2,1,1,0);
   htr=[htr; ht];
end

htr=logical(htr);

% Selecting each column and thresholding
htc=[];
for i=1:c
   im2=im(:,i);
   ht=Thresh(im2,1,1,0);
htc=[htc ht];
end

htc=logical(htc);
% OUTPUTS
imshow(htr)
title('Row threshold')
imwrite(htr,'rowthresh.tif')

figure,imshow(htc)
title('Column threshold')
imwrite(htc,'colthresh.tif')
%-----------------------------------------------------------
%-----------------------------------------------------------
```

3.3.2 Blockwise Processing

3.3.2.1 Direction of Processing

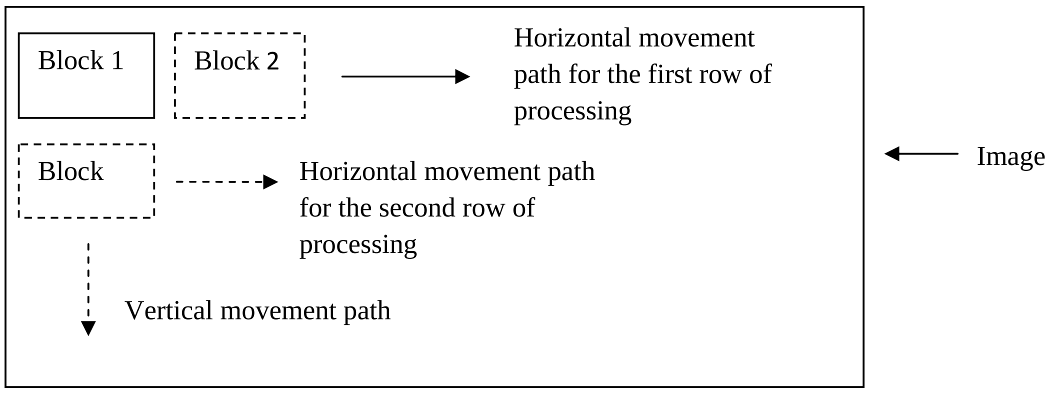

Block 1
Block 2
Horizontal movement path for the first row of processing
Image
Block
Horizontal movement path for the second row of processing
Vertical movement path

Figure 3.B1: Schematic Drawing for Blockwise Processing

In the above schematic drawing of an image (fig.3.B1), the initially the blocks used for processing moves horizontally for the first row of processing and then goes to the second row of processing till it sweeps the total image. As each of the block move the threshold is applied block wise to the image to convert it to binary image.

Table 3.2

Function: blkthresh.m

Block size (pixels x pixels)	**Resolution of original (dpi)**	**Original image name**	**Transformed image name**	**Resolution (dpi)**
2 x2	600	einstein600.tif	ebt1	300
4 x4	do	do	ebt2	do
8 x8	do	do	ebt3	do
16 x16	do	do	ebt4	do
32x32	do	do	ebt5	do
64x64	do	do	ebt6	do
128x128	do	do	ebt7	do
256x256	do	do	ebt8	do
512x512	do	do	ebt9	do

The above table 3.2 shows the block sizes used to process the original image Fig.3.B2 (einstein600.tif) with resolution 600 dpi to get the processed images like ebt1, ebt2 etc as given the above table with resolution 300dpi.

3.3.2.2 Input and Output Images

The block-wise processing of fig.3.B2 generates the following pictures (fig.3.B3-fig.3.B12) as output

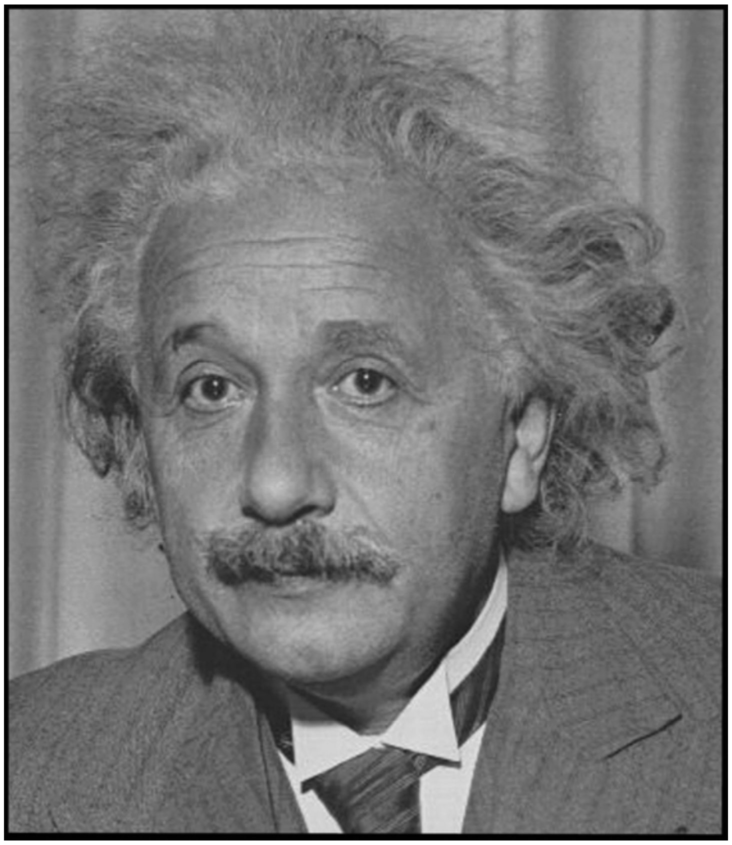


Figure 3.B2: (Einstein600.tif) Original Image

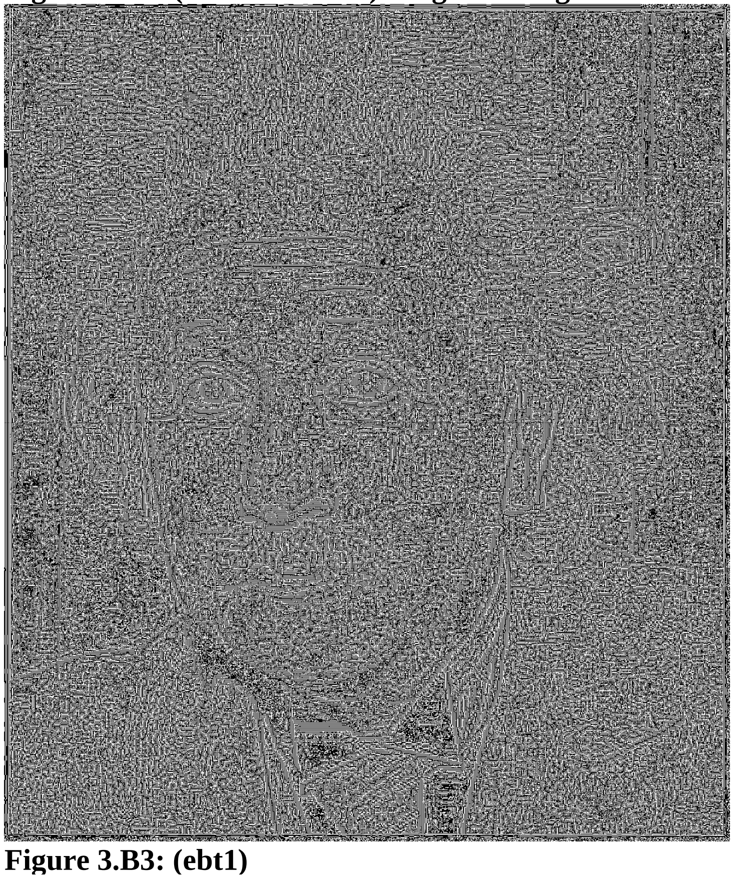


Figure 3.B3: (ebt1)

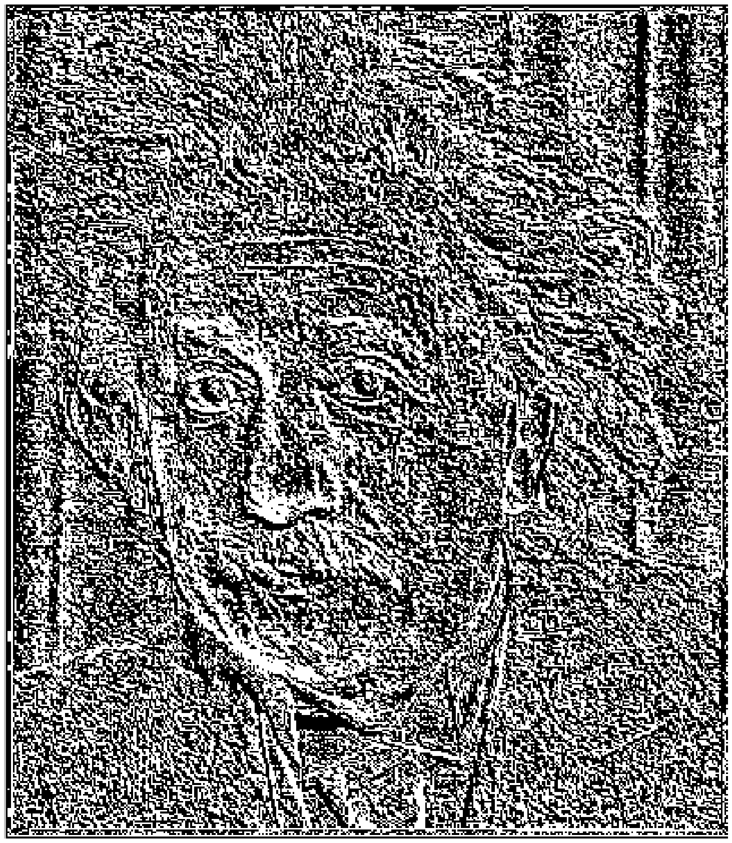


Figure 3.B4: (ebt1_zoom33p3_bitmap)

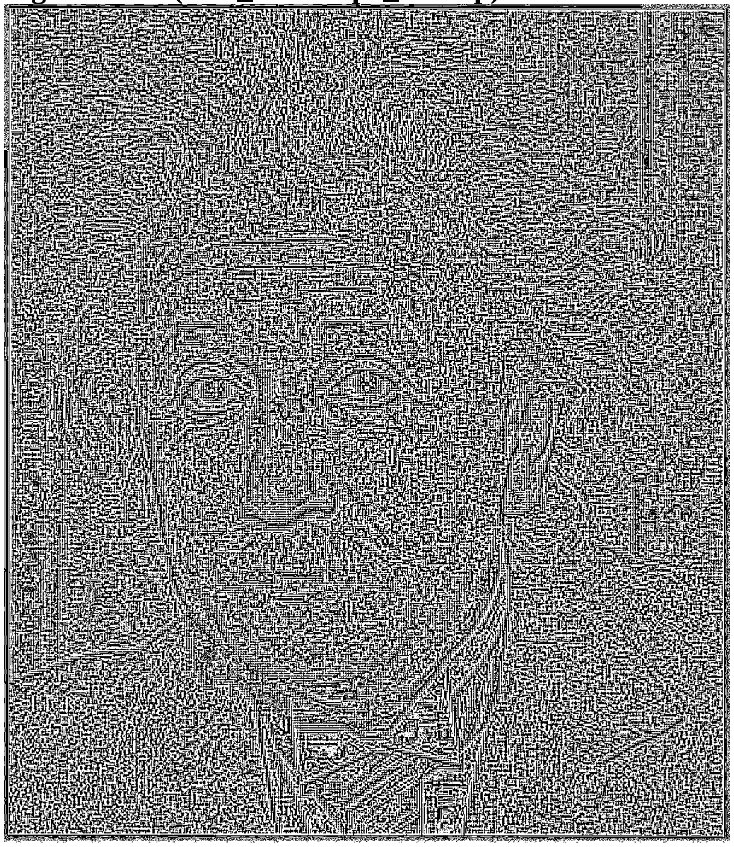


Figure 3.B5: (ebt2)

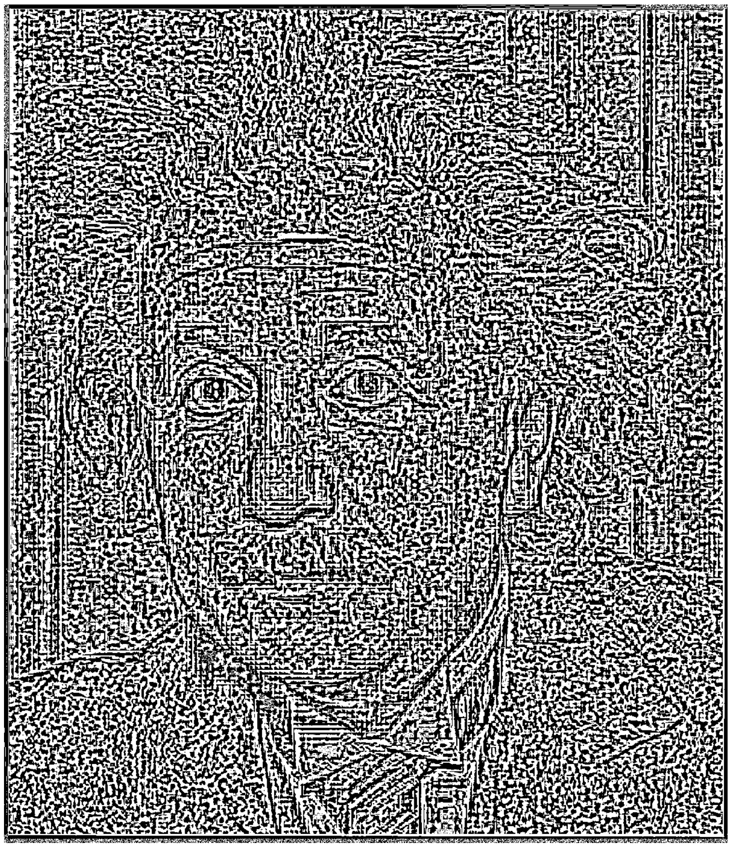


Figure 3.B6: (ebt3)

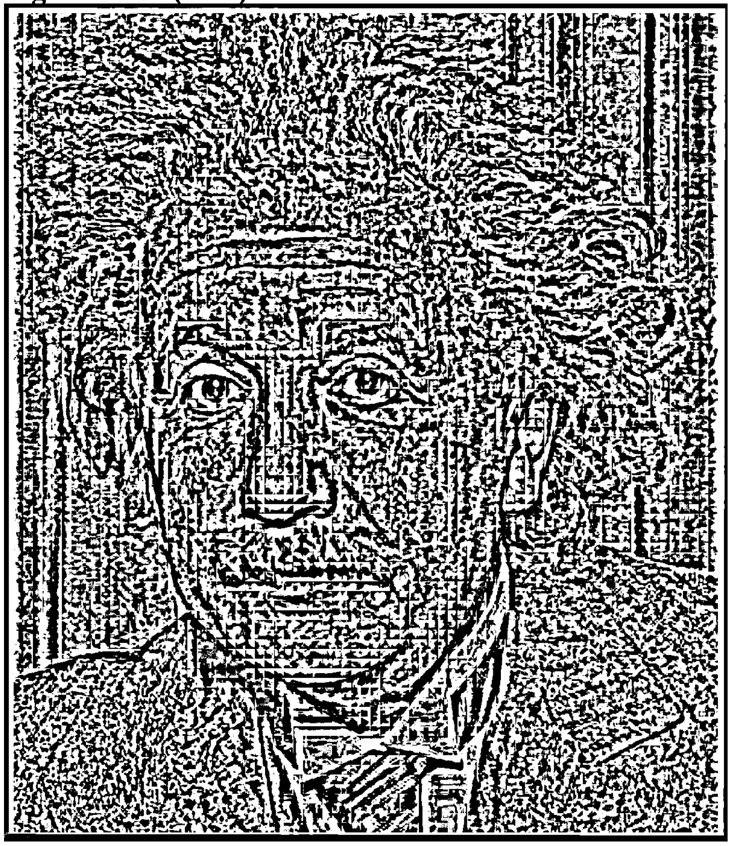


Figure 3.B7: (ebt4)

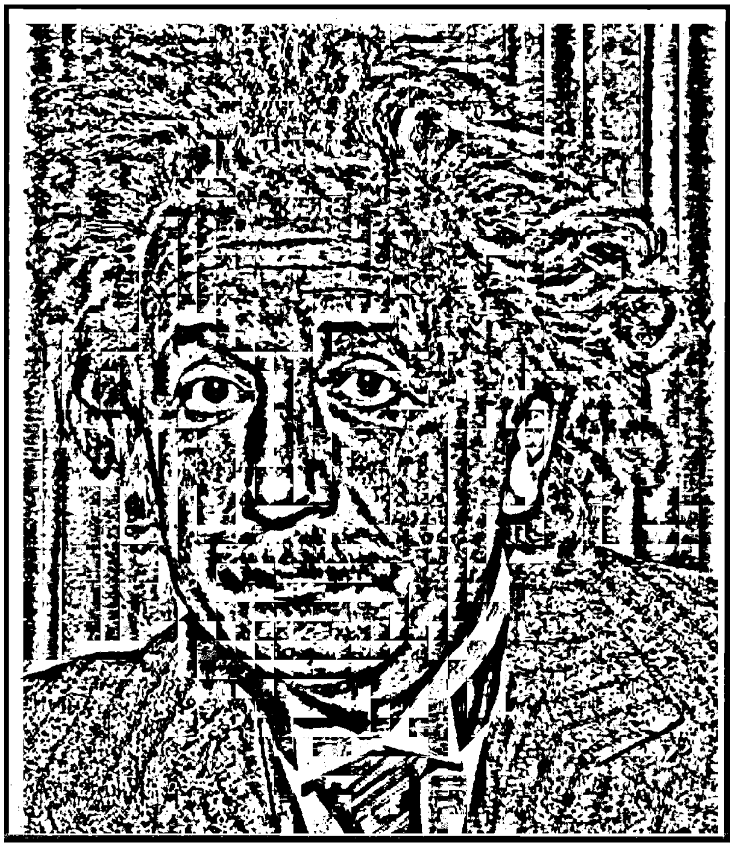


Figure 3.B8: (ebt5)

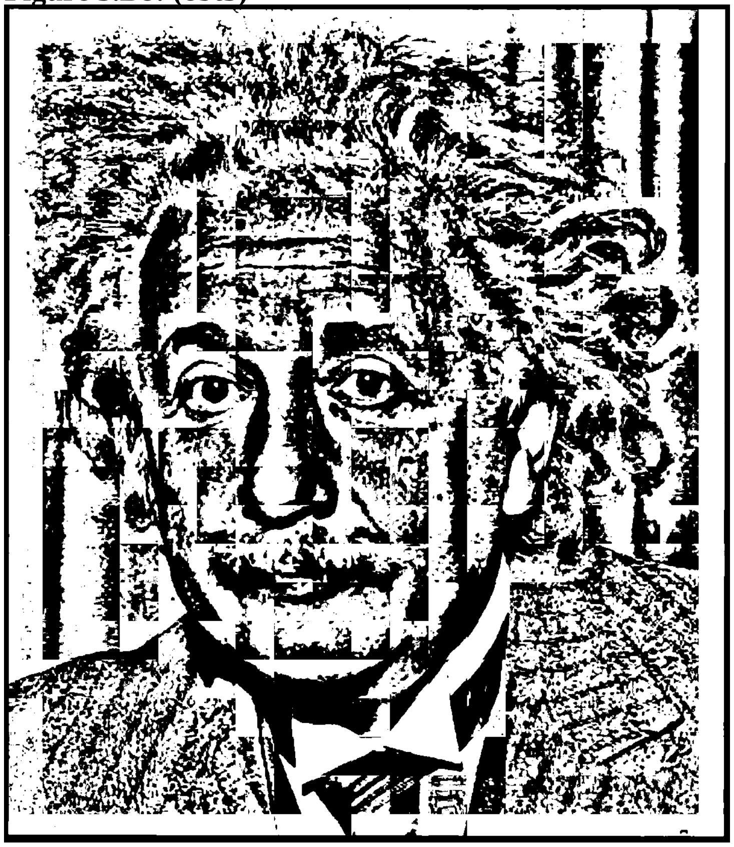


Figure 3.B9: (ebt6)

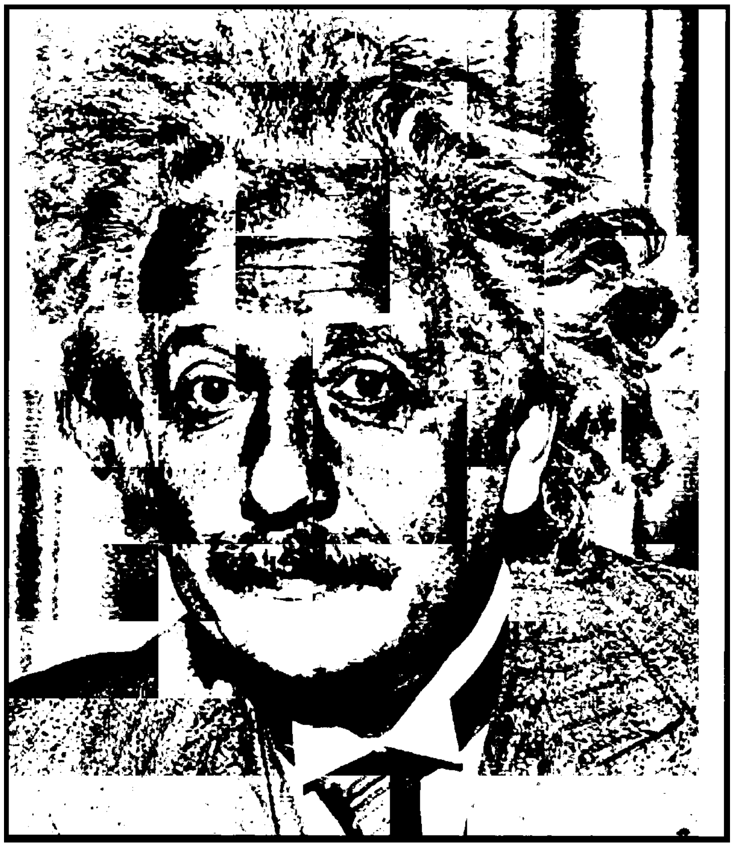


Figure 3.B10: (ebt7)

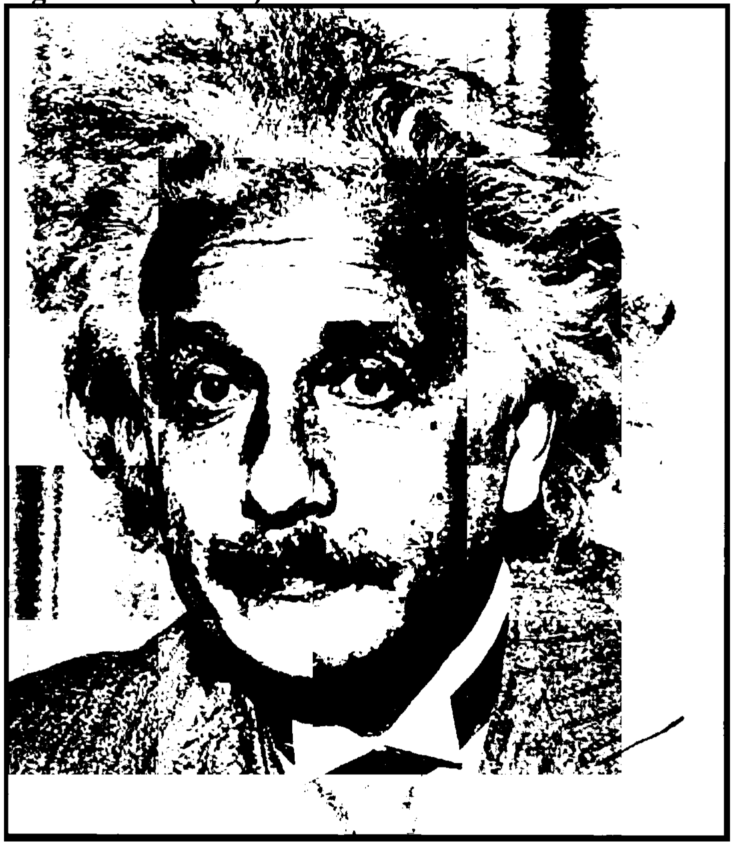


Figure 3.B11: (ebt8)

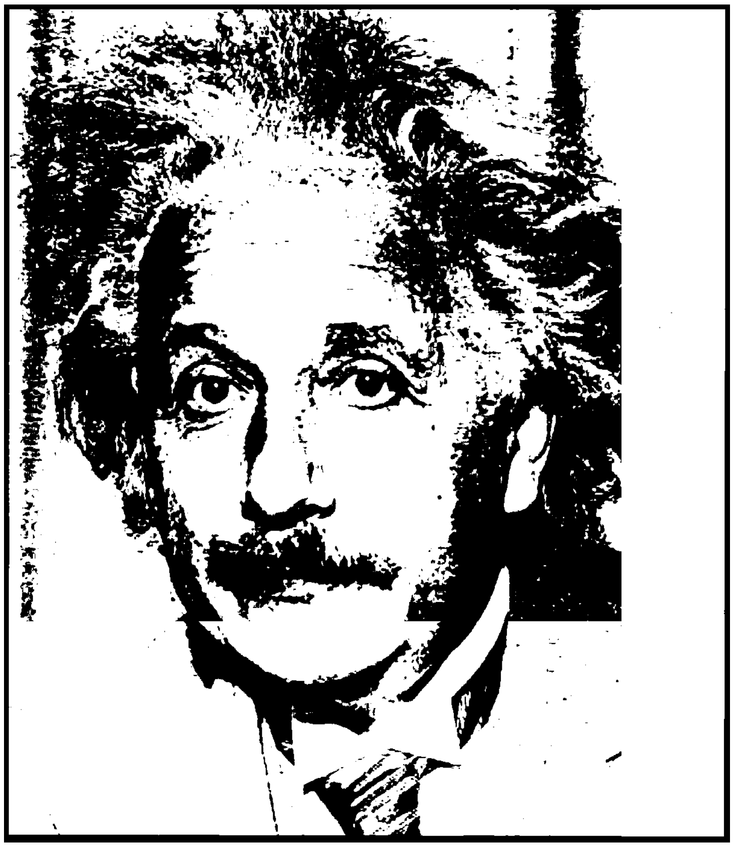


Figure 3.B12: (ebt9)

3.3.2.3 Discussions

Fig.3.B2 (Einstein600.tif) is the original sample image. Filenames are given in parenthesis for the fig.3.B2-fig.3.B12. As we increase the size of the block and process the image, the image becomes prominent but block like structures appear. In the extreme cases when block size is 256 x 256 or more the image details are also lost. Image fig.3.B4 (ebt1_zoom33p3_bitmap) is the image fig.3.B3 (ebt1) zoomed at 33.3% through Photoshop 7. It contains more details which are absent in the image fig.3.B3 (ebt1).

3.3.2.4 Program 3.3

The following program function (in matlab 6.1) does blockwise processing of gray images to generate a binary image.

```
function BlkThresh(im,bs)


%BLKTHRESH : Block Thresholding
%--------------------------------------------------------------
%--------------------------------------------------------------
% BlkThresh(im,bs)
% BLKTHRESH (function)
%
% This function does dynamic (adaptive) thresholding through
% block wise processing of a gray image to generate a binary
% image.
%
% INPUT ARGUMENT:
% im: gray image file or matrix of m by n
% bs: one side of the square blocksize, in pixels
%
% EXAMPLE:
% BlkThresh('pout.tif',4)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%--------------------------------------------------------------
%--------------------------------------------------------------
if  ischar(im)
   im=imread(im);
else
   im;
end

[r c]=size(im);
```

```
tr=bs;
tc=tr; % tr and tc: row and column size of threshold matrix
trc=tr*tc;

% Converts mxn blocks into column with padding
imc=im2col(im,[tr tc],'distinct');
[r2 c2]=size(imc);

fun = @Thresh; % invoking the function Thresh.m
% uses: thresh(m,op,thmax,thmin)

imb=blkproc(imc,[trc 1],fun,1,1,0);

imu=unstkc(imb,r2,c2);
imi=col2im(imu,[tr tc],[r c],'distinct');
imbin=logical(imi);

% Outputs

out={'imshow' 'imwrite' imi imbin 'bit1.tif' 'bit2.tif' ...
    'resolution' 72 'figure'};

i=1;
% For gray output of grayimage file
  feval(out{i},out{i+2})
  feval(out{i+1},out{i+2},out{i+4},out{i+6},out{i+7})

% For bitmap output of grayimage file
  feval(out{i+8})
  feval(out{i},out{i+3})
  feval(out{i+1},out{i+3},out{i+5},out{i+6},out{i+7})
%-----------------------------------------------------------
```

3.3.3 Processing Based on Pixel Location and Path Based on Location

In this section image processing is mainly based on pixel location or path based on pixel locations [27]. Pixel location is defined in the Section 3.3.3.1.1. In different directions (e.g. row-wise, diagonally etc) processing path is taken based on pixel locations.

3.3.3.1. Processing the Image Column-wise

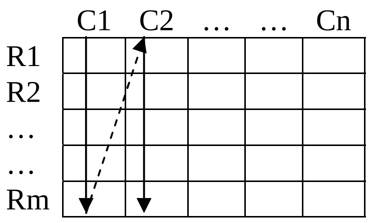


Figure 3.C1

3.3.3.1.1 Image Processing Path

Figure 3.C1 is a gray image matrix (m by n). Total number of pixels=m*n. Serial numbers of the pixel locations are from 1 to m*n, that run column wise from top to bottom, i.e. 1 (R1C1), 2 (R2C1) etc. The program processes the image by the pixels locations as described. Advantage of this method is that single number is used to identify the pixel location. One term is proposed, that is LISN (Location Identification by Single Number). Fig. 3.C1 shows image processing direction with arrows.

Table 3.3			
Function: locthresh.m			
Nop (No of Pixels)	**Original Resolution (dpi)**	**Original Filename**	**Transformed Filename**
Thresh opt=1			
10	600	einstein600.tif	ltein-1
50	do	do	ltein0
100	do	do	ltein1
200	do	do	ltein2
400	do	do	ltein3
600	do	do	ltein4
750	do	do	ltein5
1200	do	do	ltein6
1500	do	do	ltein7
1800	do	do	ltein8
2400	do	do	ltein9
10	72	gs3.tif	locthresh_nop10

50	do	do	locthresh_nop50
100	do	do	locthresh_nop100

This above table 3.3 shows the transformed images corresponding to the number of pixels used to process original images einstein600.tif and gs3.tif.

3.3.3.1.2 Output Images

The column-wise processing of fig.3.B2 generates the following pictures (fig.3.C2-fig.3.C12) as output similarly processing of fig.3.A1.1 generates fig.3.C2.1, fig.3.C3.1 and fig.3.C4.1

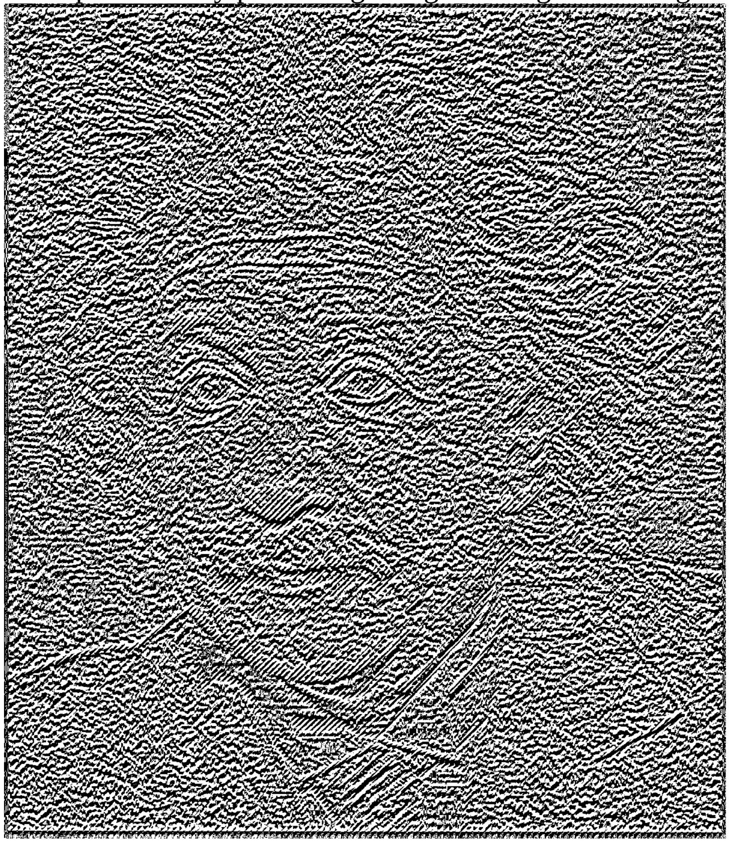


Figure 3.C2: (ltein-1)

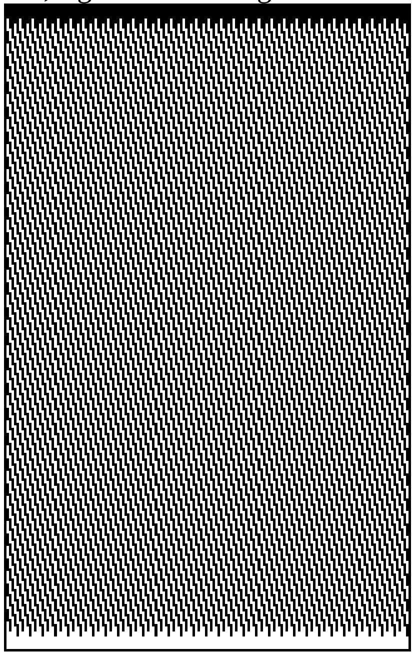


Figure 3.C2.1 (locthresh_nop10)

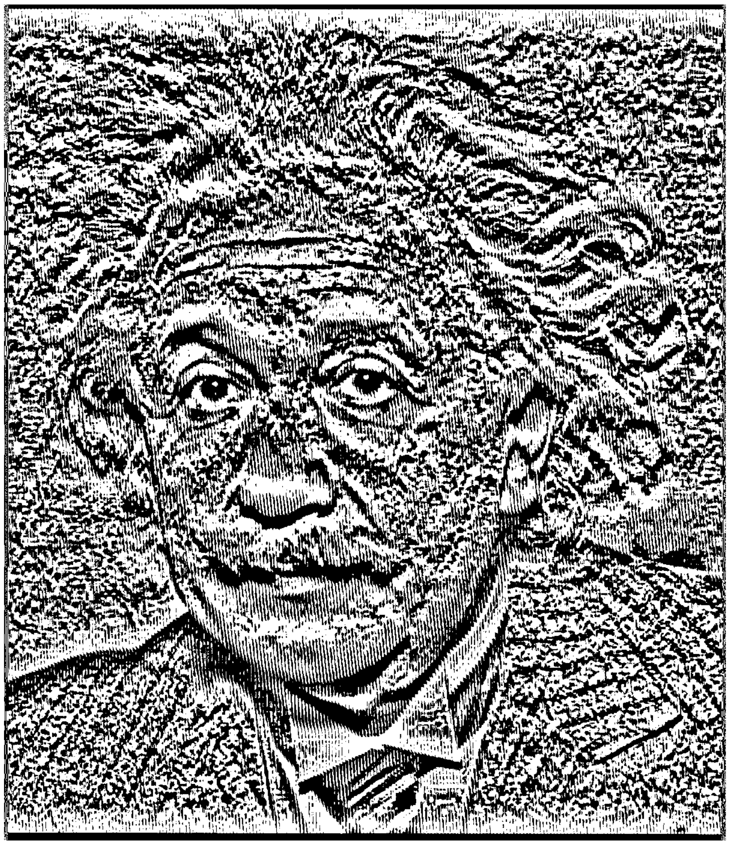


Figure 3.C3: (ltein0)

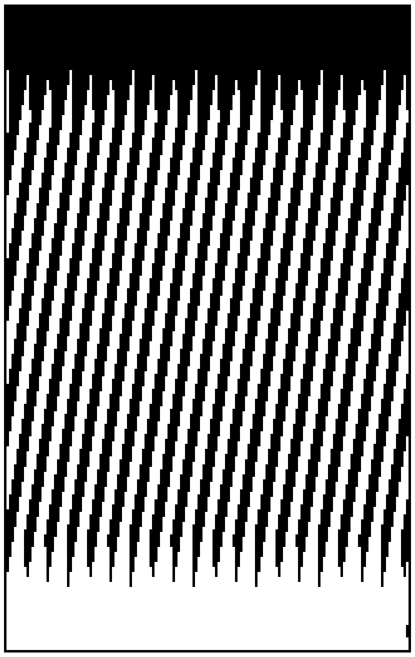


Figure 3.C3.1: (locthresh_nop50)

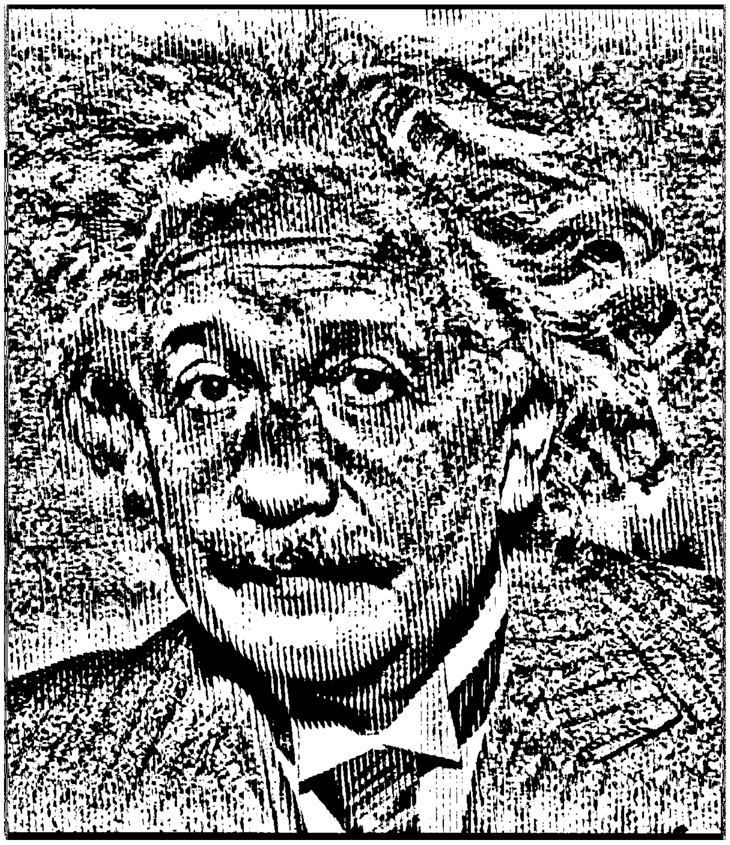


Figure 3.C4: (ltein1)

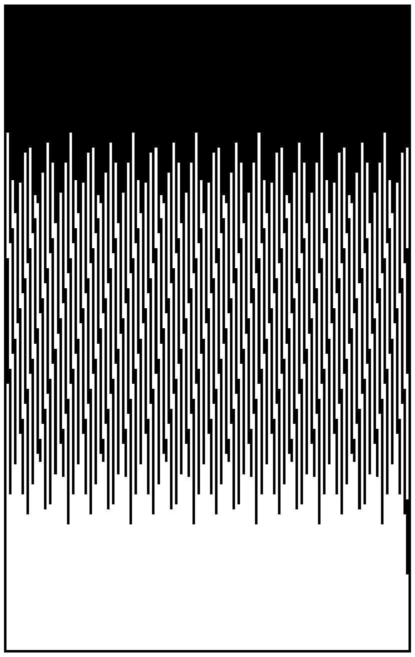


Figure 3.C4.1: (lockthresh_nop100)

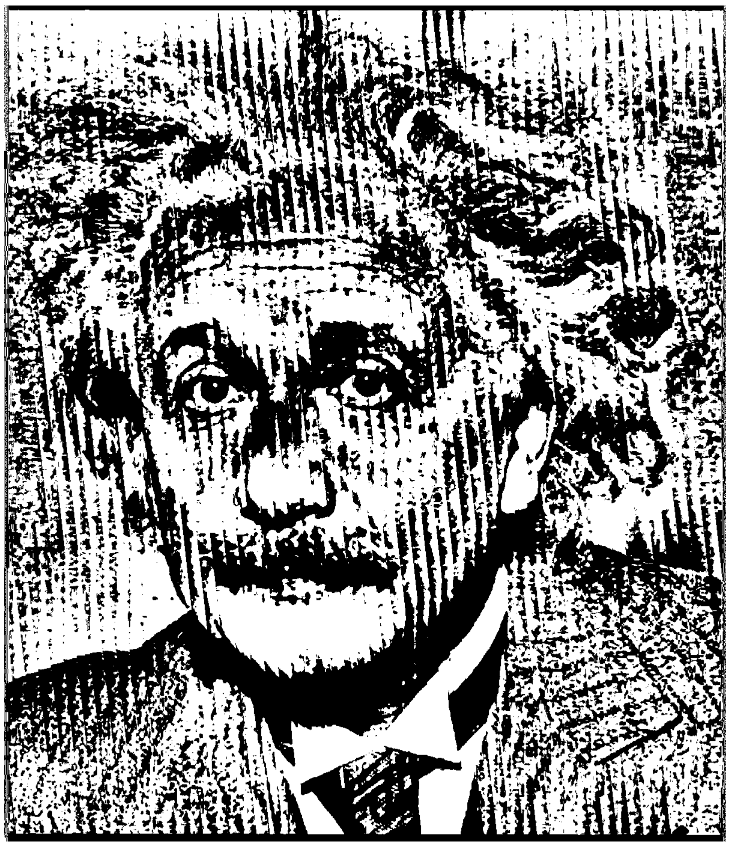


Figure 3.C5: (ltein2)

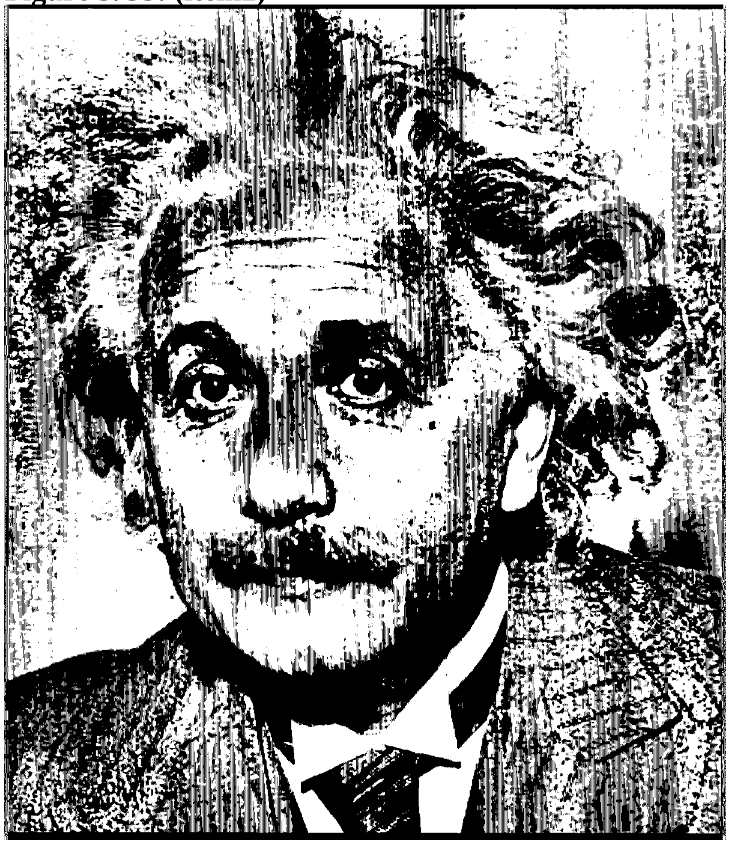


Figure 3.C6: (ltein3)

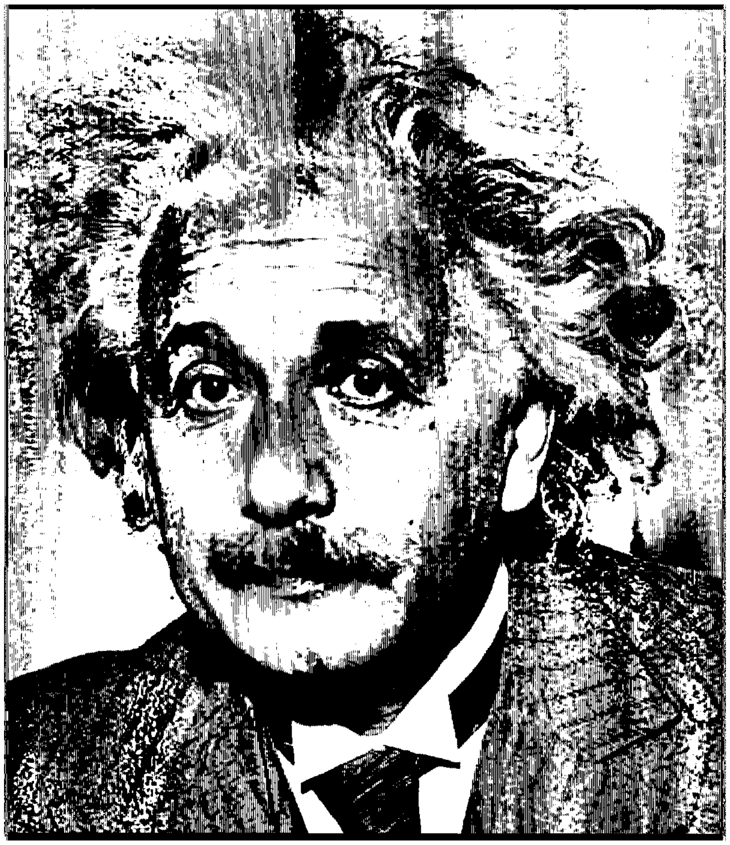


Figure 3.C7: (ltein4)

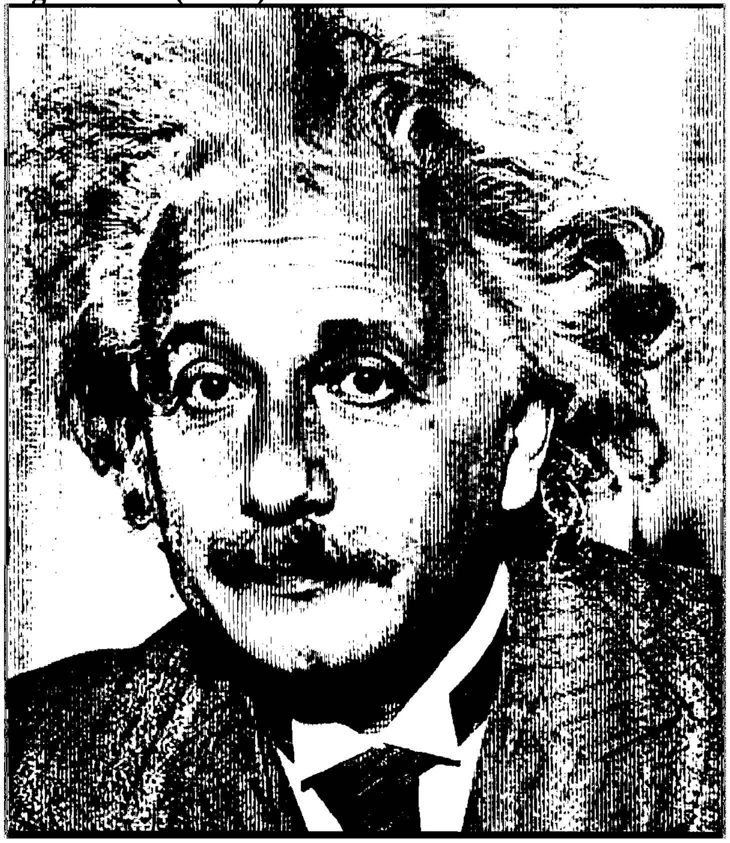


Figure 3.C8: (ltein5)

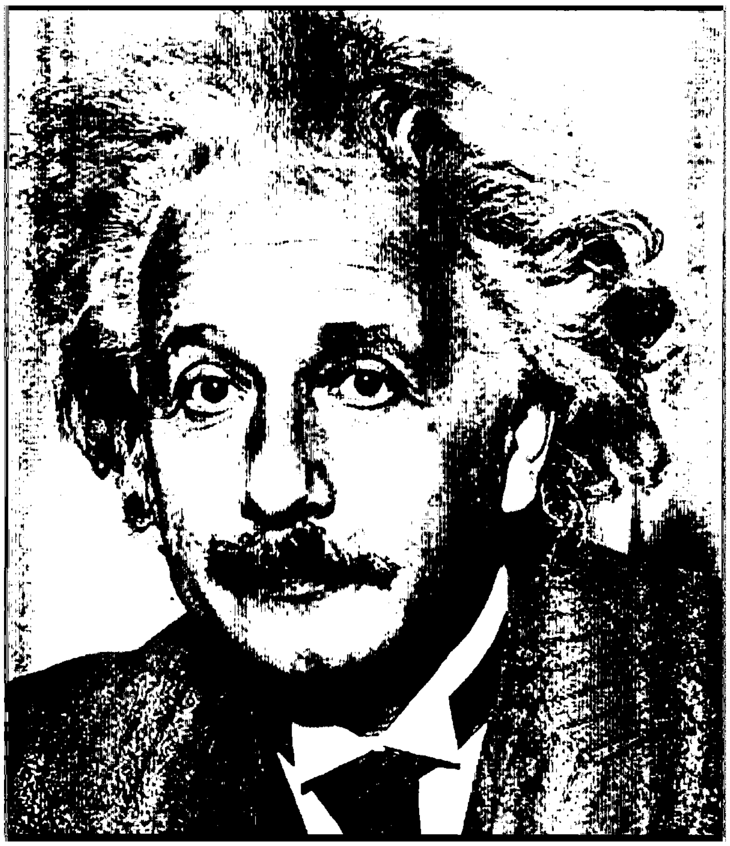


Figure 3.C9: (ltein6)

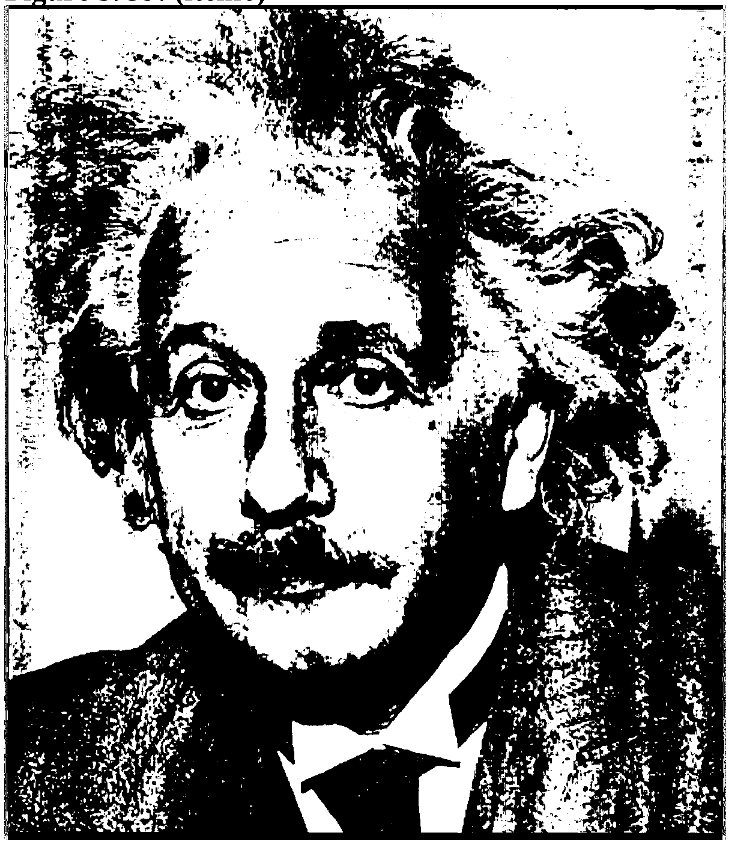


Figure 3.C10: (ltein7)

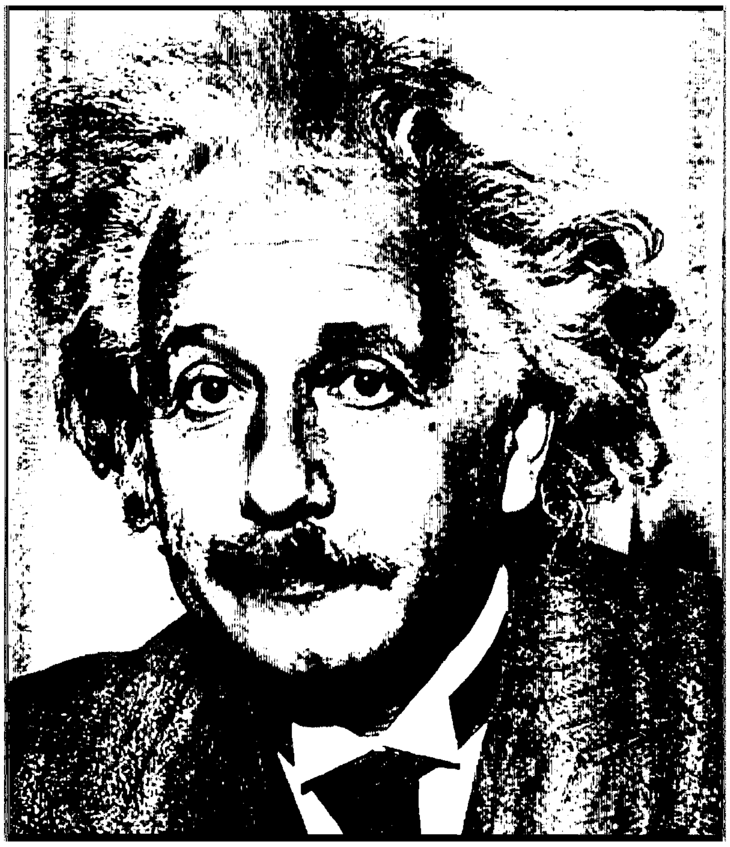


Figure 3.C11: (ltein8)

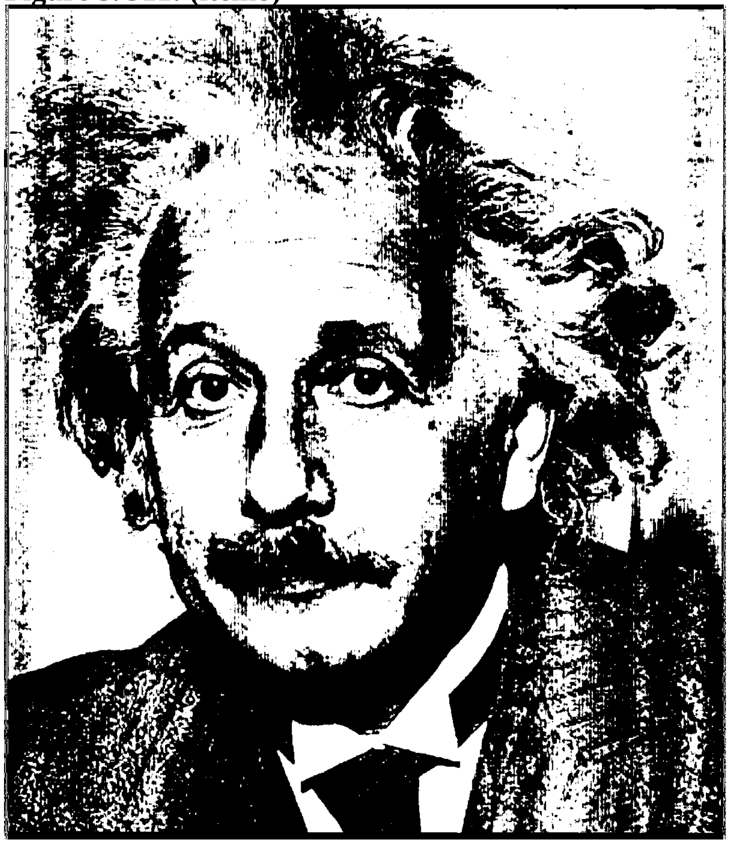


Figure 3.C12: (ltein9)

3.3.3.1.3 Discussions

Filenames are given in parenthesis for the fig.3.C2-fig.3.C12, fig.3.C2.1, fig.3.C3.1 and fig.3.C4.1.

Image details of the processed images are lost slowly as we increase the number of pixels processed at a time, which is evident from fig.3.C3 (ltein0) to fig.3.C12 (ltein9). The image ltein-1 looks like mildly engraved image. Visual artifacts are more prominent in the images fig.3.C4 (ltein1) and fig.3.C5 (ltein2) in comparison to other processed images. Fig.3.C2.1 (Locthresh_nop10), fig.3.C3.1 (locthresh_nop50), fig.3.C4.1 (locthresh_nop100) are grayscale image ramp (Fig.3.A1.1) non-halftoned using this lockthresh.m function (Section 3.3.3.1.4, Program 3.4)

3.3.3.1.4 Program 3.4

The following program function (in Matlab 6.1) does pixel location-wise processing of gray images to generate a binary image.

```
function LocThresh(filename, nop)
%-----------------------------------------------------------
%-----------------------------------------------------------
% LocThresh(filename,nop)
%
% LOCTHRESH (function) : LOCation THRESHold
%
% This function generates a thresholded image from
% gray or rgb image by selecting group of pixels
% locationwise.
%
% INPUT ARGUMENTS:
% filename: gray or rgb image
% nop :number of pixels thresholded at a time,
% (if nop is column size it is equivalent to rcthresh-column)
% except the last group, which may be less.
% EXAMPLES:
% 1.
% LocThresh('pout.tif',4)
% 2.
% LocThresh('flowers.tif',100)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%-----------------------------------------------------------
%-----------------------------------------------------------
im=imread(filename);
[r c dim]=size(im);

if dim==3
   im=rgb2gray(im);
end
```

```
pix=numel(im);


nop;
rm=rem(pix,nop);


ht2=[];
im3=[];
imend=[];
htend=[];
for i=0:nop:pix


  if  nop+i<=pix
    im2=im(1+i:nop+i);
    im3=[im3 im2];
    ht=Thresh(im2,1,1,0);
    ht2=[ht2 ht];
  end


  if rm~=0
    imend=im(end-rm+1:end);
    htend=Thresh(imend,1,1,0);
  end
    ht3=[ht2 htend];
    im4=[im3 imend];


end


im4;
ht3;
ht4=reshape(ht3,r,c);
imshow(ht4)
imwrite(ht4,'locthresh.tif','resolution',300)
%-----------------------------------------------------------
```

3.3.3.2 Processing the Image Row-wise

3.3.3.2.1 Image Processing Path

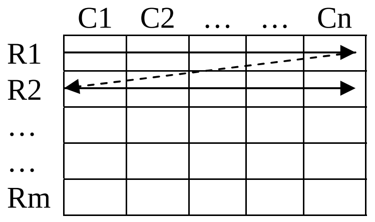


Figure 3.R1

The above fig.3.R1 shows the direction of processing row-wise while selecting locations column-wise using arrows.

Table 3.4

Input Image	nop (no of pixels)	Output Image
einstein600.tif	100	LocThreshRow_ein1
do	200	LocThreshRow_ein2
do	400	LocThreshRow_ein3
do	600	LocThreshRow_ein4
do	750	LocThreshRow_ein5
do	1200	LocThreshRow_ein6
do	1500	LocThreshRow_ein7
do	1800	LocThreshRow_ein8
do	2400	LocThreshRow_ein9
do	5000	LocThreshRow_ein10

This above table 3.4 shows the processed images corresponding to the number of pixels (nop) used to process the input image einstein600.tif

3.3.3.2.2 Output Images

The row-wise processing of fig.3.B2 generates the following pictures (fig.3.R2-fig.3.R11) as output similarly processing of fig.3.A1.1 generates fig.3.R2.1 and fig.3.R6.1.

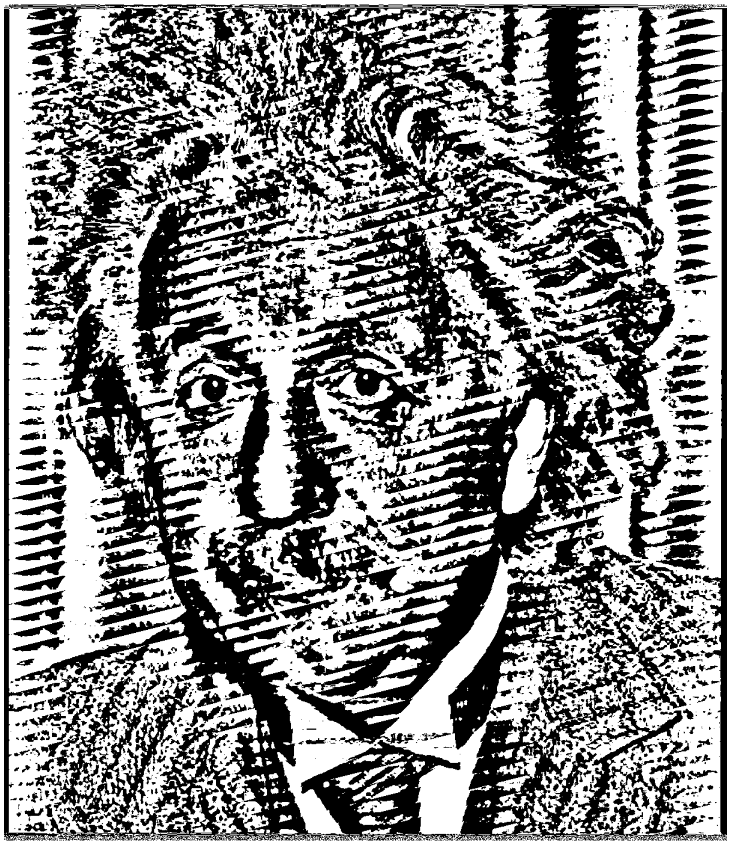


Figure 3.R2: (LocThreshRow_ein1)

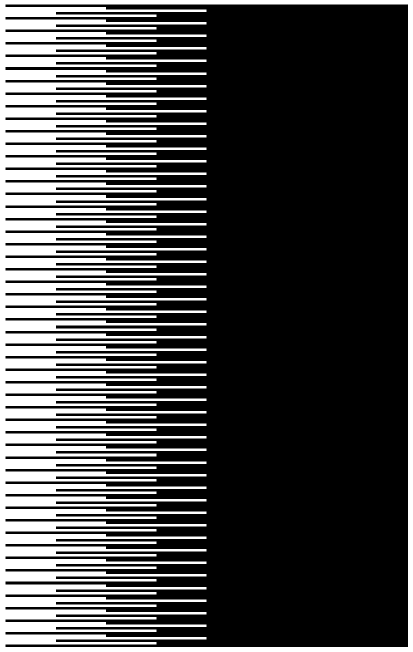


Figure 3.R2.1:(Locthreshrow_nop100)

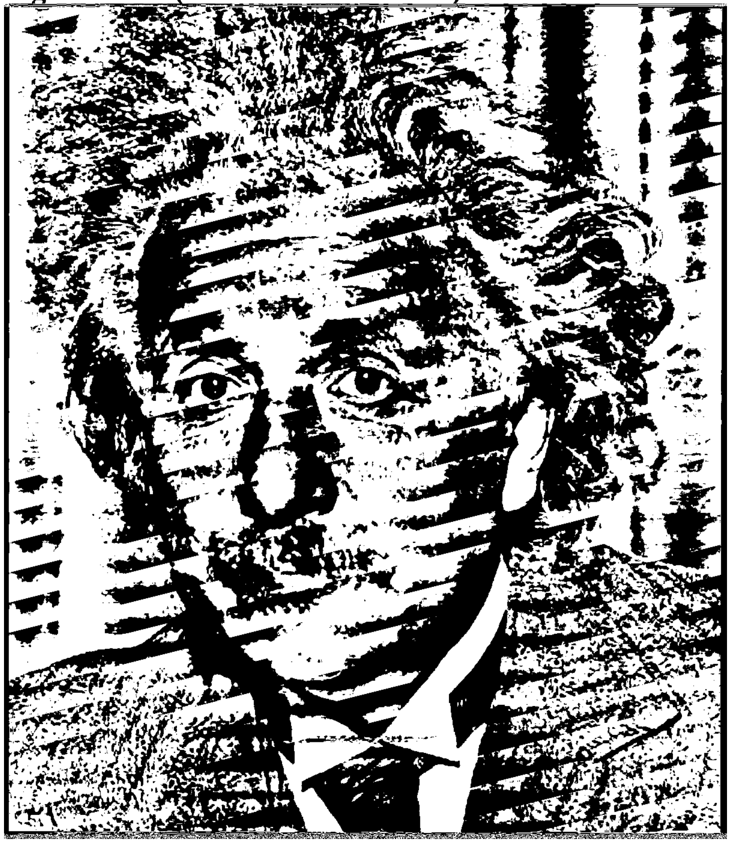


Figure 3.R3: (LocThreshRow_ein2)

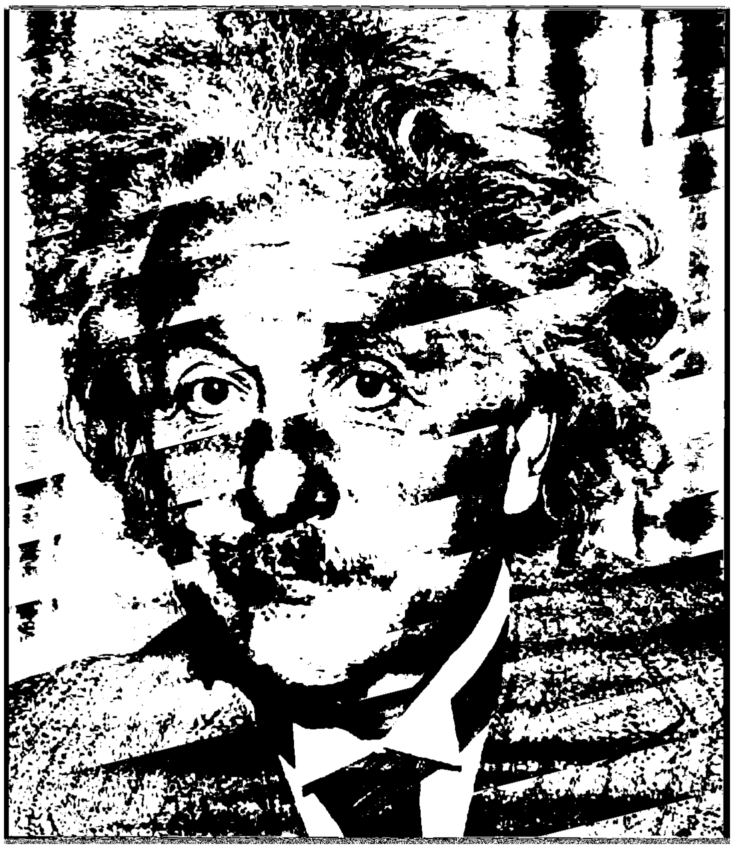


Figure 3.R4: (LocThreshRow_ein3)

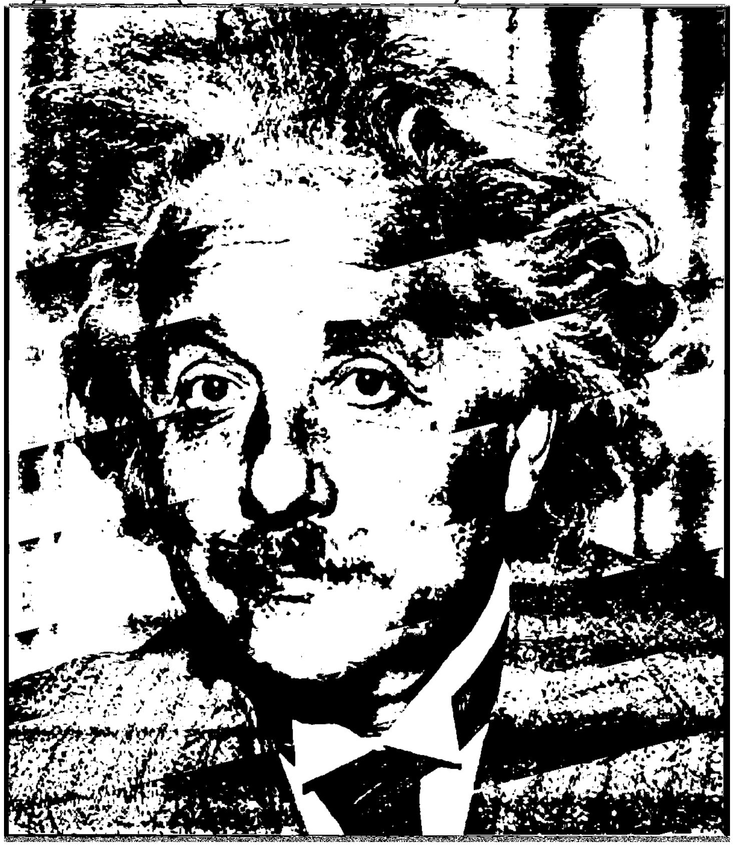


Figure 3.R5: (LocThreshRow_ein4)

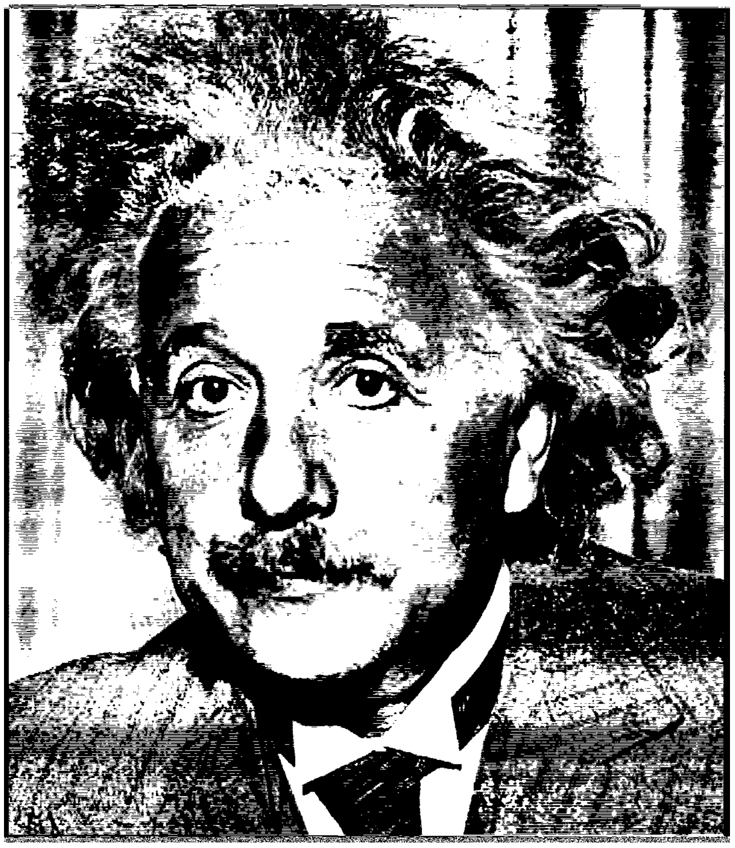


Figure 3.R6: (LocThreshRow_ein5)

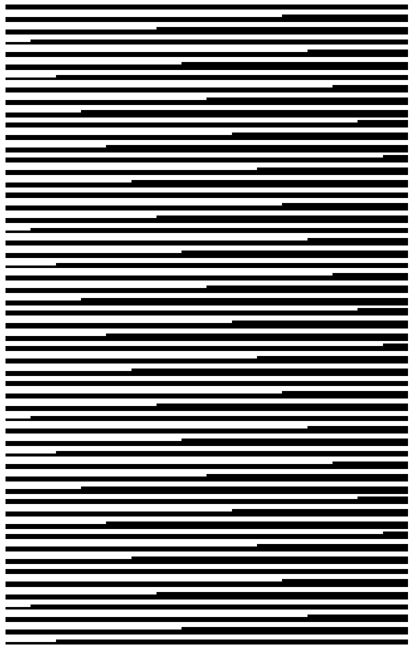


Figure 3.R6.1:(Locthreshrow_nop750)

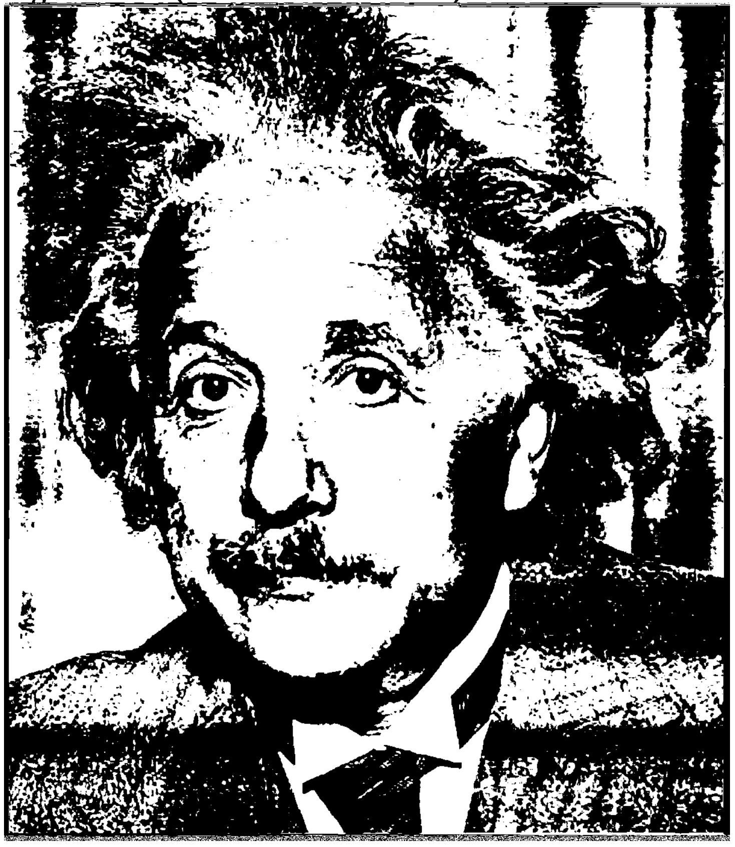


Figure 3.R7: (LocThreshRow_ein6)

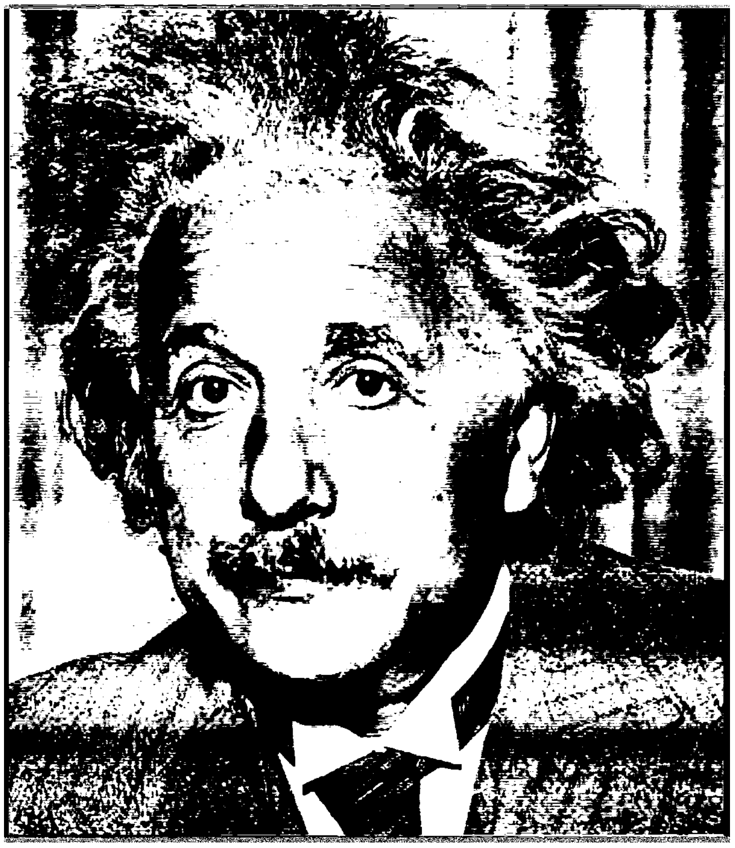


Figure 3.R8: (LocThreshRow_ein7)

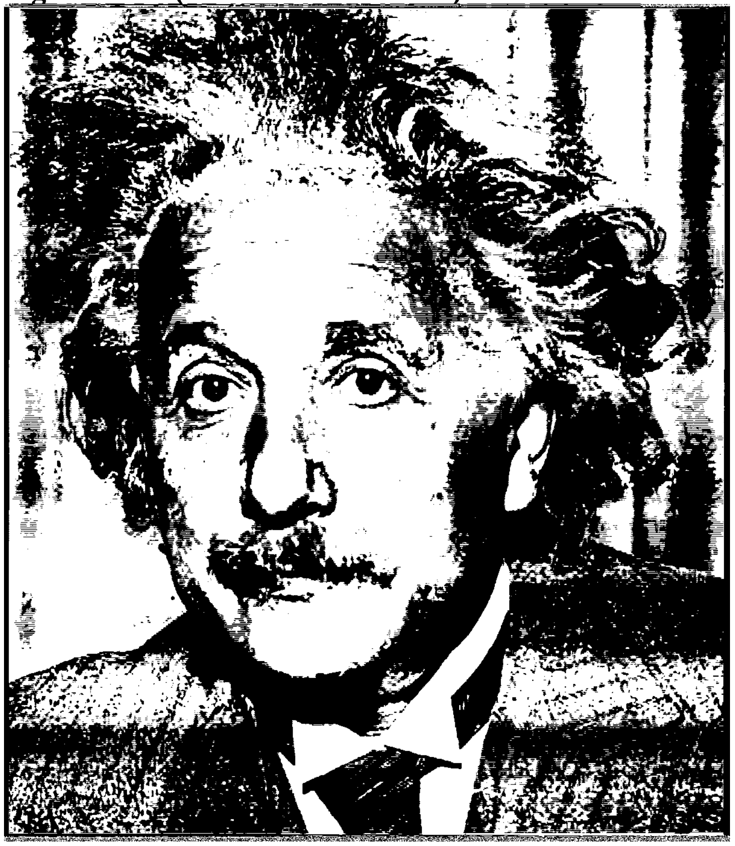


Figure 3.R9: (LocThreshRow_ein8)

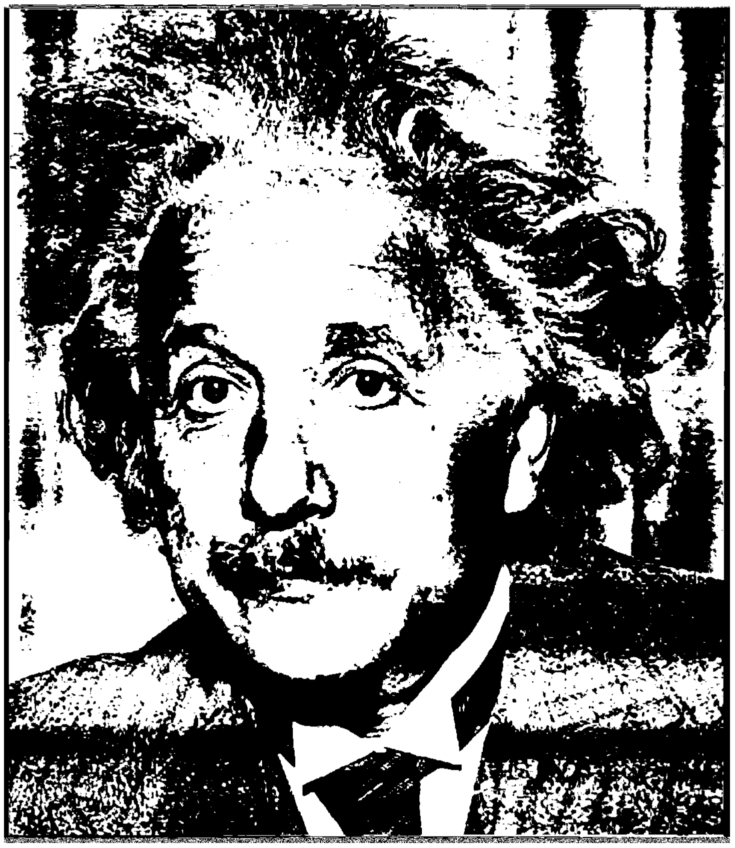


Figure 3.R10: (LocThreshRow_ein9)

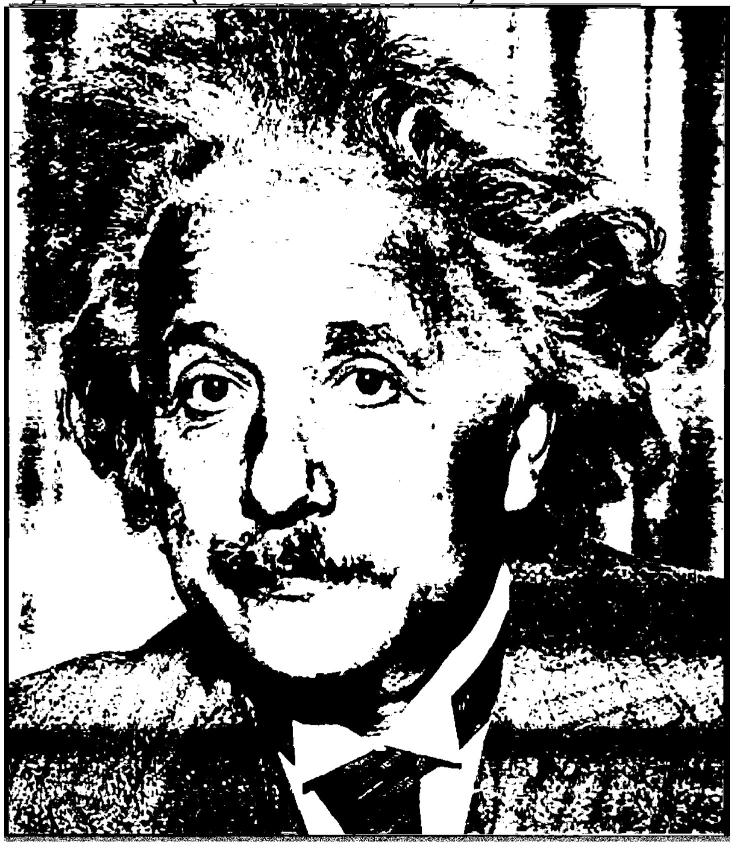


Figure 3.R11: (LocThreshRow_ein10)

3.3.3.2.3 Discussions

Filenames are given in parenthesis for the fig.3.R2-fig.3.R11, fig.3.R2.1 and fig.3.R6.1. Image details are more as we process less numbers of pixels per group. Visual artifacts are more in the processed images fig.3.R2 (LocThreshRow_ein1) to fig.3.R5 (LocThreshRow_ein4), artifacts have disappeared from the process image fig.3.R6 (LocThreshRow_ein5) to fig. 3.R11 (LocThreshRow_ein10).

3.3.3.2.4 Program 3.5

The following program function (in matlab 6.1) does row-wise processing of gray images, while selecting group of pixels locationwise (along the column) to generate a binary image.

```
function LocThreshRow(im,nop)
%------------------------------------------------------------
%------------------------------------------------------------
% LocThreshRow(im,nop)
% LOCTHRESHROW (function) : LOCation THRESHold ROWwise
%
% This function generates a thresholded image from
% gray or rgb image by selecting group of pixels
% locationwise (along the column) but processing is done
% along the row.
%
% INPUT ARGUMENTS:
% im: gray or rgb image or matrix of m by n.
%
% nop :number of pixels thresholded at a time,
% (if nop is column size it is equivalent to rcthresh-column)
% except the last group, which may be less.
%
% EXAMPLES:
% 1.
% locthreshrow('pout.tif',4)
% 2.
% locthreshrow('flowers.tif',100)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%------------------------------------------------------------
%------------------------------------------------------------
if  ischar(im)
```

```
    im=imread(im);
    [r c dim]=size(im);

    if  dim==3
        im=rgb2gray(im);
    end

else
    im=im;
end

im=im.';

pix=numel(im);

nop;
rm=rem(pix,nop);

nht2=[];
im3=[];
imend=[];
htend=[];
for i=0:nop:pix

    if  nop+i<=pix
        im2=im(1+i:nop+i);
        im3=[im3 im2];
        nht=thresh(im2,1,1,0);
        nht2=[nht2 nht];
    end
```

```
  if rm~=0
    imend=im(end-rm+1:end);
    htend=thresh(imend,1,1,0);
  end
    nht3=[nht2 htend];
    im4=[im3 imend];

end

im4;
nht3;

nht4=unstkr(nht3,r,c);
nht4=logical(nht4);
imshow(nht4)

imwrite(nht4,'LocThreshRow.tif','Resolution',72)
%-----------------------------------------------------------
%-----------------------------------------------------------
```

3.3.3.3 Processing the Image in Arbitrary Path

Here image is processed in arbitrary path as done through Matlab 6.1 functions Paththresh (Section 3.3.3.3.3, Program 3.6) and Patpath (Section 3.3.3.3.4, Program 3.7)

Table 3.5

Function	**Original file**	**pm**	**Nop (Number of pixels)**	**Output File name**
Paththresh.m & patpath.m				
Threshold opt=1	einstein72.tif	rand	10	pathth_ein1
do	do		50	pathth_ein2
do	do		100	pathth_ein3
do	do		500	pathth_ein4
do	do		2000	pathth_ein5
do	do		5000	pathth_ein6
do	gs3.tif		10	gs3_pathth_10nop
do	gs3.tif		100	gs3_pathth_100nop

This above table 3.5 shows output file name (output images) processed corresponding to different nop (number of pixels) used to process the images from the original file einstein72.tif (fig.3.A1) and gs3.tif (fig.3.A1.1) with paththresh function and threshold option=1.

3.3.3.3.1 Output Images

The processing of fig.3.A1 in arbitrary path generates the following pictures (fig.3.A1-fig.3.A7) as output similarly processing of fig.3.A1.1 generates fig.3.A2.1 and fig.3.A4.1

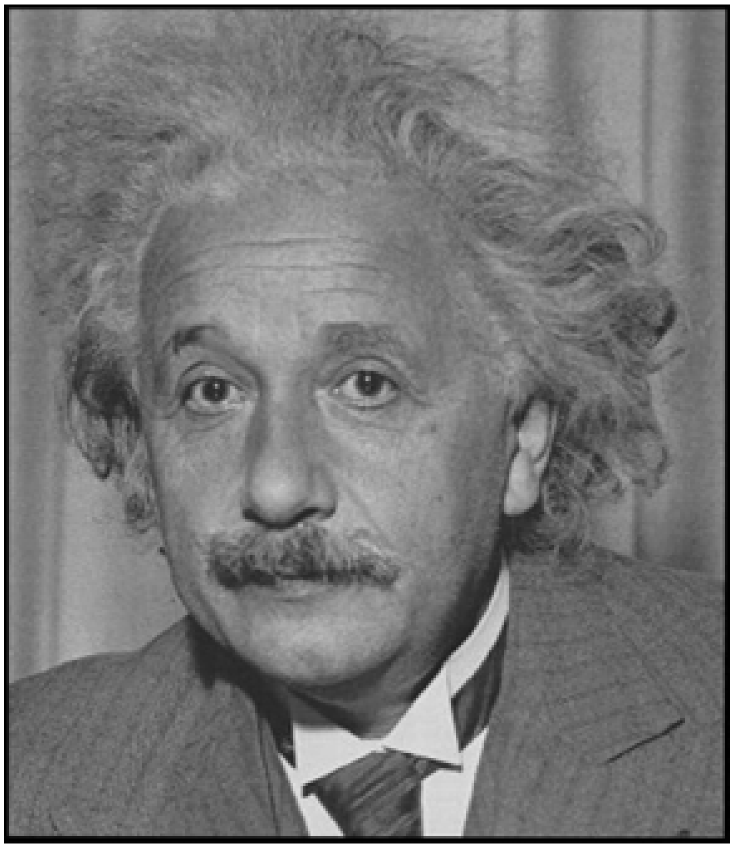


Figure 3.A1 : Sample Image (einstein72.tif)

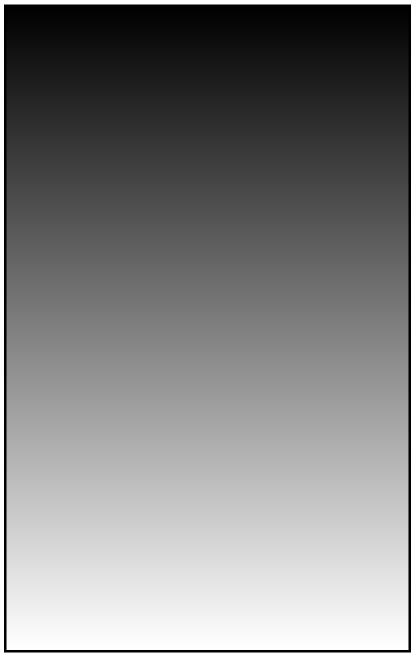


Figure 3.A1.1 : Grayscale (gs3.tif)

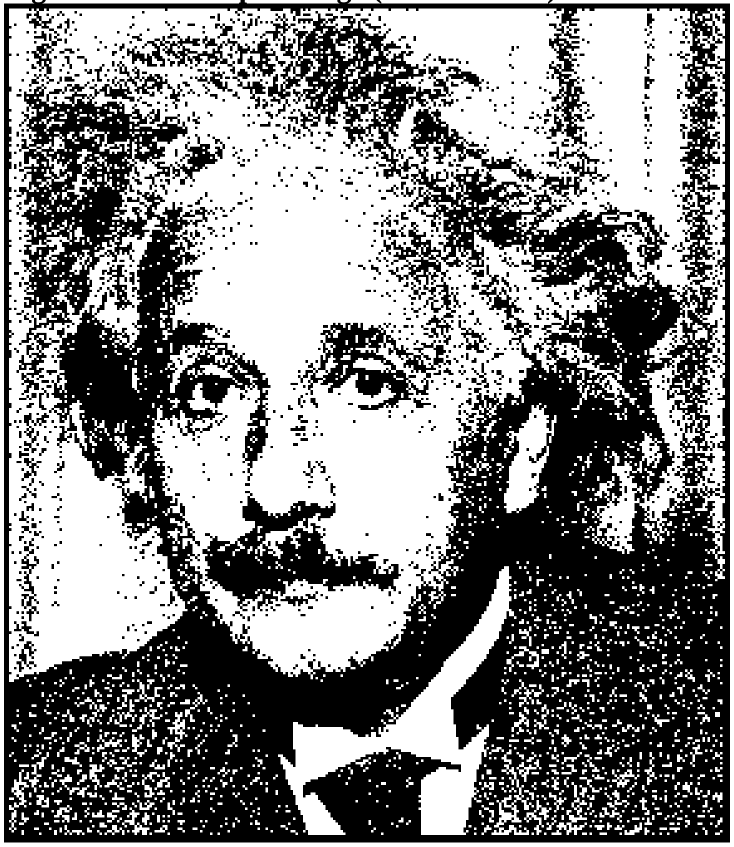


Figure 3.A2: (pathth_ein1)

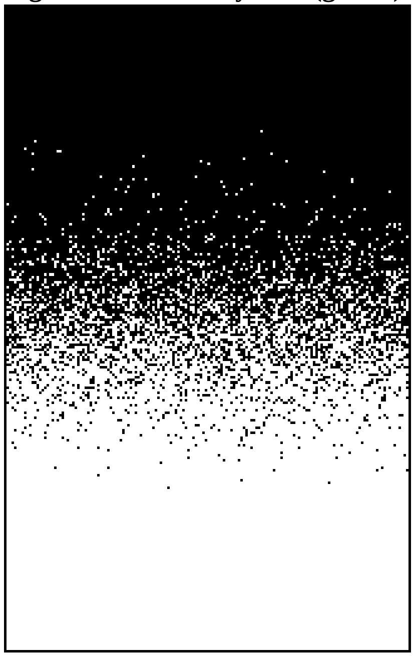


Figure 3.A2.1 : (gs3_pathth_10nop)

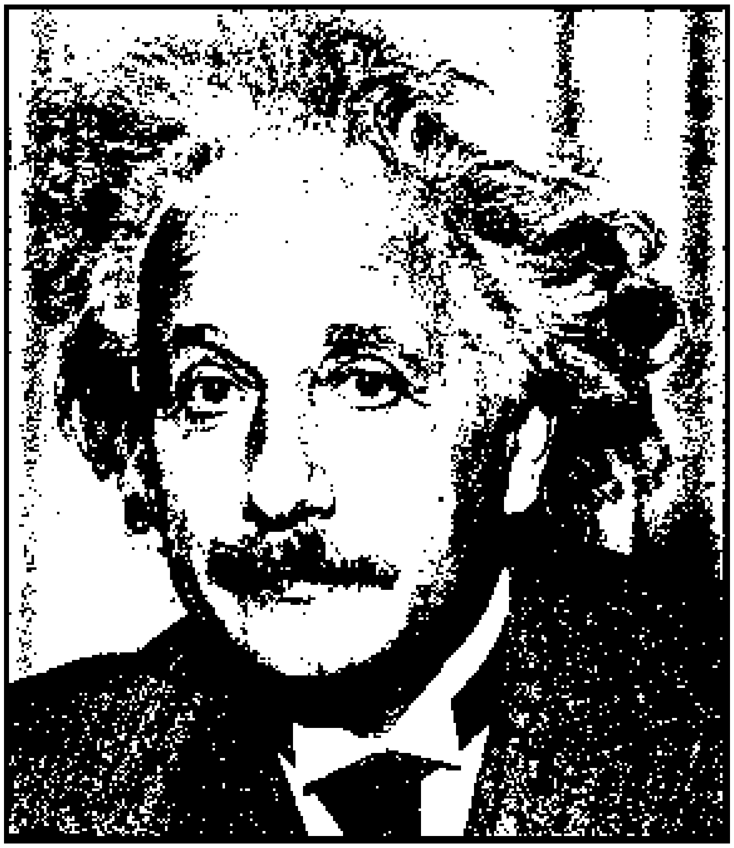


Figure 3.A3: (pathth_ein2)

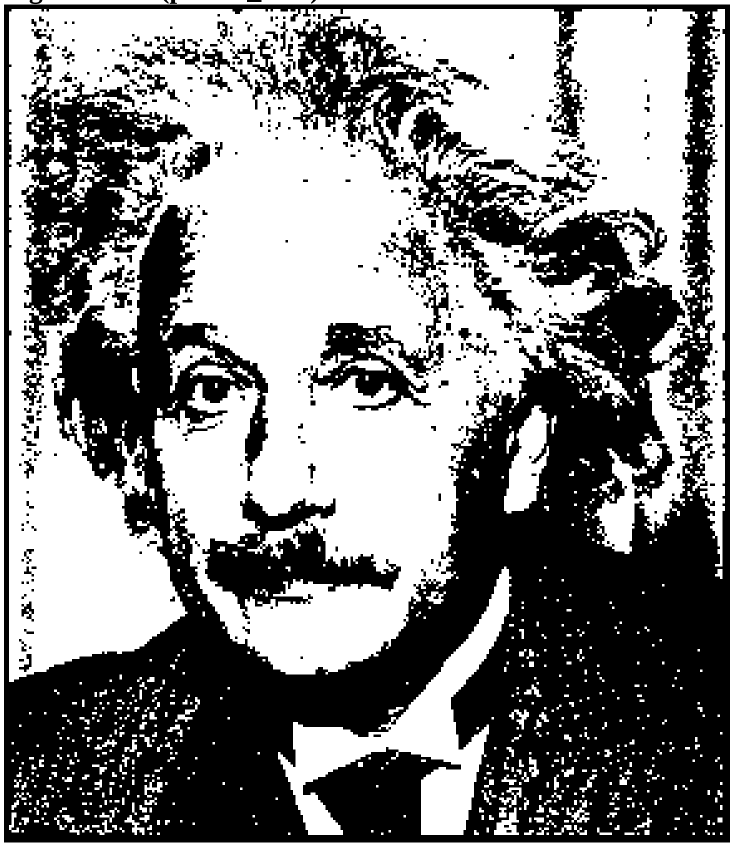


Figure 3.A4: (pathth_ein3)

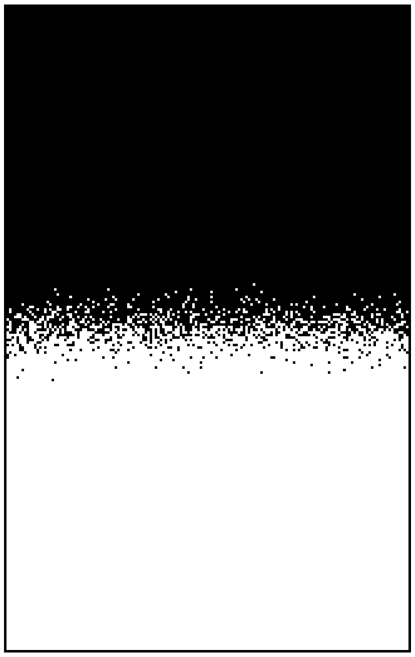


Figure 3.A4.1: (gs3_paththh_100nop)

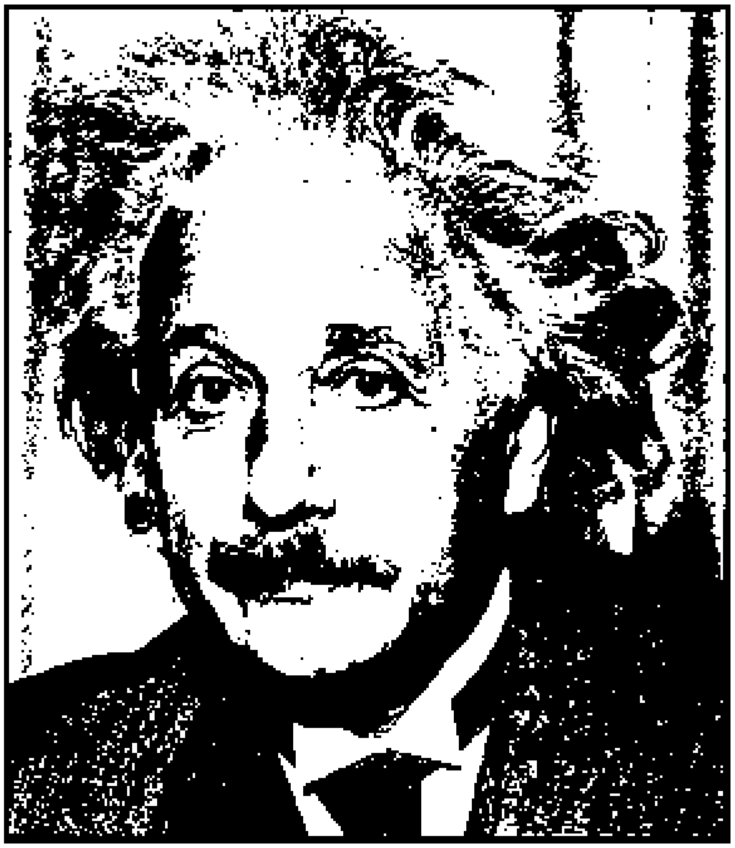


Figure 3.A5: (pathth_ein4)

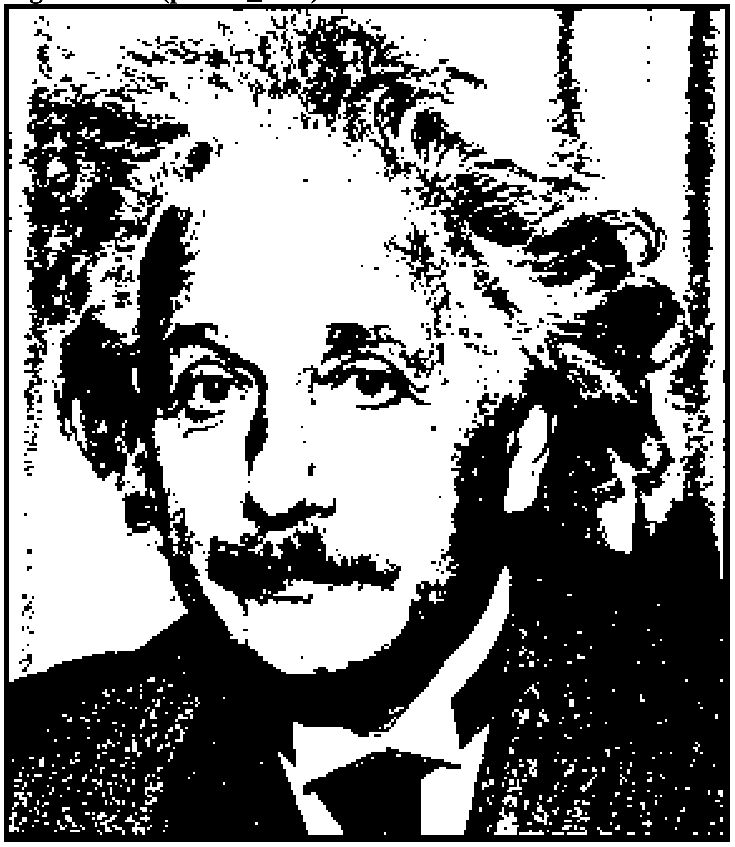


Figure 3.A6: (pathth_ein5)

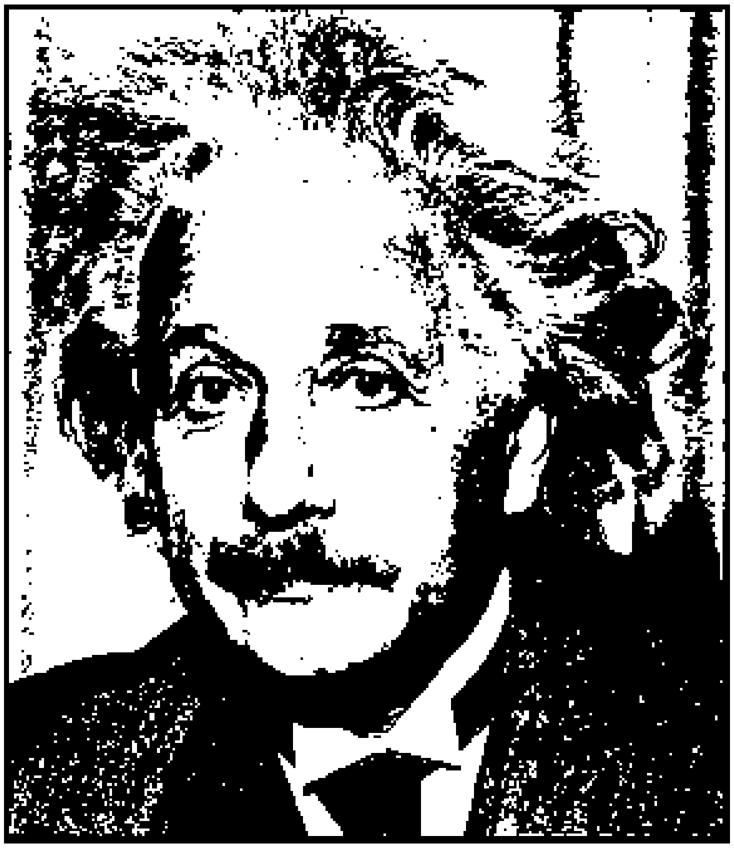


Figure 3.A7: (pathth_ein6)

3.3.3.3.2 Discussions

Filenames are given in parenthesis for the fig.3.A1-fig.3.A7, fig.3.A1.1, fig.3.A2.1 and fig.3.A4.1. Here a low resolution input image file fig.3.A1 (einstein72.tif) is used with resolution 72 dpi to process the images titled from fig.3.A2 (pathth_ein1) to fig.3.A7 (pathth_ein6). As image processing time is very high, low resolution image is used. In the first processed image fig.3.A2 (pathth_ein1) image details are more which is processed with nop (number of pixels) = 10. Image details are slowly reduced from fig.3.A2 (pathth_ein1) to fig.3.A7 (pathth_ein6) as we increase the nop.

The fig. 3.A2.1 (gs3_pathth_10nop) is a grayscale (fig. 3.A1.1) processed with nop = 10 shows more details in the middle tone region and fig. 3.A4.1 (gs3_pathth_100nop) is a grayscale processed with nop = 100 shows less image details.

3.3.3.3.3 Program 3.6

The following program function (in matlab 6.1) selects sequentially groups of pixels of an image matrix or matrix along the defined path and thresholds the selected group of pixels and generate the final non halftone binary image.

```
function PathThresh(im,nop);
%--------------------------------------------------------------
%--------------------------------------------------------------
% PathThresh(im,nop)
% PATHTHRESH (function) : PATHwise THRESHold
%
% This function selects sequentially groups of pixels of an
% image matrix or matrix along the diagonal spiral or zigzag
% path  and thresholds the selected group of pixels and generate
% the final non halftone binary image.
%
% INPUT ARGUMENTS:
% im: gray image file
% nop: no of pixels
%
% finput: the input function
%      : DiagVectorLTRB or DiagVectorRTLB
%
% EXAMPLE:
% PathThresh('flgray.tif',100)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%--------------------------------------------------------------
%--------------------------------------------------------------
im=imread(im);
[r c dim]=size(im)
pix=r*c
```

```
%------
% Path functions
pm=rand(r,c);
path=PatPath(pm); % tn:  path
tn=path;
%----

rc=r*c
L=length(tn)
nop

rm=rem(rc,nop) % left over pixels
div=rc/nop
fl=floor(div)

[r2 c2 dim]=size(tn)
th2=[];
th4=[];
th5=[];

for i=0:nop:rc

   if  i+nop<=rc

       tn2=tn(:,i+1:i+nop);

       % thresholding the image pixels in the path
       th=thresh(im(tn2),1,1,0);
       th2=[th2 th];
   end
```

```
  if rm~=0
    tn3=tn(:,end-rm+1:end);

    % thresholding the image pixels in the path
    th4=Thresh(im(tn3),1,1,0);
  end

end
th5=[th2 th4];

LL=length(th5)
tn;

im(tn)=th5(:);

im=logical(im);
imshow(im)

imwrite(im,'pathth.tif')
%------------------------------------------------------------
```

3.3.3.3.4 Program 3.7

The following program function (in matlab 6.1) generates path in location matrix from any square or rectangular matrix (spiral etc) for image processing

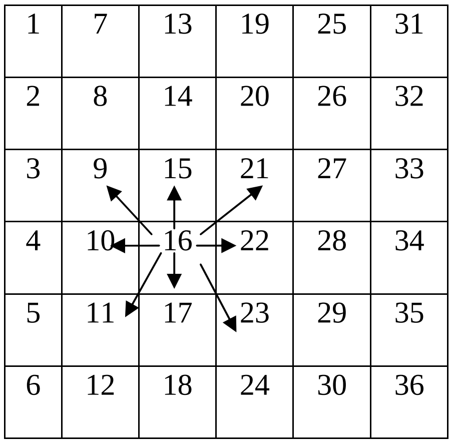
```
function path=patpath(pm)
%----------------------------------------------------------
%----------------------------------------------------------
% This function generates path in location matrix from any
% square or rectangular matrix (spiral etc) for image
% processing
%
% INPUT ARGUMENT:
% pm: pattern matrix
%
% OUTPUT ARGUMENT:
% path: final pattern path in location matrix
%
% EXAMPLES:
% 1.
% pm=spiral(4);
% path=patpath(pm);
% 2.
% pm=rand(4,5);
% path=patpath(pm);
%
% Tested using Matlab 6.1.0.450 Release 12.1
%----------------------------------------------------------
%----------------------------------------------------------
[r c dim]=size(pm)

Lpm=numel(pm)
```

```
Ln=[];
pmv=[];
for i=1:Lpm

   % Serial location numbers
   Ln=[Ln; i];

   % Pattern matrix values
   pmv=[pmv; pm(i)];
end

ps=[pmv Ln];

% Arranging for final location path
pss=sortrows(ps);

% Extracting final location based path
path=pss(:,2);
path=path.';

% Uses:
lm(path)=1:r*c;
%-----------------------------------------------------------
```

3.3.3.4 Processing the Image Diagonally

Image is processed in various diagonal directions [27]. Location mapping helps in this regard. The idea of pixel location is described in the Section 3.3.3.1.1.

3.3.3.4.1 Location Mapping



1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35
6	12	18	24	30	36

Figure 3.d1: 6x6 Location Matrix

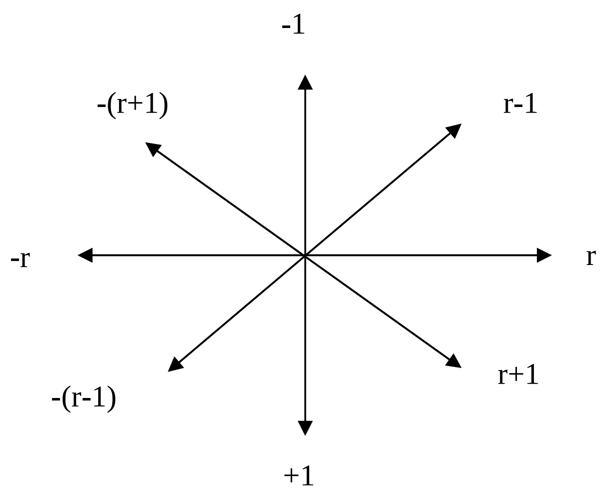


Figure 3.d2: Eight Directions within a Location Matrix

Figure 3.d2 shows graphical representation of eight directions within a location matrix as shown in the figure 3.d1. This concept is used in the Matlab 6.1 function locmap (Section 3.3.3.4.26 Program 3.11)

3.3.3.4.2 Types of Diagonal Processing

1. Starting point of the diagonal is Left Top or Right Bottom (LTRB)
    1.1. LTDAVFTC (Left top diagonal alternate vector spiral with first turn clockwise)
    1.2. LTDAVFTAC (Left diagonal alternate vector spiral with first turn anti-clockwise)
    1.3. LTDVLB2RT (Left top diagonal vectors, left bottom to right top)
    1.4. LTDVRT2LB (Left top diagonal vectors, right top to left bottom)
    1.5. RBDAVFTAC (Right bottom (starting point) diagonal alternate vector spiral with first turn anti-clockwise. 5 is reverse of 1).
    1.6. RBDAVFTC (Right bottom diagonal alternate vector spiral with first turn clockwise. 6 is reverse of 2)
    1.7. RBDVRT2LB (Right bottom diagonal vectors, right top to left bottom. 7 is reverse of 3)
    1.8. RBDVLB2RT (Right bottom diagonal vectors, left bottom to right top. 8 is reverse of 4)
2. Starting point of the diagonal is Right Top or Left Bottom (RTLB)
    2.1. RTDAVFTC (Right top diagonal alternate vector spiral with first turn clockwise)
    2.2. RTDAVFTAC (Right top diagonal alternate vector spiral with first turn anticlockwise)
    2.3. RTDVRB2LT (Right top diagonal vectors, right bottom to left top)
    2.4. RTDVLT2RB (Right top diagonal vectors, left top to right bottom)
    2.5. LBDAVFTAC (Left bottom (starting point) diagonal alternate vector spiral with first turn anti-clockwise. 5 is reverse of 1).
    2.6. LBDAVFTC (Left bottom diagonal alternate vector spiral with first turn clockwise. 6 is reverse of 2)
    2.7. LBDVLT2RB (Left bottom diagonal vectors, left top to right bottom. 7 is reverse of 3).
    2.8. LBDVRB2LT (Left bottom diagonal vectors, right bottom to left top. 8 is reverse of 4)

3.3.3.4.3 Algorithm for the above methods

1. Selecting the group of pixel diagonally
2. Then thresholding is done on the above group of pixels serially to obtain the final binary image.

3.3.3.4.4 Image Processing Directions and Matrices

1. LTRB Opt1 (LTDAVFTC)

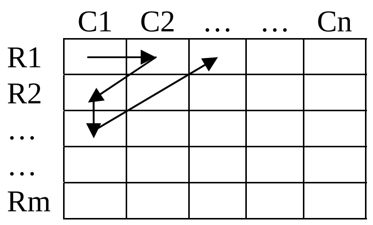


Figure 3.1La: Image processing direction

1. LTRB Opt1 (LTDAVFTC)

Figure 3.1Lb

1	2	6	7	15	16	28
3	5	8	14	17	27	29
4	9	13	18	26	30	42
10	12	19	25	31	41	43
11	20	24	32	40	44	56
21	23	33	39	45	55	57
22	34	38	46	54	58	70
35	37	47	53	59	69	71
36	48	52	60	68	72	84
49	51	61	67	73	83	85
50	62	66	74	82	86	95
63	65	75	81	87	94	96
64	76	80	88	93	97	102
77	79	89	92	98	101	103
78	90	91	99	100	104	105

Figure 3.1Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.1La shows the image processing direction. Fig.3.1Lb is the matrix showing image processing direction serially pixel wise. Figure 3.1Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

2. LTRB Opt2 (LTDAVFTAC)

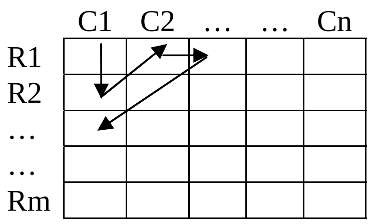


Figure 3.2La: Image processing direction

2. LTRB Opt2 (LTDAVFTAC)

Figure 3.2Lb

1	3	4	10	11	21	22
2	5	9	12	20	23	35
6	8	13	19	24	34	36
7	14	18	25	33	37	49
15	17	26	32	38	48	50
16	27	31	39	47	51	63
28	30	40	46	52	62	64
29	41	45	53	61	65	77
42	44	54	60	66	76	78
43	55	59	67	75	79	90
56	58	68	74	80	89	91
57	69	73	81	88	92	99
70	72	82	87	93	98	100
71	83	86	94	97	101	104
84	85	95	96	102	103	105

Figure 3.2Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.2La shows the image processing direction. Fig.3.2Lb is the matrix showing image processing direction serially pixel wise. Figure 3.2Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

3. LTRB Opt3 (LTDVLB2RT)

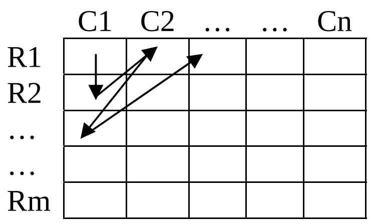


Figure 3.3La: Image processing direction

3. LTRB Opt3 (LTDVLB2RT)

Figure 3.3Lb

1	3	6	10	15	21	28
2	5	9	14	20	27	35
4	8	13	19	26	34	42
7	12	18	25	33	41	49
11	17	24	32	40	48	56
16	23	31	39	47	55	63
22	30	38	46	54	62	70
29	37	45	53	61	69	77
36	44	52	60	68	76	84
43	51	59	67	75	83	90
50	58	66	74	82	89	95
57	65	73	81	88	94	99
64	72	80	87	93	98	102
71	79	86	92	97	101	104
78	85	91	96	100	103	105

Figure 3.3Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.3La shows the image processing direction. Fig.3.3Lb is the matrix showing image processing direction serially pixel wise. Figure 3.3Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

4. LTRB Opt4 (LTDVRT2LB)

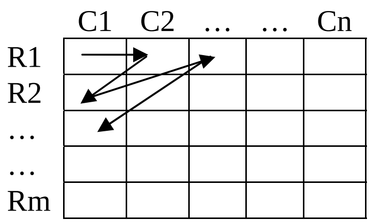


Figure 3.4La: Image processing direction

4. LTRB Opt4 (LTDVRT2LB)

Figure 3.4Lb

1	2	4	7	11	16	22
3	5	8	12	17	23	29
6	9	13	18	24	30	36
10	14	19	25	31	37	43
15	20	26	32	38	44	50
21	27	33	39	45	51	57
28	34	40	46	52	58	64
35	41	47	53	59	65	71
42	48	54	60	66	72	78
49	55	61	67	73	79	85
56	62	68	74	80	86	91
63	69	75	81	87	92	96
70	76	82	88	93	97	100
77	83	89	94	98	101	103
84	90	95	99	102	104	105

Figure 3.4Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.4La shows the image processing direction. Fig.3.4Lb is the matrix showing image processing direction serially pixel wise. Figure 3.4Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

5. LTRB opt5 (RBDAVFTAC)

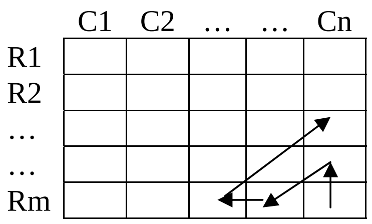


Figure 3.5La: Image processing direction

5. LTRB opt5 (RBDAVFTAC)

Figure 3.5Lb

105	103	102	96	95	85	84
104	101	97	94	86	83	71
100	98	93	87	82	72	70
99	92	88	81	73	69	57
91	89	80	74	68	58	56
90	79	75	67	59	55	43
78	76	66	60	54	44	42
77	65	61	53	45	41	29
64	62	52	46	40	30	28
63	51	47	39	31	27	16
50	48	38	32	26	17	15
49	37	33	25	18	14	7
36	34	24	19	13	8	6
35	23	20	12	9	5	2
22	21	11	10	4	3	1

Figure 3.5Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.5La shows the image processing direction. Fig.3.5Lb is the matrix showing image processing direction serially pixel wise. Figure 3.5Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

6. LTRB opt6 (RBDAVFTC)

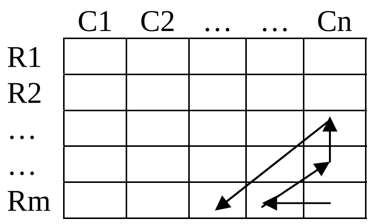


Figure 3.6La: Image processing direction

6. LTRB opt6 (RBDAVFTC)

Figure 3.6Lb

105	104	100	99	91	90	78
103	101	98	92	89	79	77
102	97	93	88	80	76	64
96	94	87	81	75	65	63
95	86	82	74	66	62	50
85	83	73	67	61	51	49
84	72	68	60	52	48	36
71	69	59	53	47	37	35
70	58	54	46	38	34	22
57	55	45	39	33	23	21
56	44	40	32	24	20	11
43	41	31	25	19	12	10
42	30	26	18	13	9	4
29	27	17	14	8	5	3
28	16	15	7	6	2	1

Figure 3.6Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.6La shows the image processing direction. Fig.3.6Lb is the matrix showing image processing direction serially pixel wise. Figure 3.6Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

7. LTRB opt7 (RBDVRT2LB)

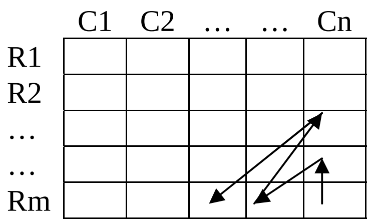


Figure 3.7La: Image processing direction

7. LTRB opt7 (RBDVRT2LB)

Figure 3.7Lb

105	103	100	96	91	85	78
104	101	97	92	86	79	71
102	98	93	87	80	72	64
99	94	88	81	73	65	57
95	89	82	74	66	58	50
90	83	75	67	59	51	43
84	76	68	60	52	44	36
77	69	61	53	45	37	29
70	62	54	46	38	30	22
63	55	47	39	31	23	16
56	48	40	32	24	17	11
49	41	33	25	18	12	7
42	34	26	19	13	8	4
35	27	20	14	9	5	2
28	21	15	10	6	3	1

Figure 3.7Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.7La shows the image processing direction. Fig.3.7Lb is the matrix showing image processing direction serially pixel wise. Figure 3.7Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

8. LTRB opt8 (RBDVLB2RT)

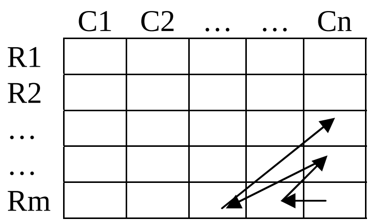


Figure 3.8La: Image processing direction

8. LTRB opt8 (RBDVLB2RT)

Figure 3.8Lb

105	104	102	99	95	90	84
103	101	98	94	89	83	77
100	97	93	88	82	76	70
96	92	87	81	75	69	63
91	86	80	74	68	62	56
85	79	73	67	61	55	49
78	72	66	60	54	48	42
71	65	59	53	47	41	35
64	58	52	46	40	34	28
57	51	45	39	33	27	21
50	44	38	32	26	20	15
43	37	31	25	19	14	10
36	30	24	18	13	9	6
29	23	17	12	8	5	3
22	16	11	7	4	2	1

Figure 3.8Lc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.8La shows the image processing direction. Fig.3.8Lb is the matrix showing image processing direction serially pixel wise. Figure 3.8Lc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

1. RTLB opt1 (RTDAVFTC)

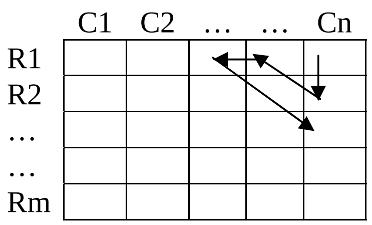


Figure 3.1Ra: Image processing direction

1. RTLB opt1 (RTDAVFTC)

Figure 3.1Rb

22	21	11	10	4	3	1
35	23	20	12	9	5	2
36	34	24	19	13	8	6
49	37	33	25	18	14	7
50	48	38	32	26	17	15
63	51	47	39	31	27	16
64	62	52	46	40	30	28
77	65	61	53	45	41	29
78	76	66	60	54	44	42
90	79	75	67	59	55	43
91	89	80	74	68	58	56
99	92	88	81	73	69	57
100	98	93	87	82	72	70
104	101	97	94	86	83	71
105	103	102	96	95	85	84

Figure 3.1Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.1Ra shows the image processing direction. Fig.3.1Rb is the matrix showing image processing direction serially pixel wise. Figure 3.1Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

2. RTLB opt2 (RTDAVFTAC)

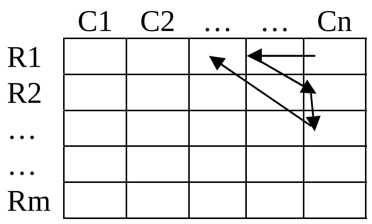


Figure 3.2Ra: Image processing direction

2. RTLB opt2 (RTDAVFTAC)

Figure 3.2Rb

28	16	15	7	6	2	1
29	27	17	14	8	5	3
42	30	26	18	13	9	4
43	41	31	25	19	12	10
56	44	40	32	24	20	11
57	55	45	39	33	23	21
70	58	54	46	38	34	22
71	69	59	53	47	37	35
84	72	68	60	52	48	36
85	83	73	67	61	51	49
95	86	82	74	66	62	50
96	94	87	81	75	65	63
102	97	93	88	80	76	64
103	101	98	92	89	79	77
105	104	100	99	91	90	78

Figure 3.2Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.2Ra shows the image processing direction. Fig.3.2Rb is the matrix showing image processing direction serially pixel wise. Figure 3.2Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

3. RTLB opt3 (RTDVRB2LT)

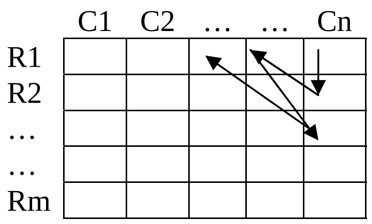


Figure 3.3Ra: Image processing direction

3. RTLB opt3 (RTDVRB2LT)

Figure 3.3Rb

28	21	15	10	6	3	1
35	27	20	14	9	5	2
42	34	26	19	13	8	4
49	41	33	25	18	12	7
56	48	40	32	24	17	11
63	55	47	39	31	23	16
70	62	54	46	38	30	22
77	69	61	53	45	37	29
84	76	68	60	52	44	36
90	83	75	67	59	51	43
95	89	82	74	66	58	50
99	94	88	81	73	65	57
102	98	93	87	80	72	64
104	101	97	92	86	79	71
105	103	100	96	91	85	78

Figure 3.3Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.3Ra shows the image processing direction. Fig.3.3Rb is the matrix showing image processing direction serially pixel wise. Figure 3.3Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

4. RTLB opt4 (RTDVLT2RB)

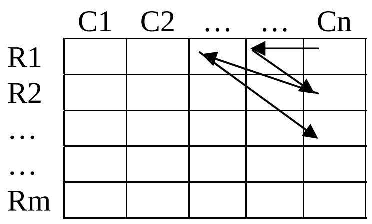


Figure 3.4Ra: Image processing direction

4. RTLB opt4 (RTDVLT2RB)

Figure 3.4Rb

22	16	11	7	4	2	1
29	23	17	12	8	5	3
36	30	24	18	13	9	6
43	37	31	25	19	14	10
50	44	38	32	26	20	15
57	51	45	39	33	27	21
64	58	52	46	40	34	28
71	65	59	53	47	41	35
78	72	66	60	54	48	42
85	79	73	67	61	55	49
91	86	80	74	68	62	56
96	92	87	81	75	69	63
100	97	93	88	82	76	70
103	101	98	94	89	83	77
105	104	102	99	95	90	84

Figure 3.4Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.4Ra shows the image processing direction. Fig.3.4Rb is the matrix showing image processing direction serially pixel wise. Figure 3.4Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

5. RTLB opt5 (LBDAVFTAC)

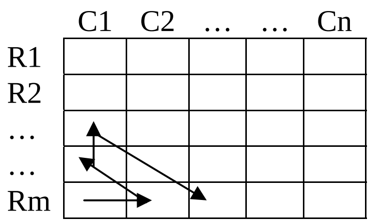


Figure 3.5Ra: Image processing direction

5. RTLB opt5 (LBDAVFTAC)

Figure 3.5Rb

78	90	91	99	100	104	105
77	79	89	92	98	101	103
64	76	80	88	93	97	102
63	65	75	81	87	94	96
50	62	66	74	82	86	95
49	51	61	67	73	83	85
36	48	52	60	68	72	84
35	37	47	53	59	69	71
22	34	38	46	54	58	70
21	23	33	39	45	55	57
11	20	24	32	40	44	56
10	12	19	25	31	41	43
4	9	13	18	26	30	42
3	5	8	14	17	27	29
1	2	6	7	15	16	28

Figure 3.5Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.5Ra shows the image processing direction. Fig.3.5Rb is the matrix showing image processing direction serially pixel wise. Figure 3.5Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

6. RTLB opt6 (LBDAVFTC)

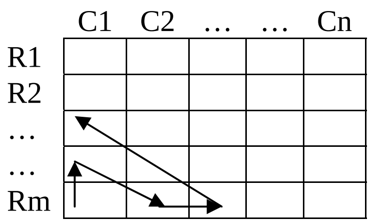


Figure 3.6Ra: Image processing direction

6. RTLB opt6 (LBDAVFTC)

Figure 3.6Rb

84	85	95	96	102	103	105
71	83	86	94	97	101	104
70	72	82	87	93	98	100
57	69	73	81	88	92	99
56	58	68	74	80	89	91
43	55	59	67	75	79	90
42	44	54	60	66	76	78
29	41	45	53	61	65	77
28	30	40	46	52	62	64
16	27	31	39	47	51	63
15	17	26	32	38	48	50
7	14	18	25	33	37	49
6	8	13	19	24	34	36
2	5	9	12	20	23	35
1	3	4	10	11	21	22

Figure 3.6Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.6Ra shows the image processing direction. Fig.3.6Rb is the matrix showing image processing direction serially pixel wise. Figure 3.6Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

7. RTLB opt7 (LBDVLT2RB)

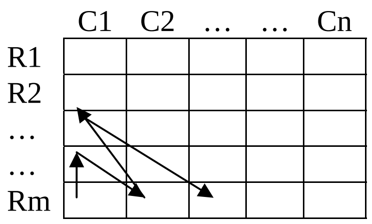


Figure 3.7Ra: Image processing direction

7. RTLB opt7 (LBDVLT2RB)

Figure 3.7Rb

78	85	91	96	100	103	105
71	79	86	92	97	101	104
64	72	80	87	93	98	102
57	65	73	81	88	94	99
50	58	66	74	82	89	95
43	51	59	67	75	83	90
36	44	52	60	68	76	84
29	37	45	53	61	69	77
22	30	38	46	54	62	70
16	23	31	39	47	55	63
11	17	24	32	40	48	56
7	12	18	25	33	41	49
4	8	13	19	26	34	42
2	5	9	14	20	27	35
1	3	6	10	15	21	28

Figure 3.7Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.7Ra shows the image processing direction. Fig.3.7Rb is the matrix showing image processing direction serially pixel wise. Figure 3.7Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

8. RTLB opt8 (LBDVRB2LT)

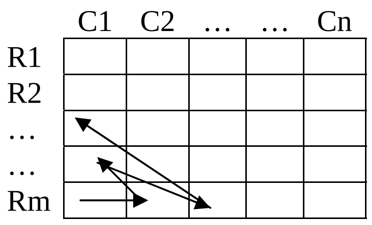


Figure 3.8Ra: Image processing direction

8. RTLB opt8 (LBDVRB2LT)

Figure 3.8Rb

84	90	95	99	102	104	105
77	83	89	94	98	101	103
70	76	82	88	93	97	100
63	69	75	81	87	92	96
56	62	68	74	80	86	91
49	55	61	67	73	79	85
42	48	54	60	66	72	78
35	41	47	53	59	65	71
28	34	40	46	52	58	64
21	27	33	39	45	51	57
15	20	26	32	38	44	50
10	14	19	25	31	37	43
6	9	13	18	24	30	36
3	5	8	12	17	23	29
1	2	4	7	11	16	22

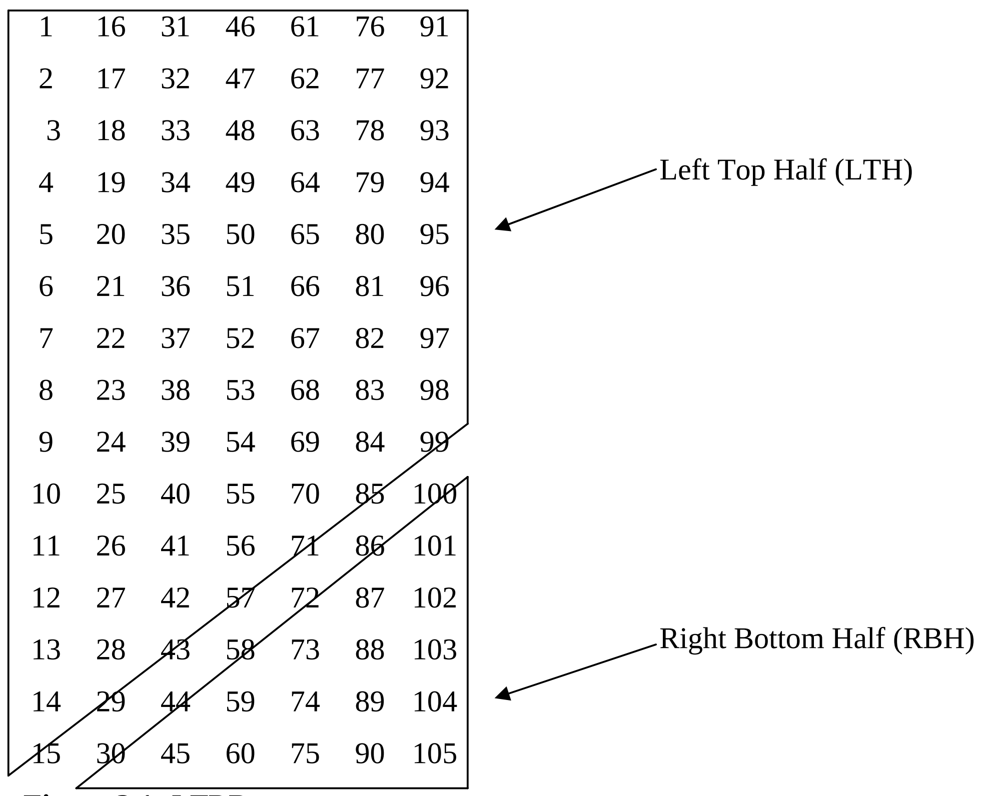
Figure 3.8Rc

1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.8Ra shows the image processing direction. Fig.3.8Rb is the matrix showing image processing direction serially pixel wise. Figure 3.8Rc is the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

3.3.3.4.5 Image Processing Techniques

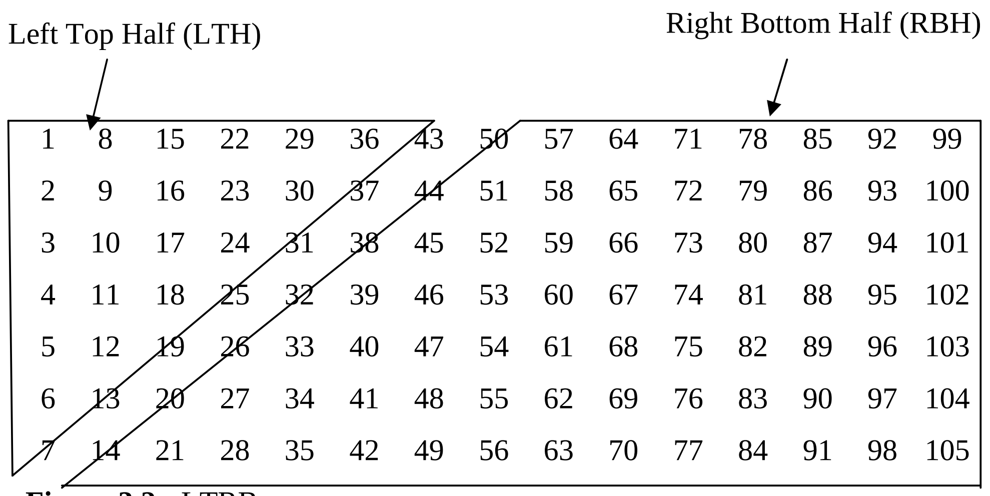
In the following figures (fig.3.1, fig.3.2, fig.3.5 and fig.3.6) image is splitted into two halves having pixel location numbers for further processing through Matlab 6.1 functions DiagvectorLTRB and DiagVector RTLB. Fig 3.3 and fig.3.4 show two halves of a real image.



1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Figure 3.1: LTRB

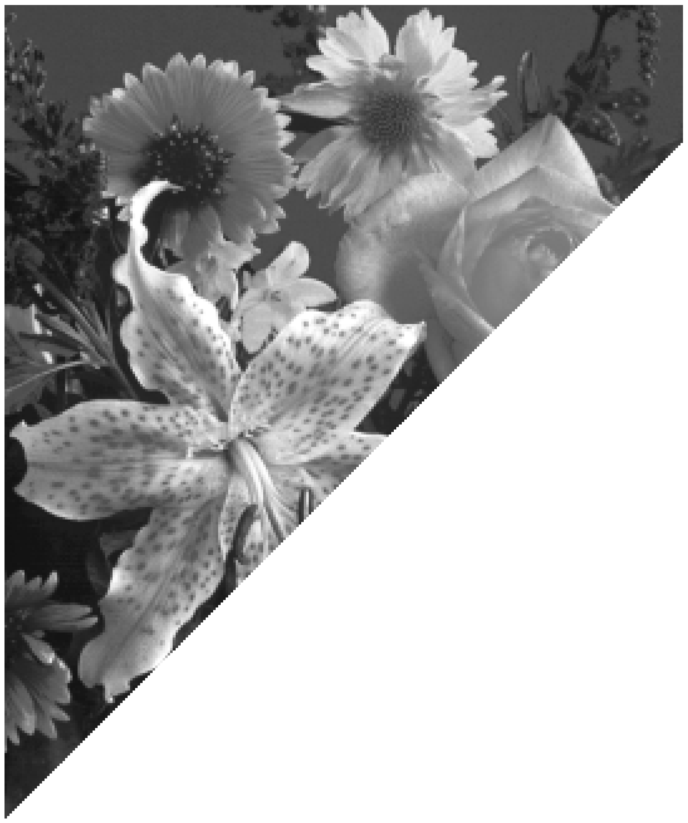
Fig.3.1 shows Left Top Half (LTH) and Right Bottom Half (RBH) of a portrait image with pixel location numbers for processing through the Matlab 6.1 function DiagVectorLTRB (Section 3.3.3.4.24 Program 3.9).



1	8	15	22	29	36	43	50	57	64	71	78	85	92	99
2	9	16	23	30	37	44	51	58	65	72	79	86	93	100
3	10	17	24	31	38	45	52	59	66	73	80	87	94	101
4	11	18	25	32	39	46	53	60	67	74	81	88	95	102
5	12	19	26	33	40	47	54	61	68	75	82	89	96	103
6	13	20	27	34	41	48	55	62	69	76	83	90	97	104
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105

Figure 3.2: LTRB

Fig.3.2 shows Left Top Half (LTH) and Right Bottom Half (RBH) of a landscape image with pixel location numbers for processing through the Matlab 6.1 function DiagVectorLTRB (Section 3.3.3.4.24 Program 3.9)



Left Top Half (LTH)

Fig.3.3 and fig.3.4 shows Left Top Half (LTH) and Right Bottom Half (RBH) of a real portrait image.

Figure 3.3

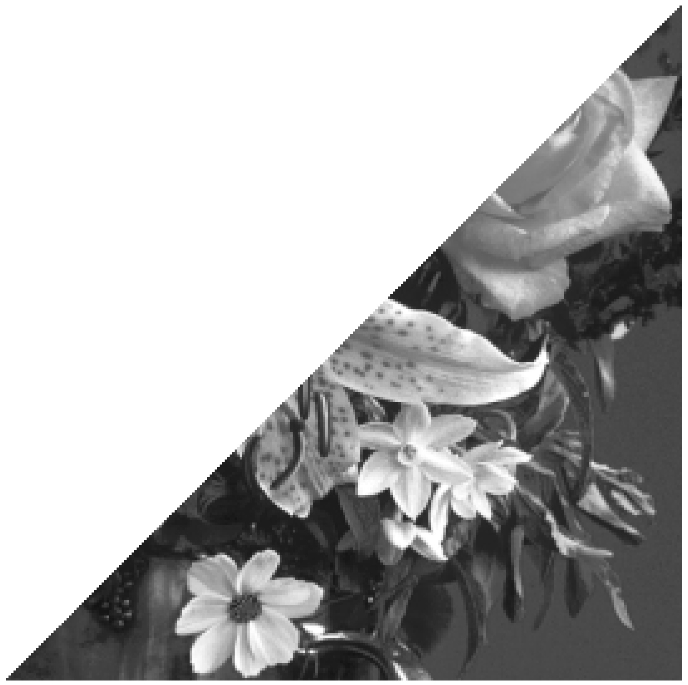
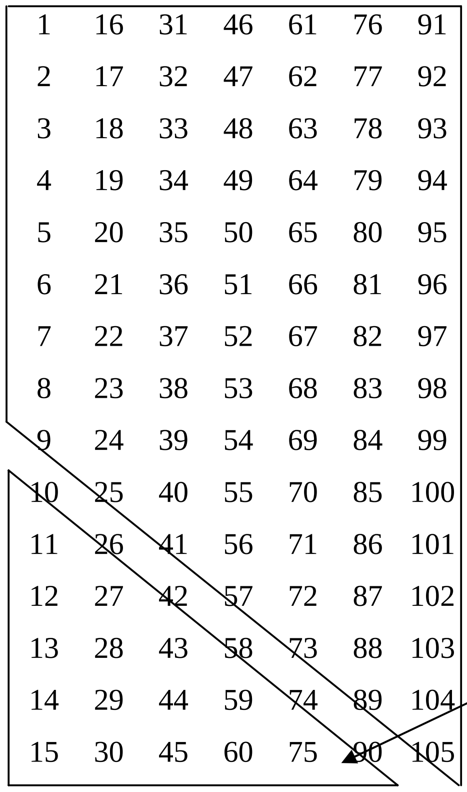


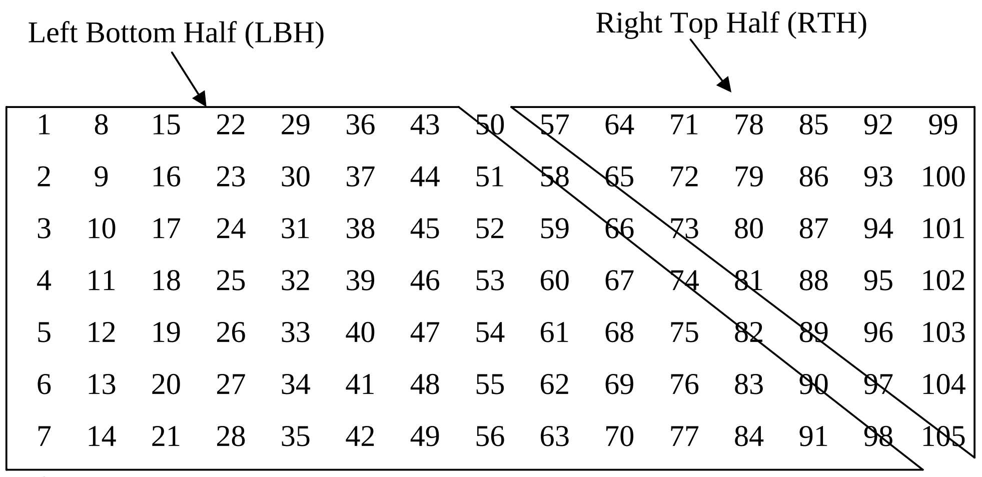
Figure 3.4



1	16	31	46	61	76	91
2	17	32	47	62	77	92
3	18	33	48	63	78	93
4	19	34	49	64	79	94
5	20	35	50	65	80	95
6	21	36	51	66	81	96
7	22	37	52	67	82	97
8	23	38	53	68	83	98
9	24	39	54	69	84	99
10	25	40	55	70	85	100
11	26	41	56	71	86	101
12	27	42	57	72	87	102
13	28	43	58	73	88	103
14	29	44	59	74	89	104
15	30	45	60	75	90	105

Fig.3.5 shows Right Top Half (RTH) and Left Bottom Half (LBH) of a portrait image with pixel location numbers for processing through the Matlab6.1 function DiagVectorRTLB (Section 3.3.3.4.25 Program 3.10)

Figure 3.5: RTLB



1	8	15	22	29	36	43	50	57	64	71	78	85	92	99
2	9	16	23	30	37	44	51	58	65	72	79	86	93	100
3	10	17	24	31	38	45	52	59	66	73	80	87	94	101
4	11	18	25	32	39	46	53	60	67	74	81	88	95	102
5	12	19	26	33	40	47	54	61	68	75	82	89	96	103
6	13	20	27	34	41	48	55	62	69	76	83	90	97	104
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105

Figure 3.6: RTLB

Fig.3.6 shows Right Top Half (RTH) and Left Bottom Half (LBH) of a landscape image with pixel location numbers for processing through the Matlab 6.1 function DiagVectorRTLB (Section 3.3.3.4.25 Program 3.10)

1 (1) [20]	2 (5) [150]	6 (9) [50]	7 (13) [12]	14 (17) [35]
3 (2) [100]	5 (6) [28]	8 (10) [120]	13 (14) [55]	15 (18) [76]
4 (3) [40]	9 (7) [69]	12 (11) [200]	16 (15) [88]	19 (19) [46]
10 (4) [25]	11 (8) [86]	17 (12) [180]	18 (16) [92]	20 (20) [130]

Figure 3.7: Three Matrices for Image Processing

In fig.3.7 open numerals shows the matrix having image processing direction serially pixel wise for the option LTDAVFTC (Left Top Diagonal Alternate Vector spiral with First Turn Clockwise) of the Matlab 6.1 function DiagVectorLTRB (Section 3.3.3.4.24 Program 3.9)

In fig.3.7 numerals in the round brackets show the location matrix. Within the location matrix serial numbers show the path of processing which is column-wise from top to bottom.

In fig.3.7 numerals in the square brackets shows pixel intensity values of a gray image.

3.3.3.4.6 Image Processing Tables

These following tables 3.6 and table 3.7 show output filenames corresponding to various nop (number of pixels), options, input filename and input functions.

Table 3.6					
Diagvector		**Opt 1**	**Opt 2**	**Opt 3**	**Opt 4**
Function: DiagVectorLTRB	**Nop**	**Result**	**Result**	**Result**	**Result**
einstein600	50	ein1L1	ein1L2	ein1L3	ein1L4
	100	ein2L1	ein2L2	ein2L3	ein2L4
	500	ein3L1	ein3L2	ein3L3	ein3L4
	1000	ein4L1	ein4L2	ein4L3	ein4L4
	2000	ein5L1	ein5L2	ein5L3	ein5L4
	5000	ein6L1	ein6L2	ein6L3	ein6L4
	10000	ein7L1	ein7L2	ein7L3	ein7L4
	20000	ein8L1	ein8L2	ein8L3	ein8L4
		Opt 5	**Opt 6**	**Opt 7**	**Opt 8**
	Nop	**Result**	**Result**	**Result**	**Result**
	50	ein1L5	ein1L6	ein1L7	ein1L8
	100	ein2L5	ein2L6	ein2L7	ein2L8
	500	ein3L5	ein3L6	ein3L7	ein3L8
	1000	ein4L5	ein4L6	ein4L7	ein4L8
	2000	ein5L5	ein5L6	ein5L7	ein5L8
	5000	ein6L5	ein6L6	ein6L7	ein6L8
	10000	ein7L5	ein7L6	ein7L7	ein7L8
	20000	ein8L5	ein8L6	ein8L7	ein8L8
		Opt1			
	Nop	**Result**			
einstein300	5	ein300_dthLTRB_opt1_nop5			
do	10	ein300_dthLTRB_opt1_nop10			
gs3	100	dth_ltrb_nop100			

Table 3.7					
Diagvector		**Opt 1**	**Opt 2**	**Opt 3**	**Opt 4**
Function: DiagVectorRTLB	**Nop**	**Result**	**Result**	**Result**	**Result**
einstein600	50	ein1R1	ein1R2	ein1R3	ein1R4
	100	ein2R1	ein2R2	ein2R3	ein2R4
	500	ein3R1	ein3R2	ein3R3	ein3R4
	1000	ein4R1	ein4R2	ein4R3	ein4R4
	2000	ein5R1	ein5R2	ein5R3	ein5R4
	5000	ein6R1	ein6R2	ein6R3	ein6R4
	10000	ein7R1	ein7R2	ein7R3	ein7R4
	20000	ein8R1	ein8R2	ein8R3	ein8R4
		Opt 5	**Opt 6**	**Opt 7**	**Opt 8**
	Nop	**Result**	**Result**	**Result**	**Result**
	50	ein1R5	ein1R6	ein1R7	ein1R8
	100	ein2R5	ein2R6	ein2R7	ein2R8
	500	ein3R5	ein3R6	ein3R7	ein3R8
	1000	ein4R5	ein4R6	ein4R7	ein4R8
	2000	ein5R5	ein5R6	ein5R7	ein5R8
	5000	ein6R5	ein6R6	ein6R7	ein6R8
	10000	ein7R5	ein7R6	ein7R7	ein7R8
	20000	ein8R5	ein8R6	ein8R7	ein8R8
		Opt1			
	Nop	**Result**			
einstein300	5	ein300_dthRTLB_opt1_nop5			
do	10	ein300_dthRTLB_opt1_nop10			
gs3	100	dth_rtlb_nop100			

3.3.3.4.7 LTRB: LTDAVFTC

3.3.3.4.7.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: LTDAVFTC option generates the following pictures (fig.3.L1.1-fig.3.L1.10) as output, similarly processing of fig.3.A1.1 generates fig.3.L1.4.1.

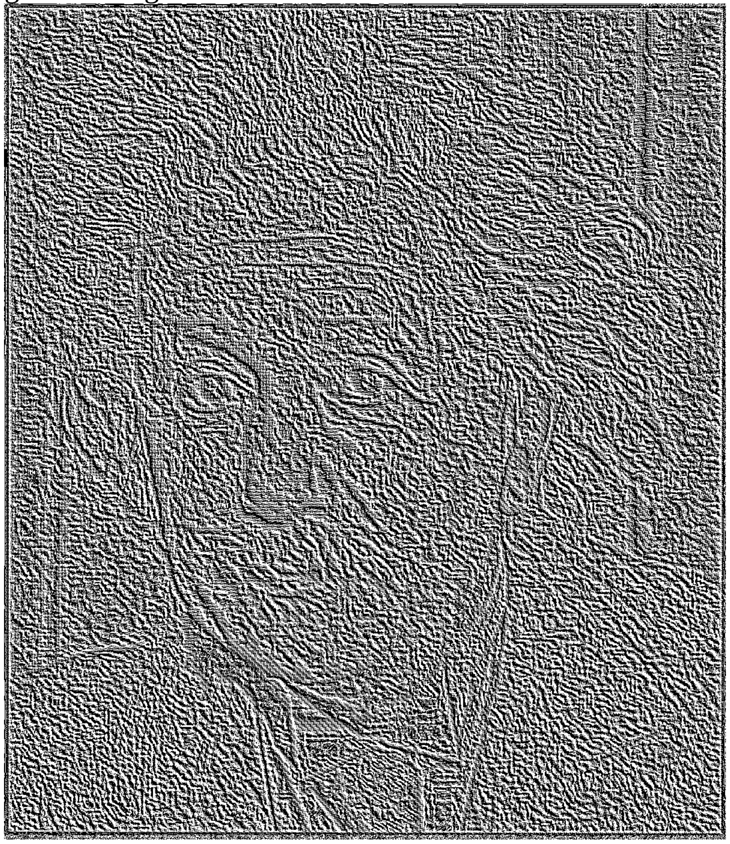


Figure 3.L1.1: (ein300_dthLTRB_nop5_opt1)

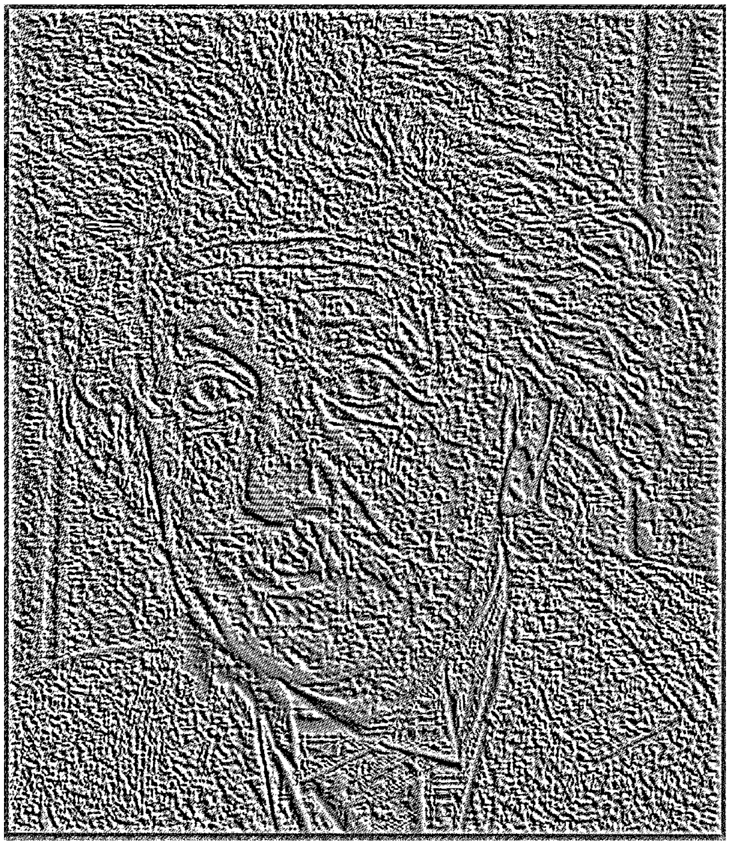


Figure 3.L1.2: (ein300_dthLTRB_nop10_opt1)

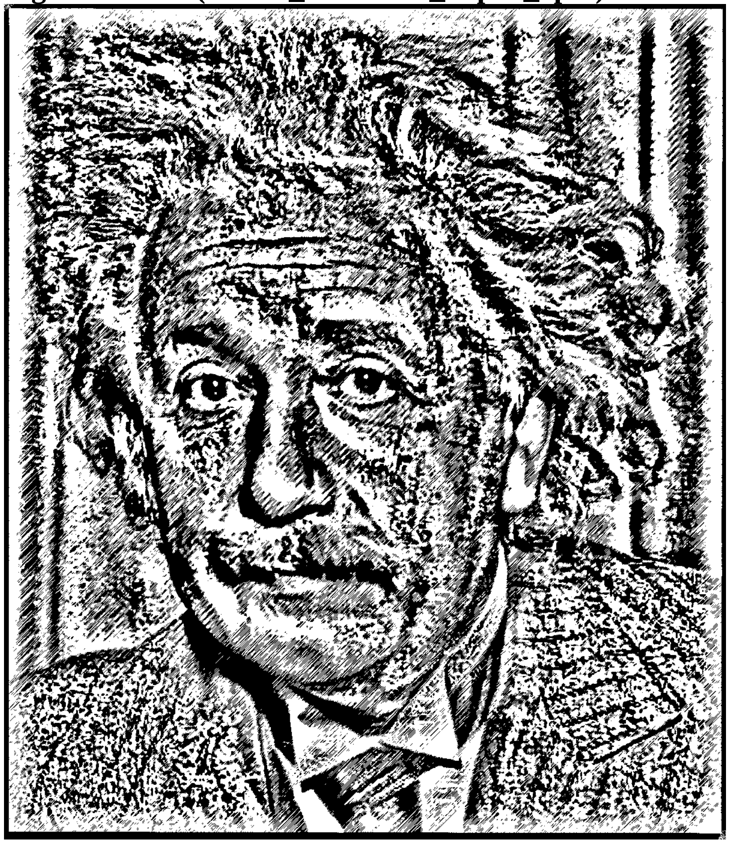


Figure 3.L1.3: (ein1L1)

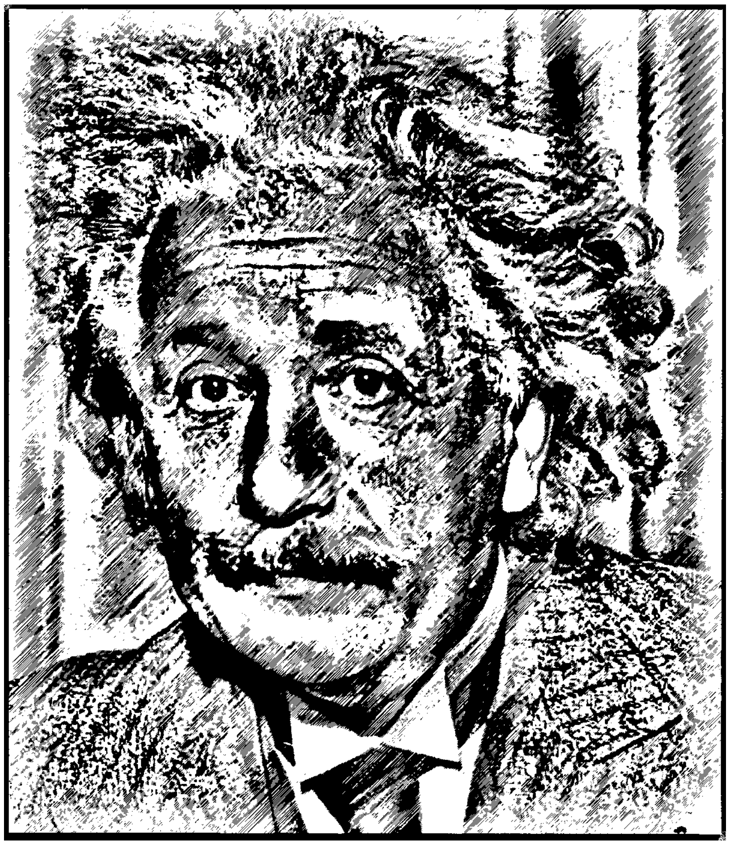


Figure 3.L1.4: (ein2L1)

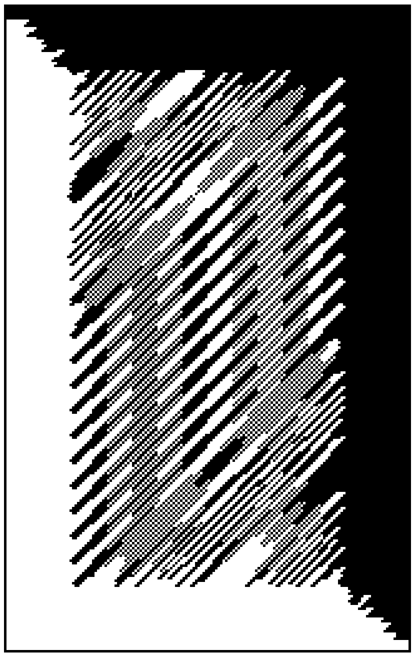


Fig 3.L1.4.1:(dth_ltrb_nop100)

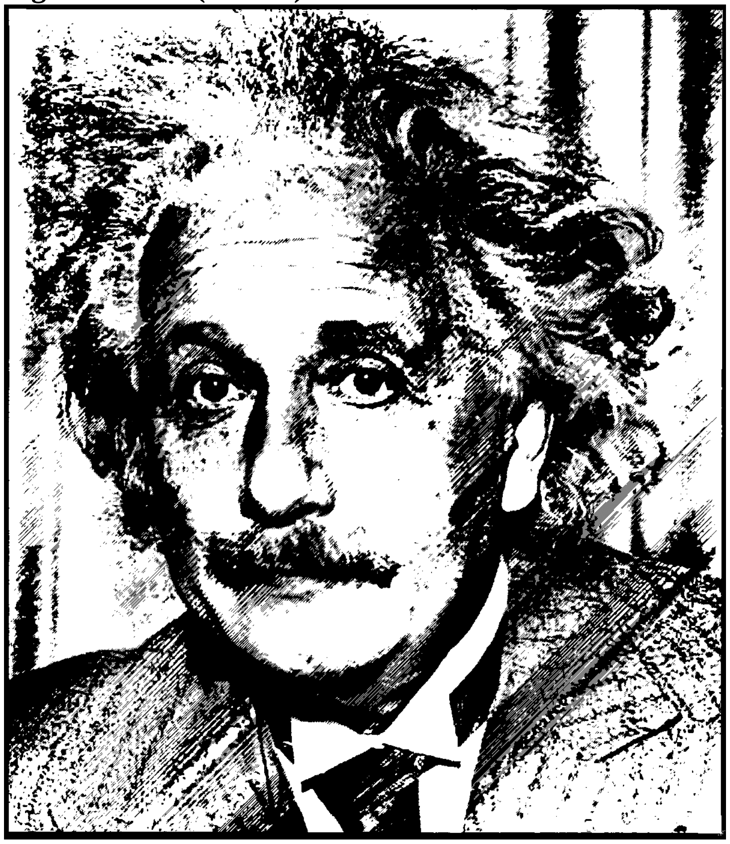


Figure 3.L1.5: (ein3L1)

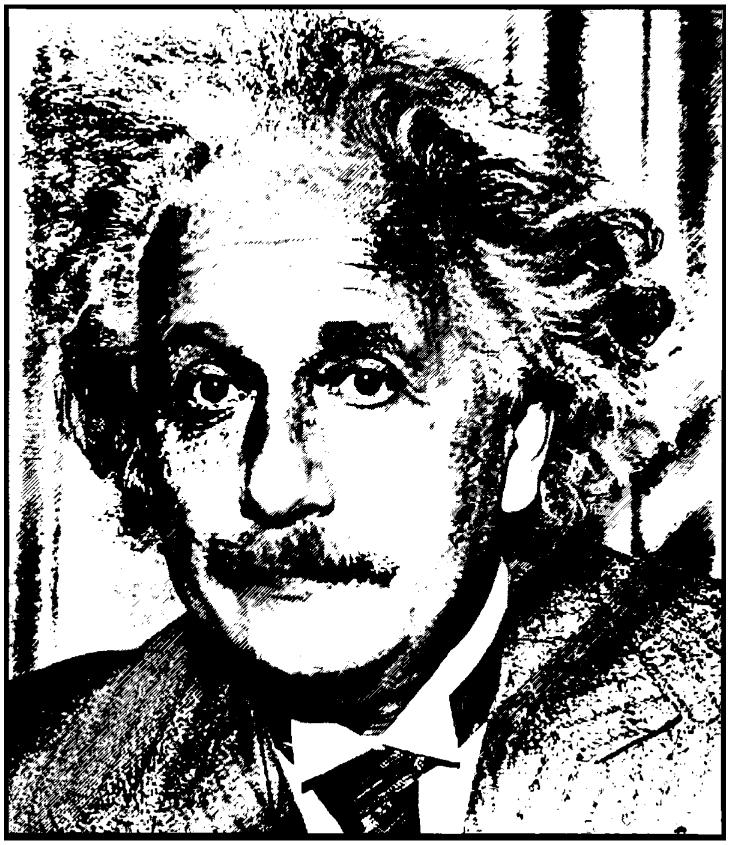


Figure 3.L1.6: (ein4L1)

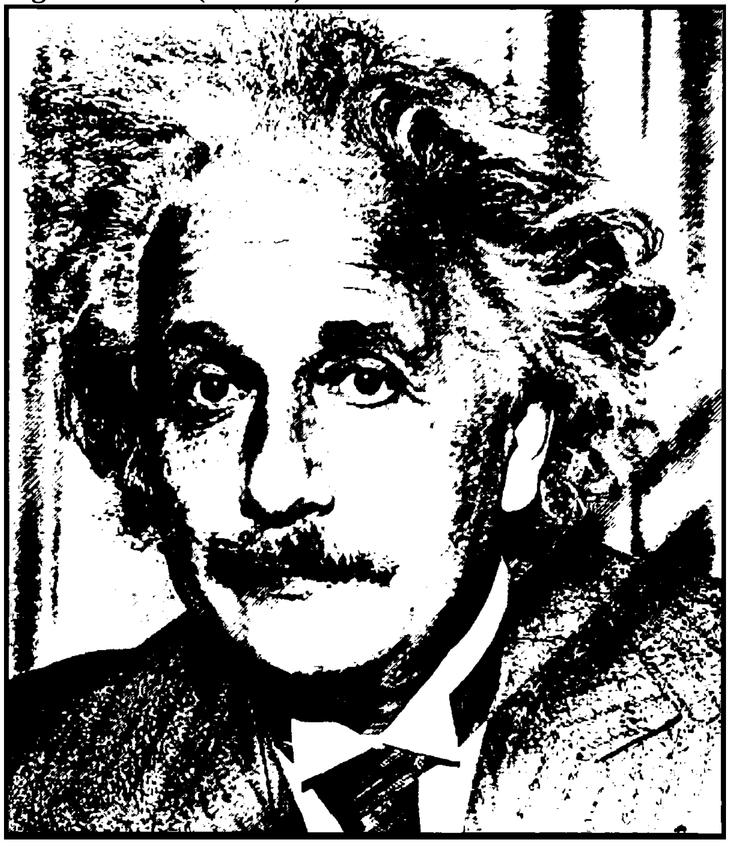


Figure 3.L1.7: (ein5L1)

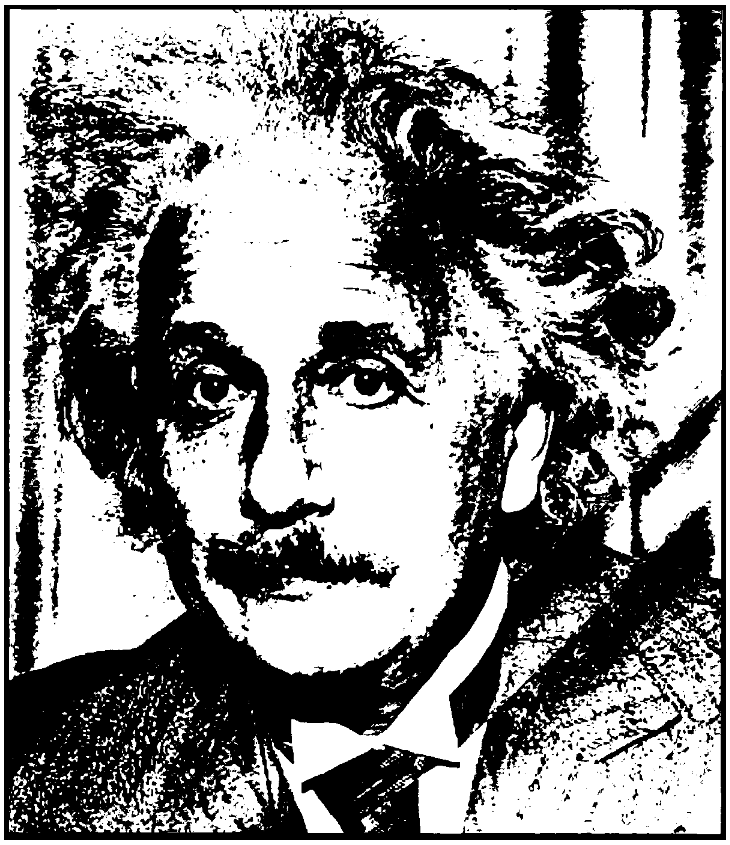


Figure 3.L1.8: (ein6L1)

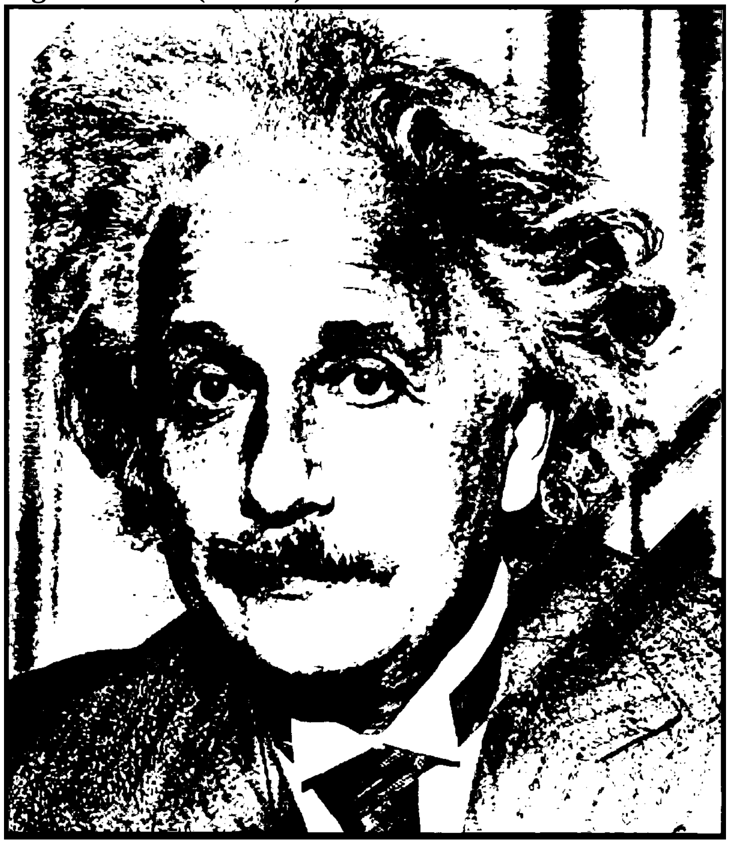


Figure 3.L1.9: (ein7L1)

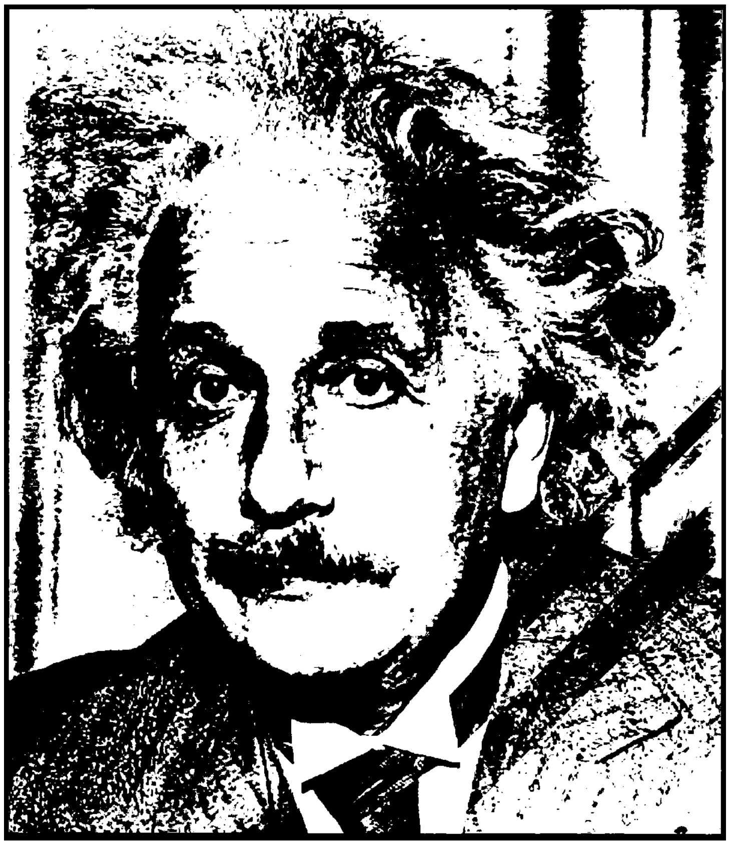


Figure 3.L1.10: (ein8L1)

3.3.3.4.7.2 Discussions

Filenames are given in parenthesis for the fig.3.L1.1-fig.3.L1.10 and fig.3.L1.4.1. Fig.3.L1.1 and fig.3.L1.2 look like engraved images. Fig.3.L1.3 contains more image details. Image details are slowly decreased from fig.3.L1.4 to fig.3.L1.7 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L1.7 to fig.3.L1.10 the decrease in image details is almost constant. Fig.3.L1.4.1 is a processed grayscale ramp (having gray values from 0-255. The grayscale ramp is generated using Program A.1 and to generate outside black border of the ramp Program A.2 is used). There is a sharp decrease is image details from fig.3.L1.4 to fig.3.L1.5 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which are more evident in fig.3.L1.3 and fig.3.L1.4. In the images left edges are more prominent which is more evident in fig.3.L1.3.

3.3.3.4.8 LTRB: LTDAVFTAC

3.3.3.4.8.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: LTDAVFTAC option generates the following pictures (fig.3.L2.1-fig.3.L2.8) as output.

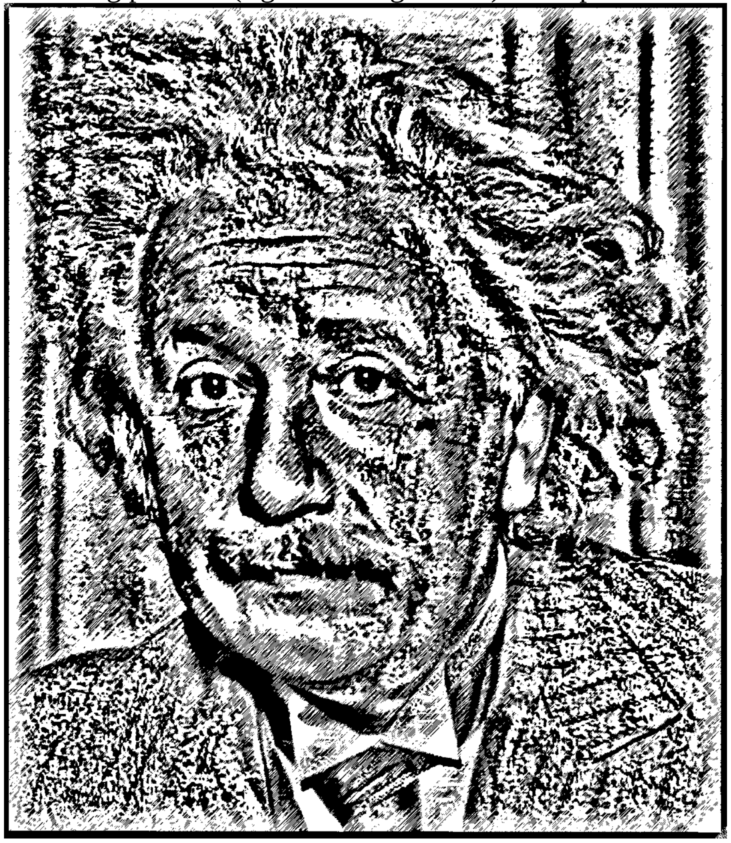


Figure 3.L2.1: (ein1L2)

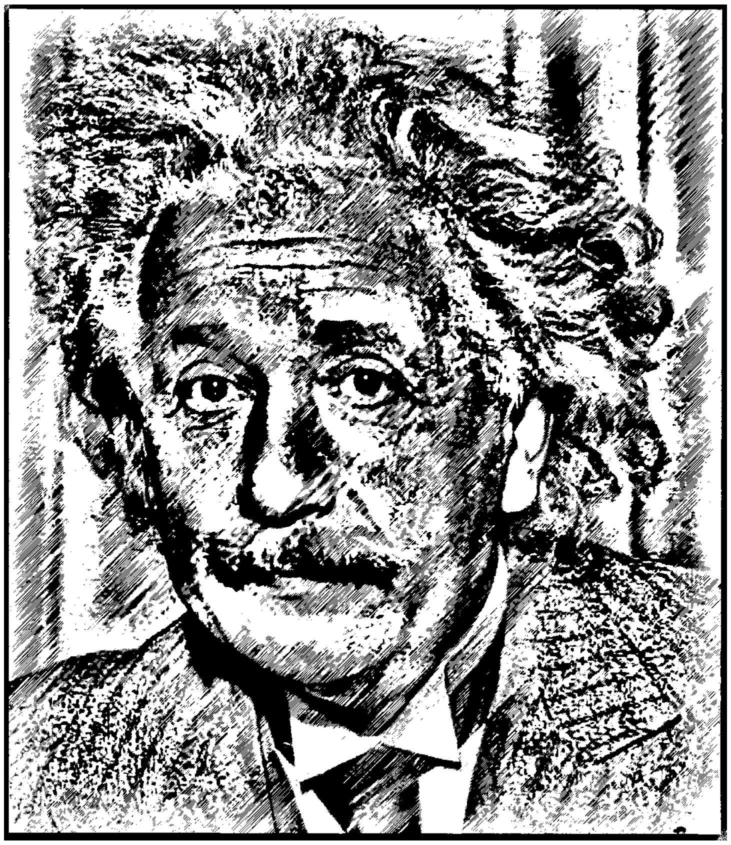


Figure 3.L2.2: (ein2L2)

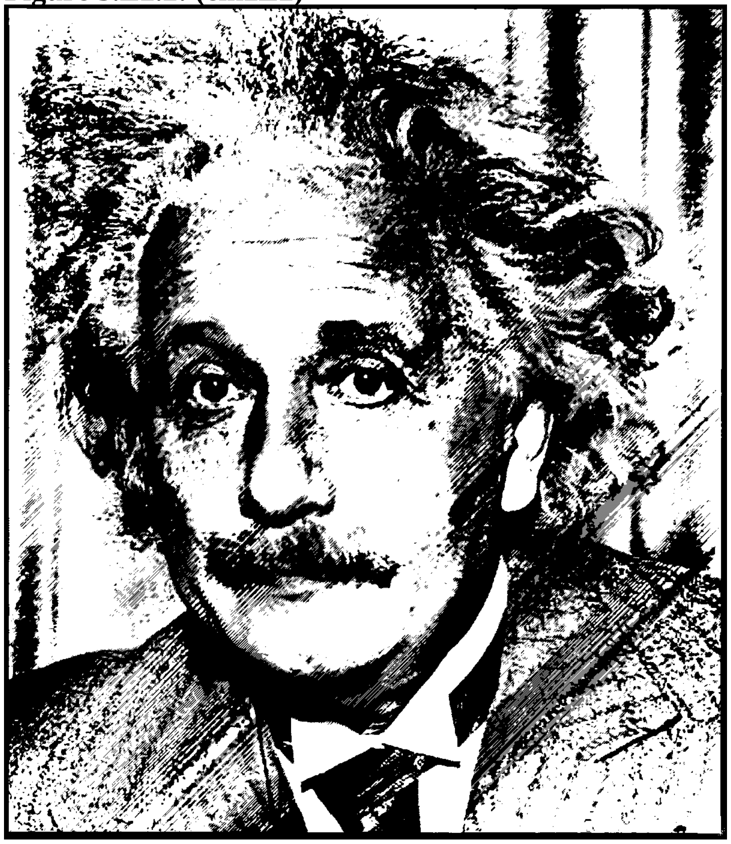


Figure 3.L2.3: (ein3L2)

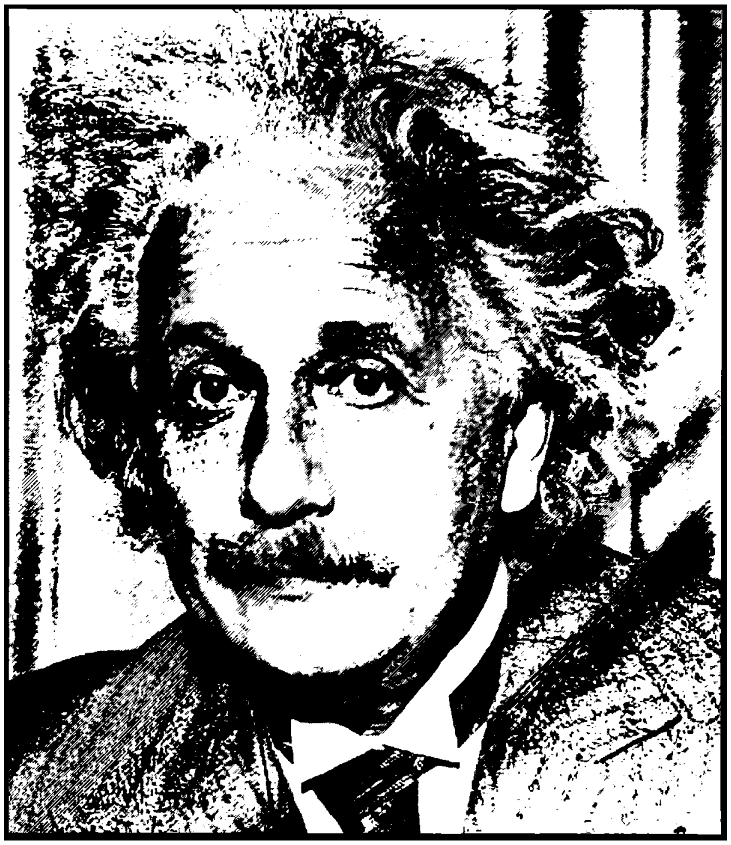


Figure 3.L2.4: (ein4L2)

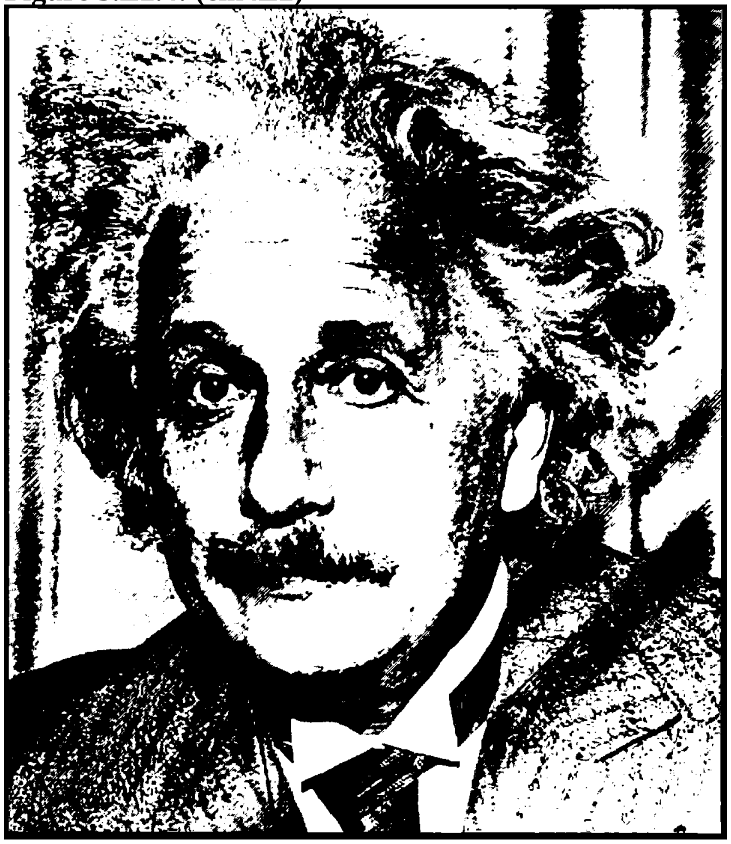


Figure 3.L2.5: (ein5L2)

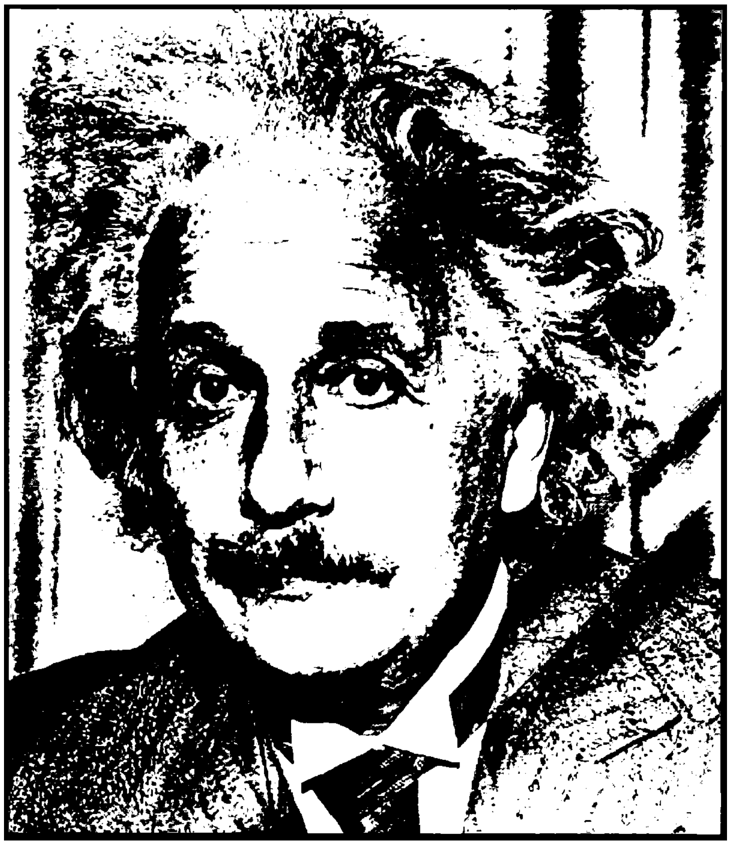


Figure 3.L2.6: (ein6L2)

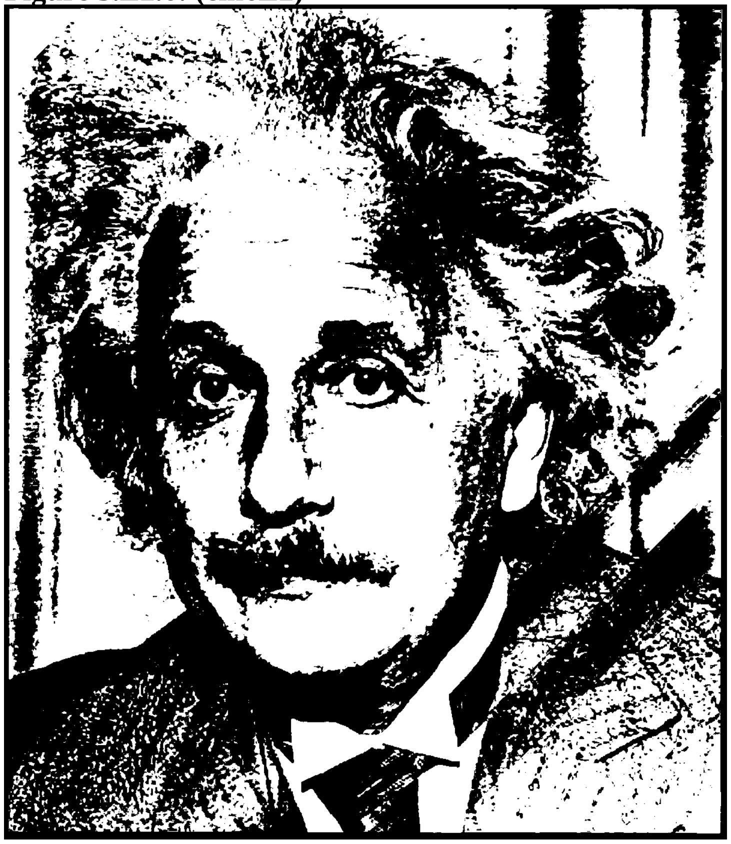


Figure 3.L2.7: (ein7L2)

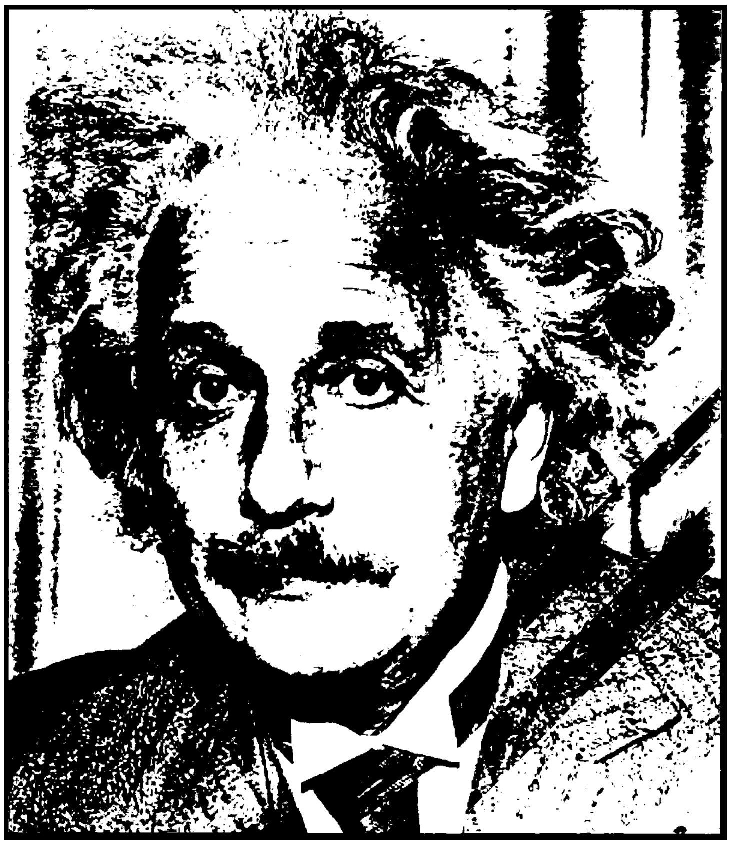


Figure 3.L2.8: (ein8L2)

3.3.3.4.8.2 Discussions

Filenames are given in parenthesis for the fig.3.L2.1-fig.3.L2.8. Fig.3.L2.1 contains more image details. Image details are slowly decreased from fig.3.L2.2 to fig.3.L2.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L2.5 to fig.3.L2.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L2.2 to fig.3.L2.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L2.1 and fig.3.L2.2. In the images left edges are more prominent which is more evident in fig.3.L2.1.

3.3.3.4.9 LTRB: LTDVLB2RT

3.3.3.4.9.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: LTDVLB2RT option generates the following pictures (fig.3.L3.1-fig.3.L3.8) as output.

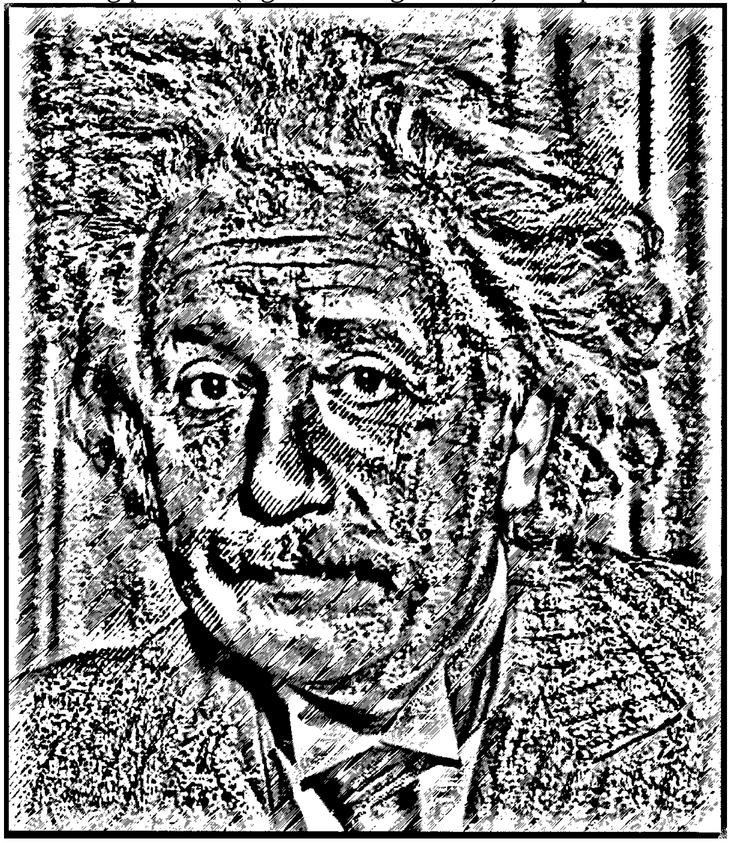


Figure 3.L3.1: (ein1L3)

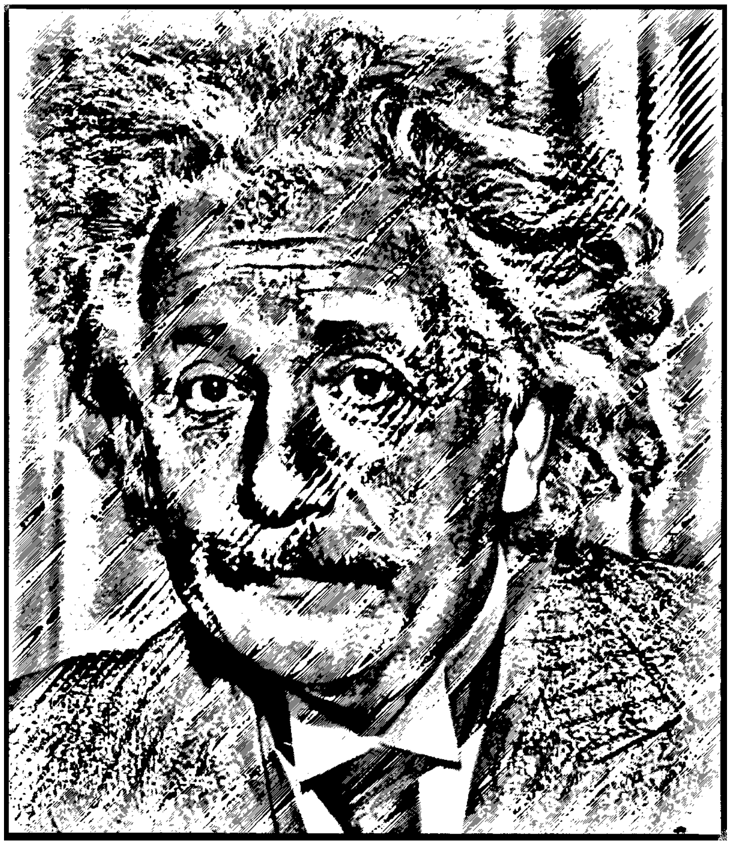


Figure 3.L3.2: (ein2L3)

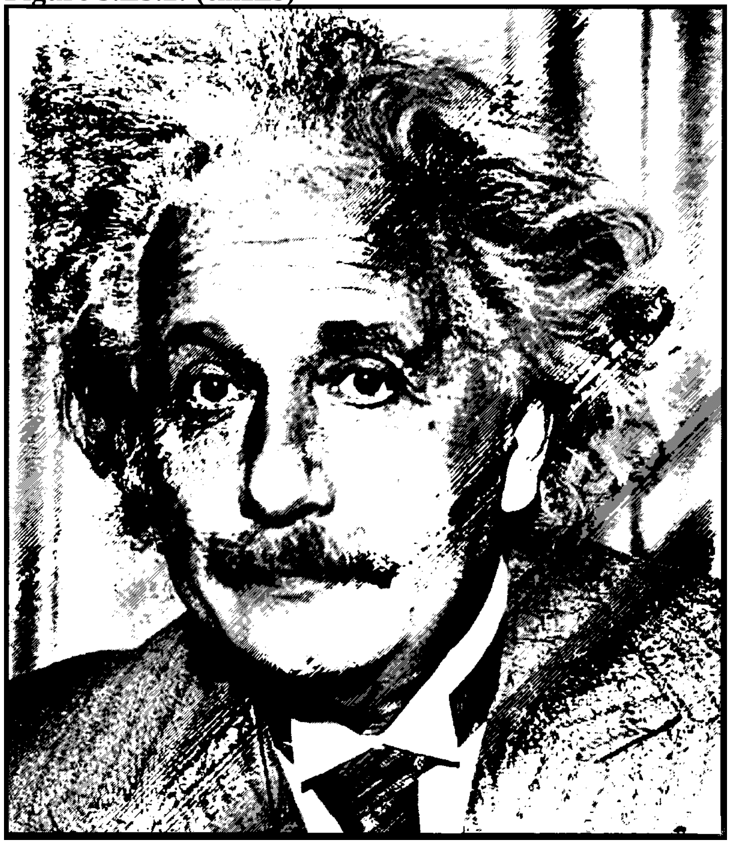


Figure 3.L3.3: (ein3L3)

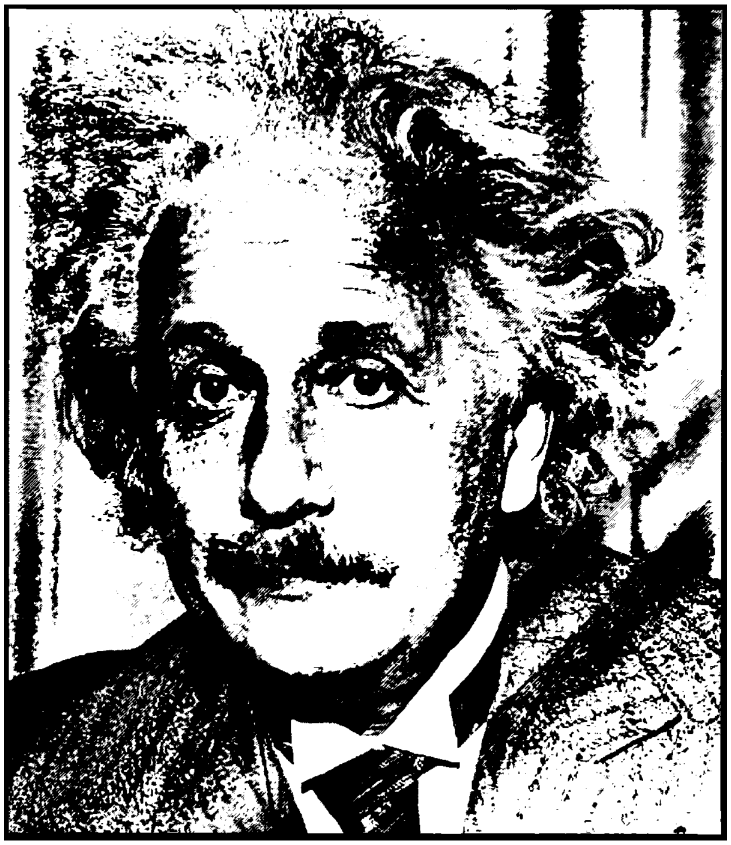


Figure 3.L3.4: (ein4L3)

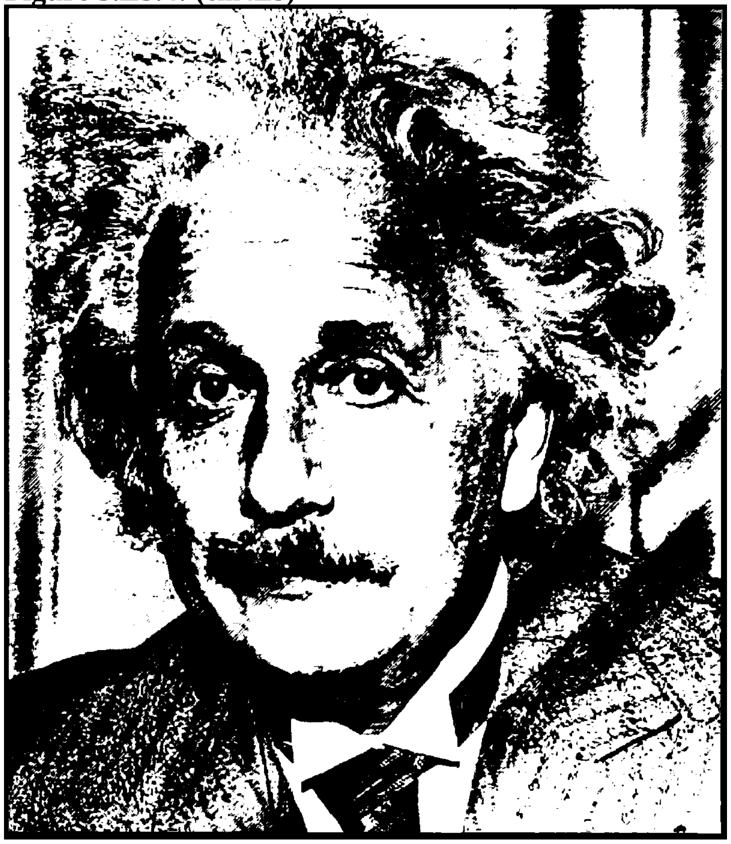


Figure 3.L3.5: (ein5L3)

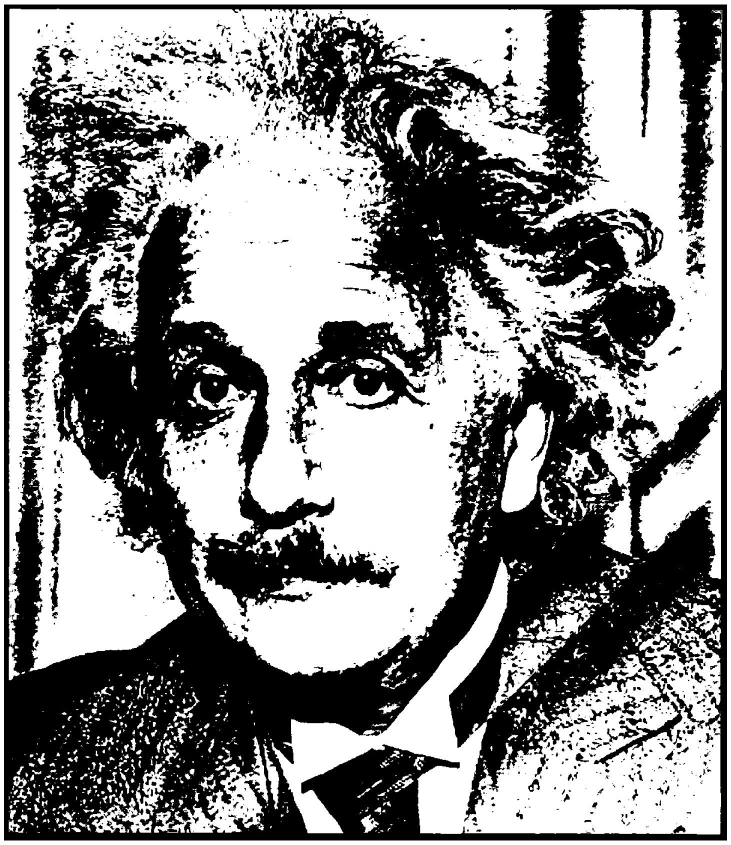


Figure 3.L3.6: (ein6L3)

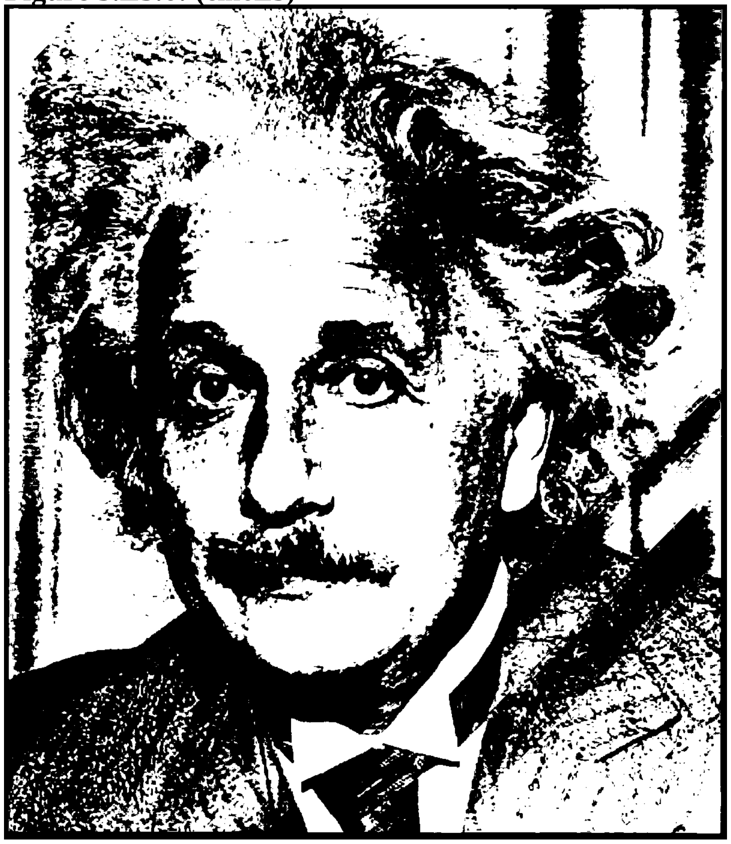


Figure 3.L3.7: (ein7L3)

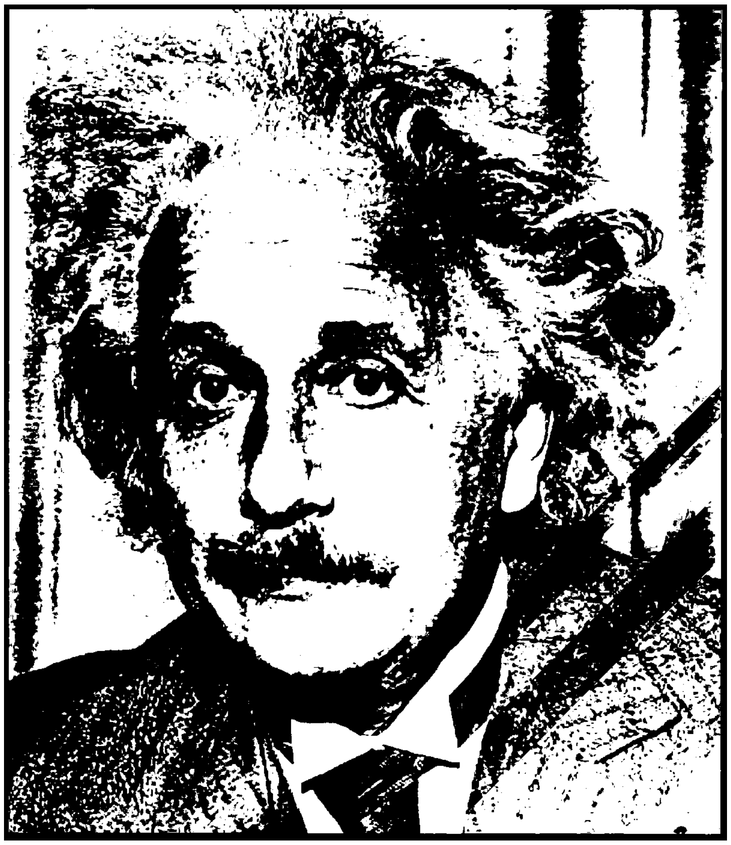


Figure 3.L3.8: (ein8L3)

3.3.3.4.9.2 Discussions

Filenames are given in parenthesis for the fig.3.L3.1-fig.3.L3.8. Fig.3.L3.1 contains more image details. Image details are slowly decreased from fig.3.L3.2 to fig.3.L3.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L3.5 to fig.3.L3.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L3.2 to fig.3.L3.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L3.1 and fig.3.L3.2. In the images left edges are more prominent which is more evident in fig.3.L3.1

3.3.3.4.10 LTRB: LTDVRT2LB

3.3.3.4.10.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: LTDVRT2LB option generates the following pictures (fig.3.L4.1-fig.3.L4.8) as output

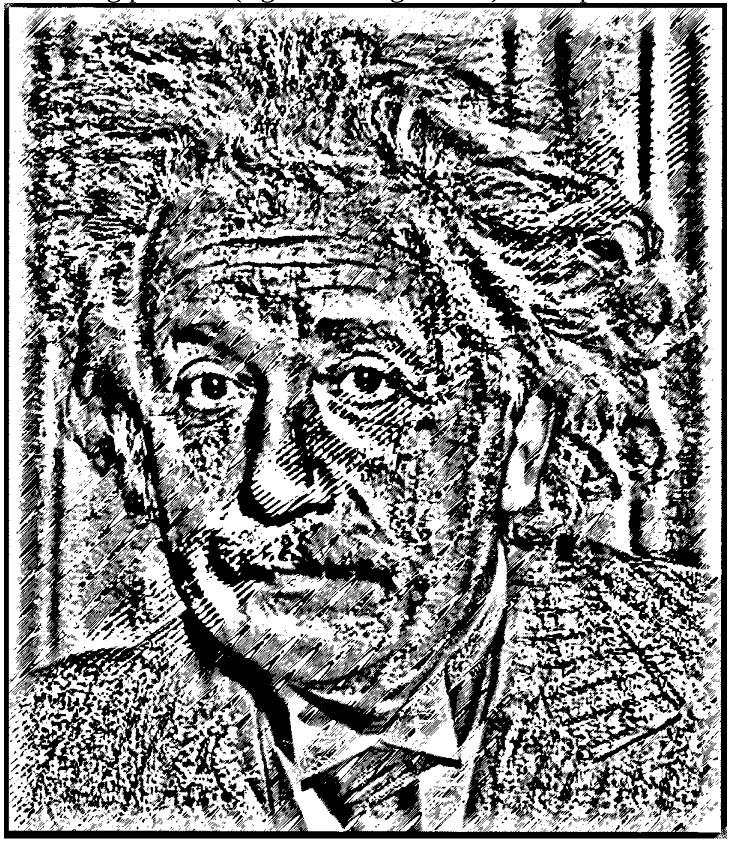


Figure 3.L4.1: (ein1L4)

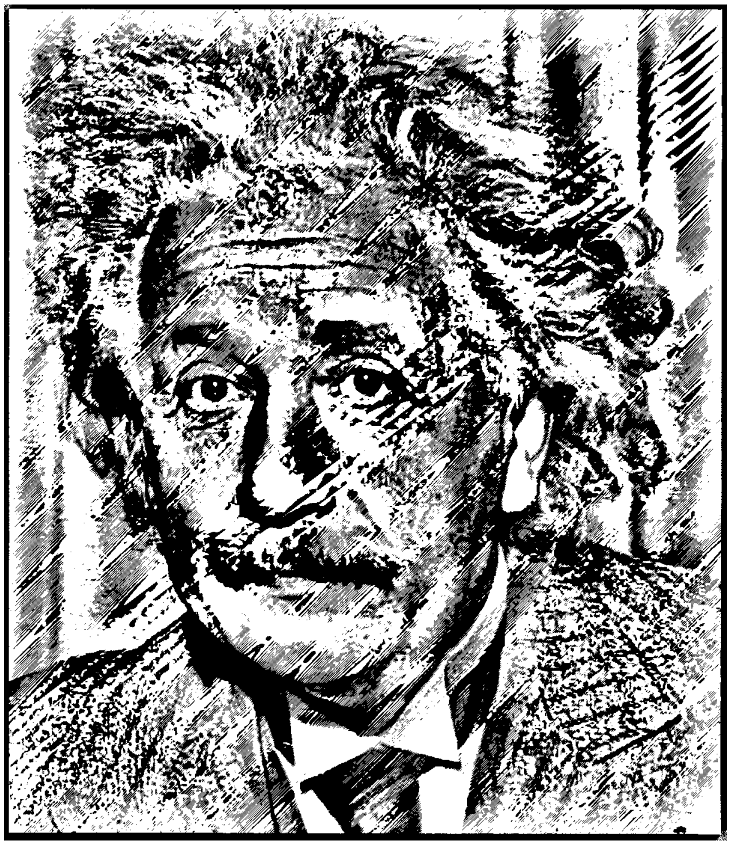


Figure 3.L4.2: (ein2L4)

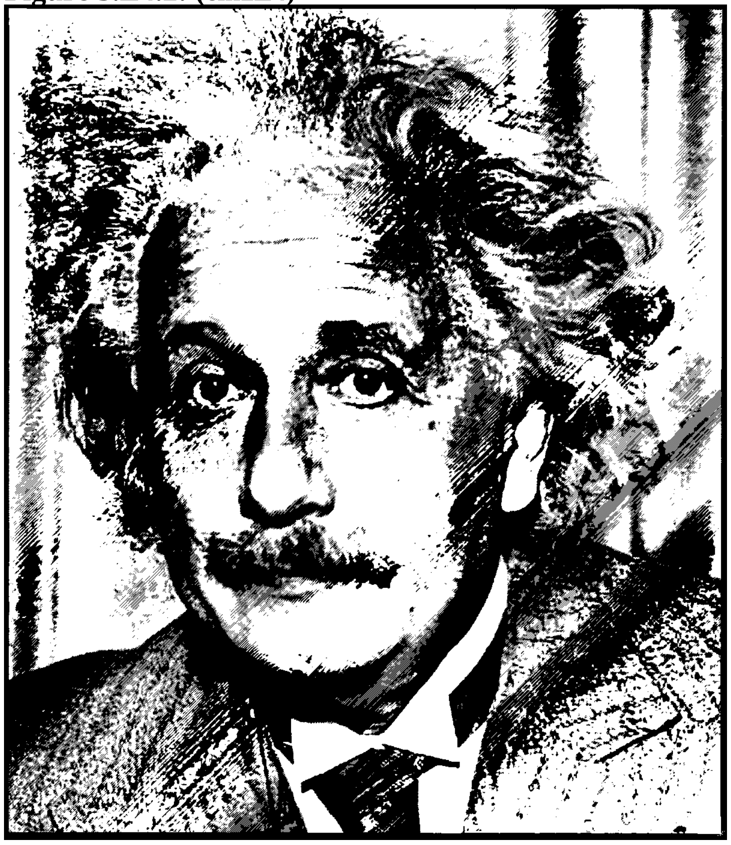


Figure 3.L4.3: (ein3L4)

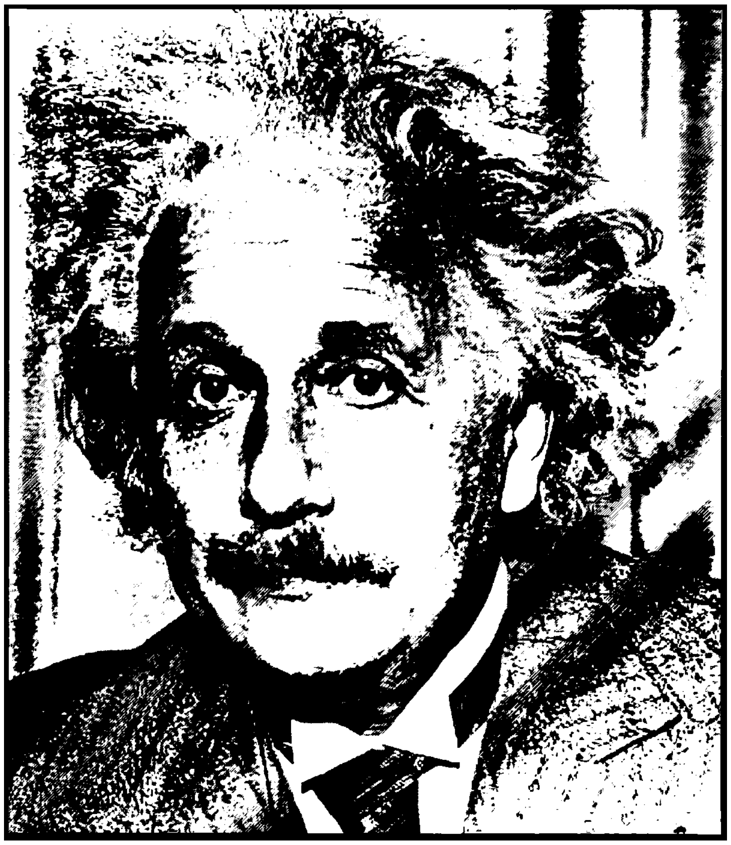


Figure 3.L4.4: (ein4L4)

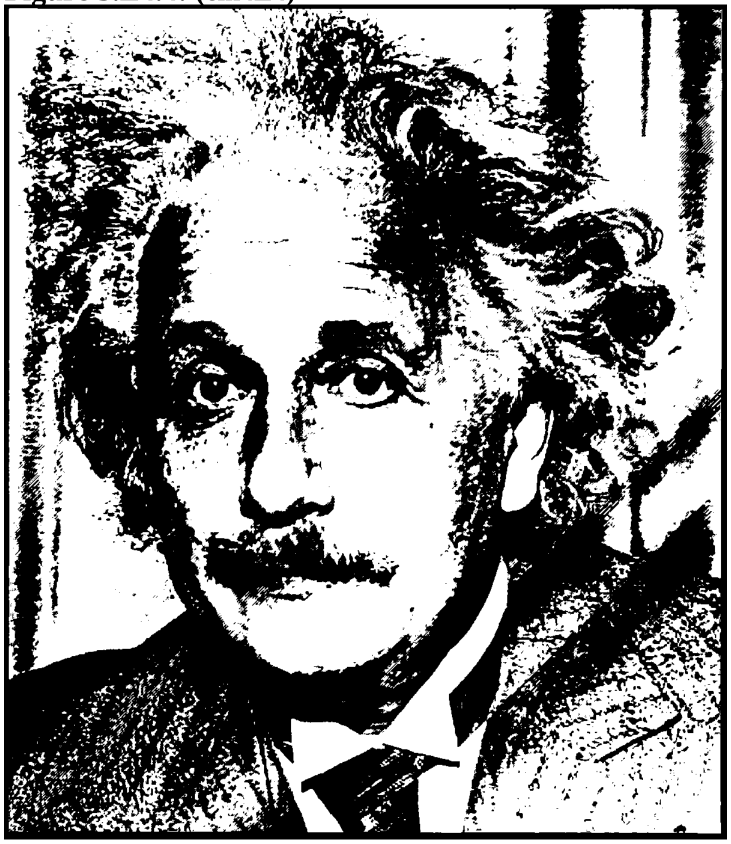


Figure 3.L4.5: (ein5L4)

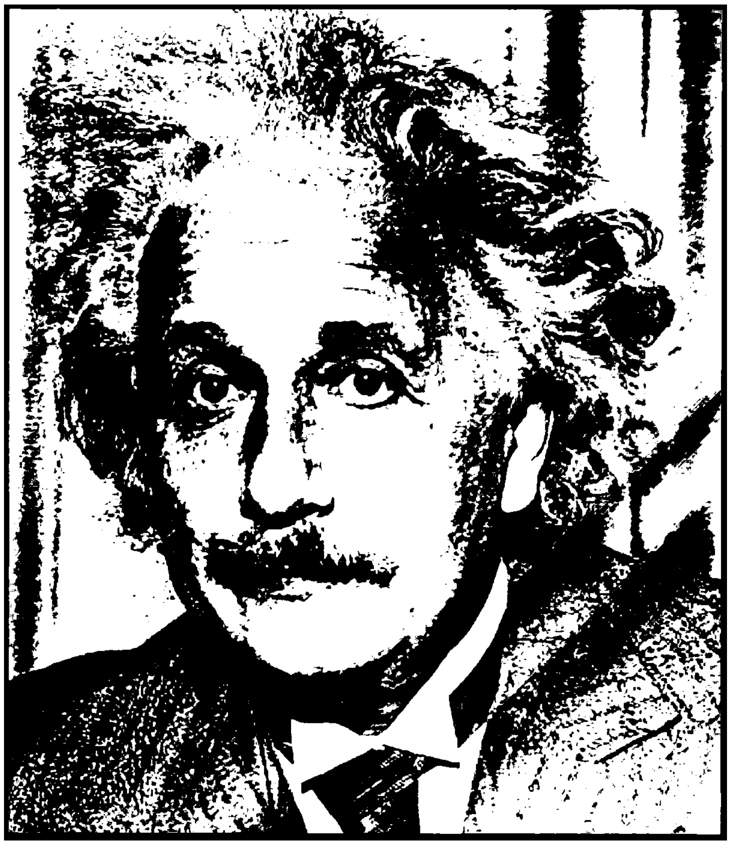


Figure 3.L4.6: (ein6L4)

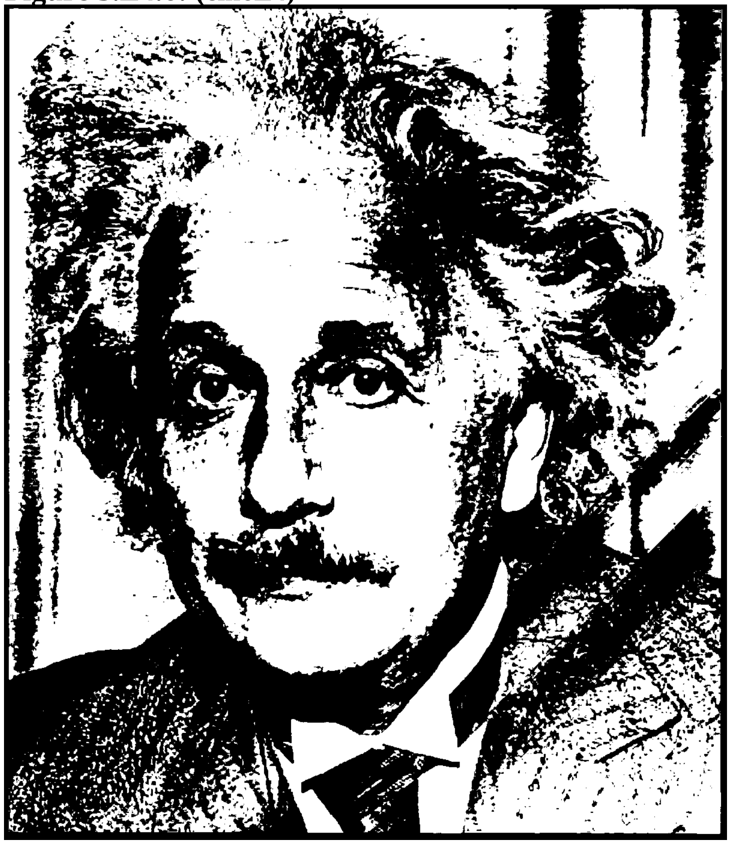


Figure 3.L4.7: (ein7L4)

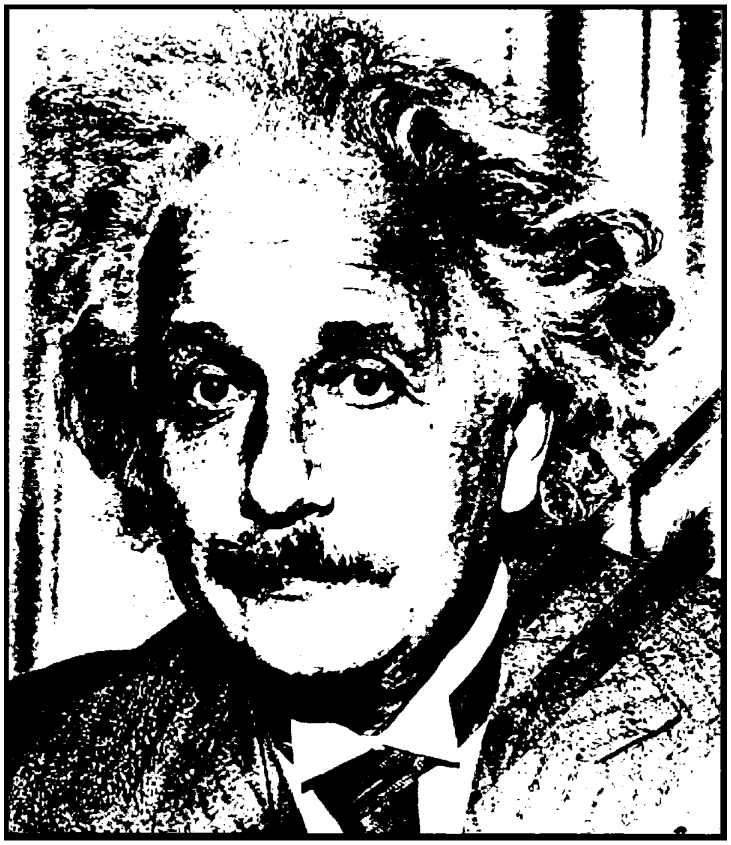


Figure 3.L4.8: (ein8L4)

3.3.3.4.10.2 Discussions

Filenames are given in parenthesis for the fig.3.L4.1-fig.3.L4.8. Fig.3.L4.1 contains more image details. Image details are slowly decreased from fig.3.L4.2 to fig.3.L4.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L4.5 to fig.3.L4.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L4.2 to fig.3.L4.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L4.1 and fig.3.L4.2. In the images left edges are more prominent which is more evident in fig.3.L4.1.

3.3.3.4.11 LTRB: RBDAVFTAC

3.3.3.4.11.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: RBDAVFTAC option generates the following pictures (fig.3.L5.1-fig.3.L5.8) as output

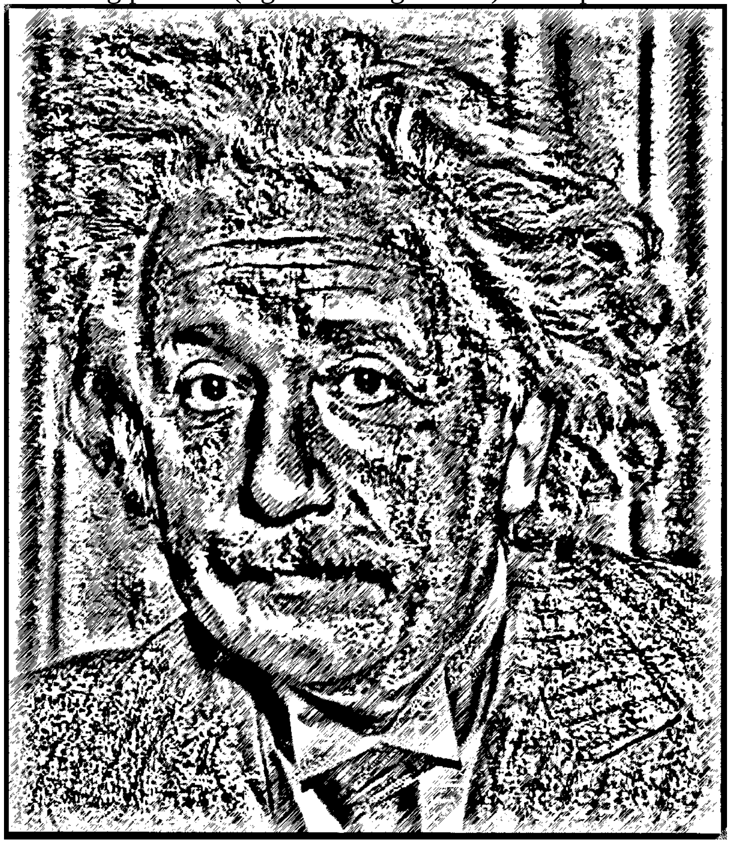


Figure 3.L5.1: (ein1L5)

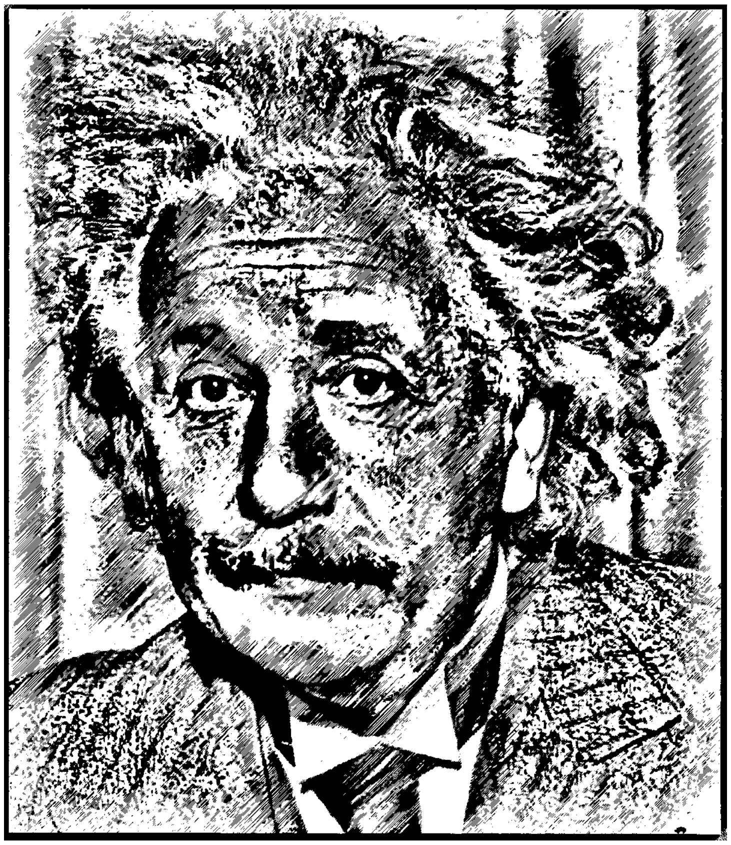


Figure 3.L5.2: (ein2L5)

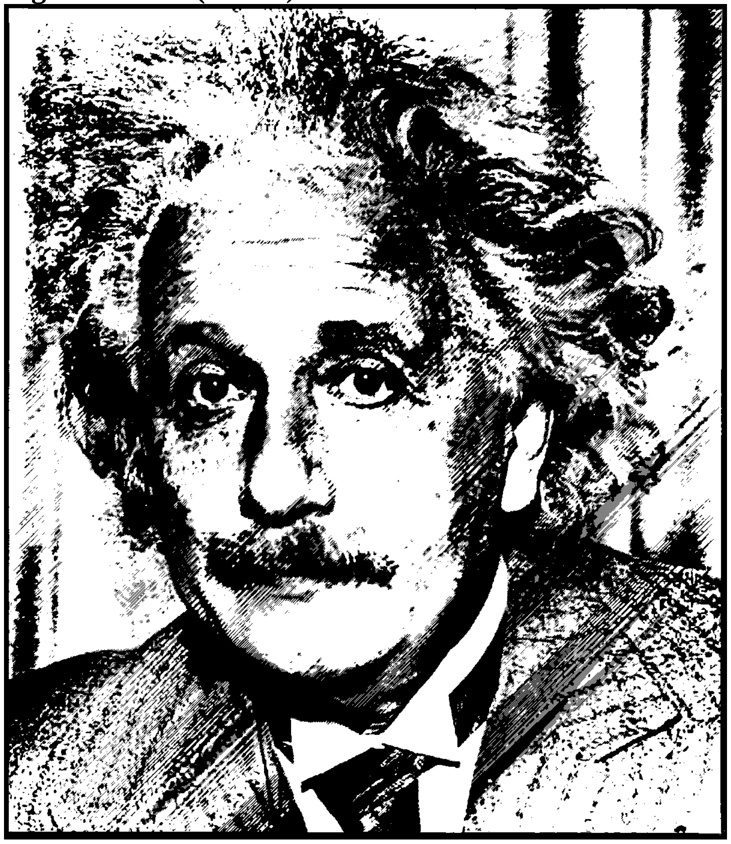


Figure 3.L5.3: (ein3L5)

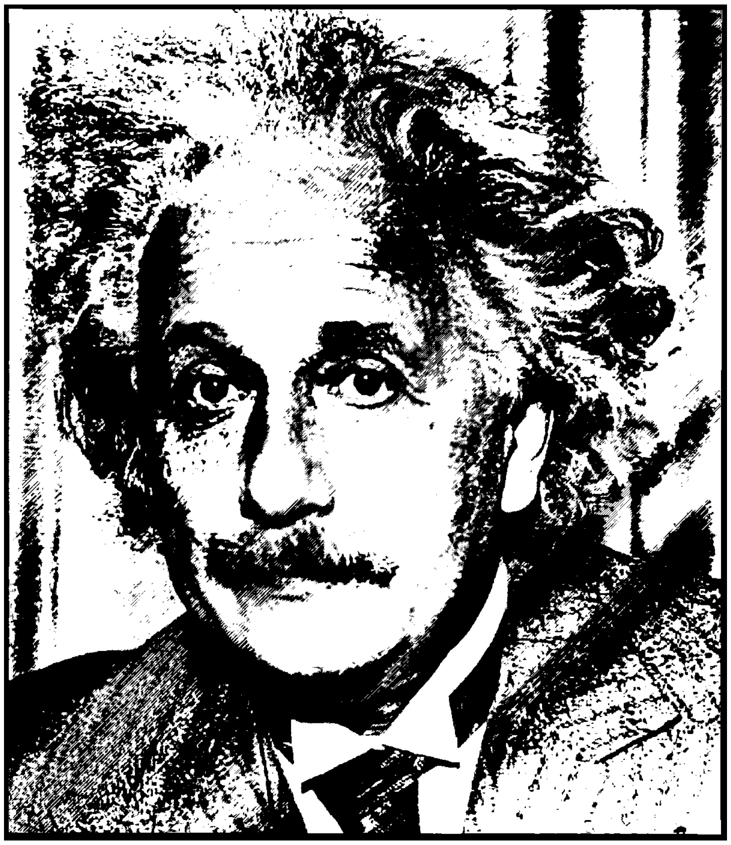


Figure 3.L5.4: (ein4L5)

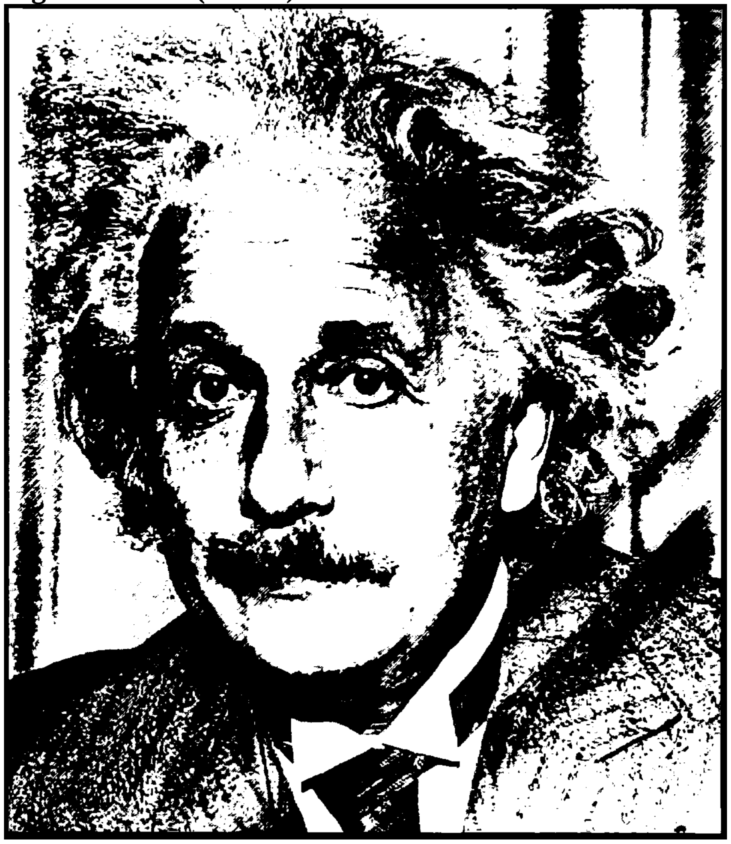


Figure 3.L5.5: (ein5L5)

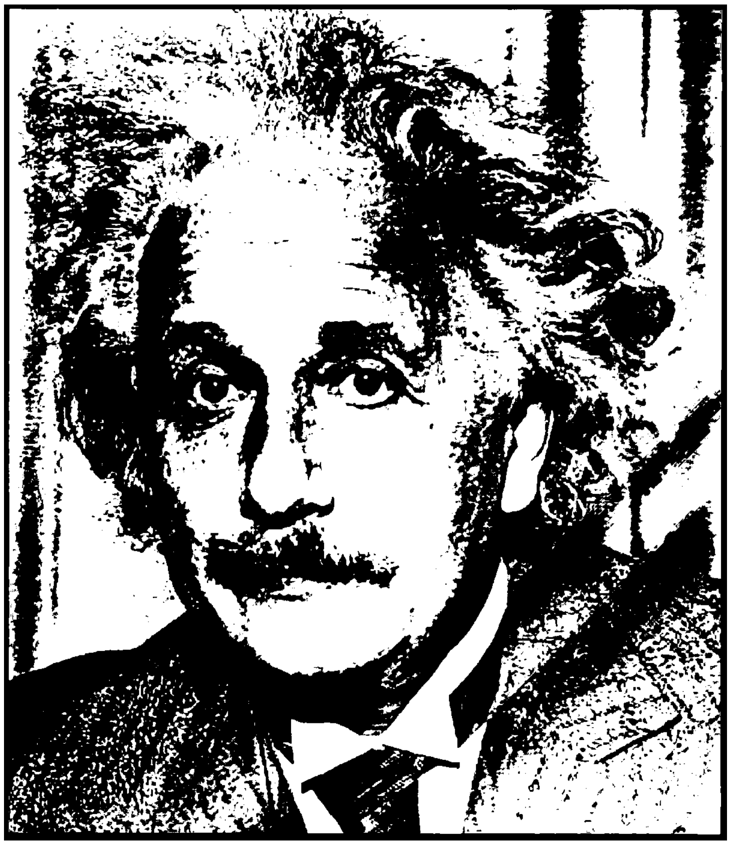


Figure 3.L5.6: (ein6L5)

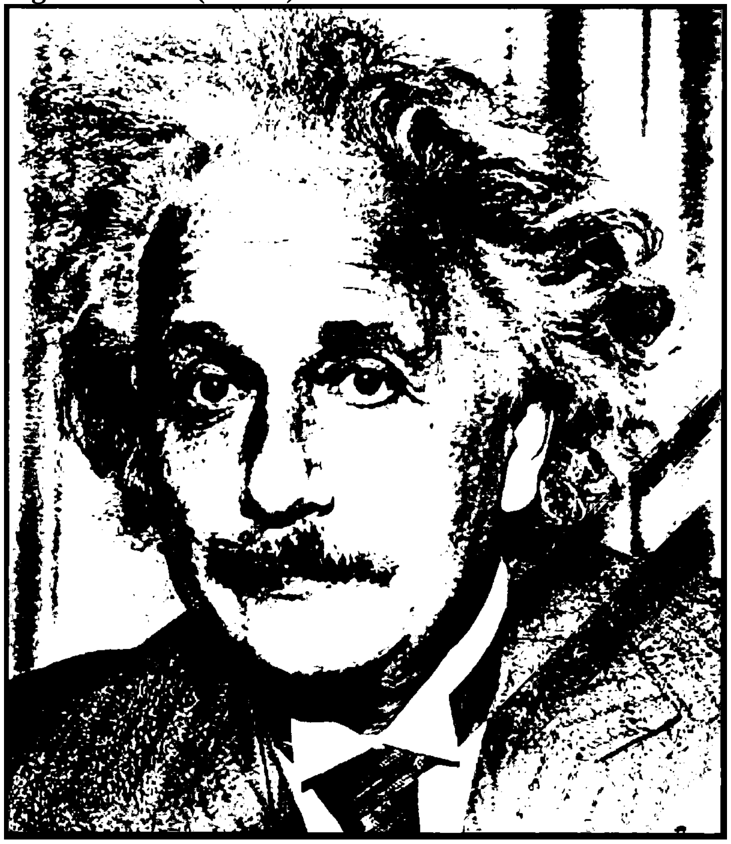


Figure 3.L5.7: (ein7L5)

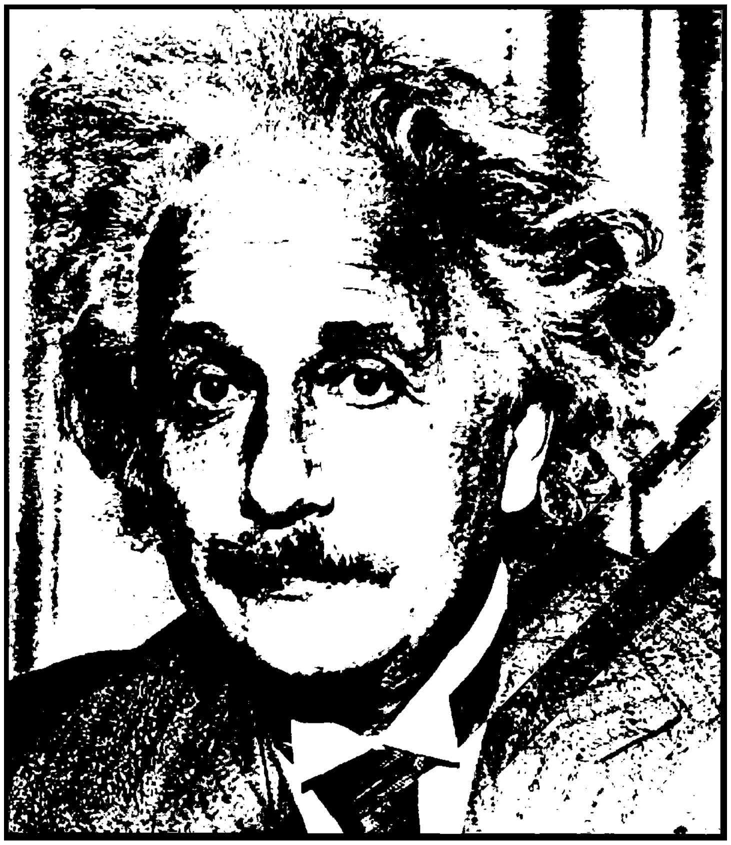


Figure 3.L5.8: (ein8L5)

3.3.3.4.11.2 Discussions

Filenames are given in parenthesis for the fig.3.L5.1-fig.3.L5.8. Fig.3.L5.1 contains more image details. Image details are slowly decreased from fig.3.L5.2 to fig.3.L5.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L5.5 to fig.3.L5.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L5.2 to fig.3.L5.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L5.1 and fig.3.L5.2. In the images left edges are more prominent which is more evident in fig.3.L5.1

3.3.3.4.12 LTRB: RBDAVFTC

3.3.3.4.12.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: RBDAVFTC option generates the following pictures (fig.3.L6.1-fig.3.L6.8) as output

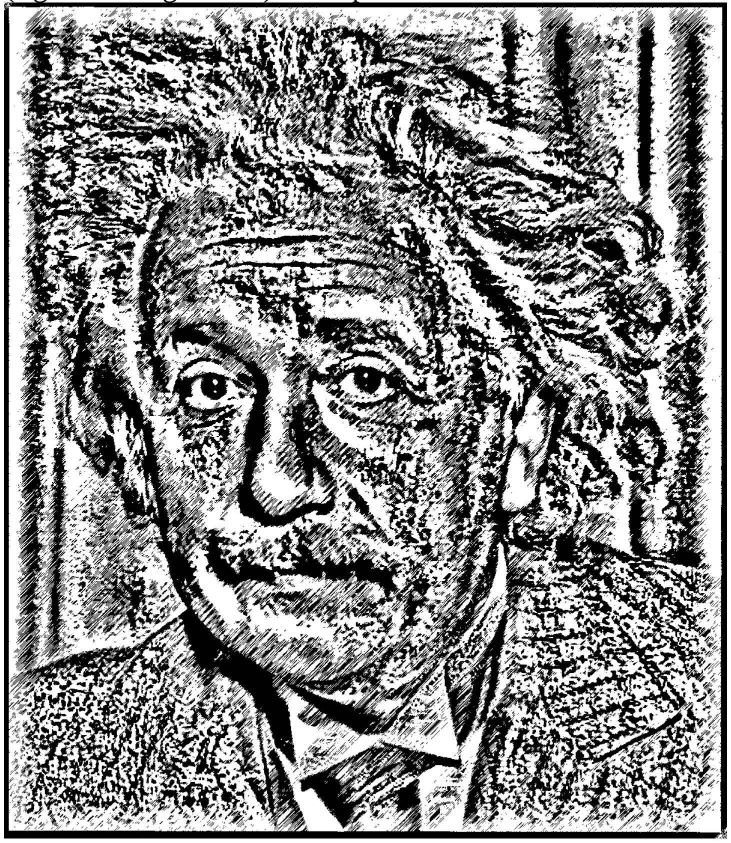


Figure 3.L6.1: (ein1L6)

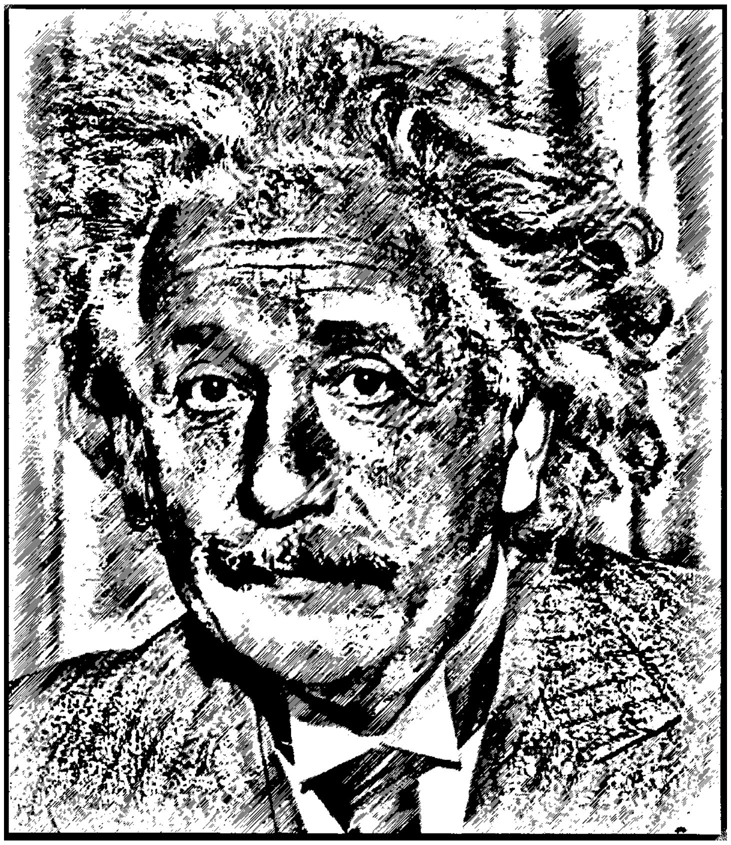


Figure 3.L6.2: (ein2L6)

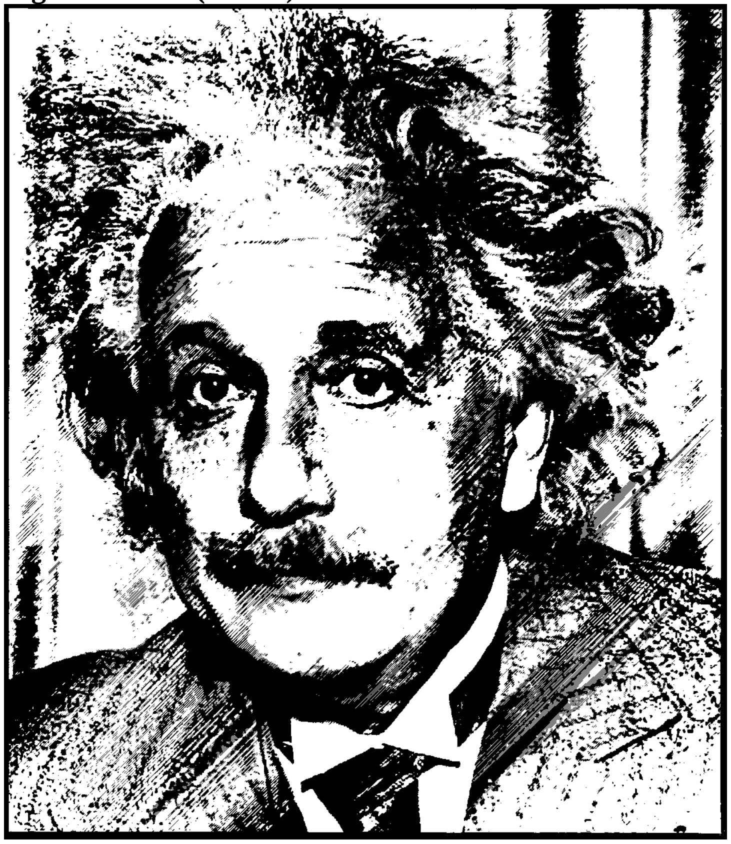


Figure 3.L6.3: (ein3L6)

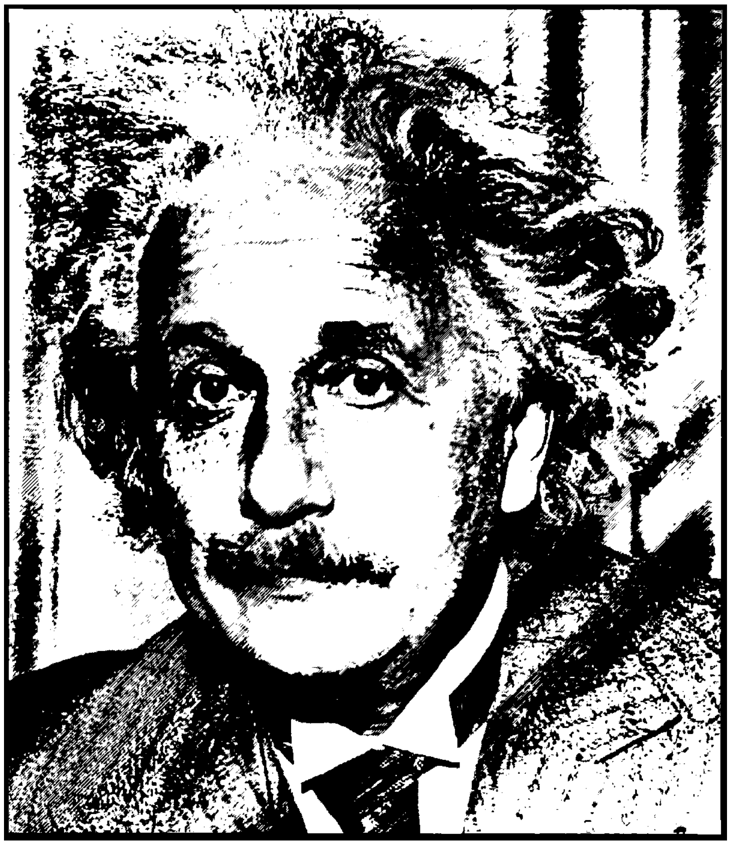


Figure 3.L6.4: (ein4L6)

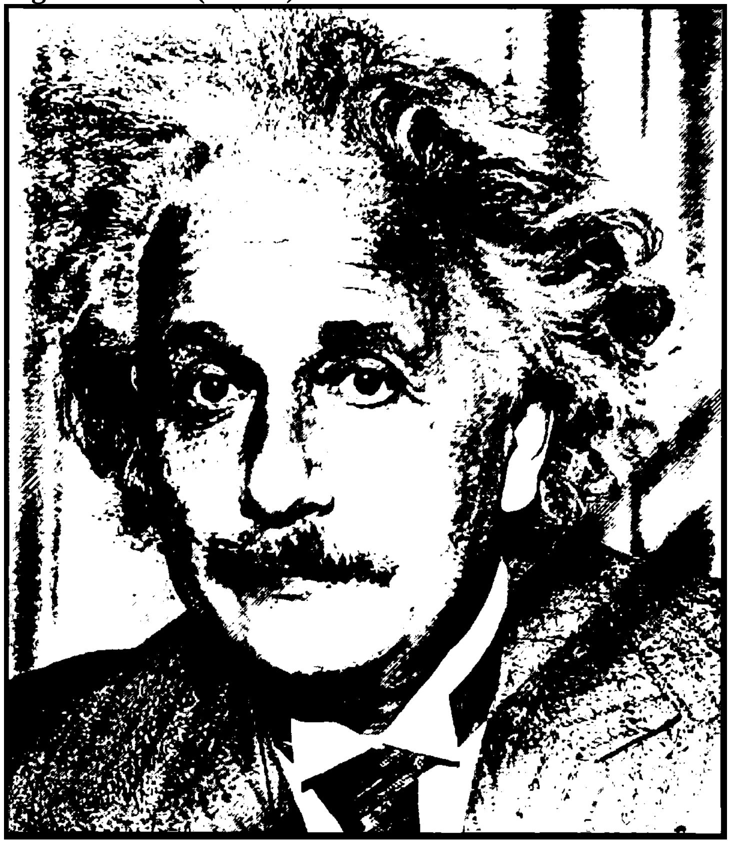


Figure 3.L6.5: (ein5L6)

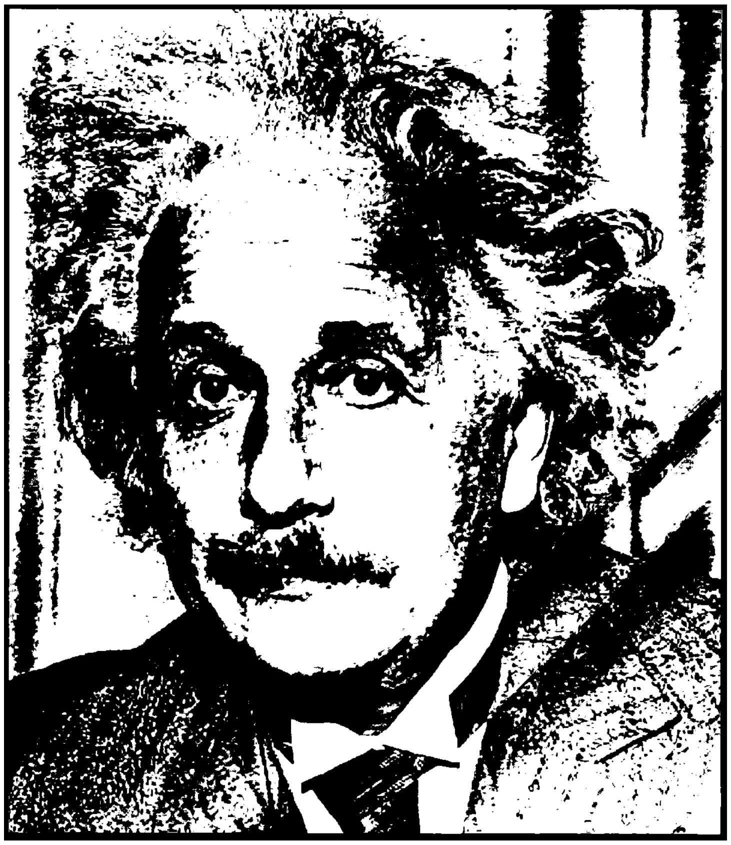


Figure 3.L6.6: (ein6L6)

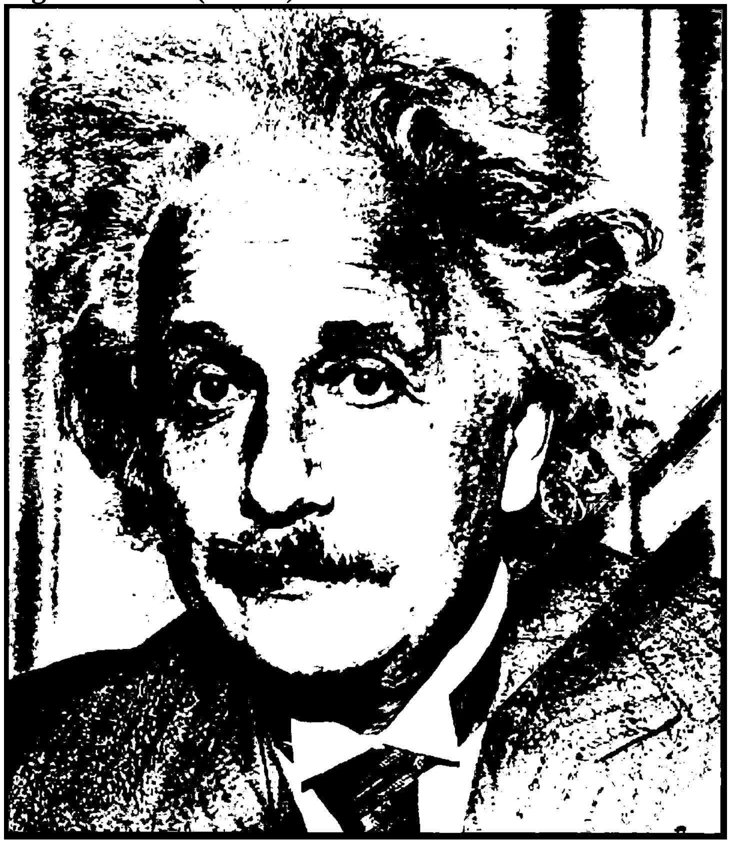


Figure 3.L6.7: (ein7L6)

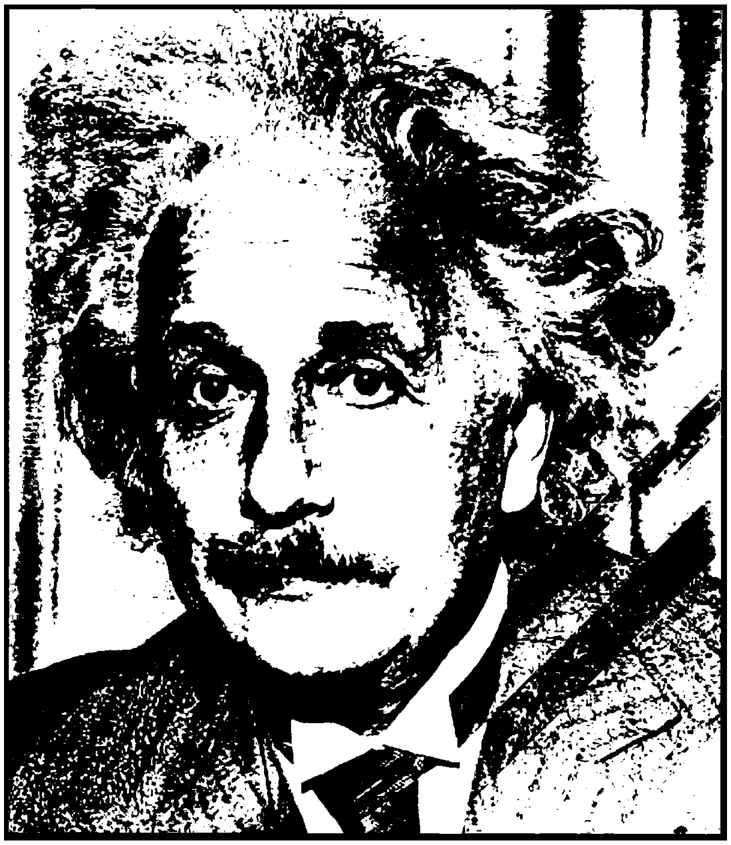


Figure 3.L6.8: (ein8L6)

3.3.3.4.12.2 Discussions

Filenames are given in parenthesis for the fig.3.L6.1-fig.3.L6.8. Fig.3.L6.1 contains more image details. Image details are slowly decreased from fig.3.L6.2 to fig.3.L6.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L6.5 to fig.3.L6.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L6.2 to fig.3.L6.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L6.1 and fig.3.L6.2. In the images left edges are more prominent which is more evident in fig.3.L6.1

3.3.3.4.13 LTRB: RBDVRT2LB

3.3.3.4.13.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: RBDVRT2LB option generates the following pictures (fig.3.L7.1-fig.3.L7.8) as output

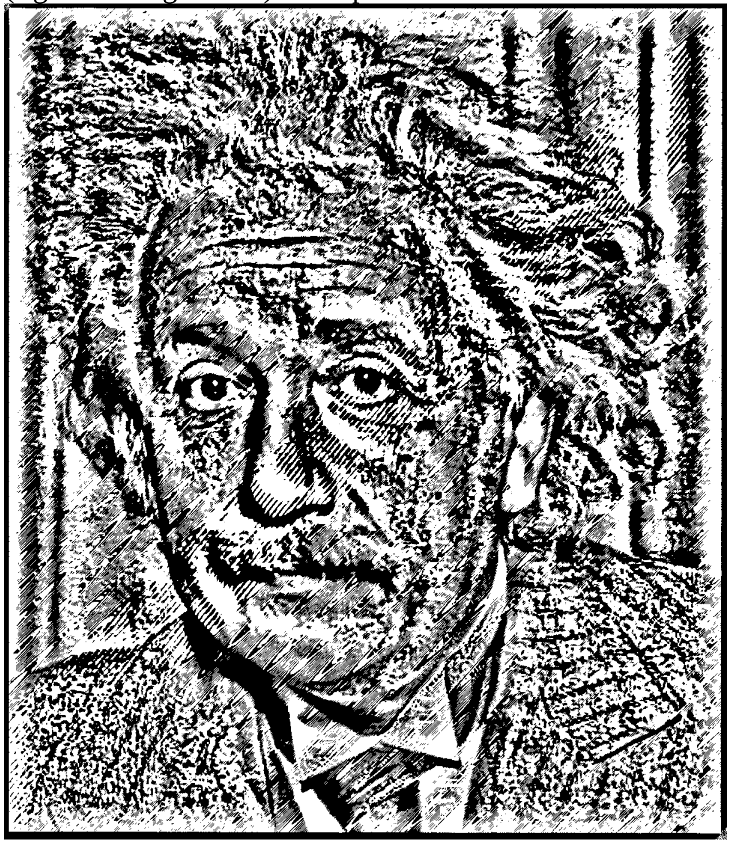


Figure 3.L7.1: (ein1L7)

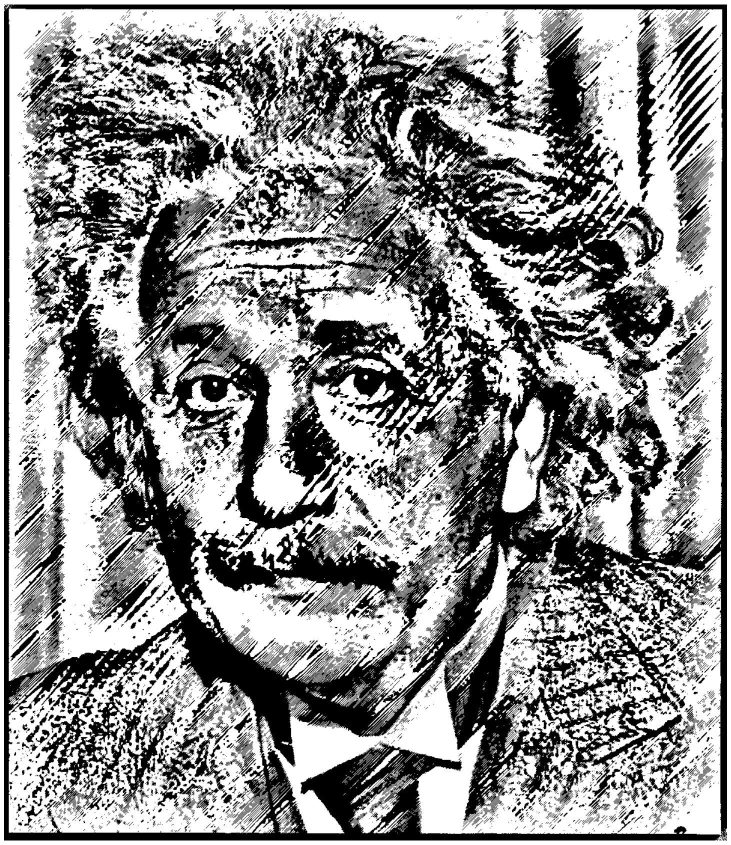


Figure 3.L7.2: (ein2L7)

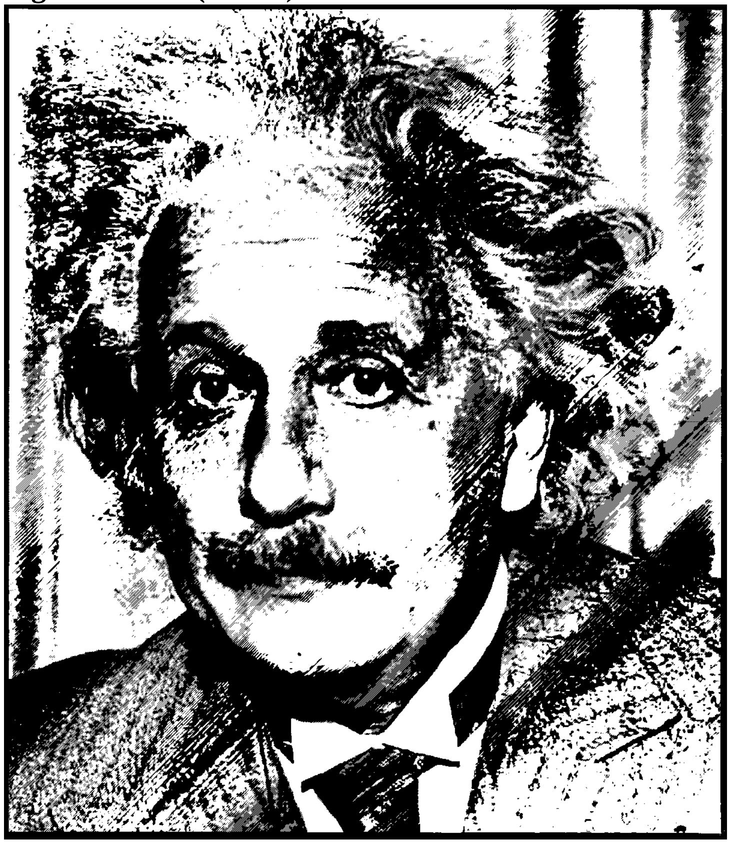


Figure 3.L7.3: (ein3L7)

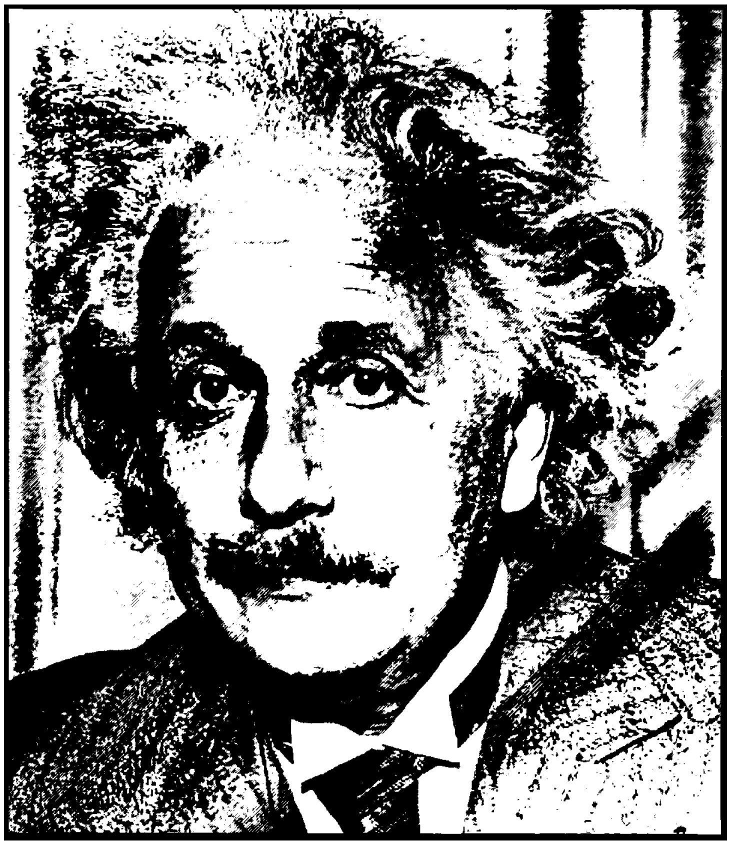


Figure 3.L7.4: (ein4L7)

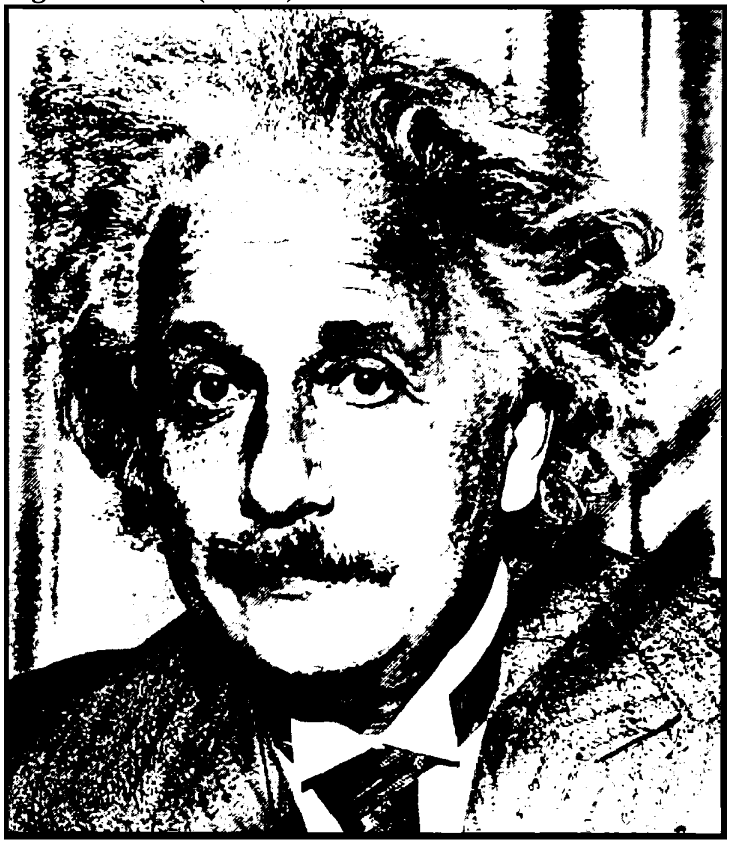


Figure 3.L7.5: (ein5L7)

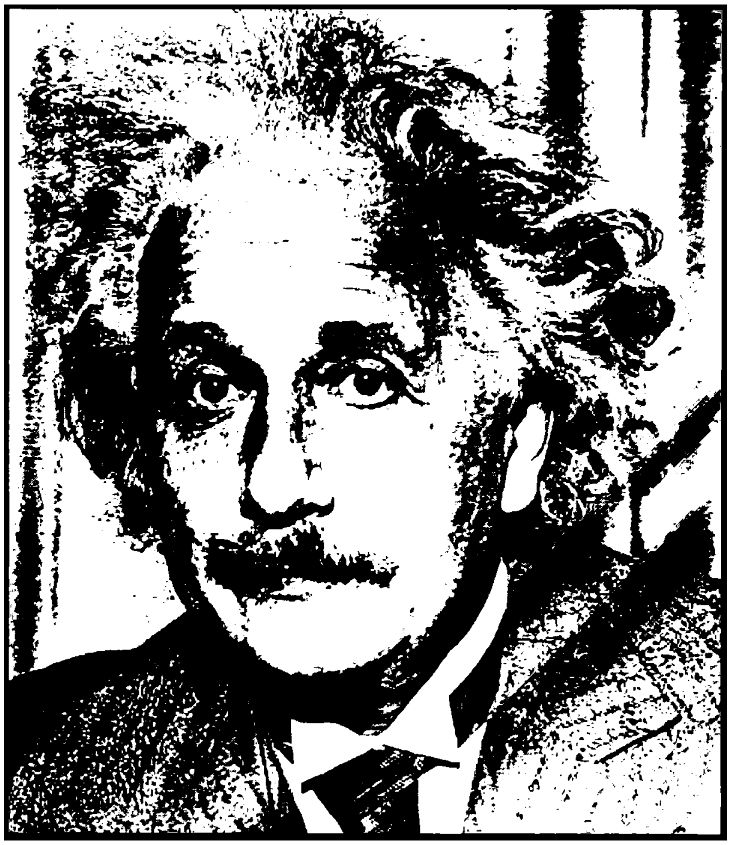


Figure 3.L7.6: (ein6L7)

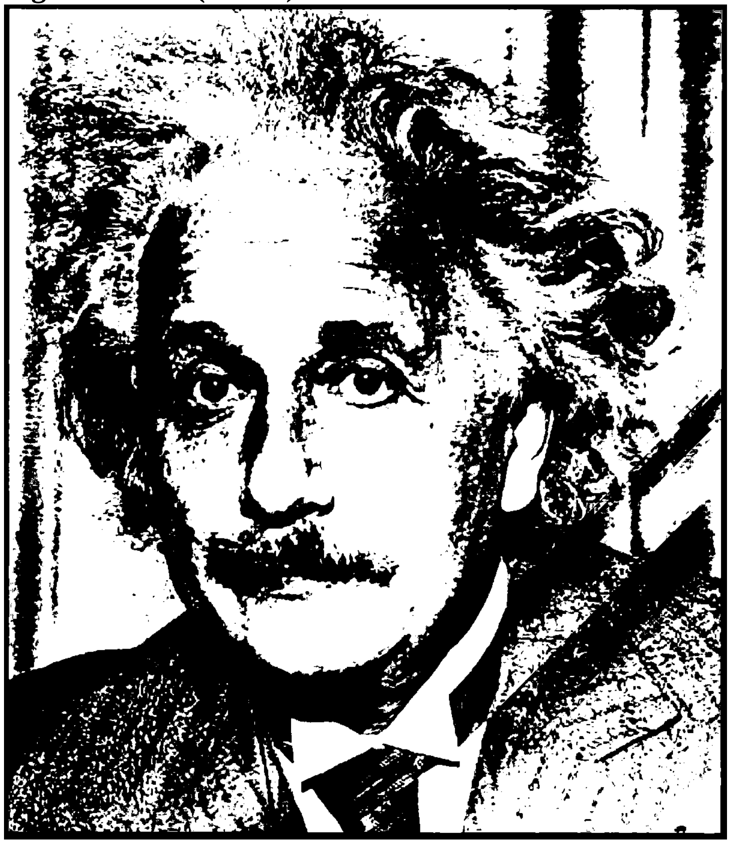


Figure 3.L7.7: (ein7L7)

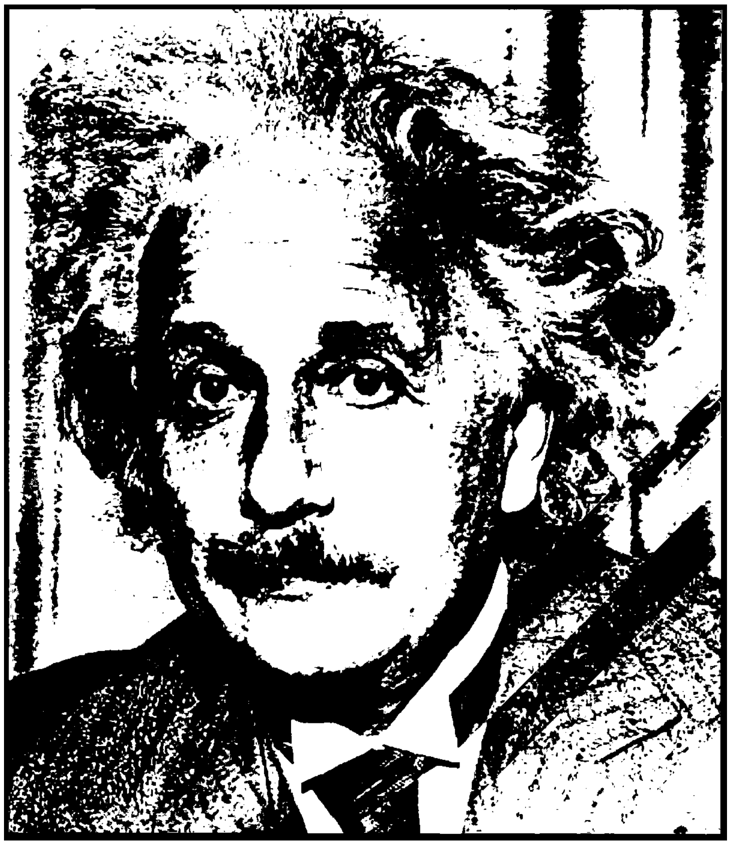


Figure 3.L7.8: (ein8L7)

3.3.3.4.13.2 Discussions

Filenames are given in parenthesis for the fig.3.L7.1-fig.3.L7.8. Fig.3.L7.1 contains more image details. Image details are slowly decreased from fig.3.L7.2 to fig.3.L7.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L7.5 to fig.3.L7.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L7.2 to fig.3.L7.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L7.1 and fig.3.L7.2. In the images left edges are more prominent which is more evident in fig.3.L7.1

3.3.3.4.14 LTRB: RBDVLB2RT

3.3.3.4.14.1 Output Images

The processing of fig.3.B2 diagonally with LTRB: RBDVLB2RT option generates the following pictures (fig.3.L8.1-fig.3.L8.8) as output

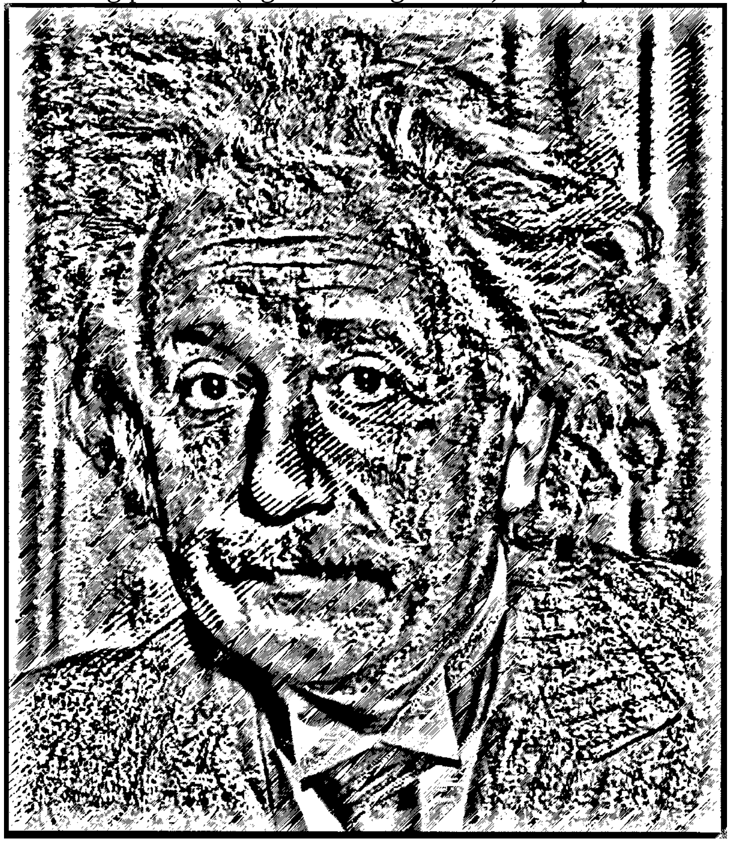


Figure 3.L8.1: (ein1L8)

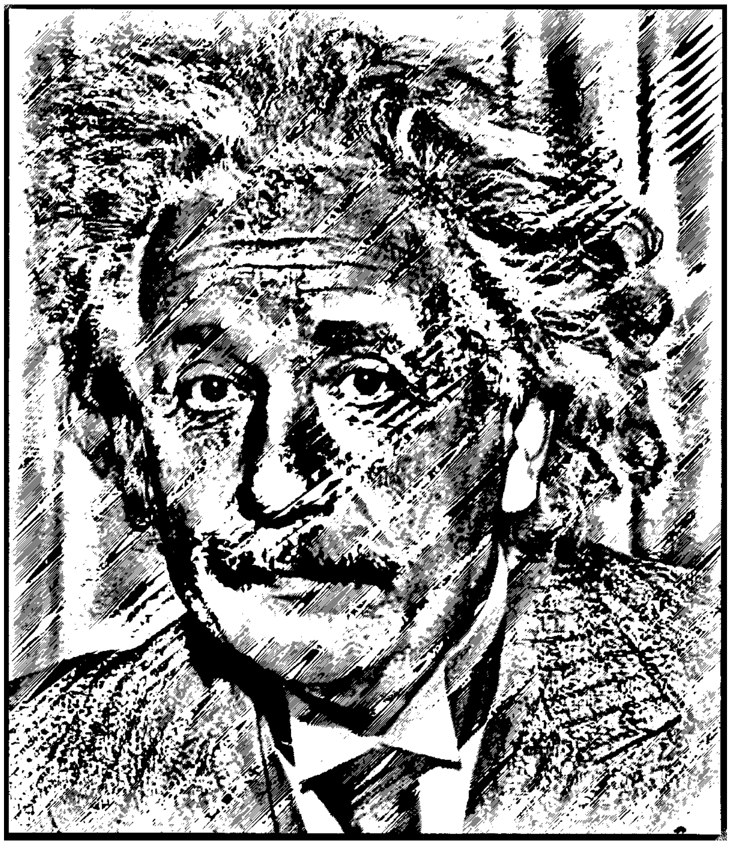


Figure 3.L8.2: (ein2L8)

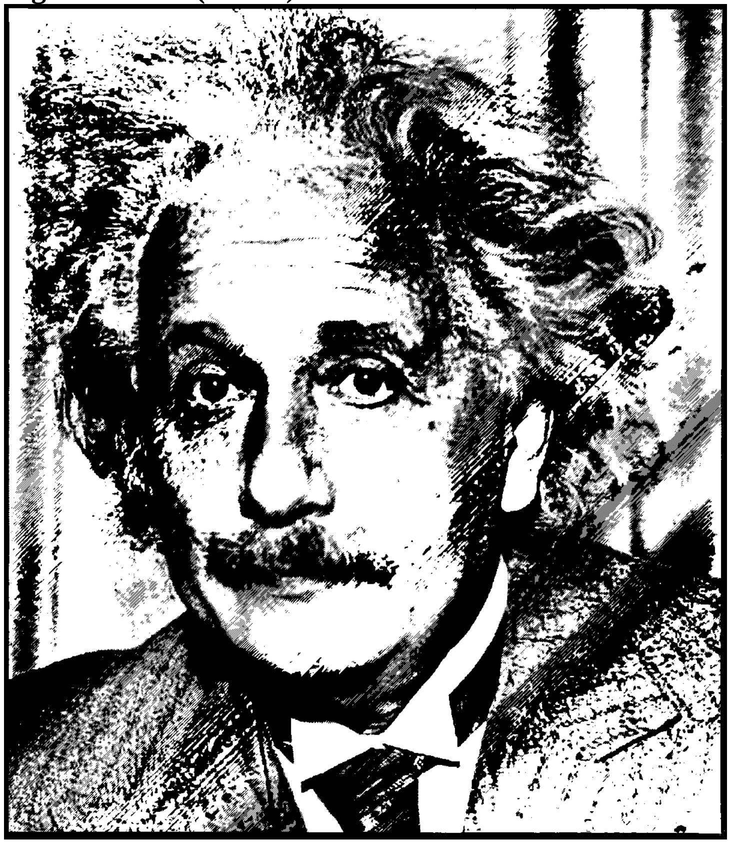


Figure 3.L8.3: (ein3L8)

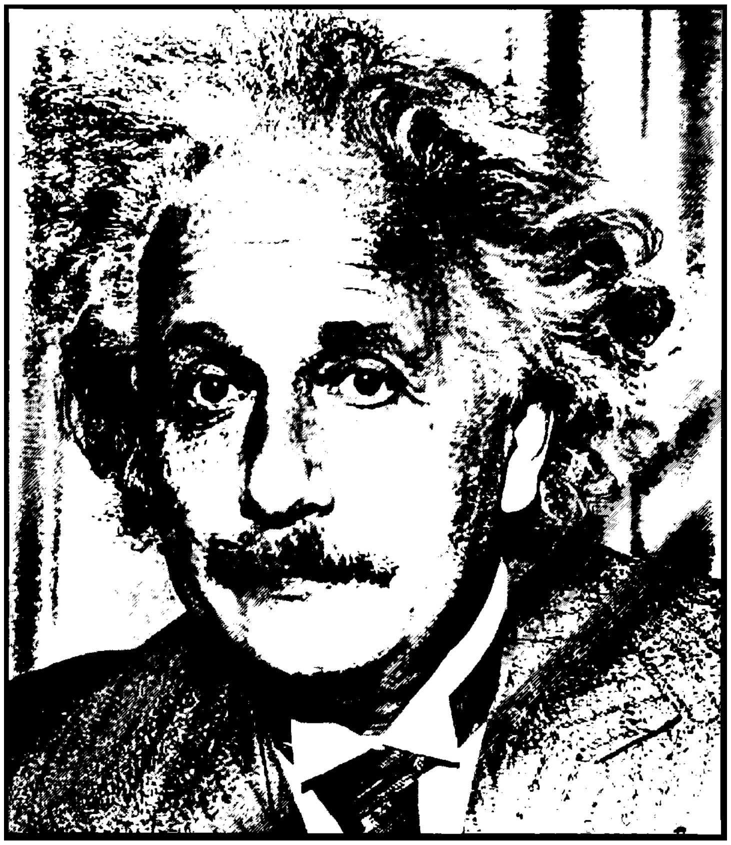


Figure 3.L8.4: (ein4L8)

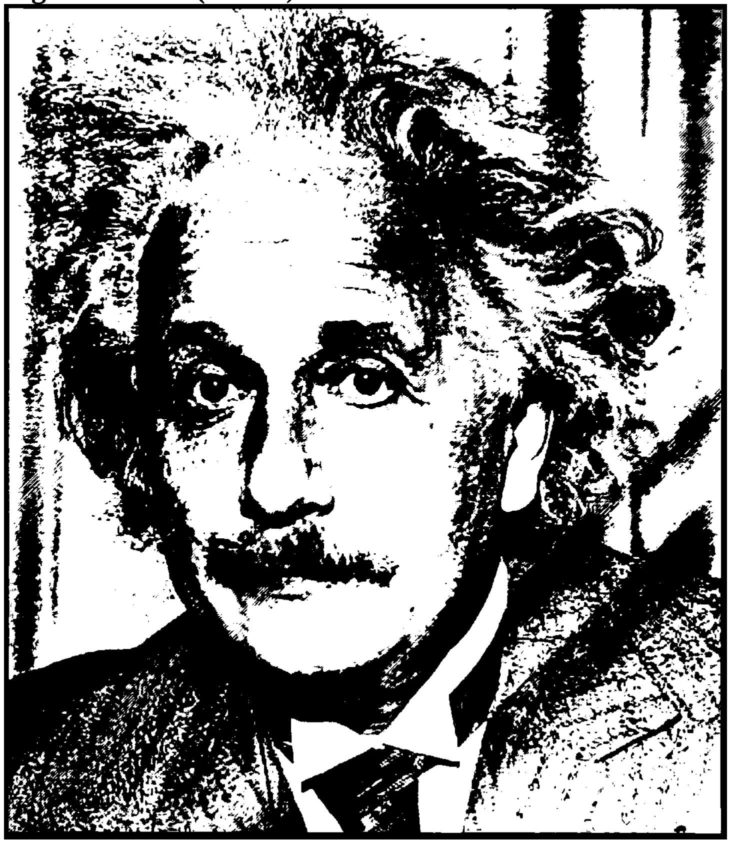


Figure 3.L8.5: (ein5L8)

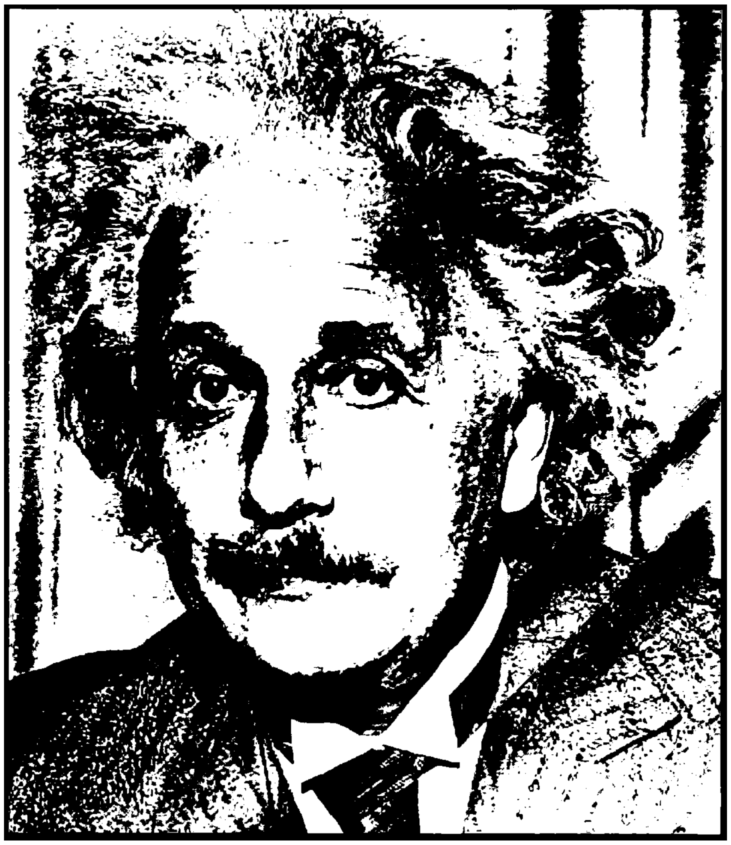


Figure 3.L8.6: (ein6L8)

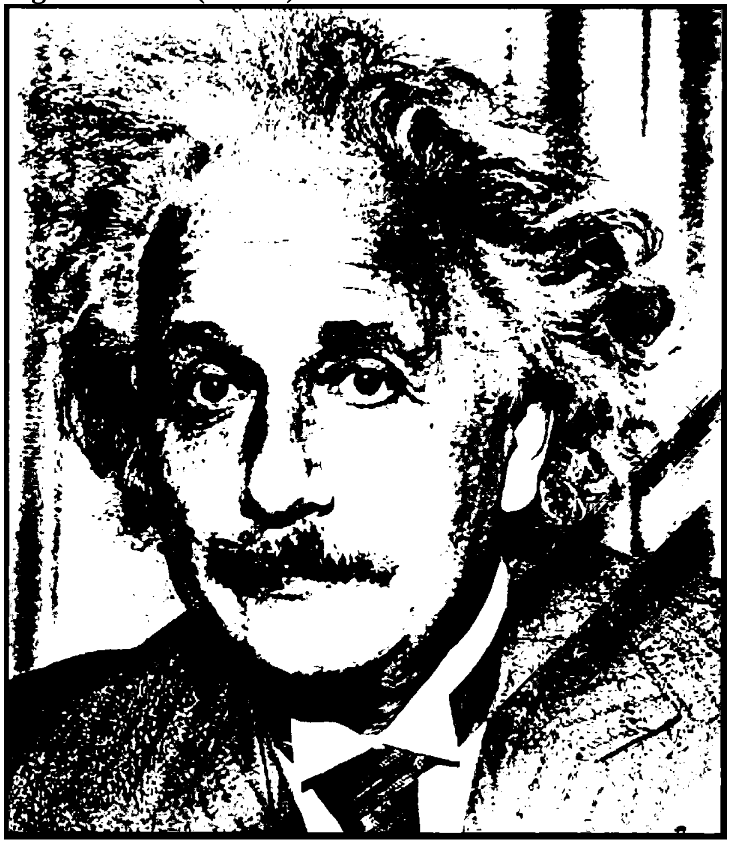


Figure 3.L8.7: (ein7L8)

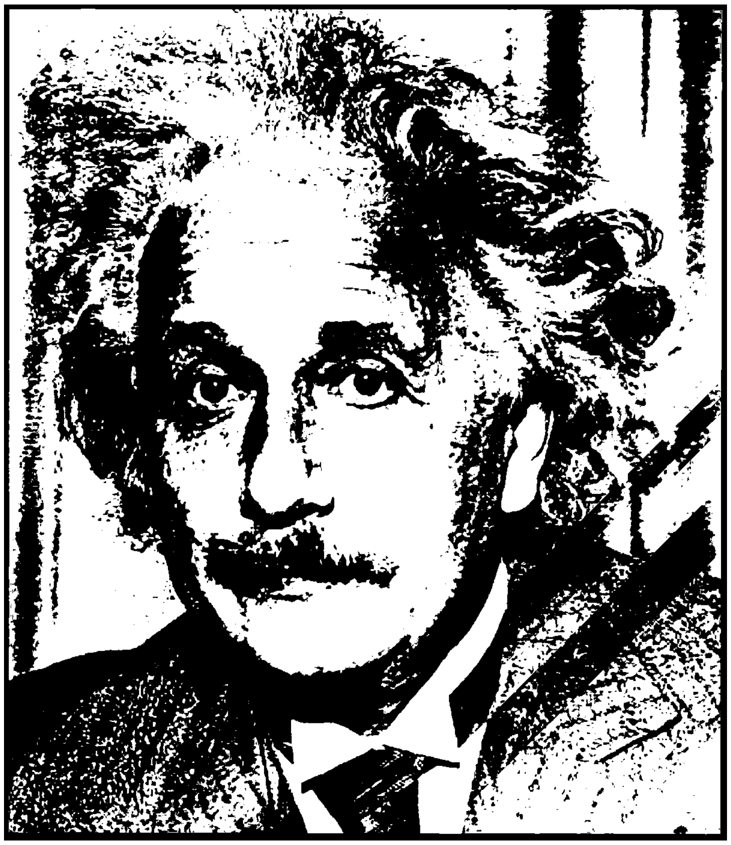


Figure 3.L8.8: (ein8L8)

3.3.3.4.14.2 Discussions

Filenames are given in parenthesis for the fig.3.L8.1-fig.3.L8.8. Fig.3.L8.1 contains more image details. Image details are slowly decreased from fig.3.L8.2 to fig.3.L8.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.L8.5 to fig.3.L8.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.L8.2 to fig.3.L8.3 because of sharp increase in nop. Lines are visible (from right to inclined towards left), in the direction of processing which is evident in fig.3.L8.1 and fig.3.L8.2. In the images left edges are more prominent which is more evident in fig.3.L8.1

3.3.3.4.15 RTLB: RTDAVFTC

3.3.3.4.15.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: RTDAVFTC option generates the following pictures (fig.3.R1.1-fig.3.R1.10) as output, similarly processing of fig.3.A1.1 generates fig.3.R1.4.1.

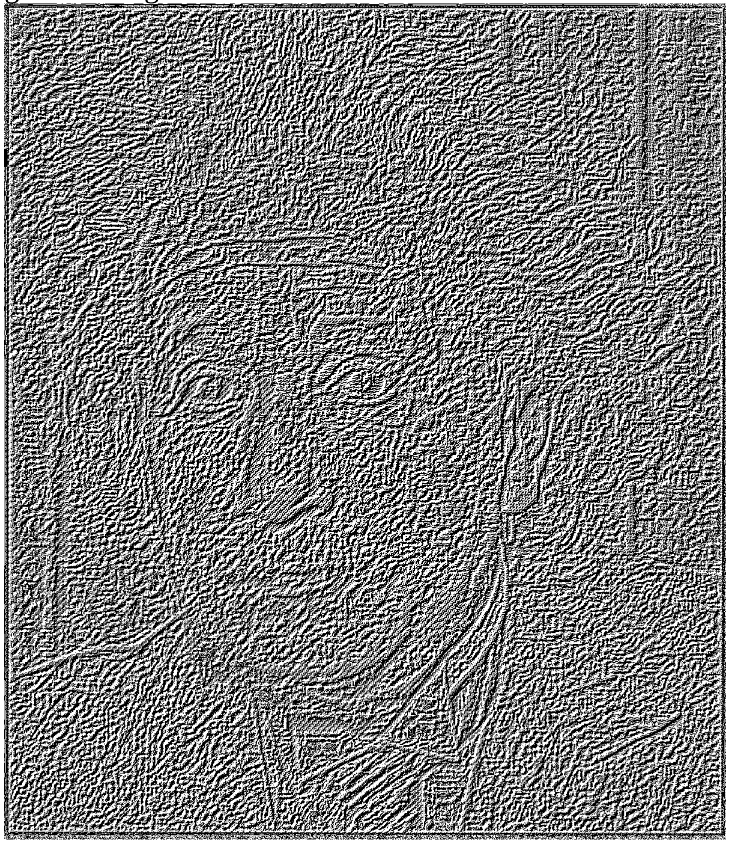


Figure 3.R1.1: (ein300_dthRTLB_nop5_opt1)

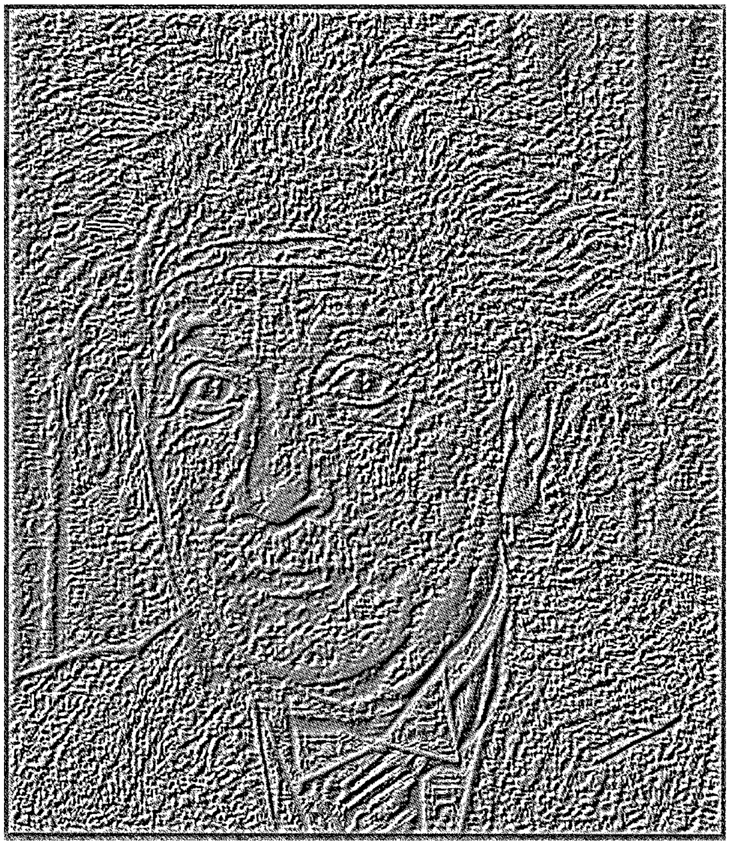


Figure 3.R1.2: (ein300_dthRTLB_nop10_opt1)

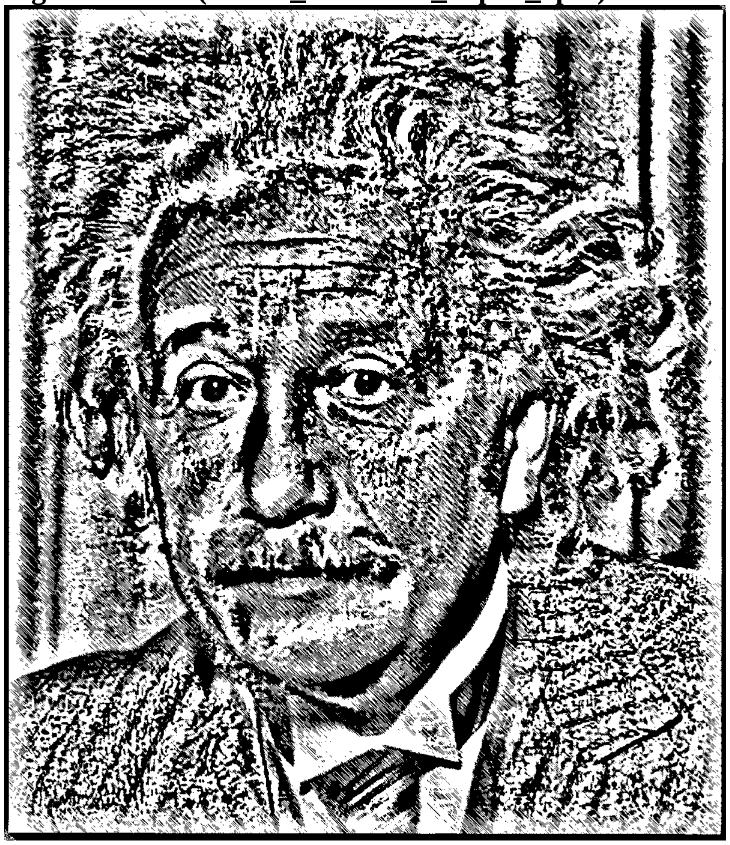


Figure 3.R1.3: (ein1R1)

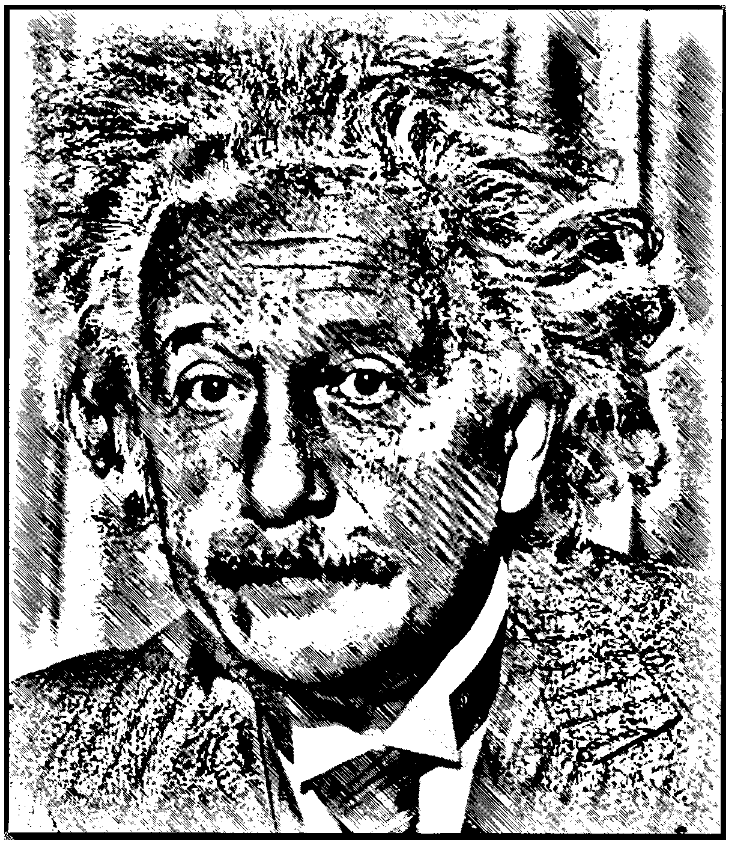


Figure 3.R1.4: (ein2R1)

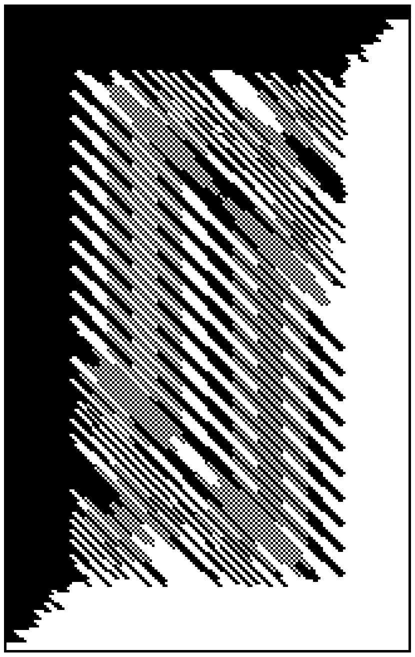


Figure 3.R1.4.1: (dth_rtlb_nop100)

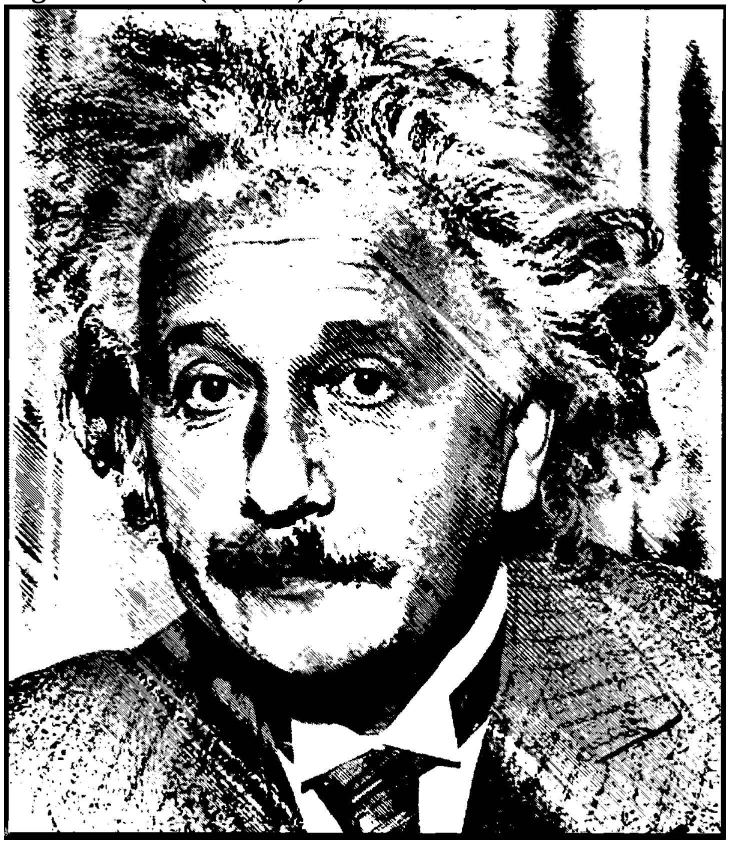


Figure 3.R1.5: (ein3R1)

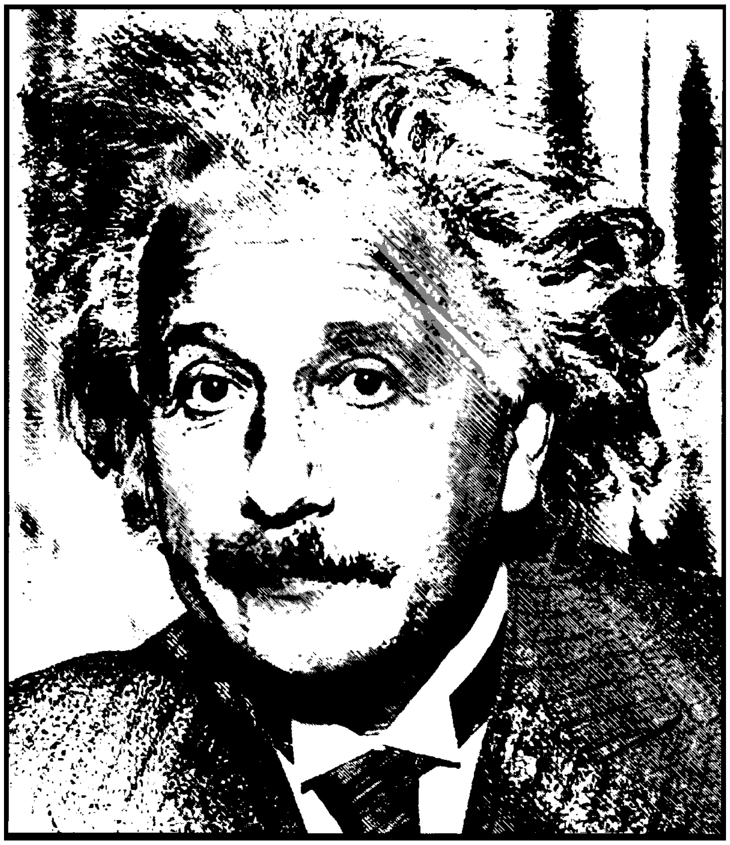


Figure 3.R1.6: (ein4R1)

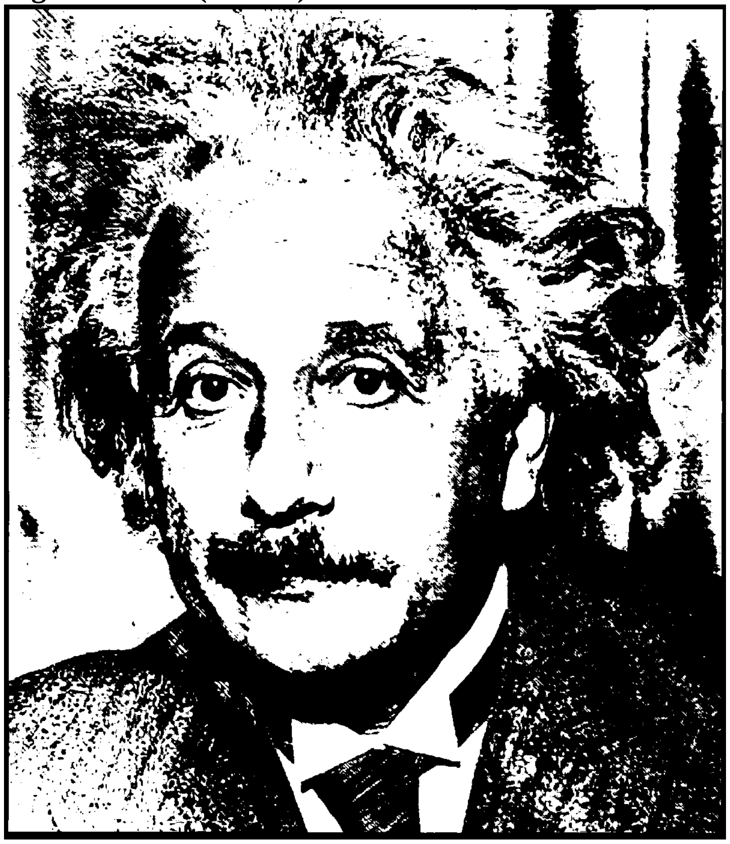


Figure 3.R1.7: (ein5R1)

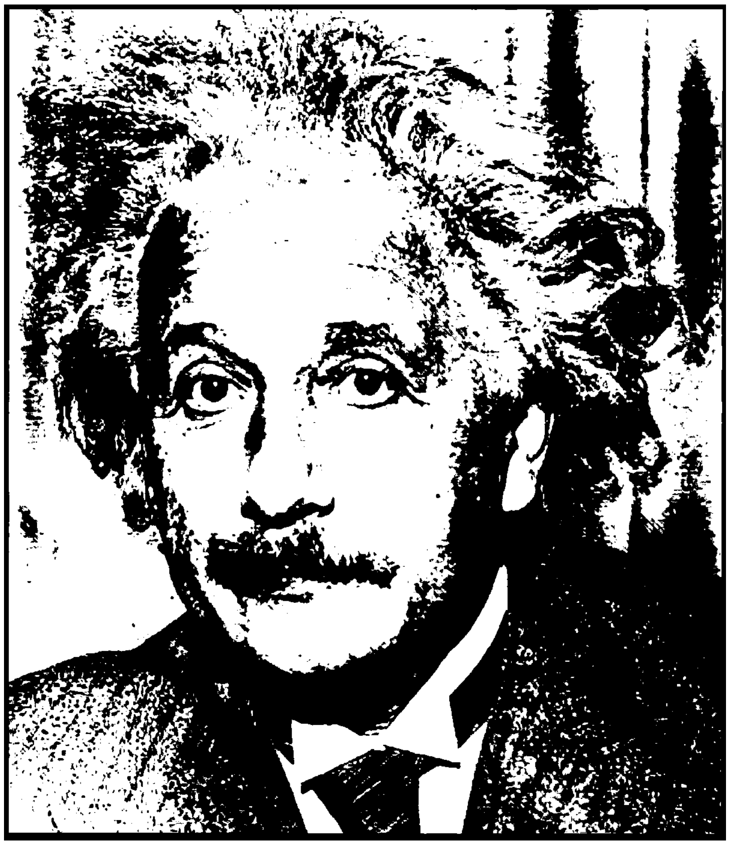


Figure 3.R1.8: (ein6R1)

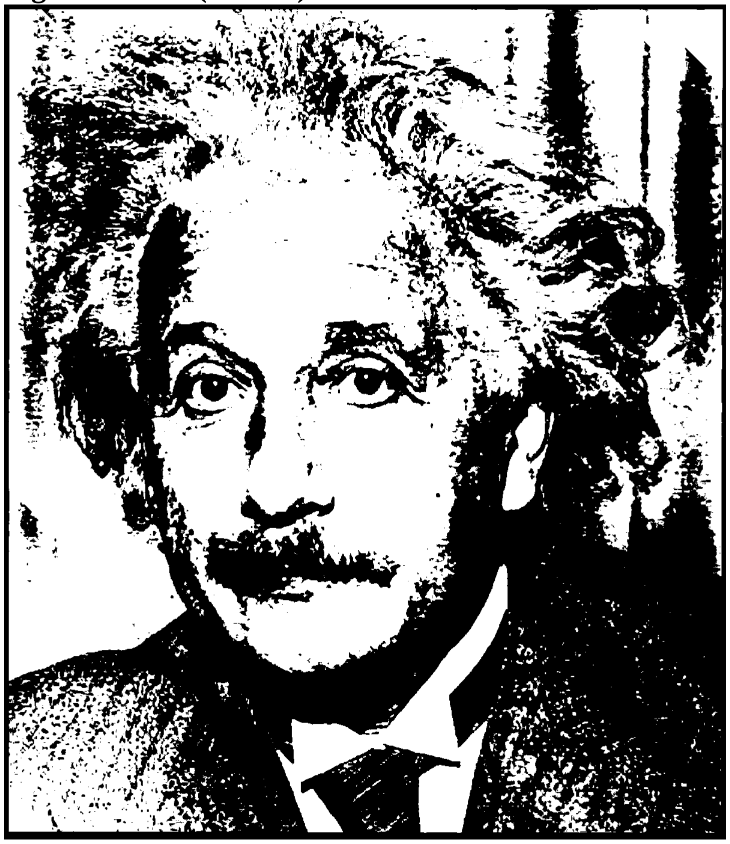


Figure 3.R1.9: (ein7R1)

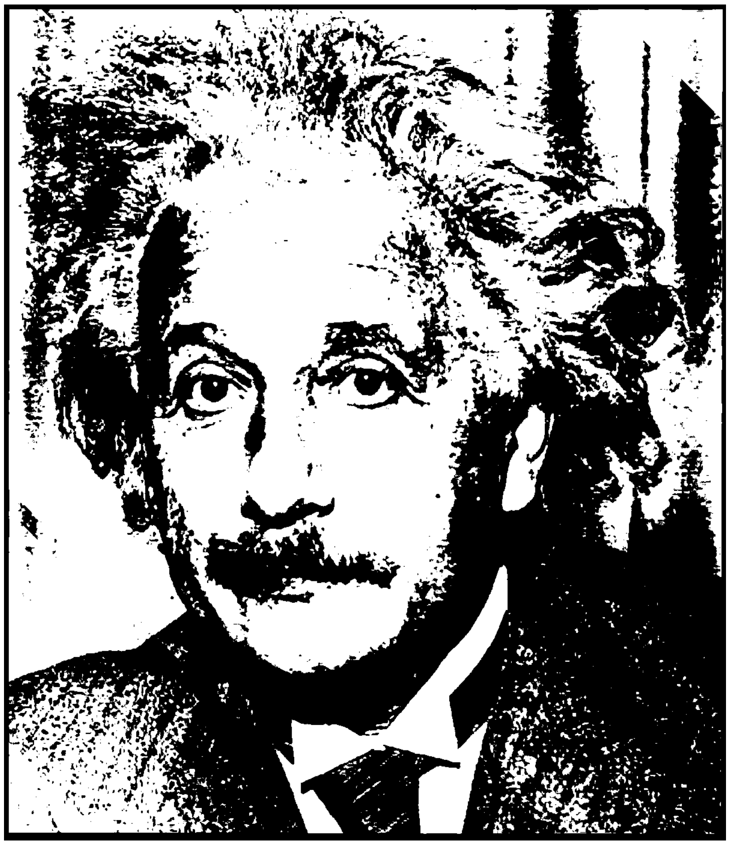


Figure 3.R1.10: (ein8R1)

3.3.3.4.15.2 Discussions

Filenames are given in parenthesis for the fig.3.R1.1-fig.3.R1.10 and fig.3.R1.4.1. Fig.3.R1.1 and fig.3.R1.2 look like engraved images. Fig.3.R1.3 contains more image details. Image details are slowly decreased from fig.3.R1.4 to fig.3.R1.7 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R1.7 to fig.3.R1.10 the decrease in image details is almost constant. Fig.3.R1.4.1 is a processed grayscale ramp (having gray values from 0-255. The grayscale ramp is generated using Program A.1 and to generate the outside black border of the ramp Program A.2 is used). There is a sharp decrease is image details from fig.3.R1.4 to fig.3.R1.5 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which are more evident in fig.3.R1.3 and fig.3.R1.4. In the images right edges are more prominent which is more evident in fig.3.R1.3

3.3.3.4.16 RTLB: RTDAVFTAC

3.3.3.4.16.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: RTDAVFTAC option generates the following pictures (fig.3.R2.1-fig.3.R2.8) as output

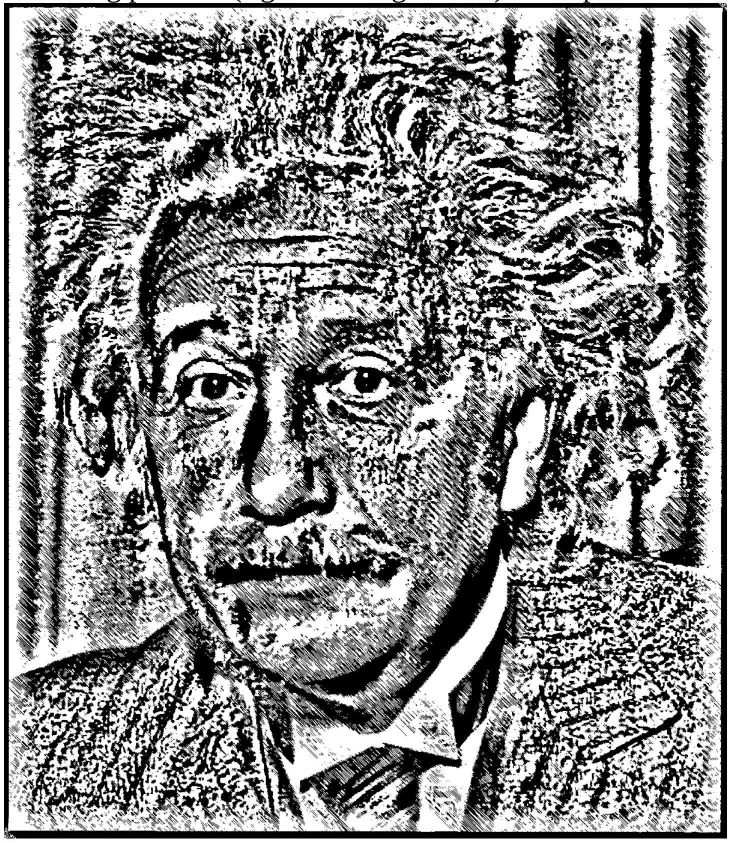


Figure 3.R2.1: (ein1R2)

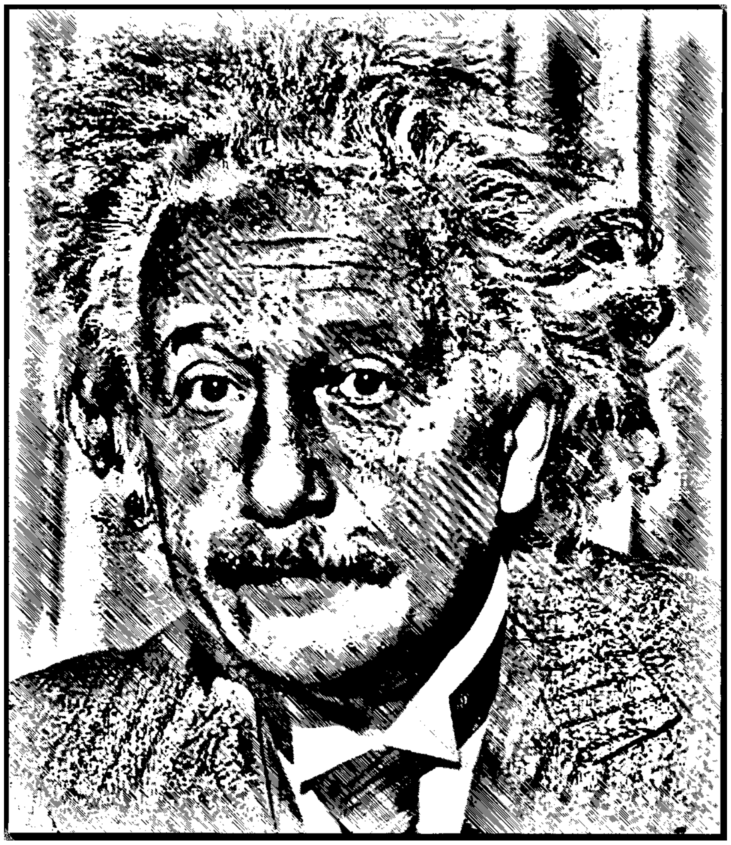


Figure 3.R2.2: (ein2R2)

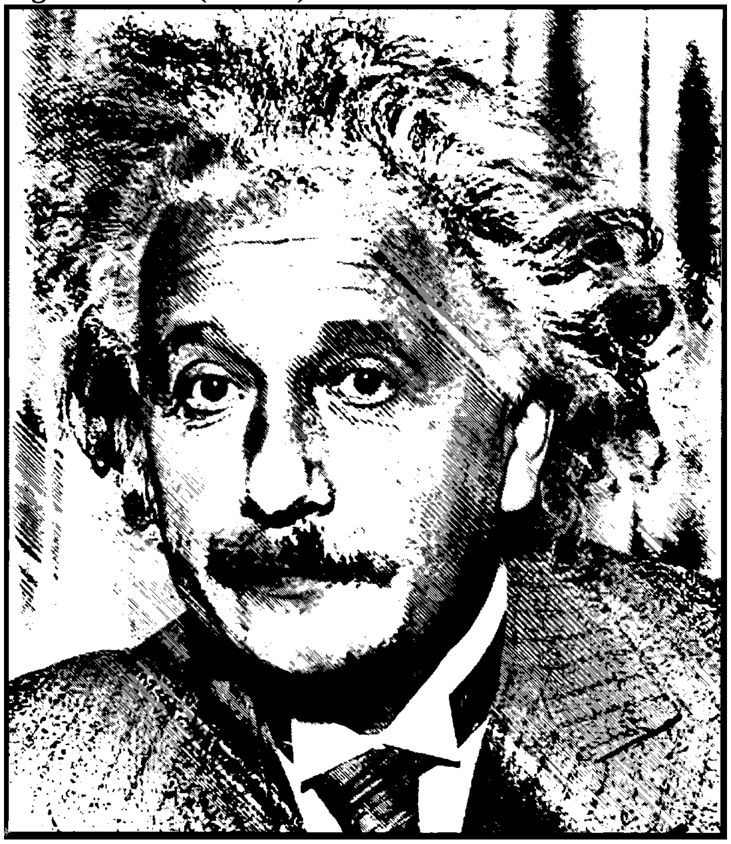


Figure 3.R2.3: (ein3R2)

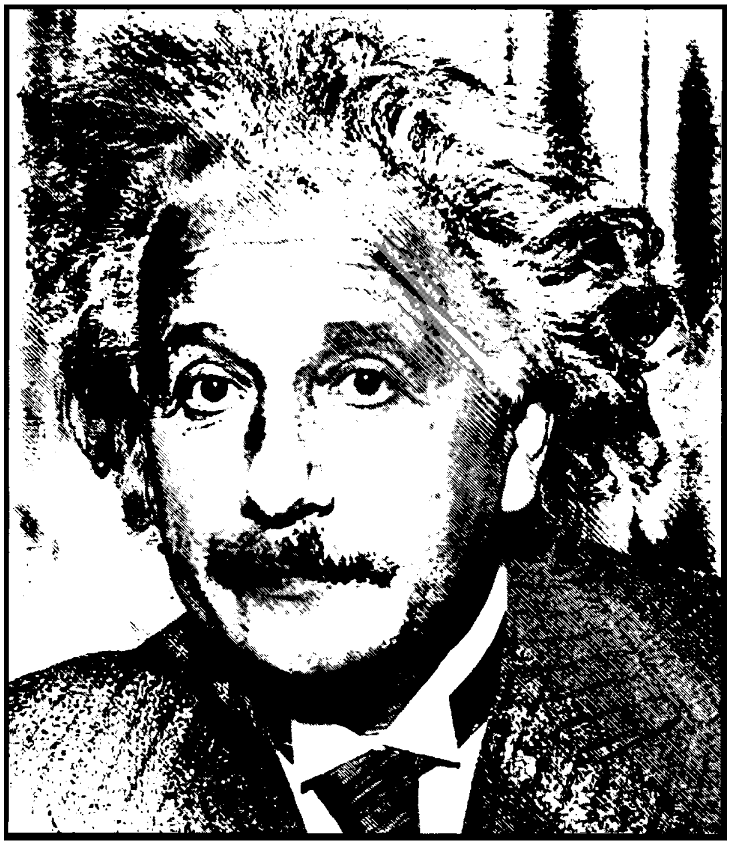


Figure 3.R2.4: (ein4R2)

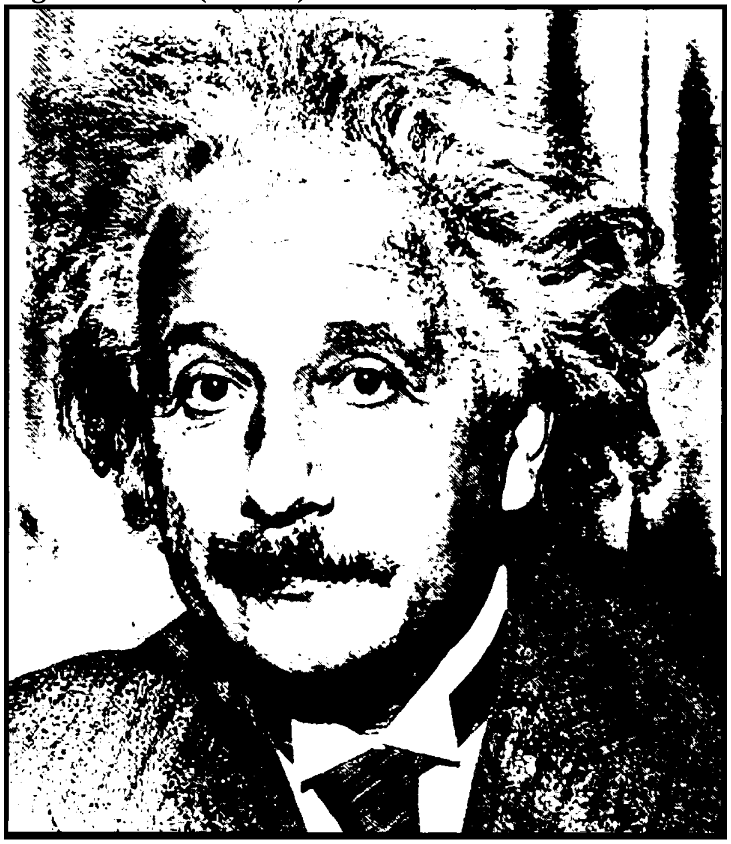


Figure 3.R2.5: (ein5R2)

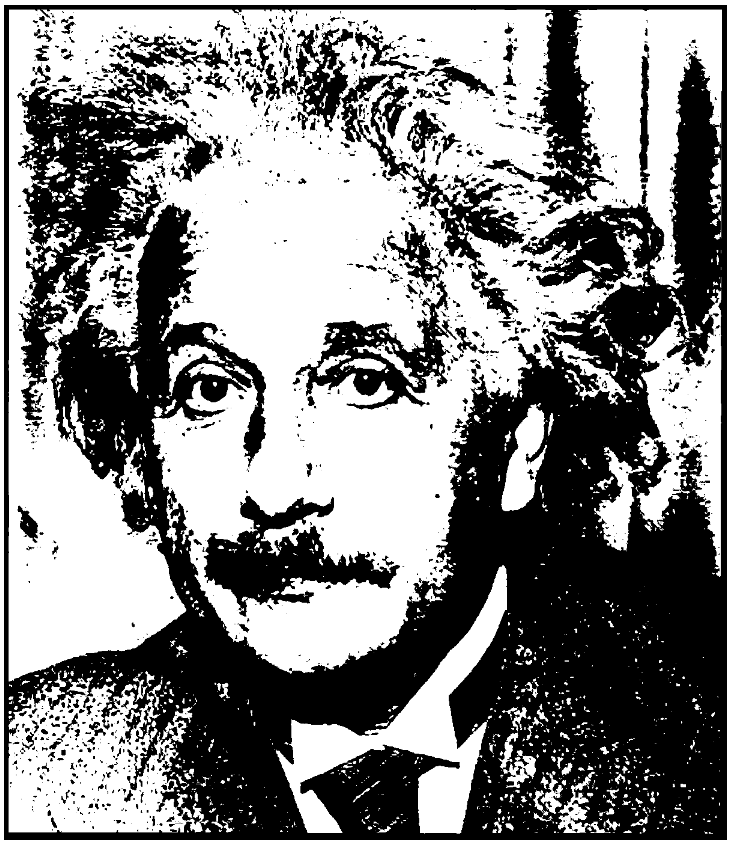


Figure 3.R2.6: (ein6R2)

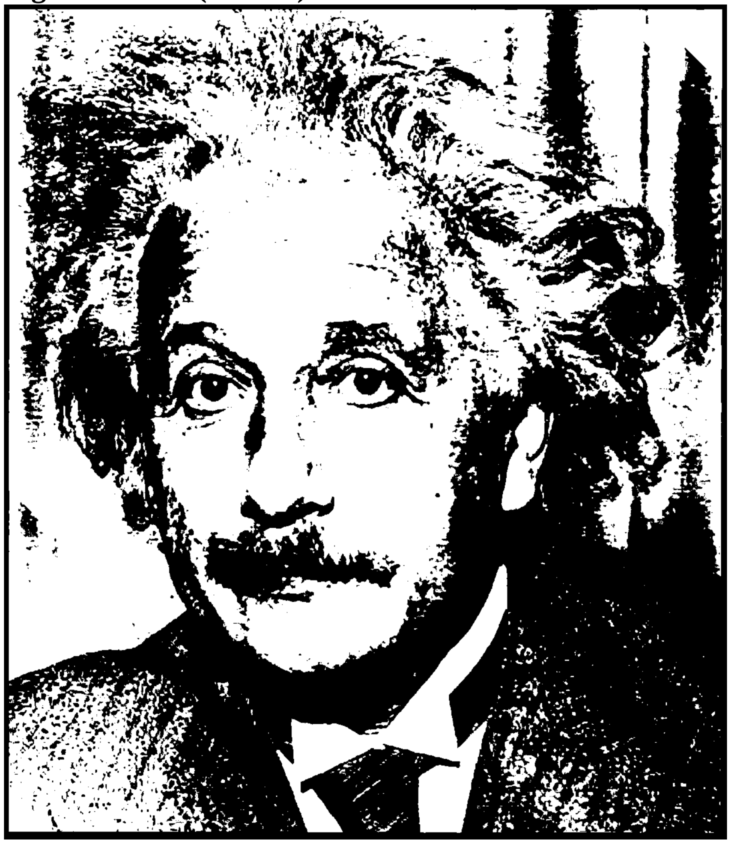


Figure 3.R2.7: (ein7R2)

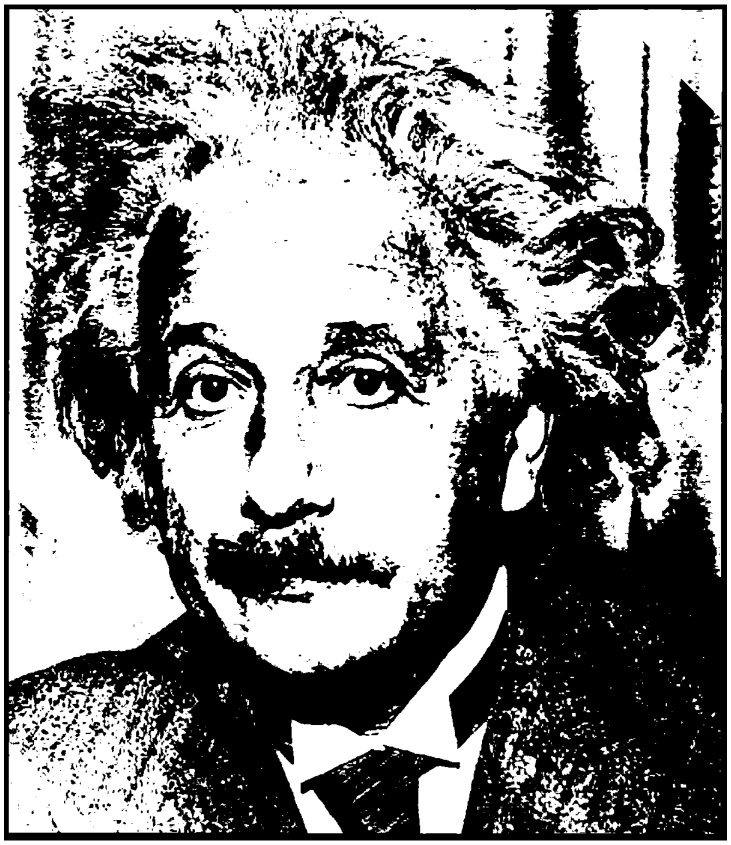


Figure 3.R2.8: (ein8R2)

3.3.3.4.16.2 Discussions

Filenames are given in parenthesis for the fig.3.R2.1-fig.3.R2.8. Fig.3.R2.1 contains more image details. Image details are slowly decreased from fig.3.R2.2 to fig.3.R2.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R2.5 to fig.3.R2.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R2.2 to fig.3.R2.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R2.1 and fig.3.R2.2. In the images left edges are more prominent which is more evident in fig.3.R2.1

3.3.3.4.17 RTLB: RTDVRB2LT

3.3.3.4.17.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: RTDVRB2LT option generates the following pictures (fig.3.R3.1-fig.3.R3.8) as output

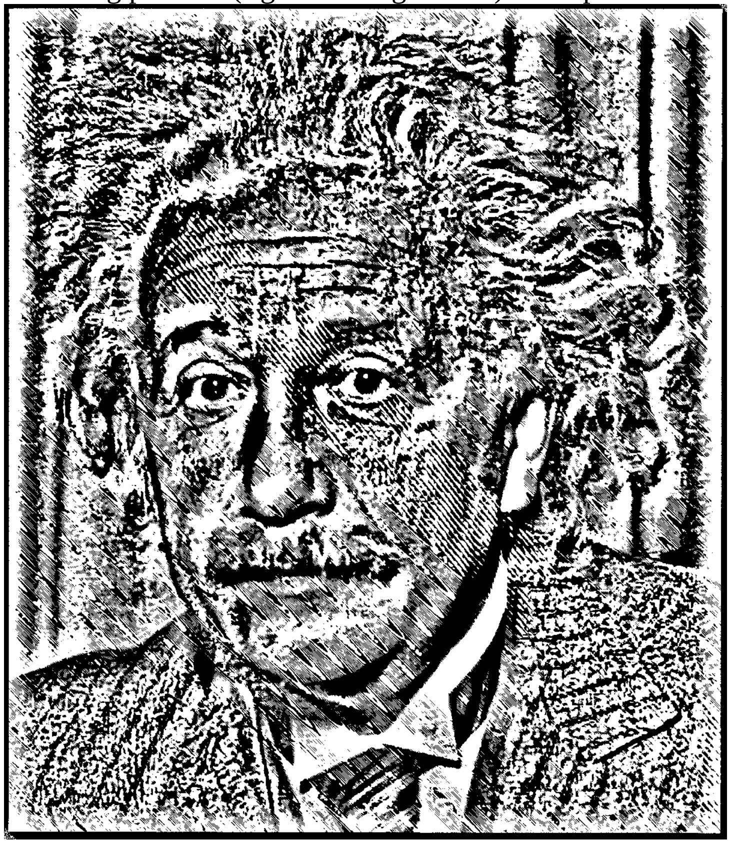


Figure 3.R3.1: (ein1R3)

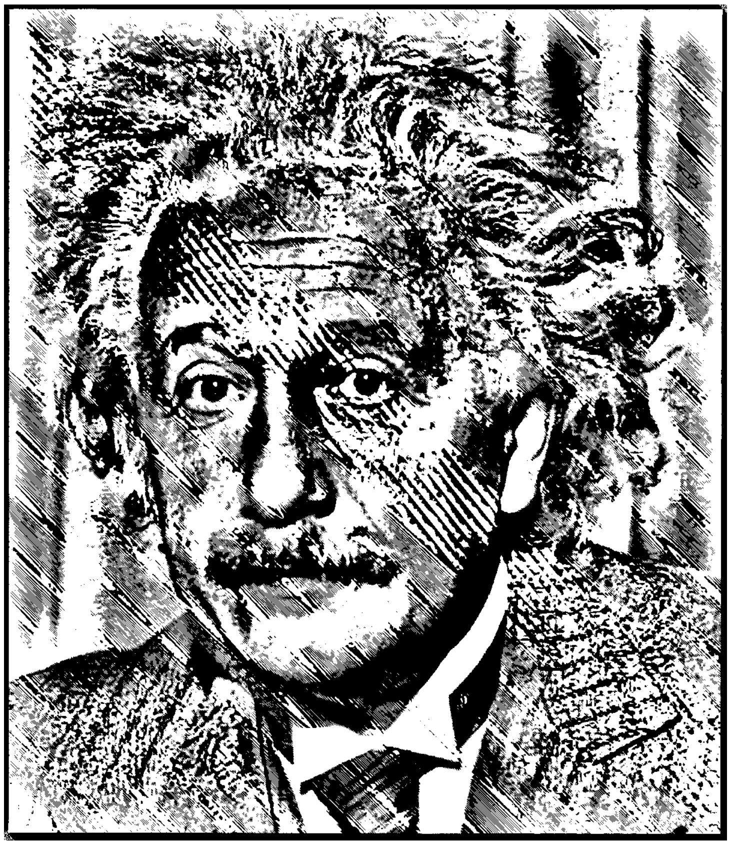


Figure 3.R3.2: (ein2R3)

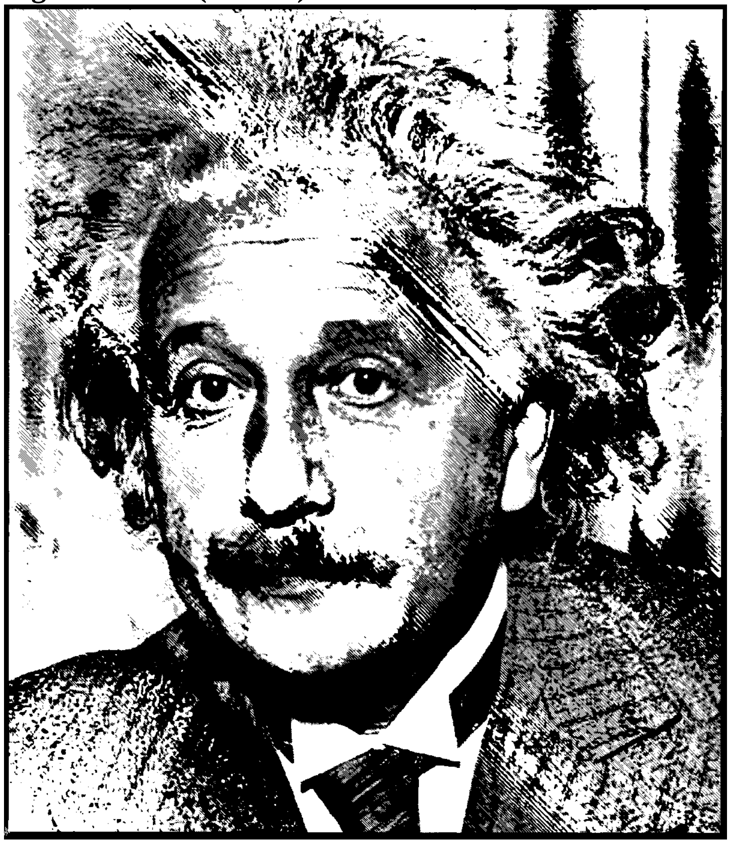


Figure 3.R3.3: (ein3R3)

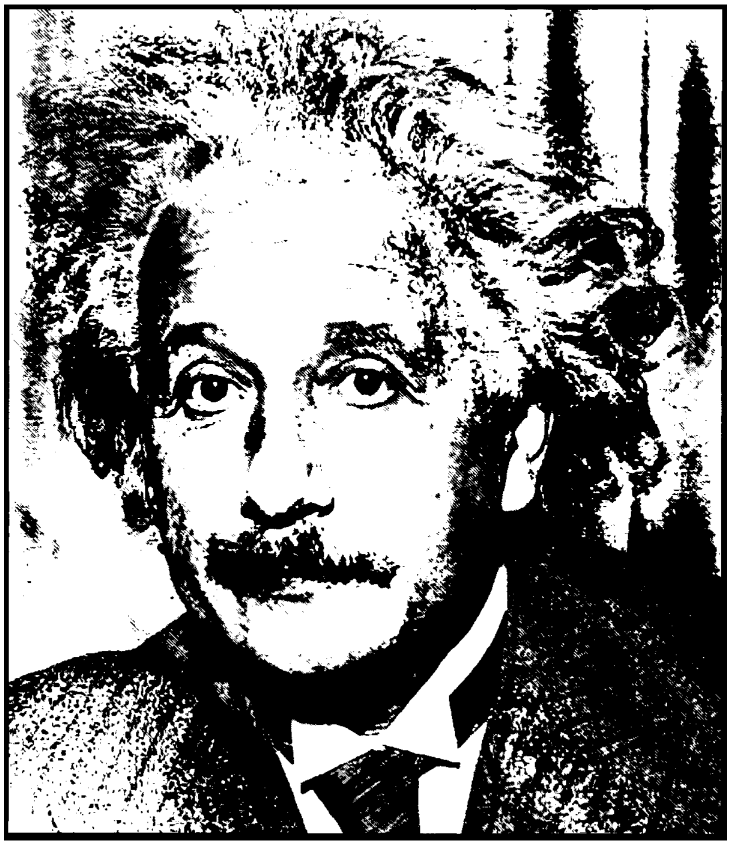


Figure 3.R3.4: (ein4R3)

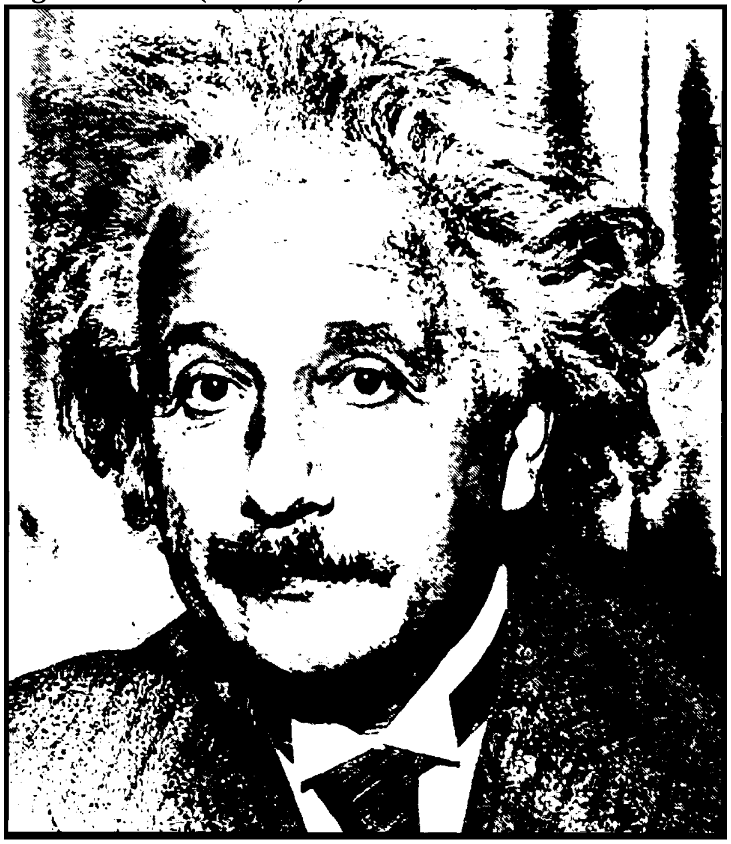


Figure 3.R3.5: (ein5R3)

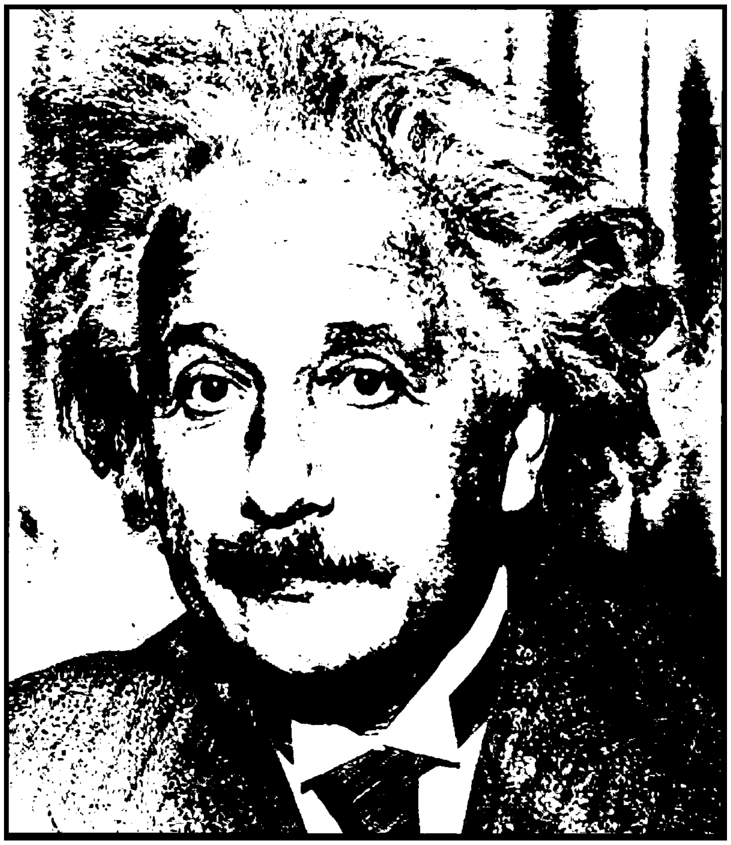


Figure 3.R3.6: (ein6R3)

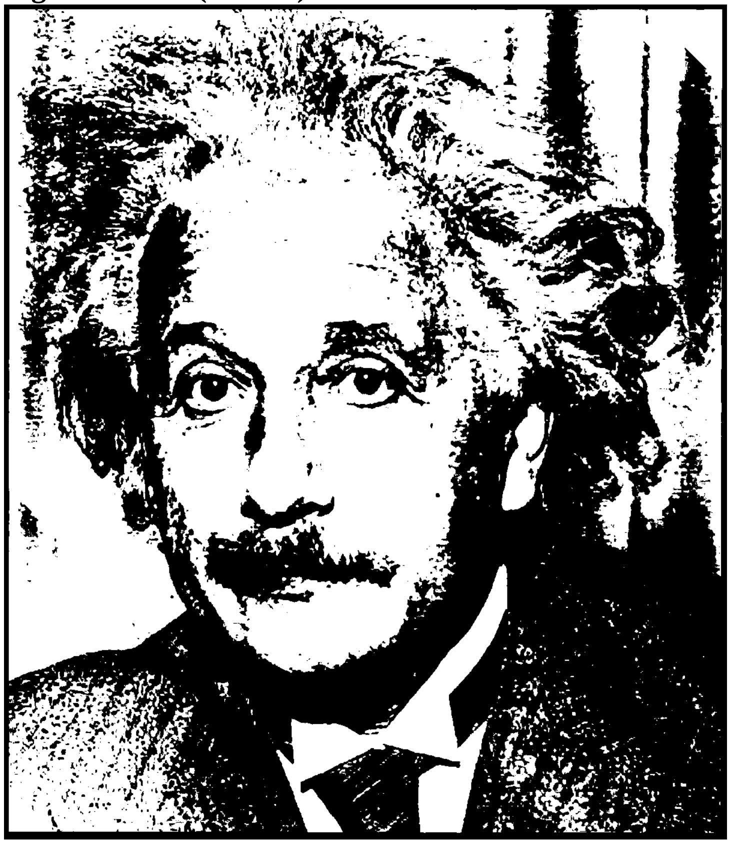


Figure 3.R3.7: (ein7R3)

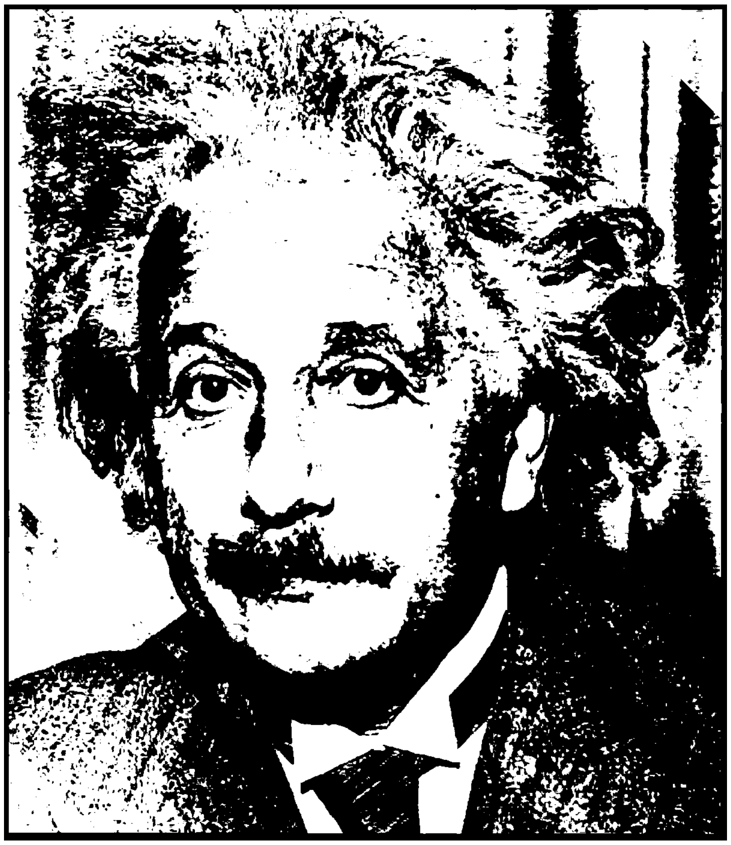


Figure 3.R3.8: (ein8R3)

3.3.3.4.17.2 Discussions

Filenames are given in parenthesis for the fig.3.R3.1-fig.3.R3.8. Fig.3.R3.1 contains more image details. Image details are slowly decreased from fig.3.R3.2 to fig.3.R3.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R3.5 to fig.3.R3.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R3.2 to fig.3.R3.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R3.1 and fig.3.R3.2. In the images left edges are more prominent which is more evident in fig.3.R3.1

3.3.3.4.18 RTLB: RTDVLT2RB

3.3.3.4.18.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: RTDVLT2RB option generates the following pictures (fig.3.R4.1-fig.3.R4.8) as output

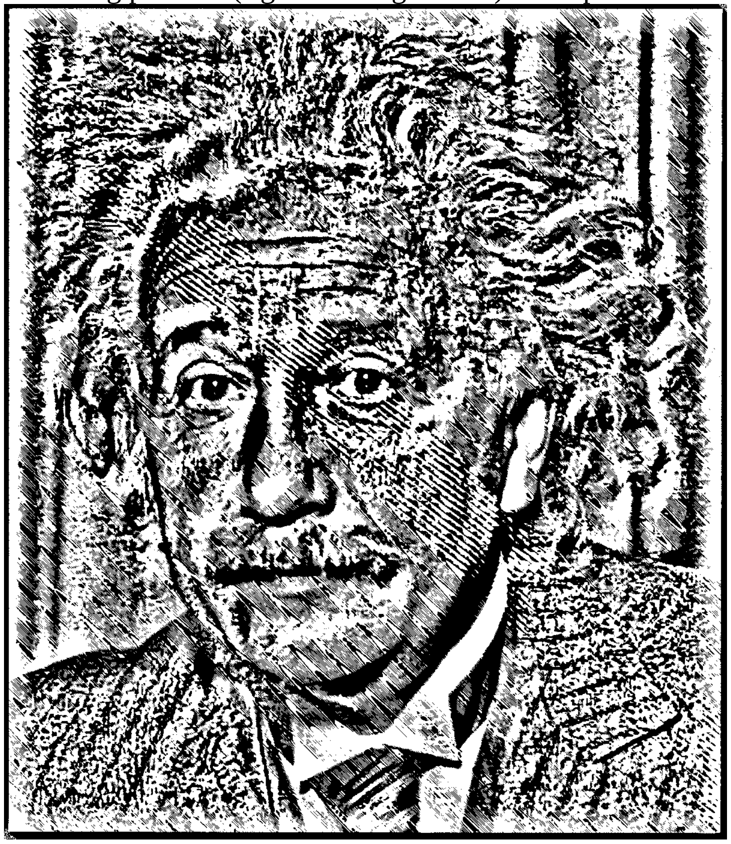


Figure 3.R4.1: (ein1R4)

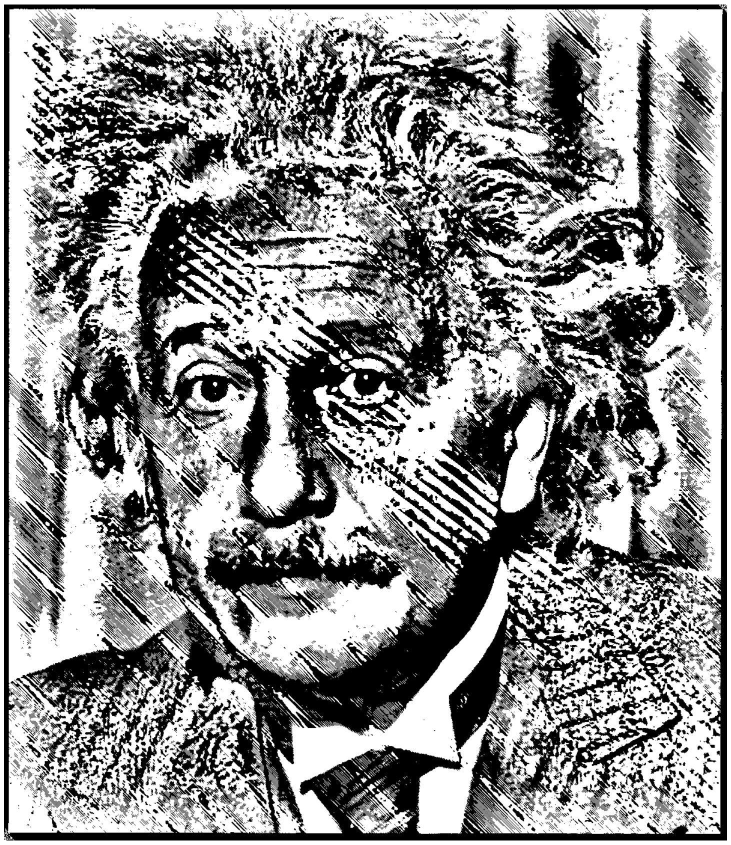


Figure 3.R4.2: (ein2R4)

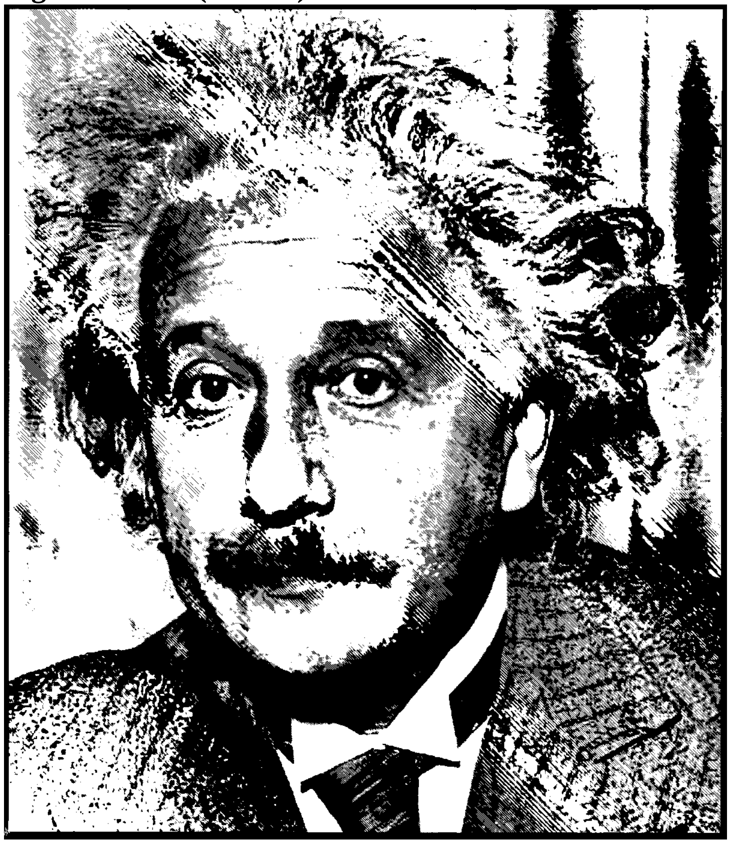


Figure 3.R4.3: (ein3R4)

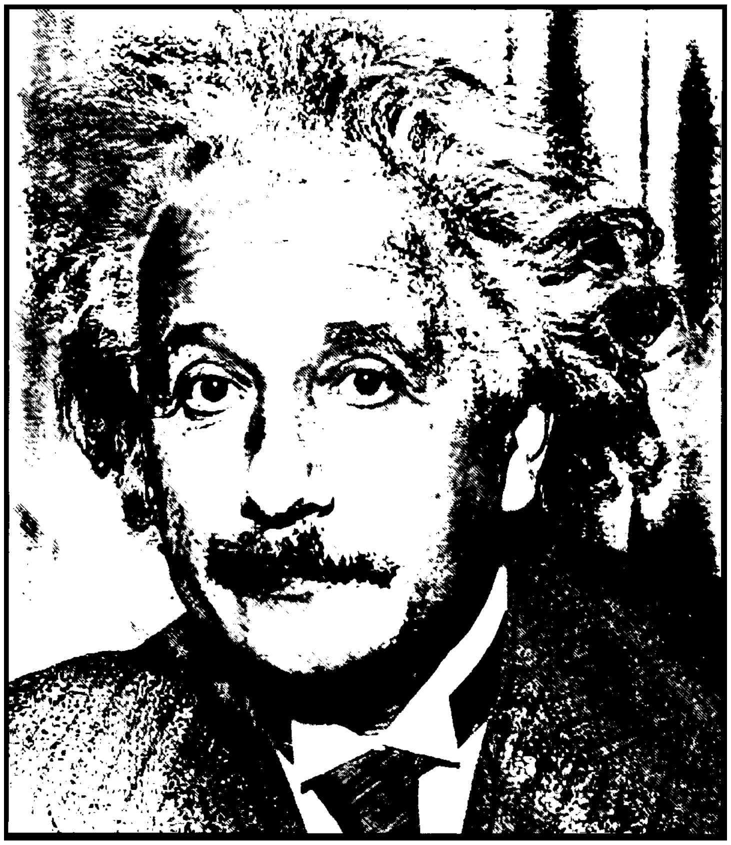


Figure 3.R4.4: (ein4R4)

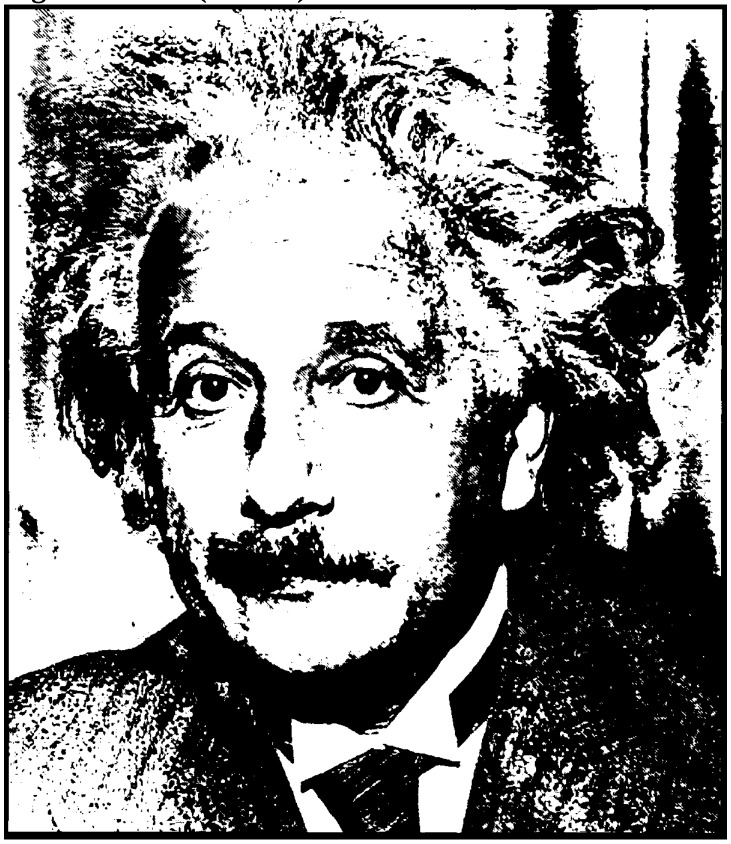


Figure 3.R4.5: (ein5R4)

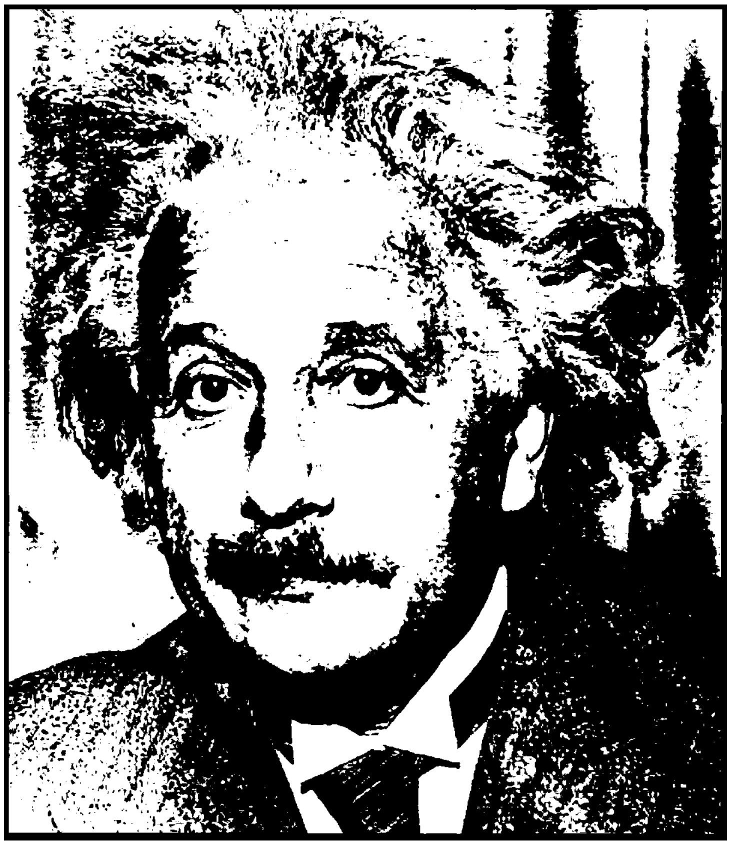


Figure 3.R4.6: (ein6R4)

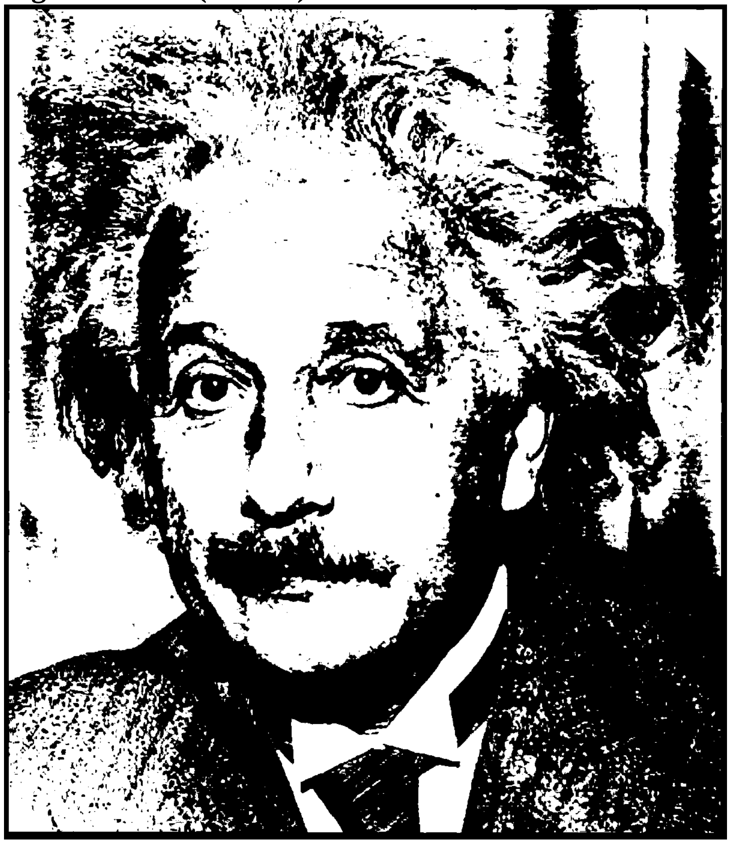


Figure 3.R4.7: (ein7R4)

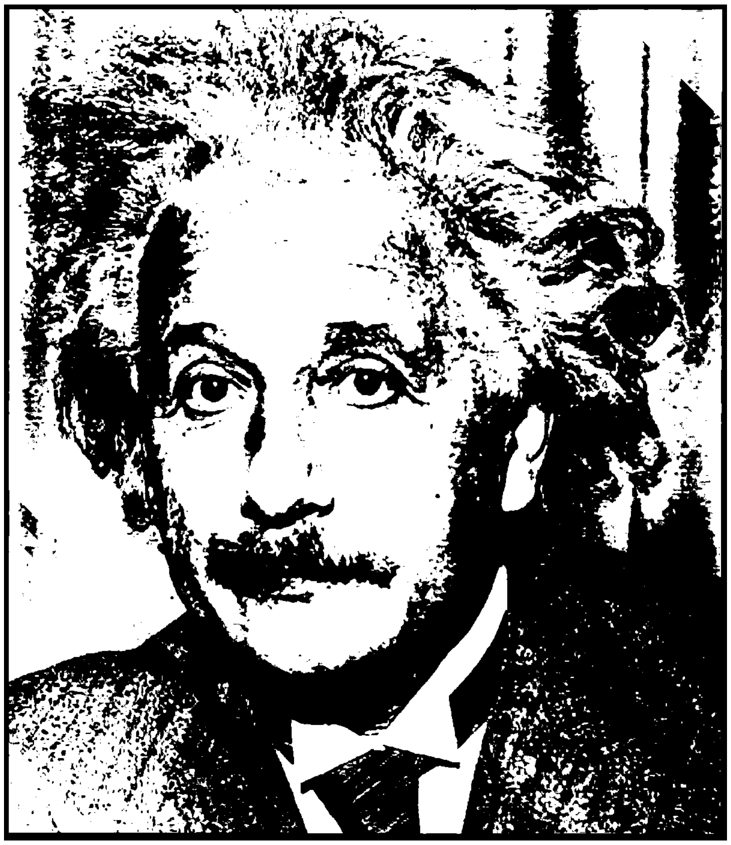


Figure 3.R4.8: (ein8R4)

3.3.3.4.18.2 Discussions

Filenames are given in parenthesis for the fig.3.R4.1-fig.3.R4.8. Fig.3.R4.1 contains more image details. Image details are slowly decreased from fig.3.R4.2 to fig.3.R4.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R4.5 to fig.3.R4.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R4.2 to fig.3.R4.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R4.1 and fig.3.R4.2. In the images left edges are more prominent which is more evident in fig.3.R4.1

3.3.3.4.19 RTLB: LBDAVFTAC

3.3.3.4.19.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: LBDAVFTAC option generates the following pictures (fig.3.R5.1-fig.3.R5.8) as output

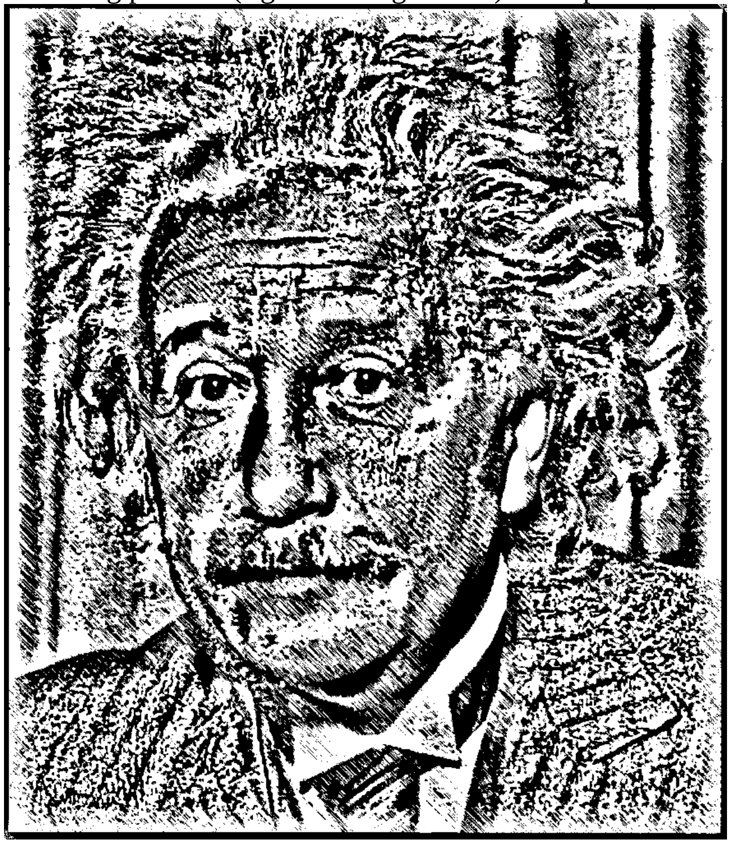


Figure 3.R5.1: (ein1R5)

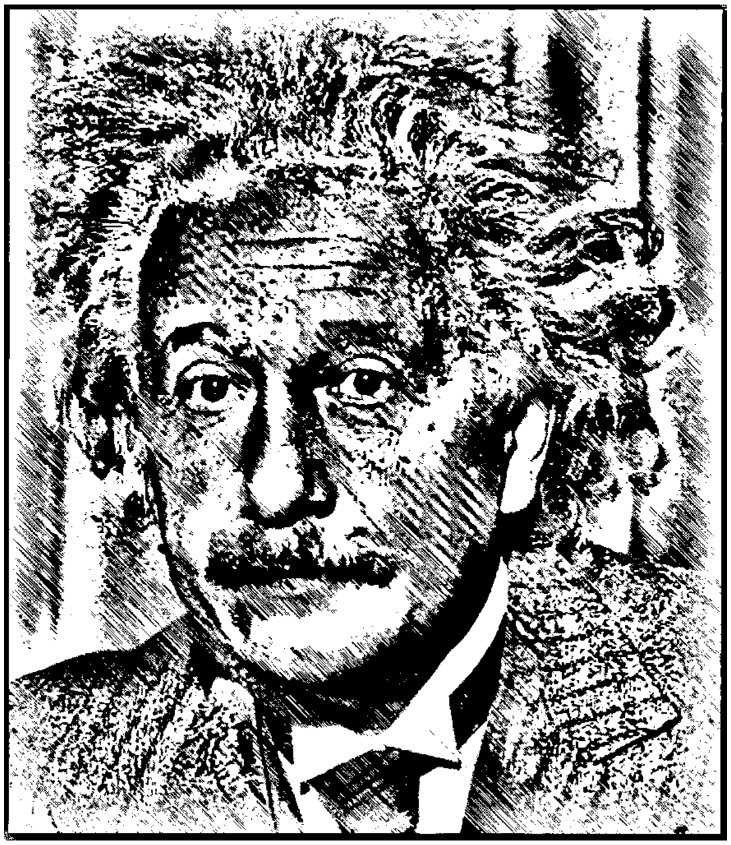


Figure 3.R5.2: (ein2R5)

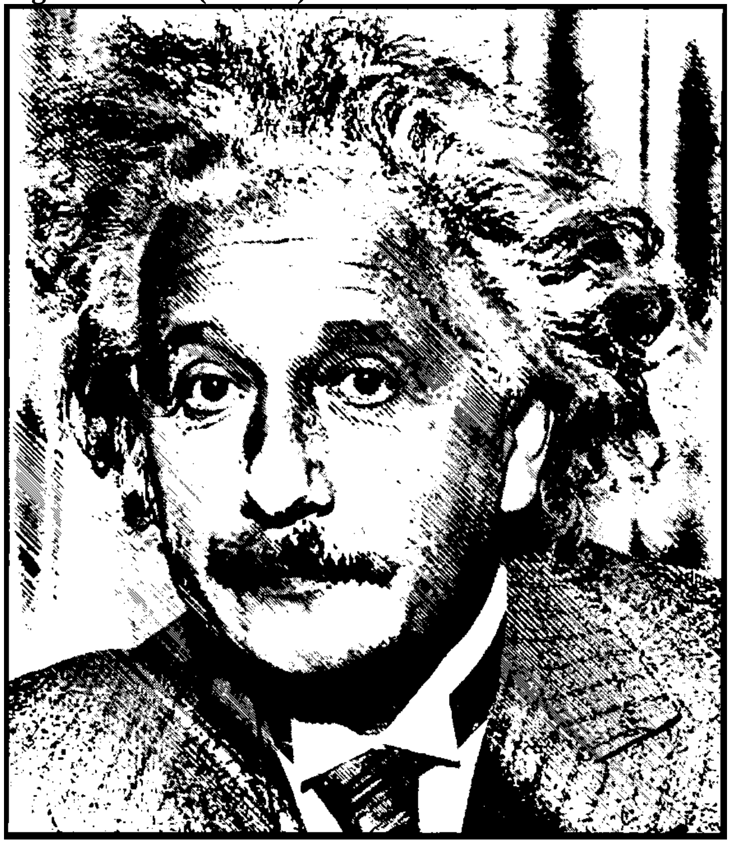


Figure 3.R5.3: (ein3R5)

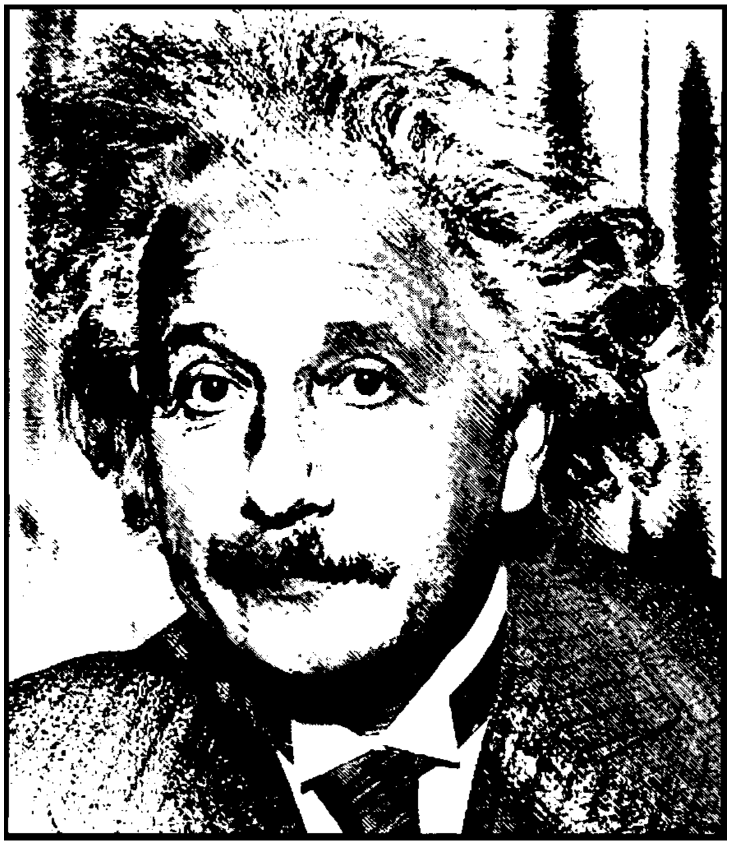


Figure 3.R5.4: (ein4R5)

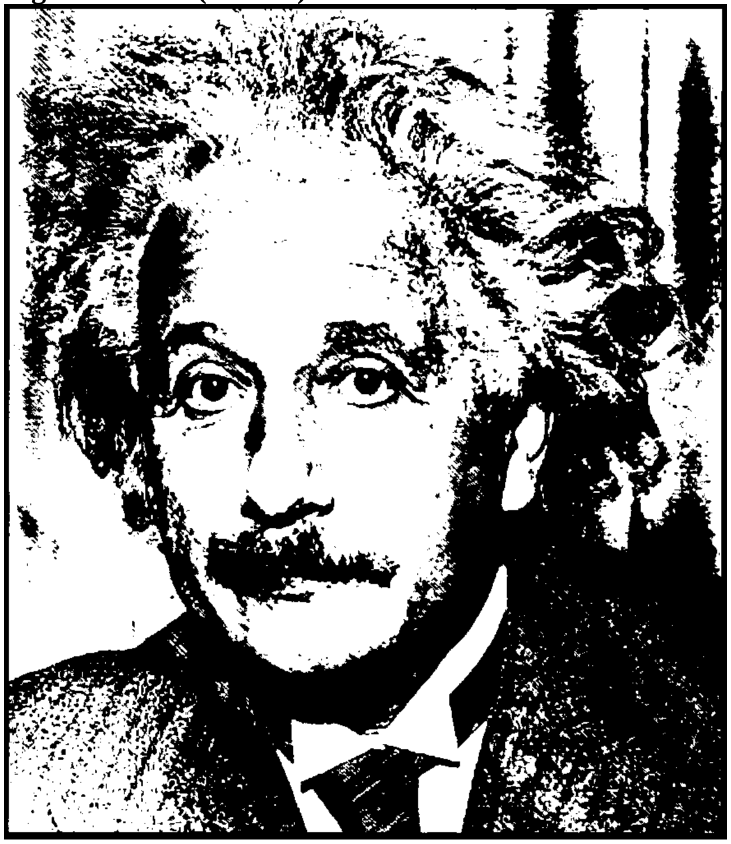


Figure 3.R5.5: (ein5R5)

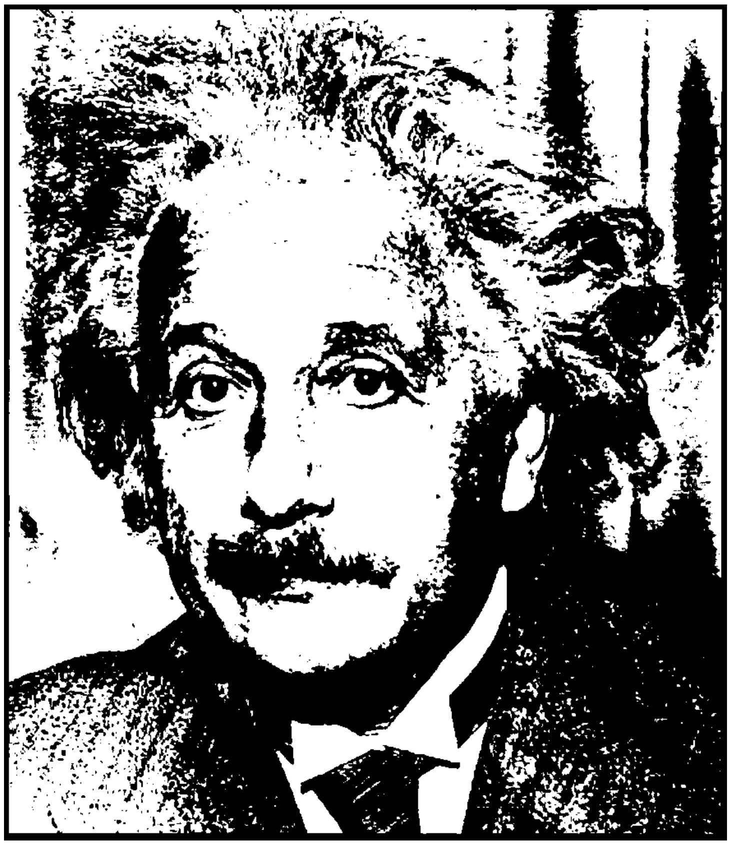


Figure 3.R5.6: (ein6R5)

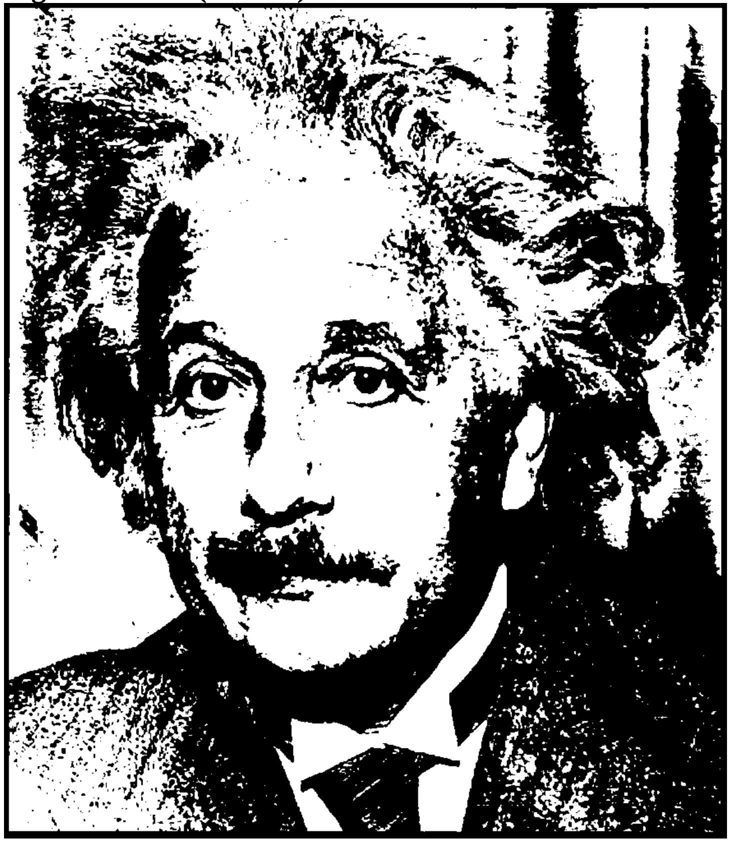


Figure 3.R5.7: (ein7R5)

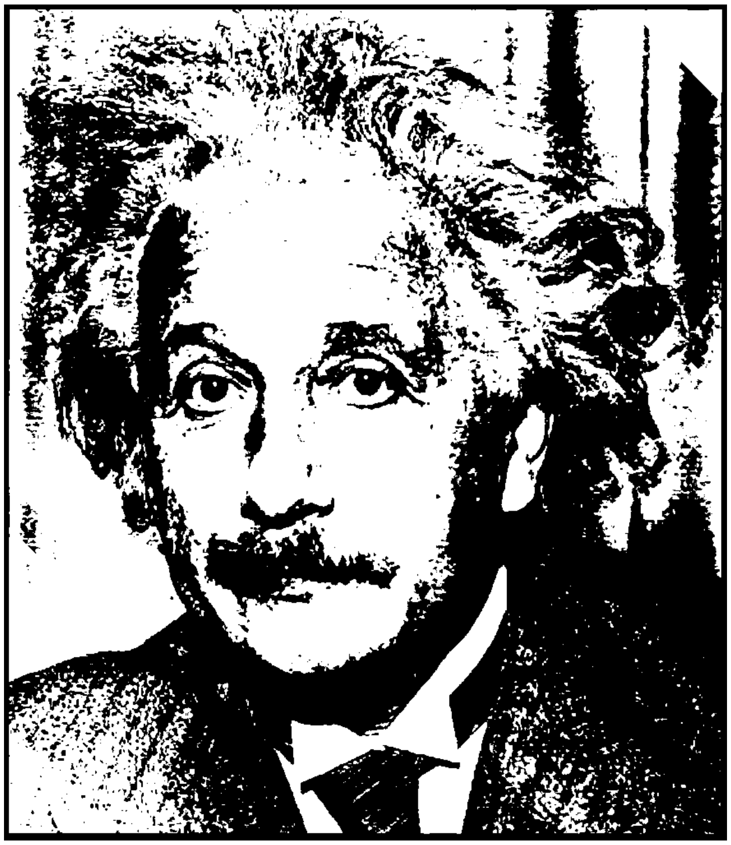


Figure 3.R5.8: (ein8R5)

3.3.3.4.19.2 Discussions

Filenames are given in parenthesis for the fig.3.R5.1-fig.3.R5.8. Fig.3.R5.1 contains more image details. Image details are slowly decreased from fig.3.R5.2 to fig.3.R5.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R5.5 to fig.3.R5.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R5.2 to fig.3.R5.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R5.1 and fig.3.R5.2. In the images left edges are more prominent which is more evident in fig.3.R5.1

3.3.3.4.20 RTLB: LBDAVFTC

3.3.3.4.20.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: LBDAVFTC option generates the following pictures (fig.3.R6.1-fig.3.R6.8) as output

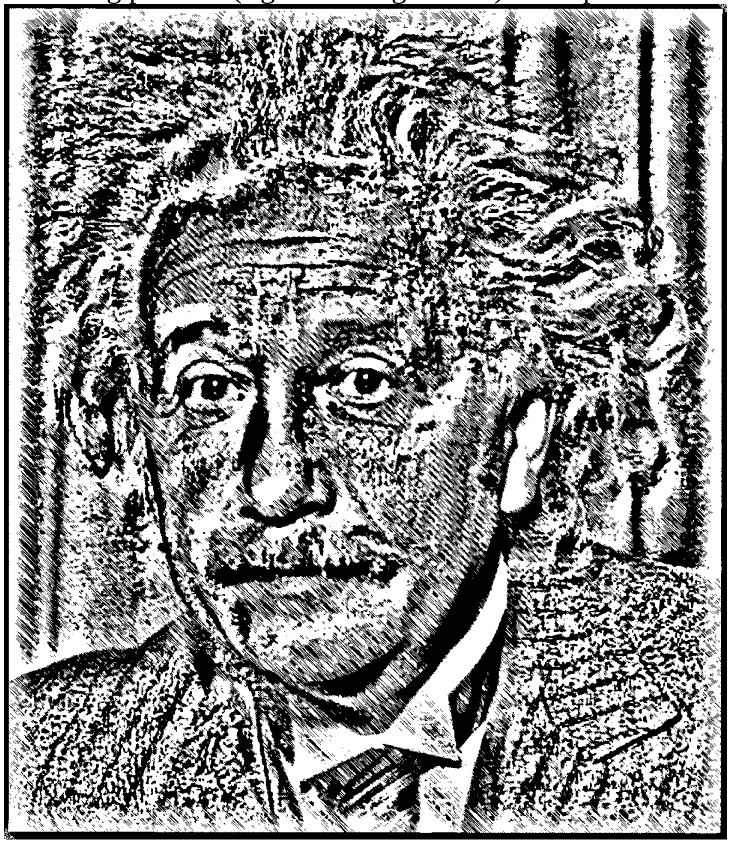


Figure 3.R6.1: (ein1R6)

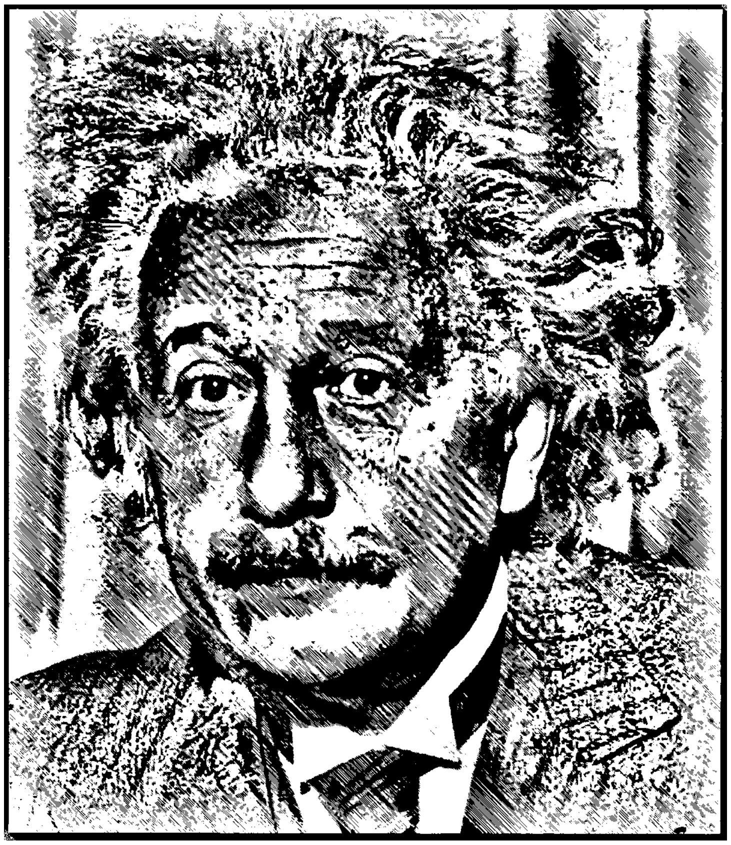


Figure 3.R6.2: (ein2R6)

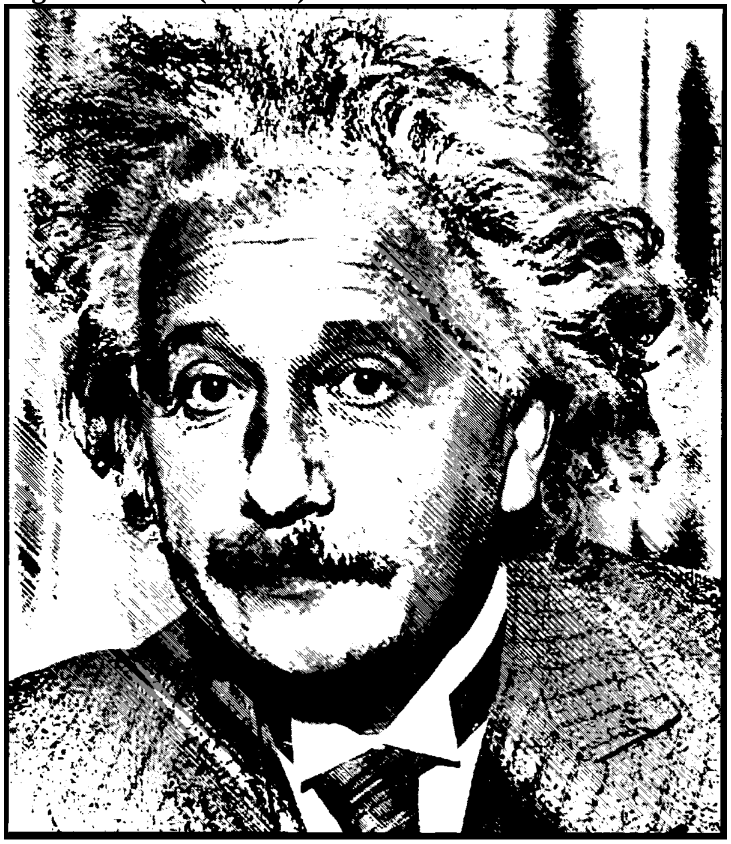


Figure 3.R6.3: (ein3R6)

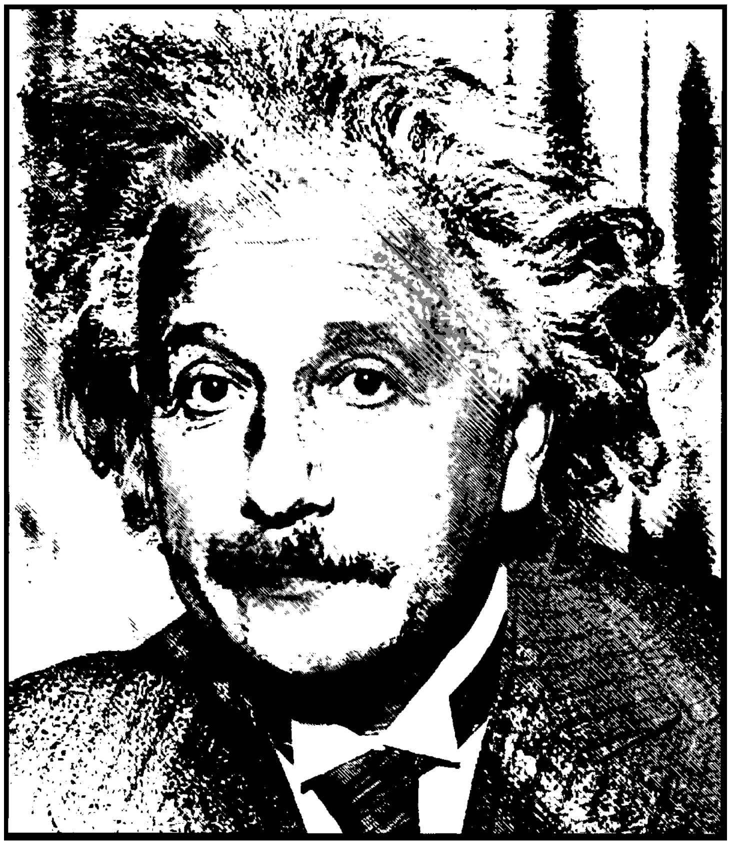


Figure 3.R6.4: (ein4R6)

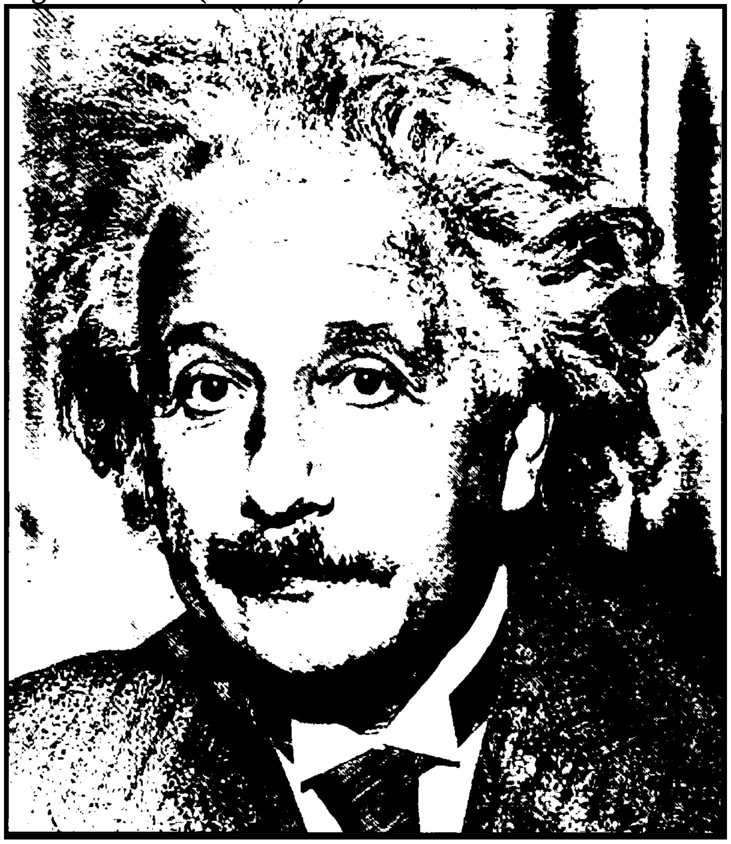


Figure 3.R6.5: (ein5R6)

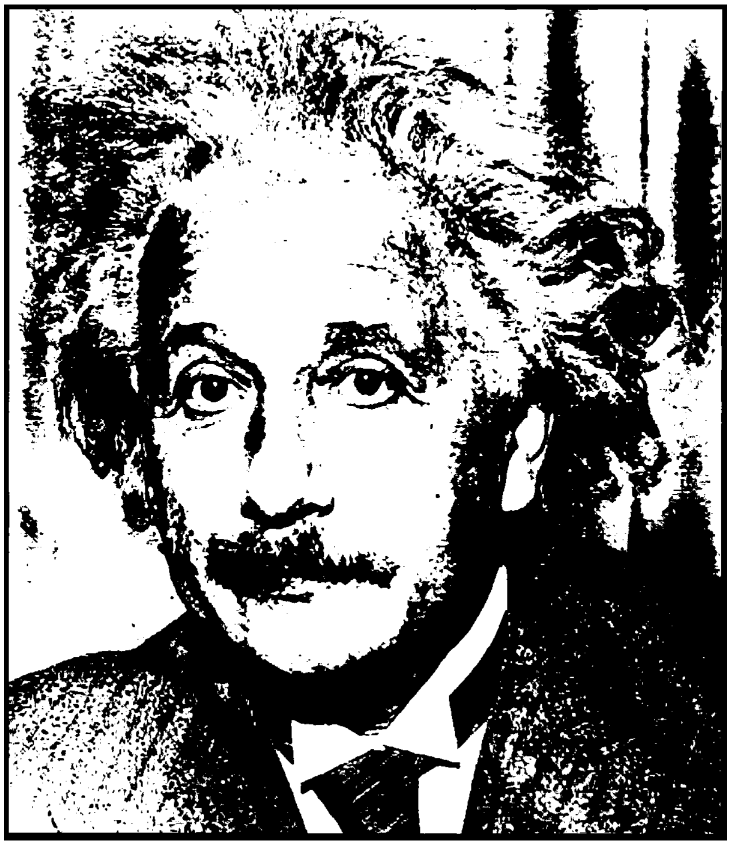


Figure 3.R6.6: (ein6R6)

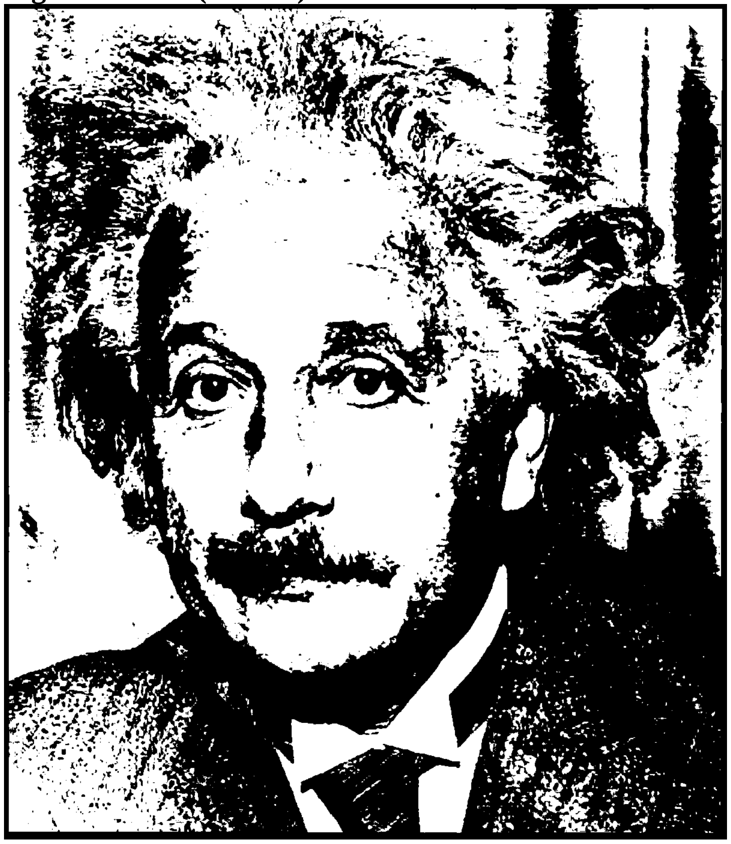


Figure 3.R6.7: (ein7R6)

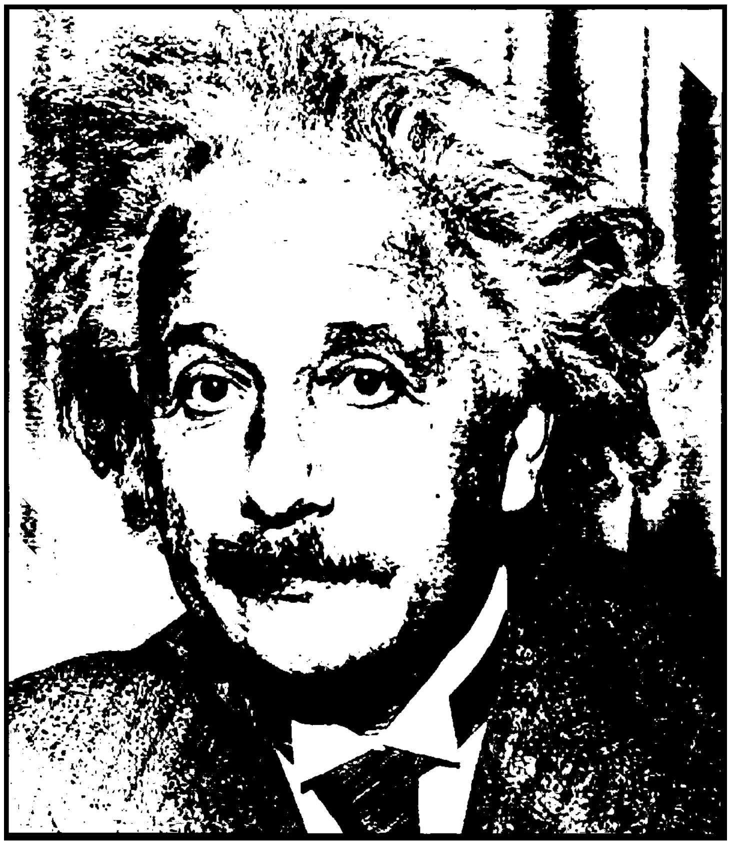


Figure 3.R6.8: (ein8R6)

3.3.3.4.20.2 Discussions

Filenames are given in parenthesis for the fig.3.R6.1-fig.3.R6.8. Fig.3.R6.1 contains more image details. Image details are slowly decreased from fig.3.R6.2 to fig.3.R6.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R6.5 to fig.3.R6.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R6.2 to fig.3.R6.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R6.1 and fig.3.R6.2. In the images left edges are more prominent which is more evident in fig.3.R6.1

3.3.3.4.21 RTLB: LBDVLT2RB

3.3.3.4.21.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: LBDVLT2RB option generates the following pictures (fig.3.R7.1-fig.3.R7.8) as output

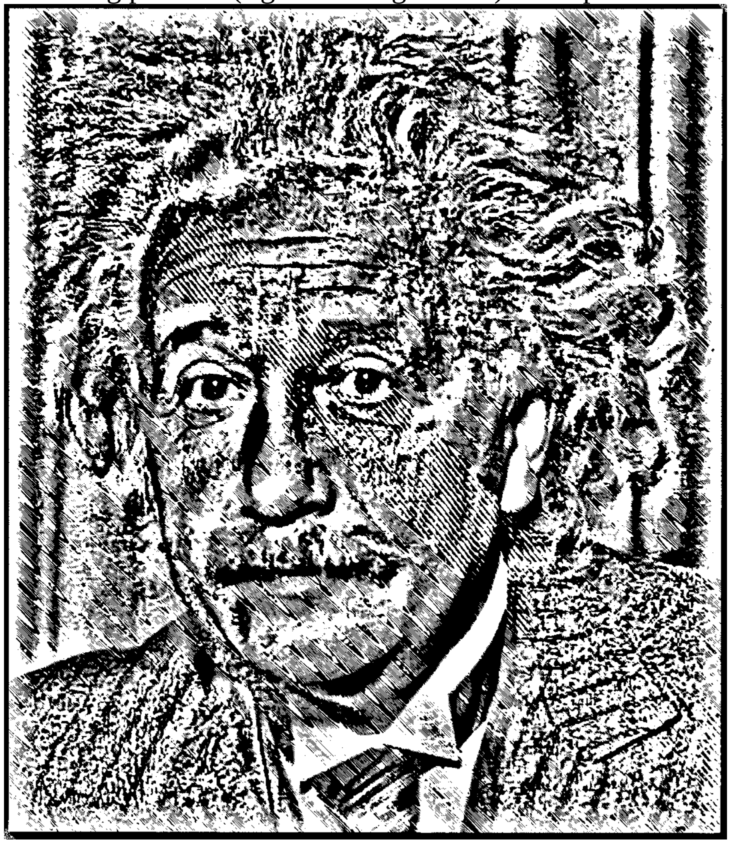


Figure 3.R7.1: (ein1R7)

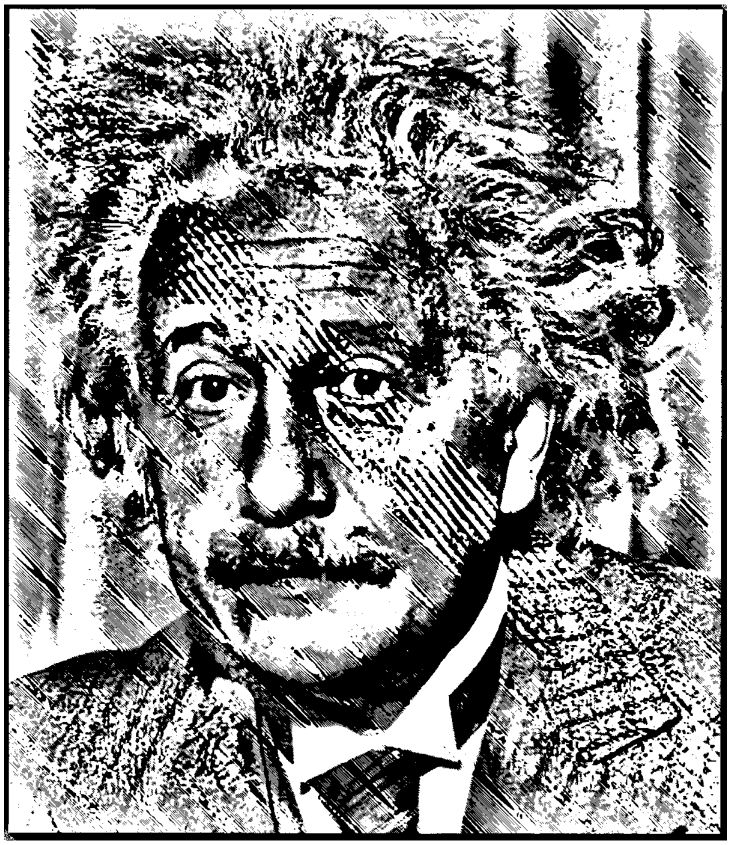


Figure 3.R7.2: (ein2R7)

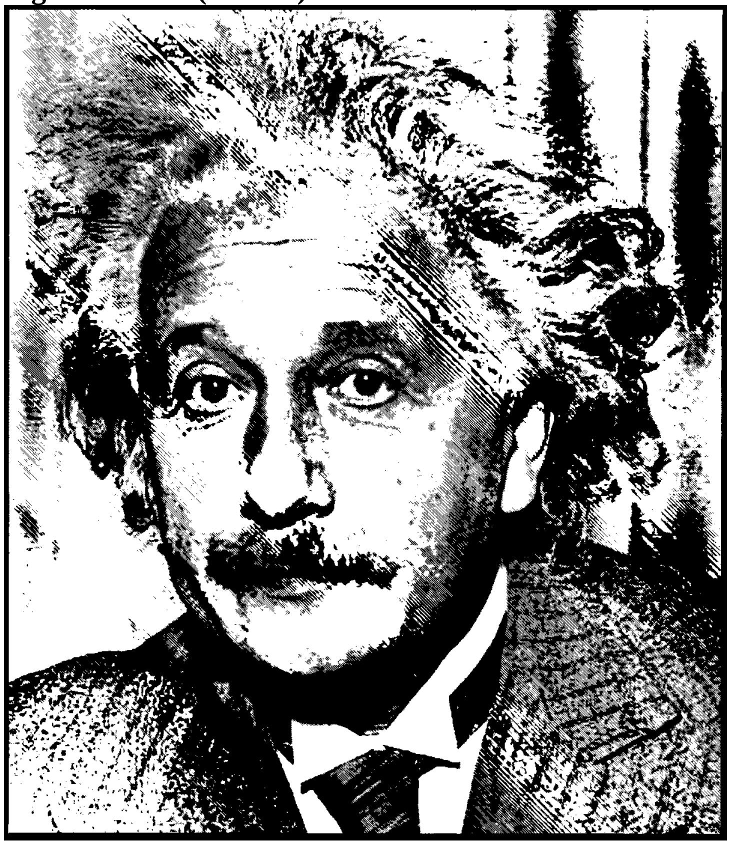


Figure 3.R7.3: (ein3R7)

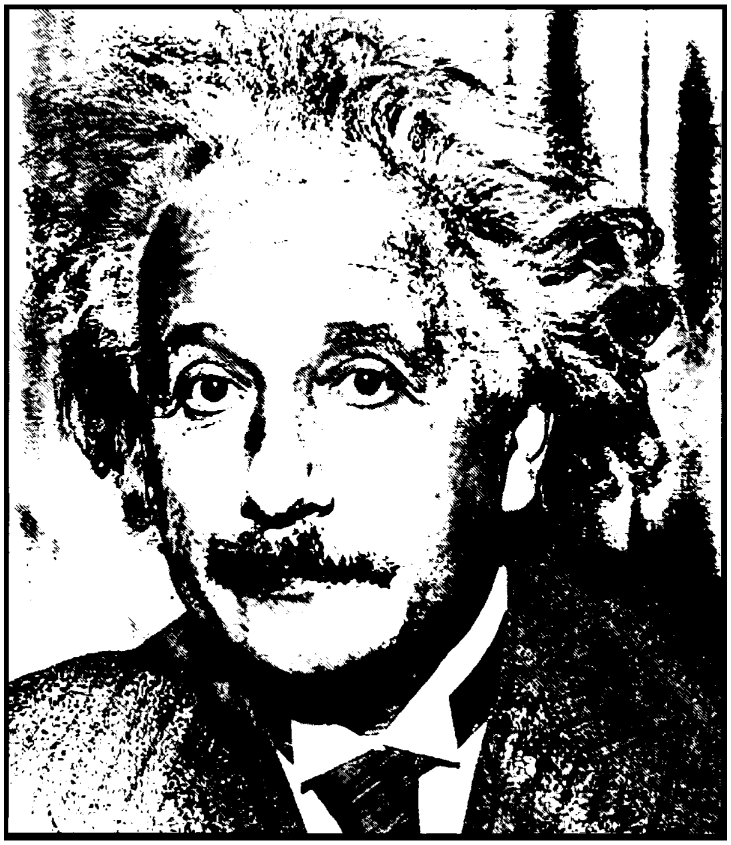


Figure 3.R7.4: (ein4R7)

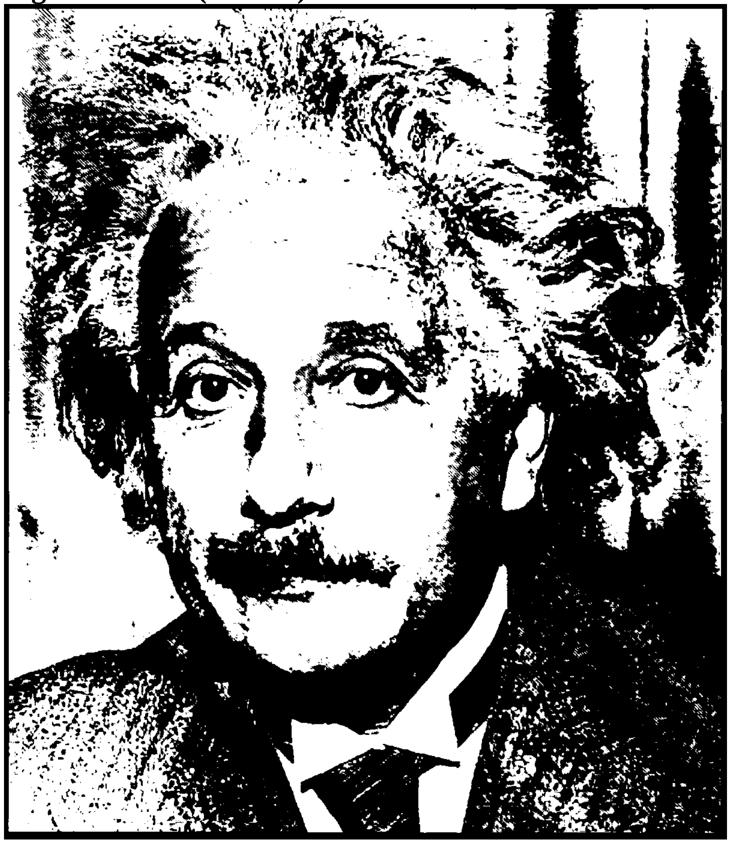


Figure 3.R7.5: (ein5R7)

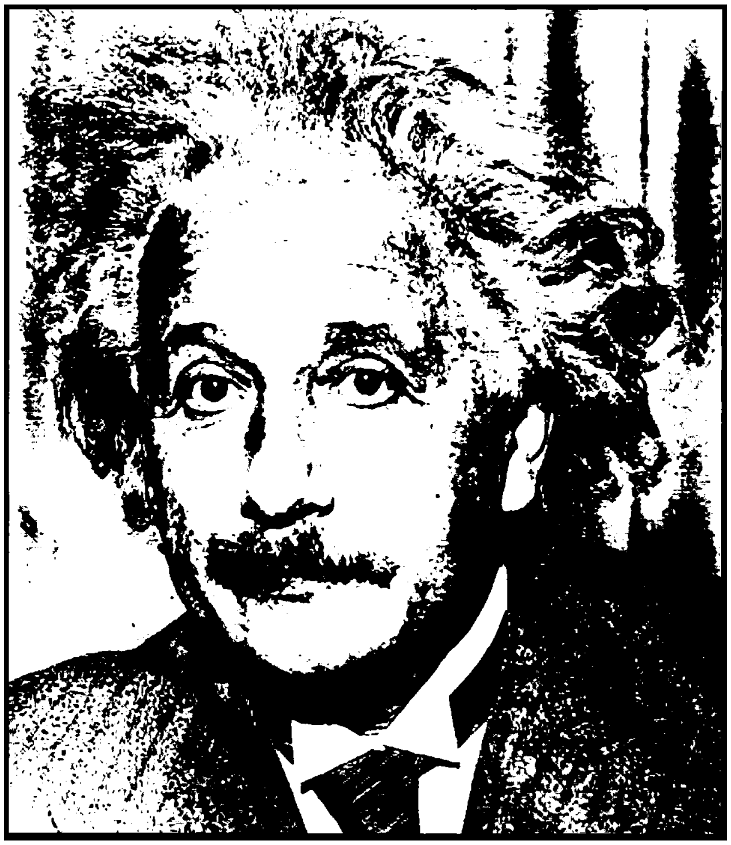


Figure 3.R7.6: (ein6R7)

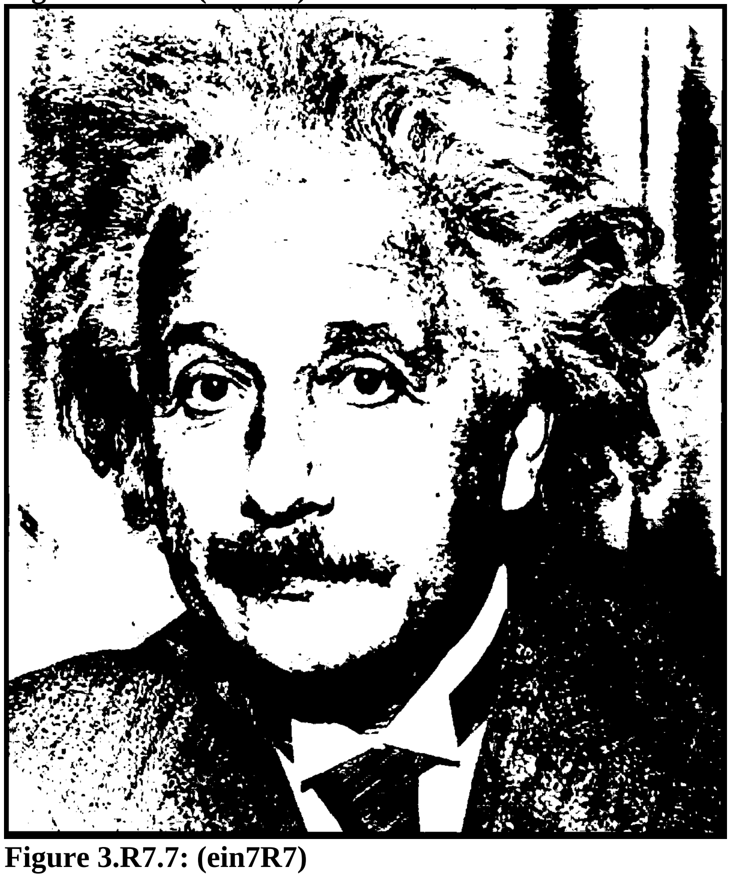


Figure 3.R7.7: (ein7R7)

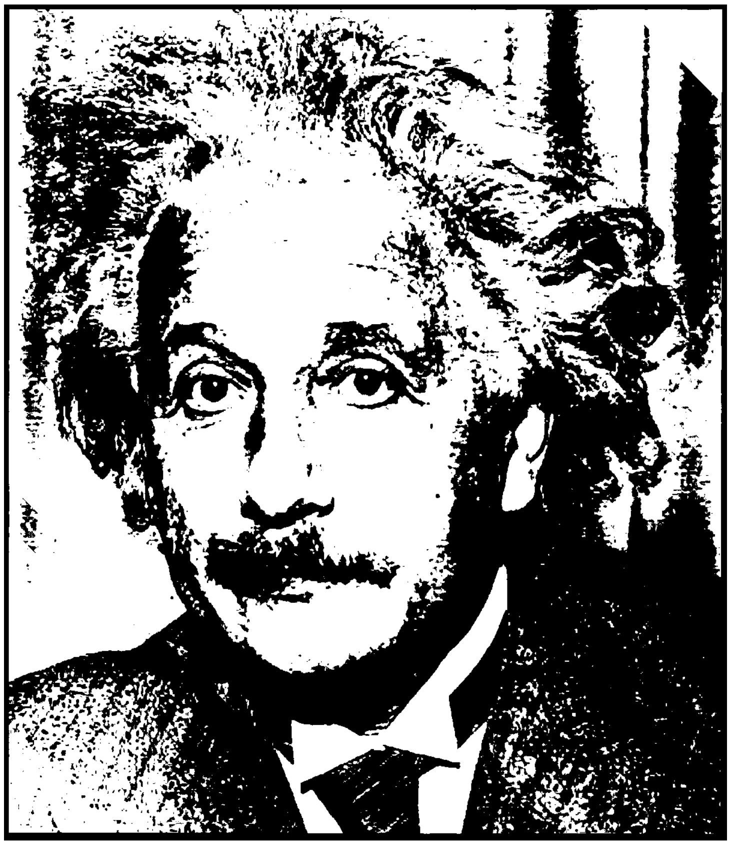


Figure 3.R7.8: (ein8R7)

3.3.3.4.21.2 Discussions

Filenames are given in parenthesis for the fig.3.R7.1-fig.3.R7.8. Fig.3.R7.1 contains more image details. Image details are slowly decreased from fig.3.R7.2 to fig.3.R7.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R7.5 to fig.3.R7.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R7.2 to fig.3.R7.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R7.1 and fig.3.R7.2. In the images left edges are more prominent which is more evident in fig.3.R7.1

3.3.3.4.22 RTLB: LBDVRB2LT

3.3.3.4.22.1 Output Images

The processing of fig.3.B2 diagonally with RTLB: LBDVRB2LT option generates the following pictures (fig.3.R8.1-fig.3.R8.8) as output

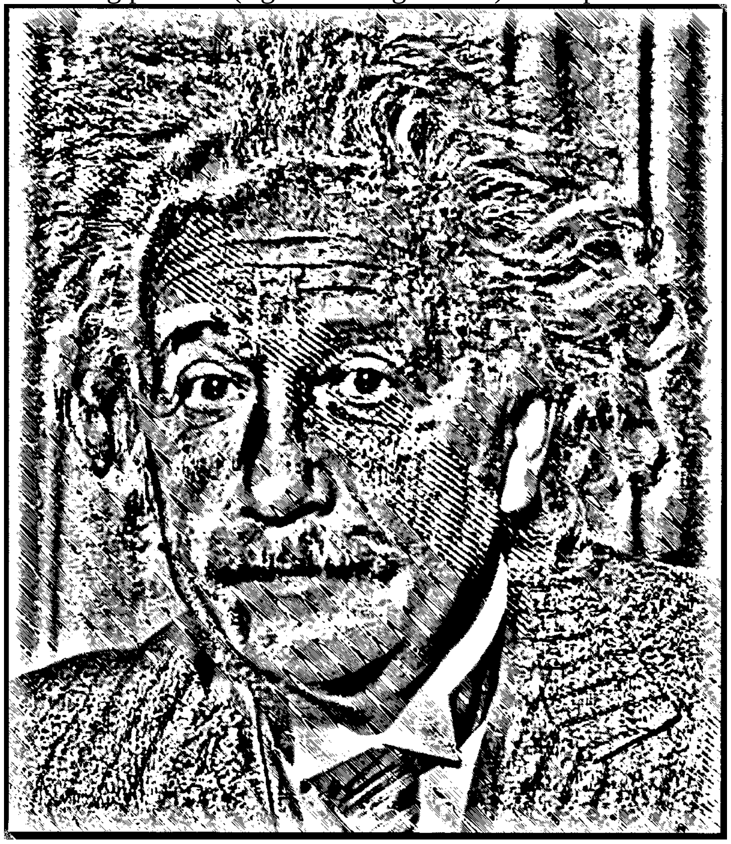


Figure 3.R8.1: (ein1R8)

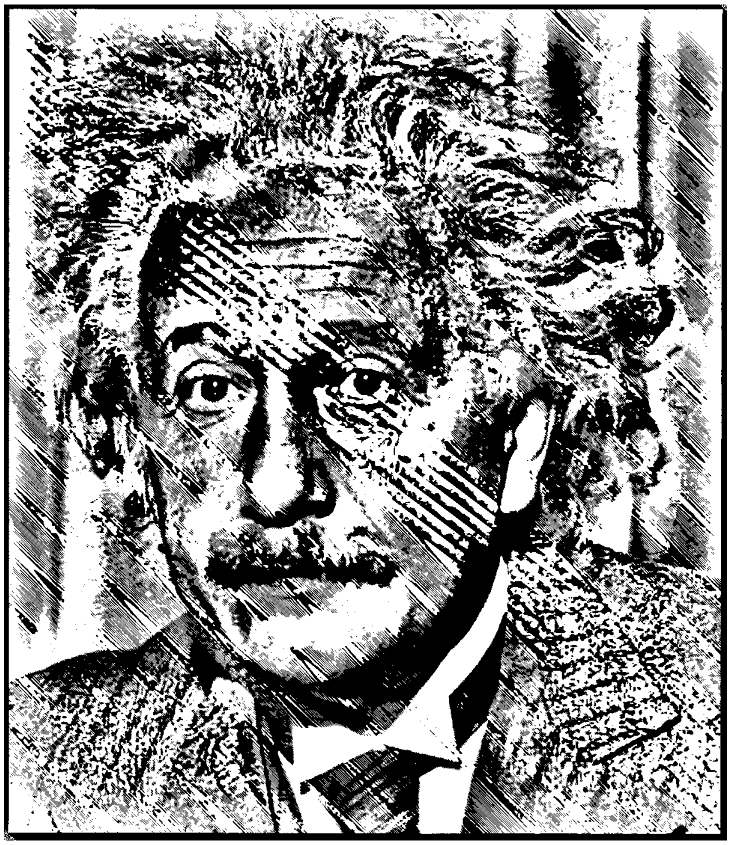


Figure 3.R8.2: (ein2R8)

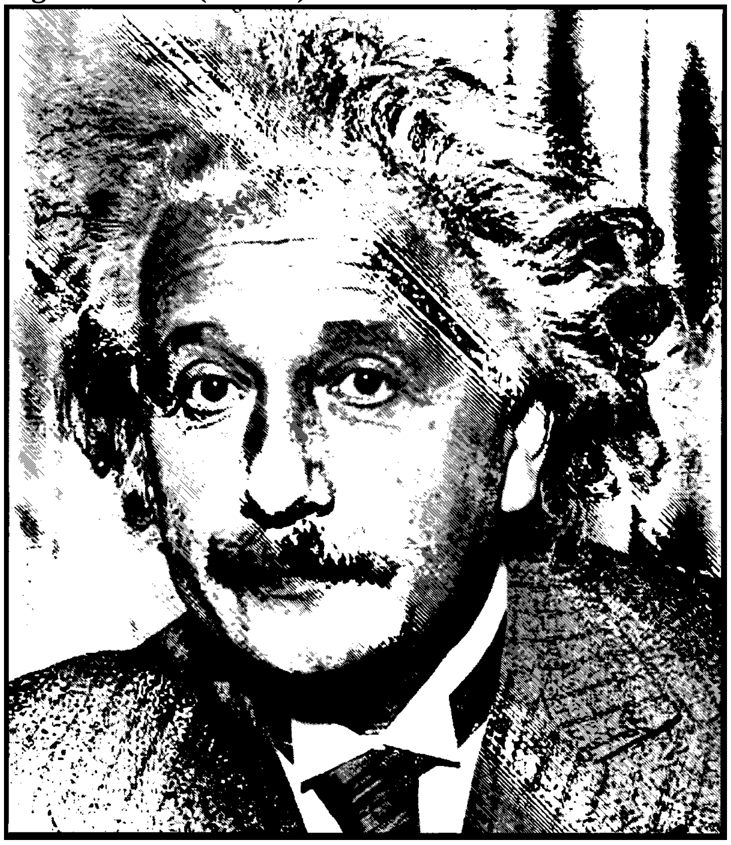


Figure 3.R8.2: (ein3R8)

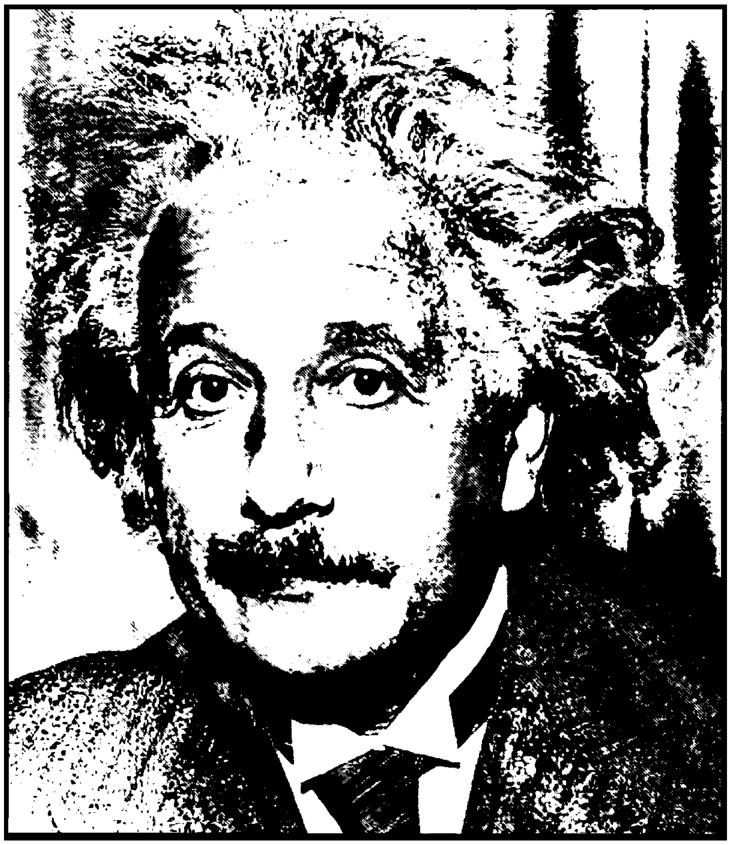


Figure 3.R8.4: (ein4R8)

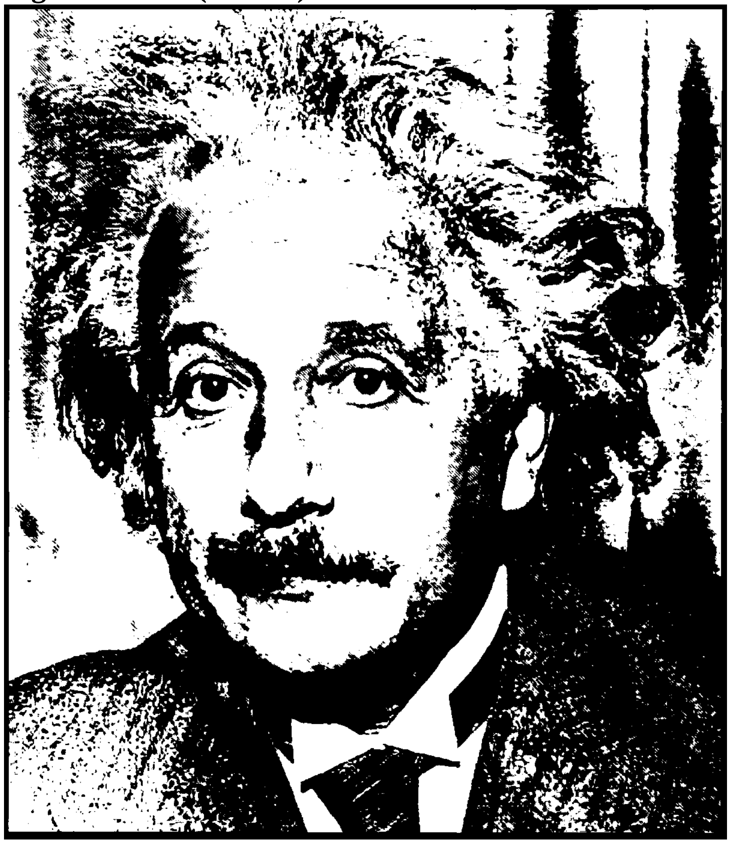


Figure 3.R8.5: (ein5R8)

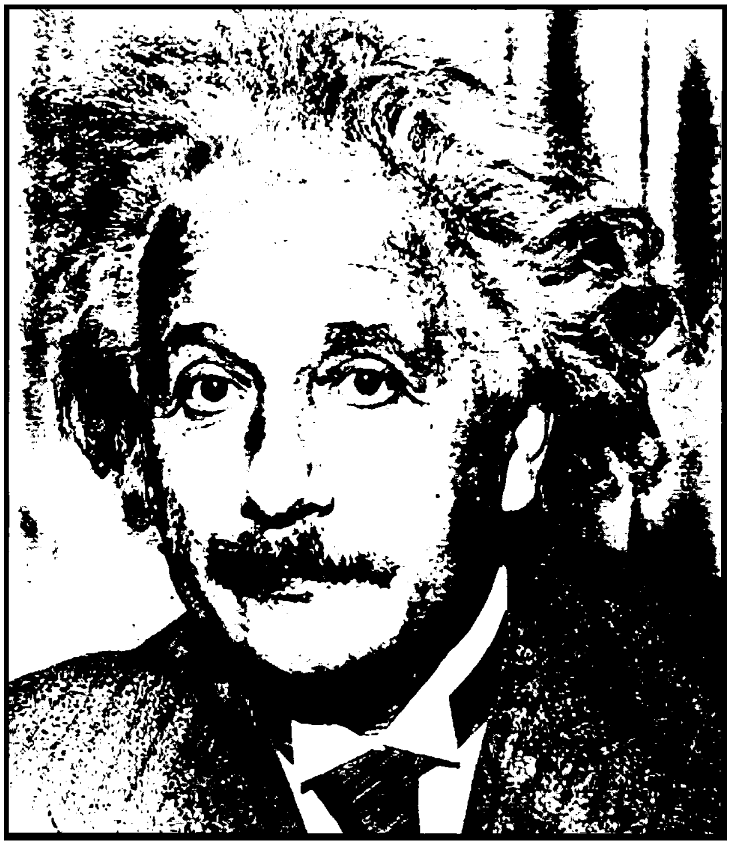


Figure 3.R8.6: (ein6R8)

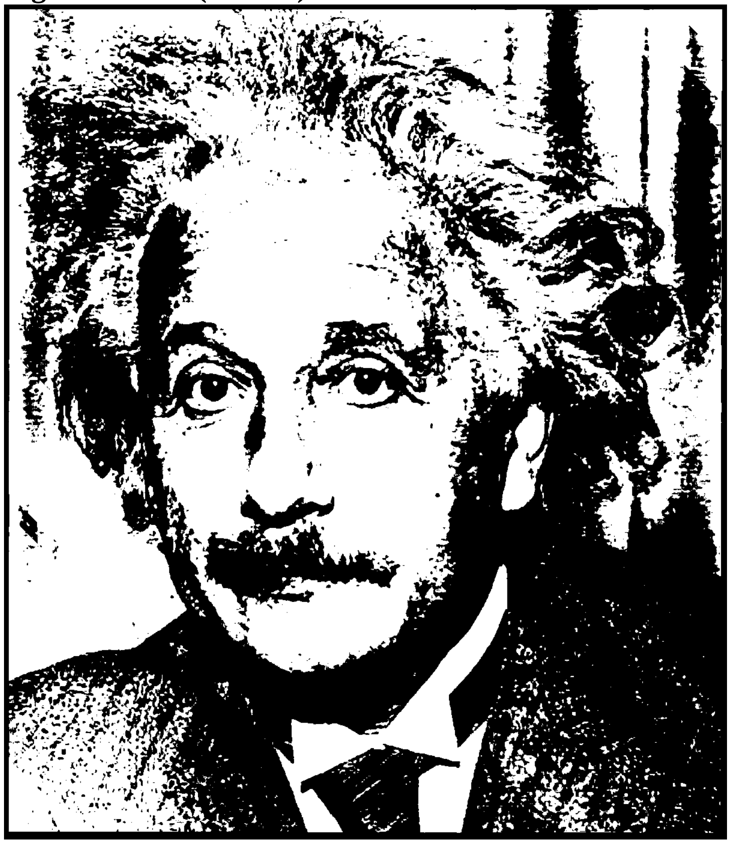


Figure 3.R8.7: (ein7R8)

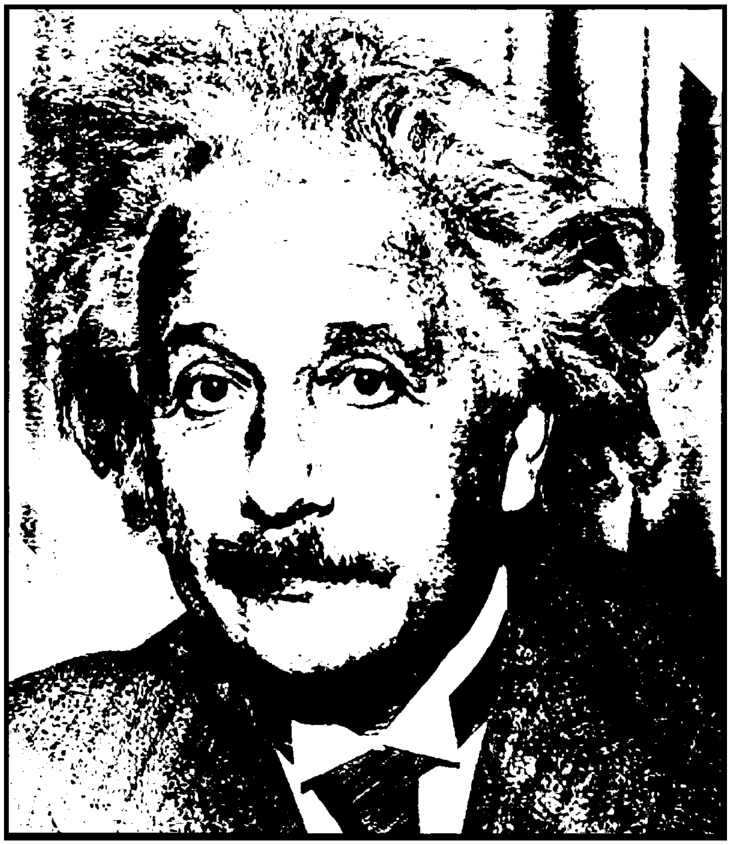


Figure 3.R8.8: (ein8R8)

3.3.3.4.22.2 Discussions

Filenames are given in parenthesis for the fig.3.R8.1-fig.3.R8.8. Fig.3.R8.1 contains more image details. Image details are slowly decreased from fig.3.R8.2 to fig.3.R8.5 as we have increased the nop (number of pixels) processed per groups for images but from fig.3.R8.5 to fig.3.R8.8 the decrease in image details is almost constant. There is a sharp decrease is image details from fig.3.R8.2 to fig.3.R8.3 because of sharp increase in nop. Lines are visible (from left to inclined towards right), in the direction of processing which is evident in fig.3.R8.1 and fig.3.R8.2. In the images left edges are more prominent which is more evident in fig.3.R8.1

3.3.3.4.23 Program 3.8

The following program function (in matlab 6.1) selects sequentially groups of pixels of an image matrix or matrix along the diagonal spiral or zigzag path and thresholds the selected group of pixels and generate the final non halftone binary image.

```
function DiagThresh(im,nop,finput,opt);
%------------------------------------------------------------
%------------------------------------------------------------
% DiagThresh(im,nop,finput,opt)
% DIAGTHRESH (function) : DIAGonal THRESHold
%
% This function selects sequentially groups of pixels of an
% image matrix or matrix along the diagonal spiral or zigzag
% path  and thresholds the selected group of pixels and generate
% the final non halftone binary image.
%
% INPUT ARGUMENTS:
% im: gray image file
% nop: no of pixels
%
% finput: the input function
%      : DiagVectorLTRB or DiagVectorRTLB
%
% EXAMPLES:
% 1.
% DiagThresh('flgray.tif',100,'DiagVectorLTRB',1)
% 2.
% im=magic(16);
% DiagThresh(im,100,'DiagVectorLTRB',1)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%------------------------------------------------------------
%------------------------------------------------------------
```

```
if  ischar(im)
   im=imread(im);
else
   im;
end

[r c dim]=size(im);

tn=feval(finput,r,c,opt)
rc=r*c;
L=length(tn);
nop;

rm=rem(rc,nop);
div=rc/nop;
fl=floor(div);
rm2=nop*fl;
rm3=rc-rm2;

[r2 c2 dim]=size(tn);
th2=[];
th4=[];
th5=[];

for i=0:nop:rc

   if  i+nop<=rc
      tn2=tn(:,i+1:i+nop);
      th=Thresh(im(tn2),1,1,0);
      th2=[th2 th];
   end
```

```
  if  rm3~=0
     tn3=tn(:,end-rm3+1:end);
     th4=Thresh(im(tn3),1,1,0);
  end

end
th5=[th2 th4];

LL=length(th5);
tn;

im(tn)=th5(:);
im=logical(im);
imshow(im)
imwrite(im,'diagthresh.tif')
%---------------------------------------------------------
```

3.3.3.4.24 Program 3.9

The following program function (in matlab 6.1) selects each diagonal vectors starting from left top or right bottom of a gray image matrix or matrix either alternately in spiral path or in a unidirectional way.

```
function [dvpath, mtp]=DiagVectorLTRB(r,c,opt)
%-----------------------------------------------------------
%-----------------------------------------------------------
% DiagVectorLTRB(r,c,opt)
% DIAGVECTORLTRB (function) : DIAGonal Vector Left Top Right Bottom
%
% This function selects each diagonal vectors starting from
% left top or right bottom of a gray image matrix or matrix either
% alternately in spiral path or in a unidirectional way.
%
% Uses: Image processing path or mask matrix for halftoning or
%      matrix for pattern generation.
%
% INPUT ARGUMENTS:
% r: no of rows
% c: no of columns
%
% Opt: options
% 1: LTDAVFTC: Left top (starting point) diagonal alternate vector spiral
%            with first turn clockwise
% 2: LTDAVFTAC: Left top diagonal alternate vector spiral with
%             first turn anti-clockwise
% 3: LTDVLB2RT: Left top diagonal vectors, left bottom to right top
% 4: LTDVRT2LB: Left top diagonal vectors, right top to left bottom
% 5: RBDAVFTAC: Right bottom (starting point) diagonal alternate vector
%             spiral with first turn anti-clockwise (5 is reverse of 1)
% 6: RBDAVFTC: Right bottom  diagonal alternate vector spiral with
%            first turn clockwise (6 is reverse of 2)
% 7: RBDVRT2LB: Right bottom diagonal vectors, right top to left bottom
%           (7 is reverse of 3)
% 8: RBDVLB2RT: Right bottom diagonal vectors, left bottom to right top
%           (8 is reverse of 4)
%
% OUTPUT ARGUMENTS:
% dvpath: diagonal vector path consisting of pixel locations
% mtp: matrix for thresholding or pattern
%
% EXAMPLE:
% [dvpath mtp]=DiagVectorLTRB(4,5,1)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%------------------------------------------------------------
%------------------------------------------------------------
dmx=min(r,c); % max size of the diagonal

% Starting locations of the diagonals for LTH (Left Top Half)
% on the location matrix
s1=1:r;

% Starting locations of the diagonals for RBH (Right Bottom Half)
% on the location matrix
s2=2*r:r:r*c;

% All the starting locations of the diagonals
% on the location matrix
s=[s1 s2];

L=length(s);
n2=0;
t2=0;
tn1=[];
tn2=[];
for st=1:L
```

```
  st2=s(st); % acessing the starting locations serially
  n=st;
  if  n>dmx
    n2=n2+1;
    n=n-n2;
  end

if  n<=dmx
  % [tn,mts,lt]=locmap(st,r,n,sg,d,fnop)
   [tn,mts,lt]=locmap(st2,r,n,'p',4,1);

  if  lt>=r*c
    te=tn(:,end-t2:end);
    t2=t2+1;
    tn;
    L2=length(te);
    for i=1:L2
      k=find(te(i)==tn);
      tn(k)=[];
    end

    if opt==1 | opt==5
      if  st~=1 & rem(st,2)~=1
        tn=fliplr(tn); % diagonals
      else
        tn;
      end
        tn2=[tn2 tn];
    end
    if opt==2 | opt==6
      if  st~=1 & rem(st,2)==1
        tn=fliplr(tn); % diagonals
      else
        tn;
        end
           tn2=[tn2 tn];
    end
    if opt==3 | opt==7
        if  st~=1
           tn;
        end
           tn2=[tn2 tn];
    end
    if opt==4 | opt==8
        if  st~=1
           tn=fliplr(tn); % diagonals
        else
           tn;
        end
           tn2=[tn2 tn];
    end

else
    if opt==1 | opt==5
        if  st~=1 & rem(st,2)~=1
           tn=fliplr(tn); % diagonals
        else
           tn; % diagonals
        end
           tn1=[tn1 tn];
    end
    if opt==2 | opt==6
        if  st~=1 & rem(st,2)==1
           tn=fliplr(tn); % diagonals
        else
           tn;
        end
           tn2=[tn2 tn];
```

```
      end
      if opt==3 | opt==7
        if  st~=1
          tn;
        end
          tn2=[tn2 tn];
      end
      if opt==4 | opt==8
        if  st~=1
          tn=fliplr(tn); % diagonals
        else
          tn;
        end
          tn2=[tn2 tn];
      end

    end

  end
end

tn1
tn2
dvpath=[tn1 tn2] % diagonal vector path

if opt==5 | opt==6 | opt==7 | opt==8
  dvpath=fliplr(dvpath)
end

% Generation of matrix for threshold or pattern
z=zeros(r,c);
z(dvpath)=1:length(dvpath);
mtp=z % mtp:matrix for threshold or pattern
%-------------------------------------------------------------
```

3.3.3.4.25 Program 3.10

The following program function (in matlab 6.1) selects each diagonal vectors starting from right top of an image matrix or matrix either spirally or in zigzag way.

```
function [dvpath,mtp]=DiagVectorRTLB(r,c,opt)
%--------------------------------------------------------
%--------------------------------------------------------
% DiagVectorRTLB(r,c,opt)
% DIAGVECTORRTLB (function) : DIAGonal Vector Right Top Left Bottom
%
% This function selects each diagonal vectors starting from
% right top of an image matrix or matrix either spirally or
% in zigzag way.
%
% INPUT ARGUMENTS:
% r: no of rows
% c: no of columns
%
% Opt: options
% 1: RTDAVFTC:  Right top diagonal alternate vector spiral with
%           first turn clockwise
% 2: RTDAVFTAC: Right top diagonal alternate vector spiral with first turn
%           anti-clockwise
% 3: RTDVRB2LT: Right top diagonal vectors, right bottom to left top
% 4: RTDVLT2RB: Right top diagonal vectors, left top to right bottom
% 5: LBDAVFTAC: Left bottom (starting point) diagonal alternate vector
%           spiral with first turn anti-clockwise (5 is reverse of 1)
% 6: LBDAVFTC:  Left bottom  diagonal alternate vector spiral with
%           first turn clockwise (6 is reverse of 2)
% 7: LBDVLT2RB: Left bottom diagonal vectors, left top to right bottom
%           (7 is reverse of 3)
% 8: LBDVRB2LT: Left bottom diagonal vectors, right bottom to left top
%             (8 is reverse of 4)
%
% OUTPUT ARGUMENTS:
% dvpath: diagonal vector path consisting of pixel locations
% mtp: matrix for thresholding or pattern
%
% EXAMPLE:
% [dvpath,mtp]=DiagVectorRTLB(4,5,1)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%-----------------------------------------------------------
%-----------------------------------------------------------
dmx=min(r,c); % max size of the diagonal

SL=r*c-r+1 % First starting location

% Starting locations of the diagonals for RTH (Right Top Half)
% on the location matrix
s1=SL:(r*c)

% Starting locations of the diagonals for LBH (Left Bottom Half)
% on the location matrix
s2=(r*c-r):-r:r

% All the starting locations of the diagonals
% on the location matrix
s=[s1 s2];

L=length(s);
n2=0;
t2=0;
tn1=[];
```

```
tn2=[];
for st=1:L
        st2=s(st); % acessing the starting locations serially
        n=st;
        if  n>dmx
            n2=n2+1;
            n=n-n2;
        end

    if  n<=dmx
        % [tn,mts,lt]=locmap(st,r,n,sg,d,fnop)
          [tn,mts,tl]=locmap(st2,r,n,'n',2,1);

       if opt==1
            if  st~=SL & rem(st,2)==1 % flip when starting position is odd
                tn=fliplr(tn); % diagonals
            else
                tn; % diagonals
            end
            tn1=[tn1 tn];
       end

       if opt==5
            if  st~=SL & rem(st,2)==1 % flip when starting position is odd
                tn=fliplr(tn); % diagonals
            else
                tn; % diagonals
            end
            tn1=[tn1 tn];
       end

       if opt==2
    if  st~=SL & rem(st,2)~=1 % flip when starting position is even
       tn=fliplr(tn); % diagonals
    else
       tn; % diagonals
    end
    tn1=[tn1 tn];
end

if opt==6
    if  st~=SL & rem(st,2)~=1 % flip when starting position is odd
       tn=fliplr(tn); % diagonals
    else
       tn; % diagonals
    end
    tn1=[tn1 tn];
end

if opt==3 | opt==7
    if  st~=SL
       tn; % diagonals
    end
    tn1=[tn1 tn];
end

if opt==4 | opt==8
    if  st~=SL % flip to change the vector direction
       tn=fliplr(tn); % diagonals
    else
       tn; % diagonals
    end
    tn1=[tn1 tn];
end
```

```
   end
end

% Final locations along the diagonal
tn1;
dvpath=tn1;

if  opt==5 | opt==6 | opt==7 | opt==8
   dvpath=fliplr(dvpath);
end

% Generation of matrix for threshold or pattern
z=zeros(r,c);

L4=length(dvpath);
sn=1:L4;

z(dvpath)=sn(:);
mtp=z % mtp:matrix for threshold or pattern
%-----------------------------------------------------------
```

3.3.3.4.26 Program 3.11

The following program function (in matlab 6.1) creates any user defined path using pixel locations as mapped by a location matrix

```
function [tn,mts,lt]=LocMap(st,r,n,sg,dom,fnop)
%----------------------------------------------------------
%----------------------------------------------------------
% [tn,mts,lt]=LocMap(st,r,n,sg,dom,fnop)
% LOCMAP (function) : LOCation MAPping
%
% This function creates any user defined path using pixel
% locations as mapped by a location matrix
%
% INPUT ARGUMENTS:
% st: starting location value
% r: number of rows of the location matrix
% n: no of numbers required to be generated for a series
% sg: sign, 'p' positive, 'n' negative
% dom: direction of movement (options for positive sign)
%    (movement in graphics coordinate plane is given in the
%    parenthesis)
%  1:  1   : vertically from top to bottom (y-)
%  2:  r+1 : top to bottom but inclined towards right (x+,y-)
%  3:  r   : left to right (x+)
%  4:  r-1 : bottom to top but inclined towards right (x+,y+)
% for negative sign movement is reverse
% fnop: if the starting number of the series is wanted, put 1
%       else 2,3 etc depending on the number location required.
%
% OUTPUT ARGUMENTS:
% tn: total numbers
% mts: middle terms
% lt: last term
%
% EXAMPLES:
% 1.
% [tn,mts,lt]=LocMap(1,7,20,'p',4,1)
% 2.
% [tn,mts,lt]=LocMap(99,7,14,'n',2,1)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%-------------------------------------------------------------
%-------------------------------------------------------------
% Defining the movement options
if      dom==1;
        d=1;
elseif  dom==2;
        d=r+1;
elseif  dom==3;
        d=r;
else    dom==4;
        d=r-1;
end

% Path generation
tn=[];

if sg=='p'

    for i=fnop:n
        tr=st+(i-1)*d;
        if  tr>0;
            tn=[tn tr];
        end
    end

    L=length(tn);
    if  L>2;
       mts=tn(:,2:end-1);
    else
       mts=[];
    end
    lt=tn(:,end);

else % 'n'

    for i=fnop:n
       tr=st-(i-1)*d;
       if  tr>0;
          tn=[tn tr];
       end
    end

    L=length(tn);
    if  L>2
       mts=tn(:,2:end-1);
    else
       mts=[];
    end
    lt=tn(:,end);

end
%------------------------------------------------------------
```

3.3.3.4.27 Comparison with the Earlier Technique

Various non-halftone binary image transformation methods have been introduced in this chapter. These methods are unique in design and have their own place in imaging science and technology. These techniques are algorithm driven and must be produced through digital computers. In comparison to earlier analog technique like Tone Line Process that has been discussed in Chapter 1 (Section 1.3.1) these novel techniques are relatively faster and with better esthetic look and less costly. The image (fig.1.2 (a)) produced by tone line process is comparable to fig.3.B7 (an output image of blockwise processing) and are of almost same category. Other methods like processing the image, columnwise (e.g. fig.3.C3), in arbitrary path (e.g. fig.3.A2) or diagonally (e.g. fig.3.L1.3), produces non-halftone binary images of better kinds (because of their nature of processing) than Tone-Line Process.

Chapter 4

Digital Halftoning (Ordered Dither)

Chapter 4

Digital Halftoning (Ordered Dither)

4.1 Introduction

Here the author has presented two novel methods of ordered dither. There are different methods for dithering an image but ordered dither is the easiest to implement, as it does not require processing or storage of neighboring pixels. Ordered dither can be divided into two types by the nature of dots produced, clustered and dispersed. The most popular method, printer’s screen is generated using clustered-dot ordered dither by following the optical process used in printing industry for over 100 years. In offset printing, clustered-dots are needed where the area of each pixel is too small to hold the ink on the printing plate. For many electronic displays dispersed dot ordered dither is preferred where single-pixel constraints are not an issue.

The two novel digital halftoning methods [25] that are attempted here are as follows

1. Halftoning by pre-embedding the pattern.
2. Halftoning by simulating character writing pattern

In the first method a pattern is embedded with the image to be halftoned which is normally not done for other earlier methods of ordered dither. Then pattern embedded halftone is generated.

In the second method a mask matrix is generated from the writing pattern of a character. Then halftoning is done using a threshold matrix generated from mask matrix.

4.2 Halftoning by Pre-Embedding the Pattern

In this section a novel digital halftoning method is presented using fig.4.1 to fig.4.5.

Figure: 4.1

Mask matrix (Spiral)			
7	8	9	10
6	1	2	11
5	4	3	12
16	15	14	13

Figure: 4.2

Threshold matrix			
112	128	144	160
96	16	32	176
80	64	48	192
256	240	224	208

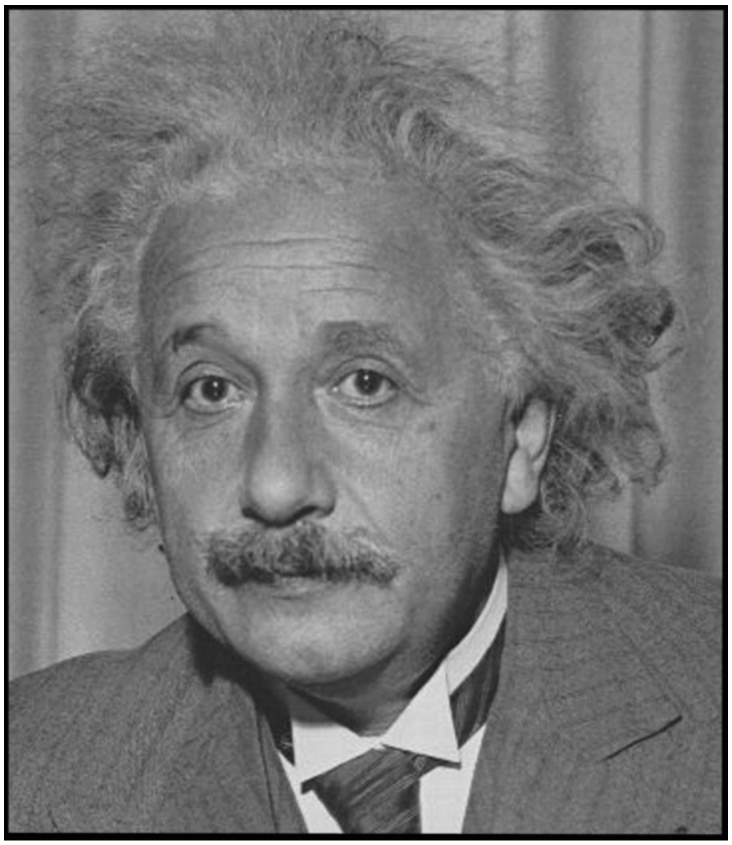


Figure 4.3: Sample Image

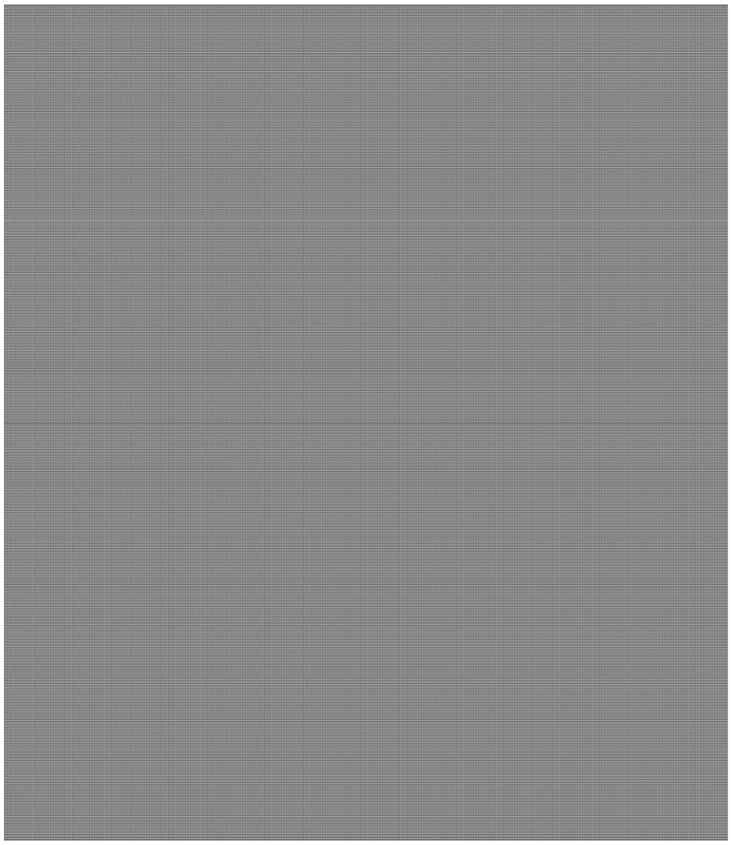


Figure 4.4 (a) Pattern to be embedded

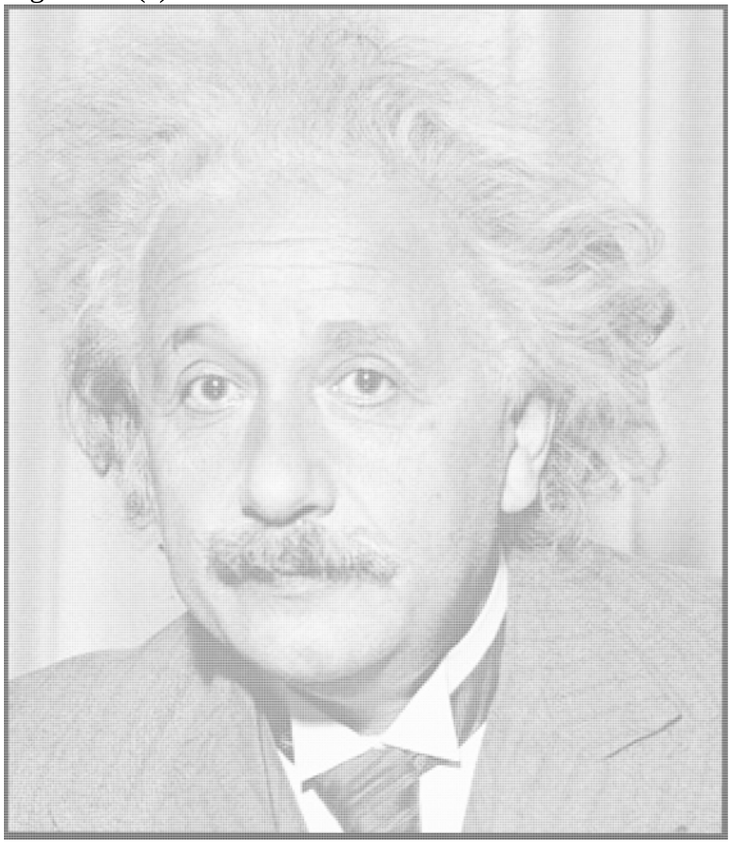


Figure 4.4 (b) Pattern embedded sample

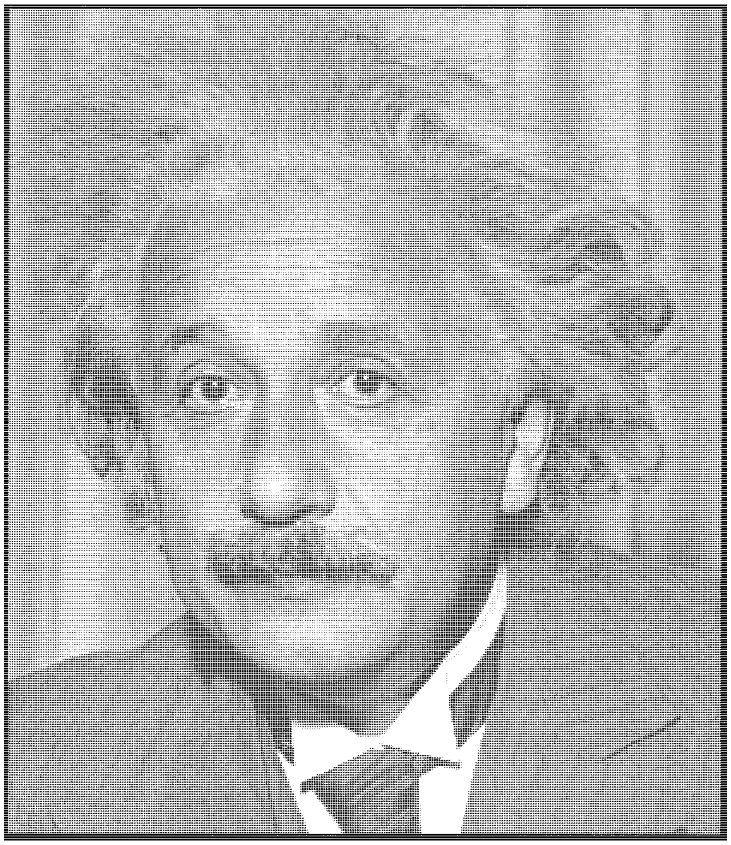


Figure 4.4 (c) Pattern embedded sample after halftoning

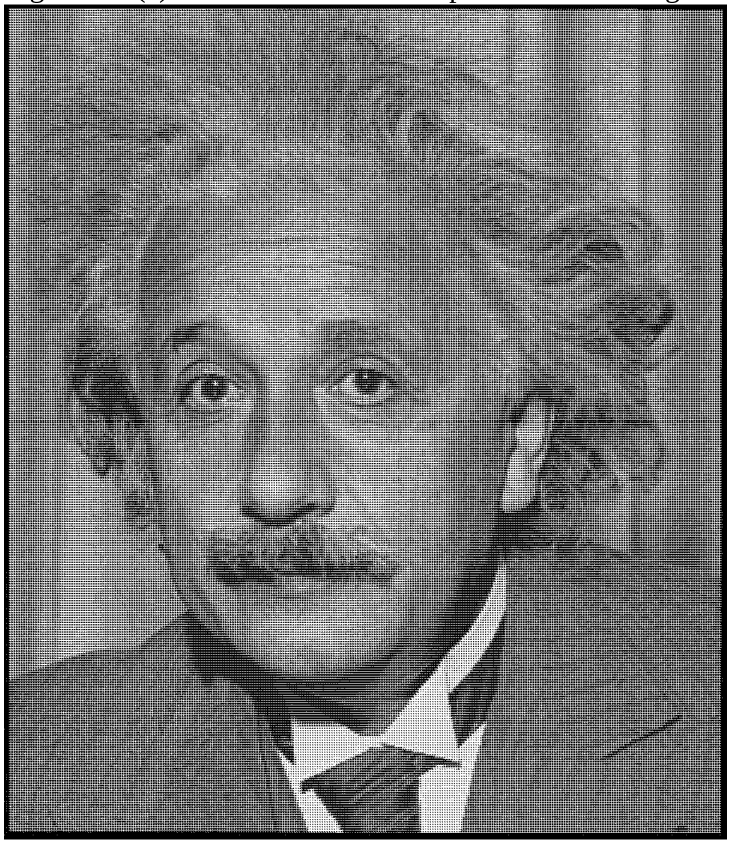


Figure 4.4 (d) Sample image of figure 3 is Halftoned using pattern

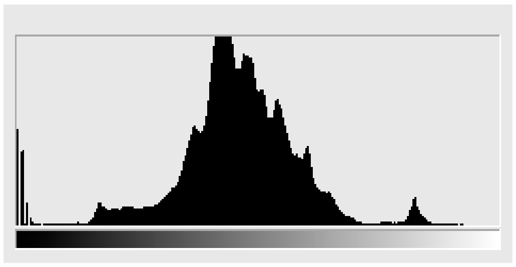


Figure 4.5(a)

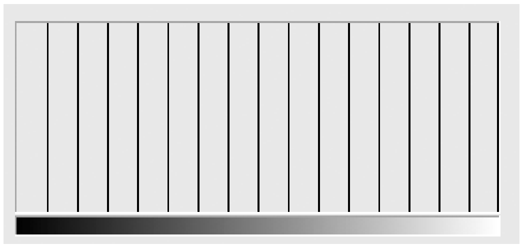


Figure 4.5(b)

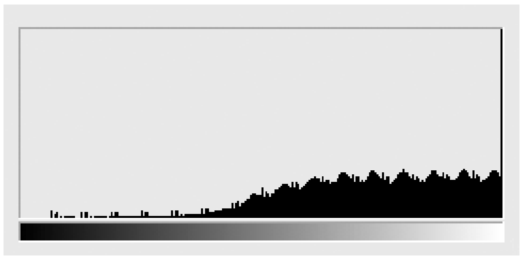


Figure 4.5(c)

Figure 4.5. (a) Histogram of the sample image of ‘figure 4.3’. (b) Histogram of the embedded pattern (c) Histogram of the embedded image

4.2.1 Algorithm (Method 1):

1. A pattern image corresponding to an order dither matrix (fig. 4.1) is created and then by adding the pattern column wise and row wise, a final pattern image (fig.4.4(a)) is obtained which is of same size as that of the gray sample image (fig.4.3).
2. Then gray sample image is embedded with the above final pattern image to get the pattern embedded sample (fig.4.4(b)).
3. After embedding the required pattern in the image then the threshold operation is done to get the final halftone (fig.4.4(c)).

4.3 Halftoning by Simulating Character-Writing Pattern

In this section a novel method of digital halftoning is presented using the fig.4.6 to fig.4.12.

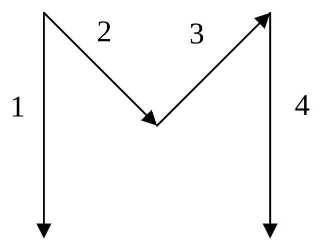


Figure: 4.6 Writing Stroke Sequences of a Character ‘M’

Figure: 4.7

Mask matrix							
0	0	0	0	0	0	0	0
0	1	7	0	0	14	15	0
0	2	8	0	0	13	16	0
0	3	0	9	12	0	17	0
0	4	0	10	11	0	18	0
0	5	0	0	0	0	19	0
0	6	0	0	0	0	20	0
0	0	0	0	0	0	0	0

Figure: 4.8

Threshold matrix with background of the character white							
255	255	255	255	255	255	255	255
255	4	28	255	255	56	60	255
255	8	32	255	255	52	64	255
255	12	255	36	48	255	68	255
255	16	255	40	44	255	72	255
255	20	255	255	255	255	76	255
255	24	255	255	255	255	80	255
255	255	255	255	255	255	255	255

Figure 4.9. Sample Image grayscale

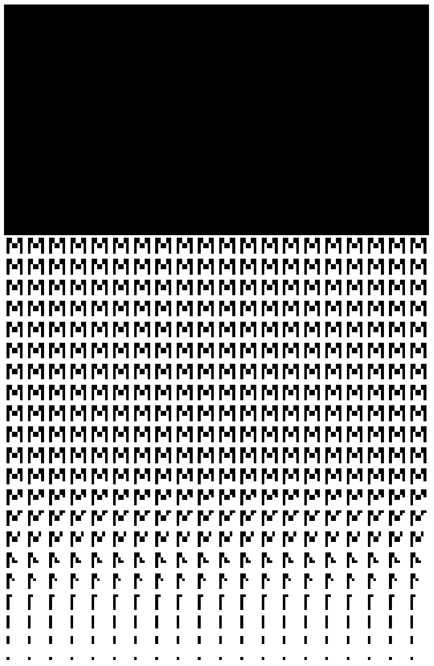


Figure 4.10. Sample image of figure 4.9 is halftoned using the matrix of figure 4.8

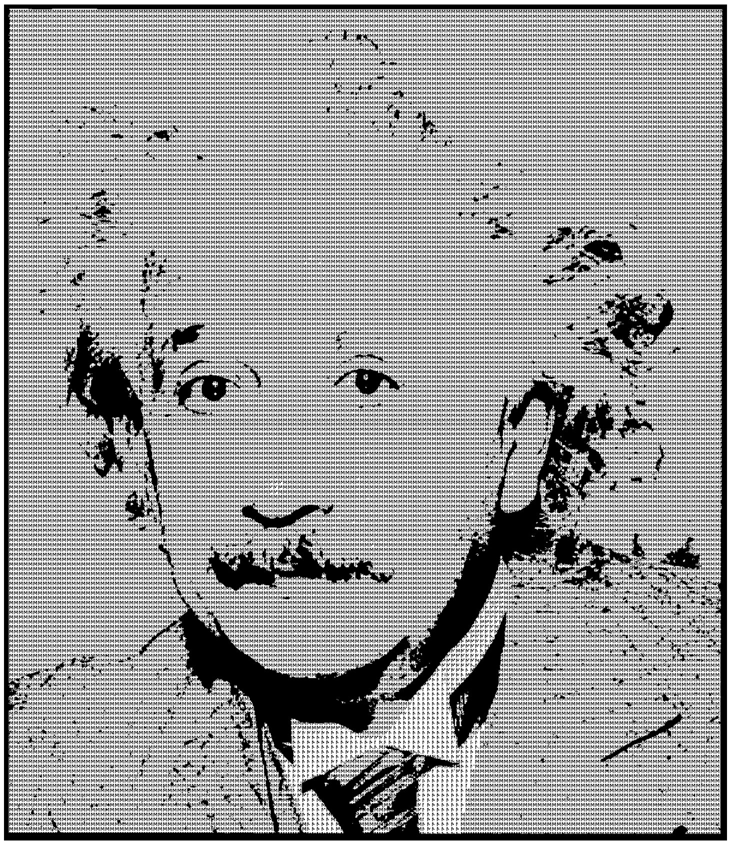
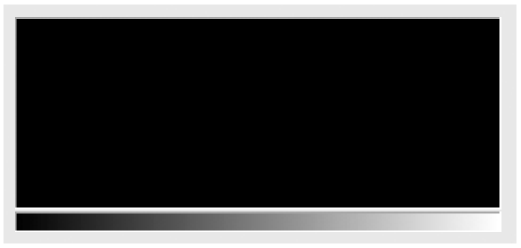


Figure 4.11. Sample image of figure 4.3 is halftoned using the matrix of figure 4.8


(a)

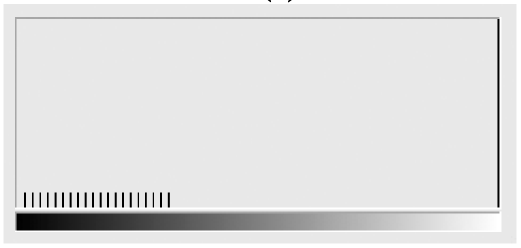

(b)

Figure 4.12. (a) Histogram of the grayscale ramp (0-255) image of ‘figure 4.9’. (b) Histogram of the image corresponding to matrix of figure 4.8.

4.3.1 Algorithm (Method 2)

1. Fig.4.6 is generated which shows the stroke sequences of writing the character ‘M’. A mask matrix (8 x 8) (fig.4.7) containing all the pixel positions is generated corresponding to the writing path of the character. Where as all other pixels which are beyond the writing path (but within the matrix) are assigned zero value.
2. Then a threshold matrix (fig.4.8) is generated from the mask matrix. The pixel positions beyond the writing path but within the matrix are assigned a value of 255.
3. A pattern image is generated from the threshold matrix.
4. Final halftones (fig.4.10 and fig.4.11) are generated by thresholding the original sample image of fig.4.3 and grayscale ramp sample of fig.4.9, using the pattern image, respectively.

4.4 Results and Discussions

4.4.1 For Method 1:

Fig.4.5(a) is the histogram of the sample image, which contains the tonal values mostly in the middle tone areas. Fig.4.5(b) is the histogram of the embedded pattern.

The image after embedding the pattern become lighter due to adding of gray values of the pattern with the gray values of the image which is evident from histogram of fig.4.5(c).

Halftone Image (fig.4.4(c)) is obtained as a result of processing, is containing lesser image details (image appears to be lighter) [25] than the halftone produced in normal way (fig.4.4(d)) i.e. without pre-embedding the pattern.

4.4.2 For Method 2:

Fig. 4.12(a) is the histogram of the fig.4.9 and fig.4.12 (b) is the histogram of the image corresponding to matrix of fig.4.8. Fig 4.12(b) shows that the image contains tonal values mostly in the darker zones. In the halftone image ‘fig.4.10’, it is seen that the character writing pattern of ‘**M**’ tends to be completed when we approach from lighter tone to darker tone [25].

Another image ‘fig.4.3’ is also halftoned using the character writing pattern to produce the image ‘fig.4.11’, generating fewer image details.

4.5 Conclusions

Here the author has presented two novel digital halftoning methods based on ordered dither.

The method 1 has produced halftone by pre embedding the pattern image. The sample image embedded with pattern may be used in special effect imaging. This method has produced halftone, which is of considerably good quality.

The method 2 has produced halftone by using a character-writing pattern. This process may be used to simulate the writing pattern of any character of alphabets.

4.6 Program 4.1

The following program function (in matlab 6.1) generates any pattern of equal size to the image size and embeds it within the image once or any number of times and produce binary image by thresholding it.

```
function PatEmbedHT(filename,patfun,bs,ntpa)
%-----------------------------------------------------------
%-----------------------------------------------------------
% PatEmbedHT(filename,patfun,bs,ntpa)
% PATEMBEDHT (function): PATtern EMBEDed HalfTone
%
% This function generates any pattern of equal size to the
% image size and embeds it within the image once or any number
% of times and produce binary image by thresholding it.
%
% INPUT ARGUMENTS
% filename: any gray or rgb image file
% patfun: pattern function or pattern matrix
% bs: linear dimension of the pattern block size (pixels).
%     Max block size: 16 pixels
% ntpa: number of the times the pattern is added
%
% EXAMPLE:
% PatEmbedHT('pout.tif','spiral',4,10)
%
% Tested using Matlab 6.1.0.450 Release 12.1
%-----------------------------------------------------------
%-----------------------------------------------------------
cell={'spiral' 'magic' [1 3; 2 4]}
filename='einstein301.tif'
patfun=cell{1}
bs=4
```

```
ntpa=1
%----------------------------------------------------------
im=imread(filename);
[r c dim]=size(im)

if dim==3
   im=rgb2gray(im);
end

if  ischar(patfun)
   m=feval(patfun, bs);
else % if patfun is a matrix
   m=patfun;
   [bs c2]=size(m) % if m is pattern matrix change the bs
                 % if not correctly given.
end
m=m*256/16

% No of horizontal blocks
nohb=c/bs

% No of vertical blocks
novb=r/bs

% Minimum no of horizontal blocks
nohbf=floor(nohb);

% Minimum no of vertical blocks
novbf=floor(novb);

% Image widthwise extra pix
wwep=c-nohbf*bs
```

```
% Image heightwise extra pix
hwep=r-novbf*bs

%---------------------------------------------------------
mh=[];
mhv=[];

% Case 1
if wwep==0 & hwep==0
% Repeating the original pattern horizontally
   for i=1:nohbf
      mh=[mh m];
   end

% Repeating the horizontal pattern vertically to create
% final pattern.
   for j=1:novbf
      mhv=[mhv; mh];
   end
end

% Case 2
if wwep==0 & hwep~=0
% Repeating the original pattern horizontally
   for i=1:nohbf
      mh=[mh m];
   end

% Repeating the horizontal pattern vertically to create
% final pattern.
   for j=1:novbf
```

```
        mhv=[mhv; mh];
    end
    m2=mhv(1:hwep,1:c);
    mhv=[mhv;m2];
end

% Case 3
if wwep~=0 & hwep==0
% Repeating the original pattern horizontally
    for i=1:nohbf
        mh=[mh m];
    end
        m2=m(1:bs,1:wwep);
        mh=[mh m2];
% Repeating the horizontal pattern vertically to create
% final pattern.
    for j=1:novbf
        mhv=[mhv; mh];
    end
end

% Case 4
if wwep~=0 & hwep~=0
% Repeating the original pattern horizontally
    for i=1:nohbf
        mh=[mh m];
    end
        m2=m(1:bs,1:wwep);
        mh=[mh m2];

% Repeating the horizontal pattern vertically to create
% final pattern.
```

```
    for j=1:novbf
        mhv=[mhv; mh];
    end
    m2=mhv(1:hwep,1:c);
    mhv=[mhv;m2];
end
%------------------------------------------------------------
mhv=uint8(mhv);

% Threshold and binarization of the original image
th1=graythresh(im);
bin1=im2bw(im,th1);
figure,imshow(bin1)

% Datatype conversion
im=double(im);
mhv=double(mhv);

mhv2=uint8(mhv);
imwrite(mhv2,'pat.tif','resolution',301)

% Enhancing the pattern by repeatedly adding
ntpa; % No of times the pattern is added
imz=zeros(r,c);
for k=1:ntpa
    imz=imz+mhv;
end

imz2=uint8(imz);
imwrite(imz2,'addpat.tif','resolution',301)

% Adding the pattern image with the original image
```

```
impe=im+imz;
impe=uint8(impe);

% Displaying the pattern embedded image
figure,imshow(impe)
imwrite(impe,'impe.tif','resolution',301)

% Threshold and binarization of the pattern embedded image
th2=graythresh(impe);
bin2=im2bw(impe,th2);
figure,imshow(bin2)
imwrite(bin2,'imht.tif','resolution',301)
%------------------------------------------------------------
%------------------------------------------------------------
```

4.7 Program 4.2

The following program function (in matlab 6.1) creates a pattern to simulate the writing of any character of any alphabet.

```
function [thm]=CharWritePat(bgcolor)
%---------------------------------------------------------
%---------------------------------------------------------
% thm=CharWritePat(bgcolor)
% CHARWRITEPAT (function)
%
% This function creates a pattern to simulate the writing of
% any character of any alphabet.
% Here it is done for 'M' within a 8 by 8 matrix
%
% INPUT ARGUMENT:
% bgcolor: background color options
% 'w': white
% 'k': black
%
% OUTPUT ARGUMENT:
% thm: threshold matrix
%
% EXAMPLE:
% CharWritePat('w')
%
% Tested using Matlab 6.1.0.450 Release 12.1
%---------------------------------------------------------
%---------------------------------------------------------
n=8; % linear dimension of the character matrix

% Matrix to hold the char design
m=zeros(n);
```

```
% Locations
loc=[10:15,18:19,28:29,37:-1:36,43:-1:42,50:55];

% Numbers
num=[length(loc):-1:1]

num2=fliplr(num)

% Inserting the numbers into the specific locations
% to generate mask matrix
m(loc)=num2(:)

% Threshold matrix
c=ceil(255/numel(m))
thm=m*c;

% Background color preferences
if  bgcolor=='w'
   k=find(thm==0);
   thm(k)=255;
elseif bgcolor=='k'
   thm;
end

thm=uint8(thm);

imwrite(thm,'CharWritePat.tif')

%-----------------------------------------------------------
```

4.8 Comparison with the Other Techniques Developed

Two new methods of digital halftoning are developed and are presented in this chapter. The method titled “Halftoning by pre-embedding the pattern” can be used for general purpose ordered dither and produces halftone of reasonable good quality with image details. In the following paragraphs this method is compared with direct clustered dot dither method of halftoning.

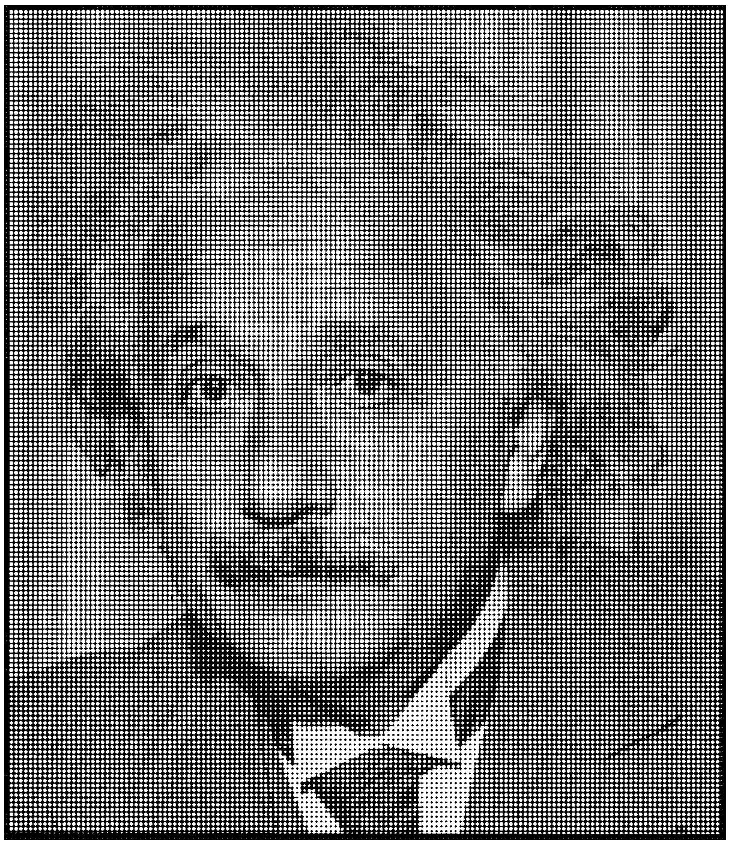


Figure 4.13: (einstein301_cdd2)

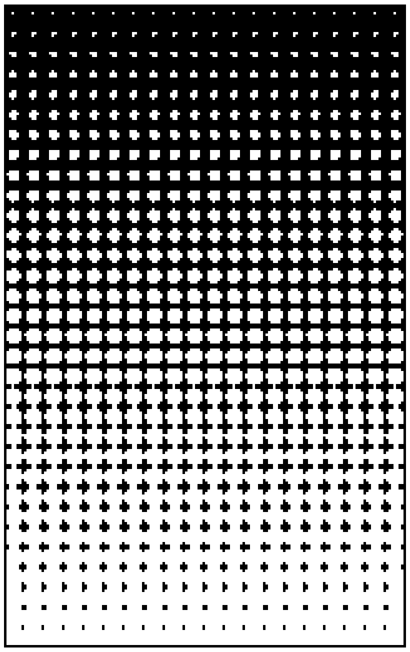


Figure 4.14: (gs3_cdd2)

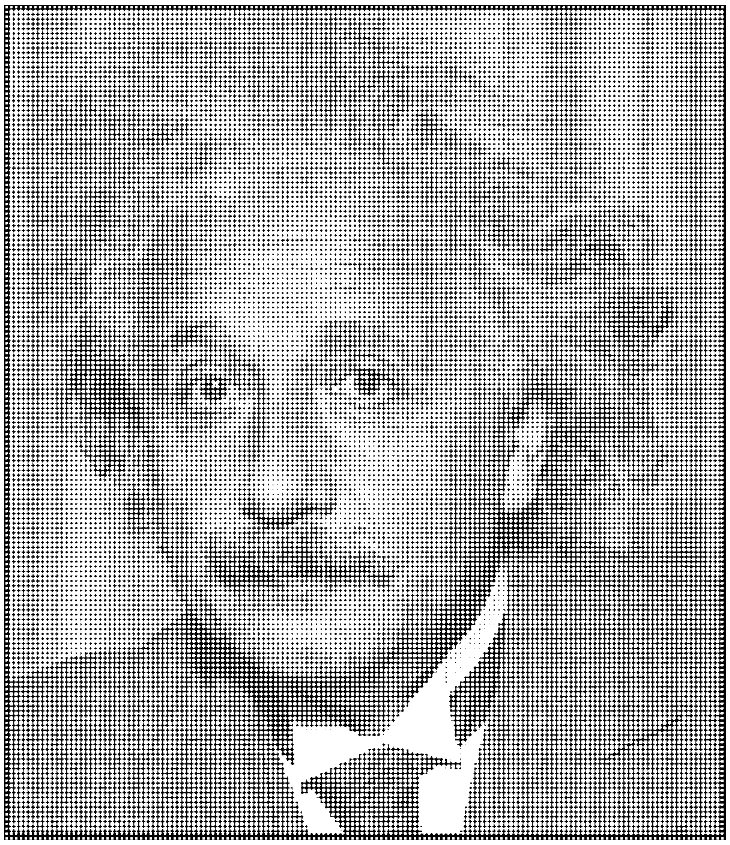


Figure 4.15: (einstein301_imht_4)

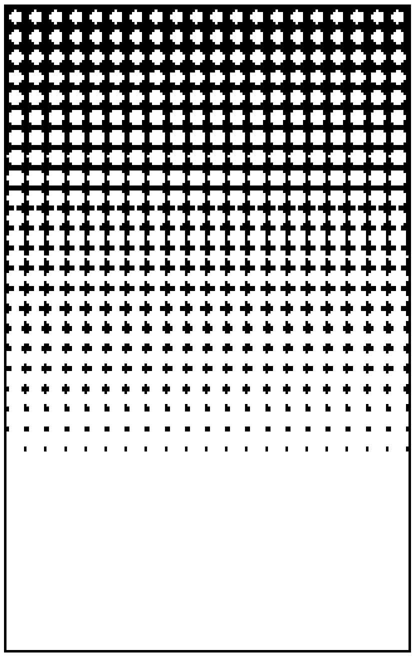


Figure 4.16: (gs3_imht_4)

Filenames are given in parenthesis for the fig.4.13-fig.4.16

Halftone of fig.4.13 is obtained by applying dither array (clustered dot) of fig.1.5 to the image of fig.4.3 and similarly halftone of fig. 4.14 is obtained by applying same dither array of fig.1.5 to the image of fig.4.9 (a grayscale ramp). Halftone of fig.4.15 is obtained by embedding the image of the dither array of fig.1.5 to the image of fig.4.3 and halftoning it. Similarly halftone of fig.4.16 is obtained by embedding the image of the dither array of fig.1.5 to the image of fig.4.9 and halftoning it. By visual comparison of the fig.4.13 and fig.4.15, it could be said that fig.4.15 is lighter than fig.4.13 and contains reasonable amount of image details. By comparing fig. 4.14 and fig.4.16 it is seen that details are lost in fig.4.16 as tonal range is compressed.

The method titled "Halftoning by simulating character-writing pattern" is meant for special application only and produces fewer image details.

Chapter: 5

Soft Halftone Proofing

Chapter: 5

Soft Halftone Proofing

5.1 Introduction

Digital color proofs are the most important requirements in predicting the results of print production. Digital proofs on monitor always take a fore seat once an image is ready for further processing especially in print production. Existing image-editing tools like Adobe Photoshop, Corel Photo-Paint, and Corel PaintShopPro etc have not provided any readily available imaging tools that may create soft halftone proofs with wide range of screening methods like amplitude modulated with various dot sizes or frequency modulated screening. The filter tool like ‘color halftone’ in Adobe Photoshop 7.0 create a special effect with amplitude modulated anti-aliased round dots with different screen angles, which is equivalent to soft-screened proof of one kind. The ‘halftone’ tool of Corel Photo-Paint 11 provides similar facility to create AM based screened proof. Corel Paint Shop Pro 9.0 also provides similar facility for creating soft halftone proofs. These above-mentioned proofing for color-screened images is not versatile as already mentioned. In an earlier work by the author [26] an attempt has been made to fill that lacuna in soft halftone proofing which is capable of producing both amplitude modulated and frequency modulated soft halftone proofs.

5.2 Experimental Procedures

The Adobe Photoshop 7.0 has been selected as the main platform to work with.

Main procedure steps are as follows

1. Editing the image in RGB mode.
2. Converting to CMYK mode using suitable setup.
3. Doing necessary corrections.
4. Separating the four plates (Black, Cyan, Magenta and Yellow) by splitting the channels, this creates four gray images corresponding to each process color.
5. Generating halftones for the four plates.
6. Converting four plates to single bit grayscale and then to CMYK mode and colorizing four halftone plates using the process colors.
7. Combining all the four images in separate layer, in a single file.

8. Apply ‘multiply’ (layer blending) effect for individual layers to create subtractive mixing of colors for the layers i.e. individual color printers as shown in fig.5.1.
9. Judge the final soft halftone proofs and progressive proofs.

Table: 5.1: Filenames and other details at various stages of AM proof generation

Original Image	**Resolution (DPI)**	**Gray Images (Splitted)**	**AM Halftone Images**	**AM Halftone Images (Colored*)**	**Dotshape**	**Screen Angle (Degree)**	**LPI**
flrgb or flcmyk	144	flcmyk_K	flcmyk_K_am	flcmyk_K_amc	ellipse	45	36
		flcmyk_C	flcmyk_C_am	flcmyk_C_amc	Do	75	36
		flcmyk_M	flcmyk_M_am	flcmyk_M_amc	Do	105	36
		flcmyk_Y	flcmyk_Y_am	flcmyk_Y_amc	Do	90	36

* Color used: Photoshop CMYK, values are given in percentages, e.g. for cyan plate C=100%, M=0%, Y=0%, K=0%

Progressive proofs: flcmyk_CM_amc, flcmyk_CY_amc, flcmyk_MY_amc

Final Proof: flcmyk_am_proof

Table: 5.2: Filenames and other details at various stages of FM proof generation

Original Image	**Resolution (DPI)**	**Gray Images (Splitted)**	**FM Halftone Images**	**FM Halftone Images (Colored*)**	**Screening**
flrgb or flcmyk	144	flcmyk_K	flcmyk_K_fm	flcmyk_K_fmc	Diffusion dither
		flcmyk_C	flcmyk_C_fm	flcmyk_C_fmc	Do
		flcmyk_M	flcmyk_M_fm	flcmyk_M_fmc	Do
		flcmyk_Y	flcmyk_Y_fm	flcmyk_Y_fmc	Do

* Color used: Photoshop CMYK, values are given in percentages, e.g. for cyan plate C=100%, M=0%, Y=0%, K=0%

Progressive proofs: flcmyk_CM_fmc, flcmyk_CY_fmc, flcmyk_MY_fmc

Final Proof: flcmyk_fm_proof

Fig.5.1 shows layer blending effect “Multiply” in Photoshop 7 of three layers containing colors in ‘Cyan’, ‘Magenta’ and ‘Yellow’ circles, to generate colors red, green and blue and black, which is equivalent to subtractive mixing of colors.

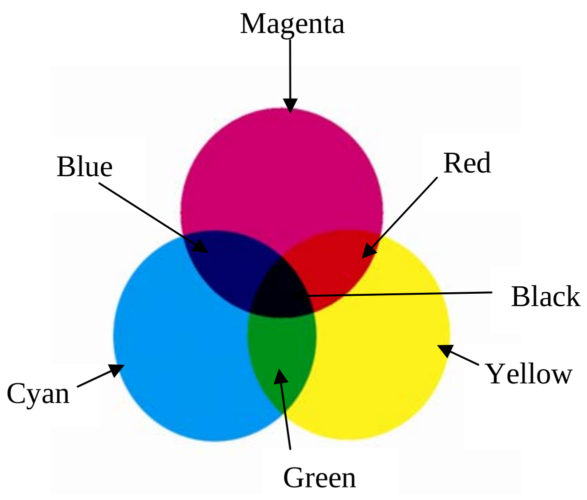


Figure 5.1: “Multiply” layer blending effect in Photoshop 7 of three layers ‘Cyan’, ‘Magenta’ and ‘Yellow’ which is equivalent to subtractive mixing of colors.

In labeling the figures from fig.5.2-fig.5.29, the filename of the images are given in the parenthesis.

Here both Amplitude Modulated and Frequency Modulated screening is done to generate both type of halftone proofs. Fig.5.2 and fig.5.3 are the rgb and cmyk color originals. Fig.5.4 to fig.5.7 are the cyan, magenta, yellow and black plates in gray respectively.

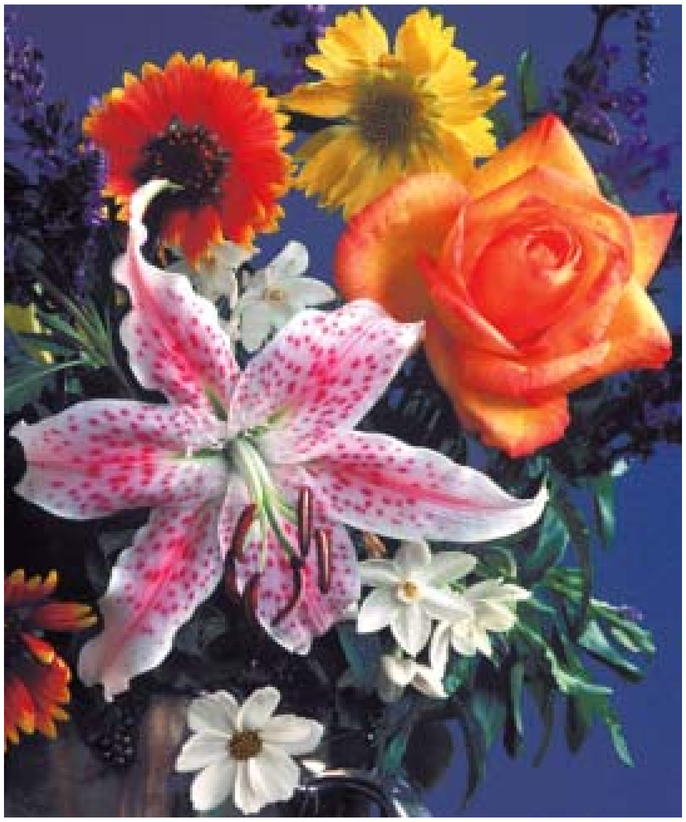


Figure 5.2: Original RGB image (flrgb)

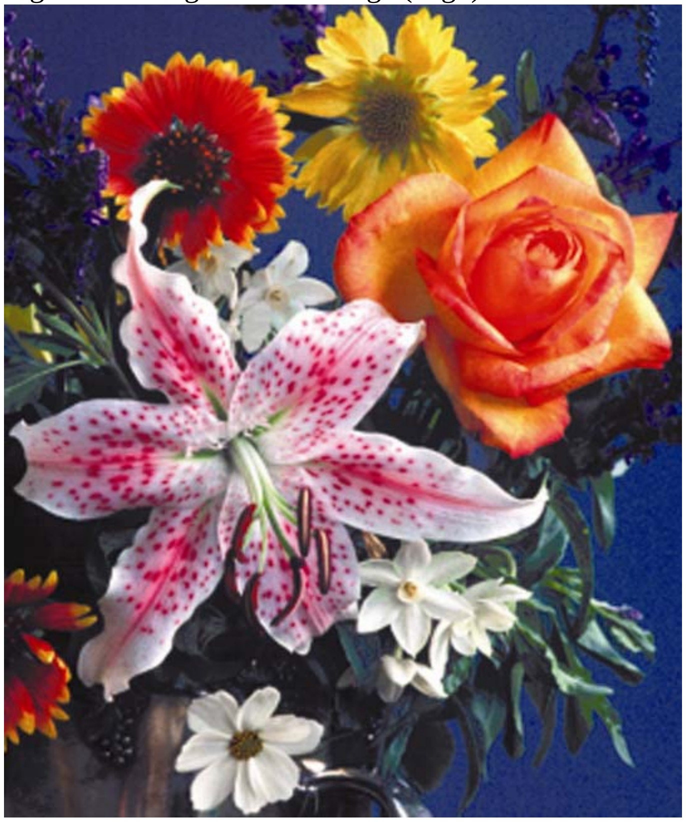


Figure 5.3: Original CMYK image (flcmyk)

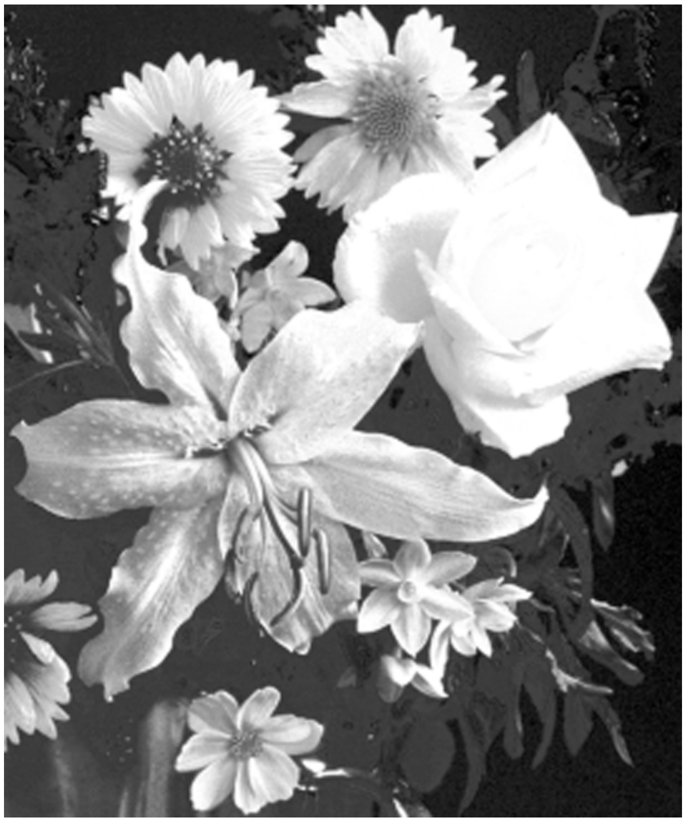


Figure 5.4: Cyan part in gray (flcmyk_C)

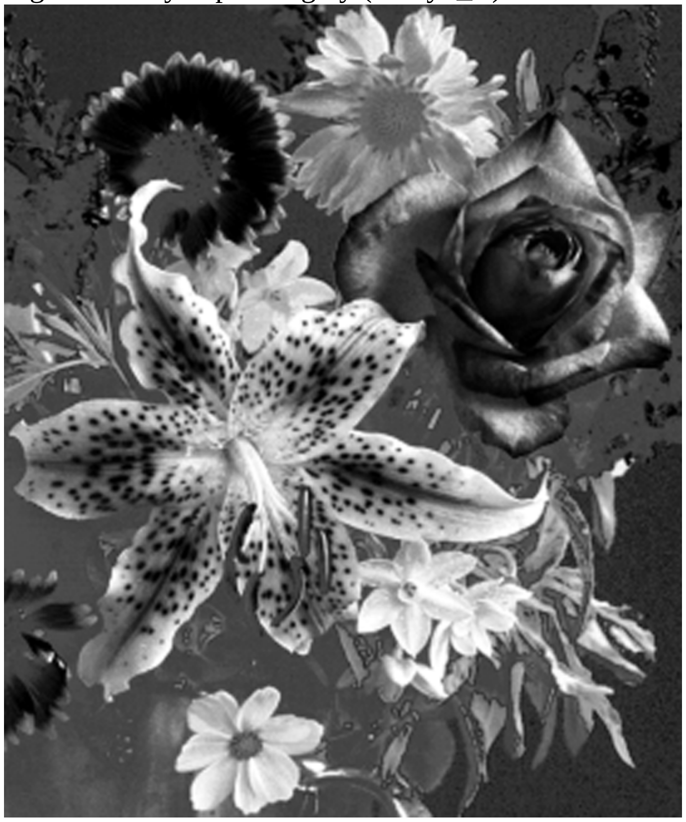


Figure 5.5: Magenta part in gray (flcmyk_M)

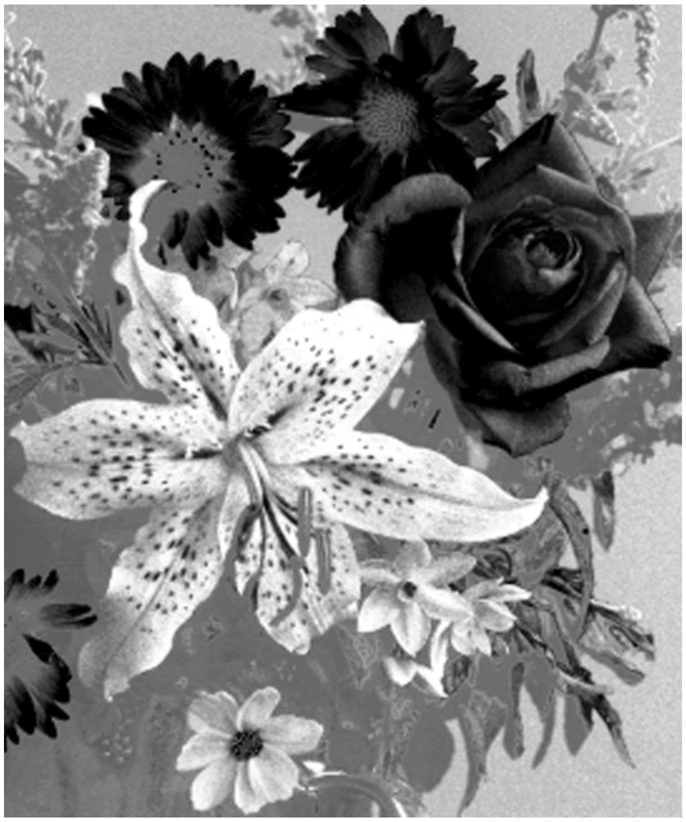


Figure 5.6: Yellow part in gray (flcmyk_Y)

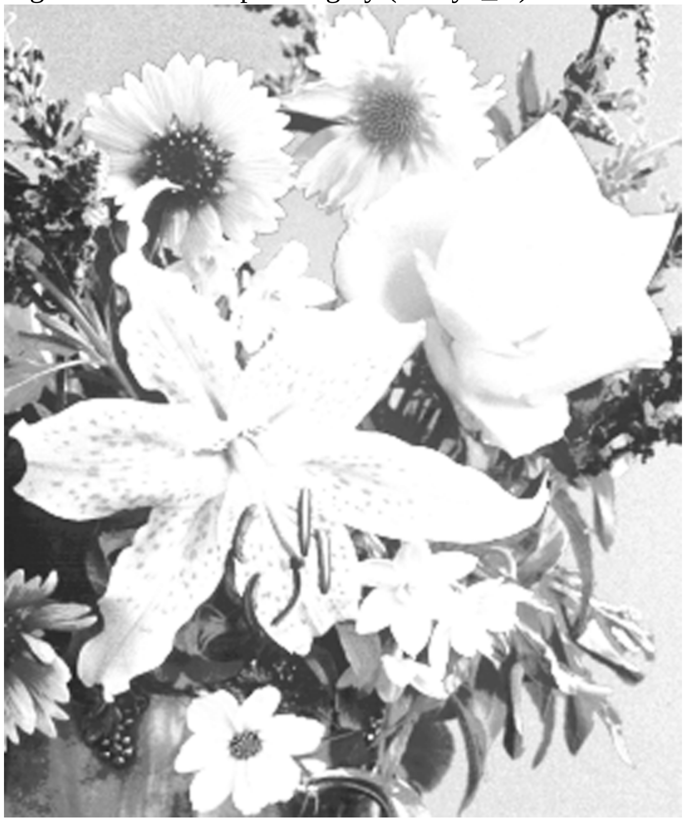


Figure 5.7: Black part in gray (flcmyk_K)

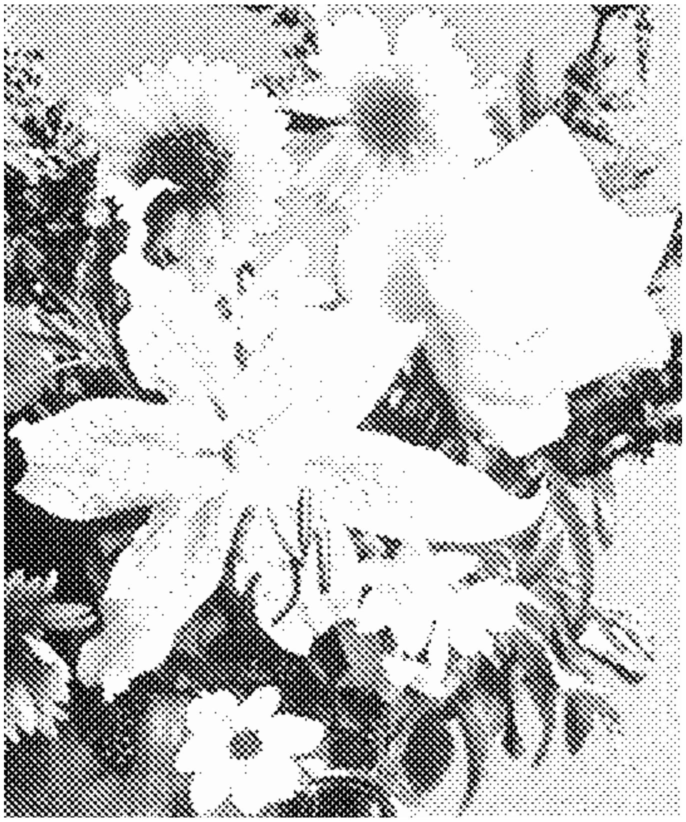


Figure 5.8: AM halftoned Black (flcmyk_K_amc)

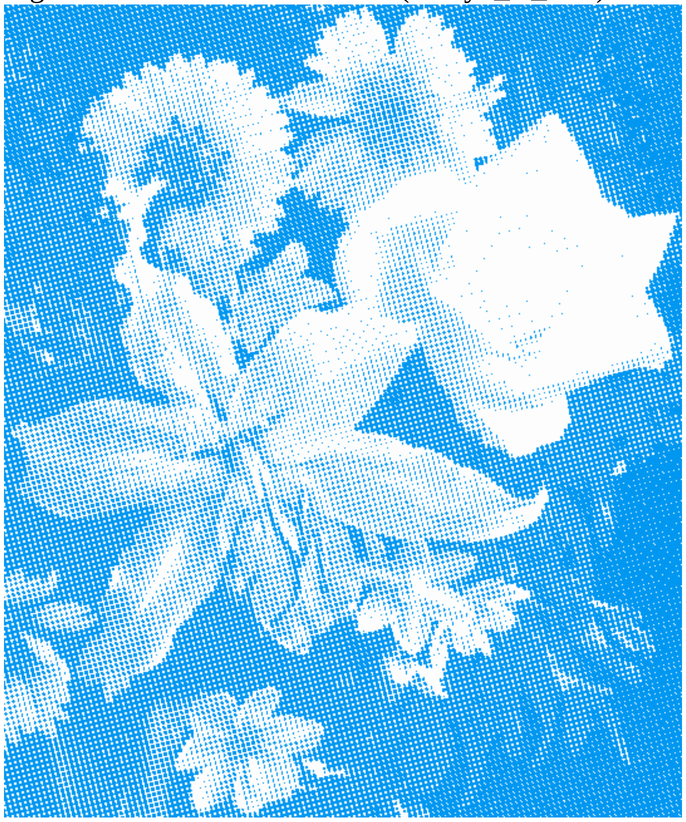


Figure 5.9: AM halftoned Cyan (flcmyk_C_amc)

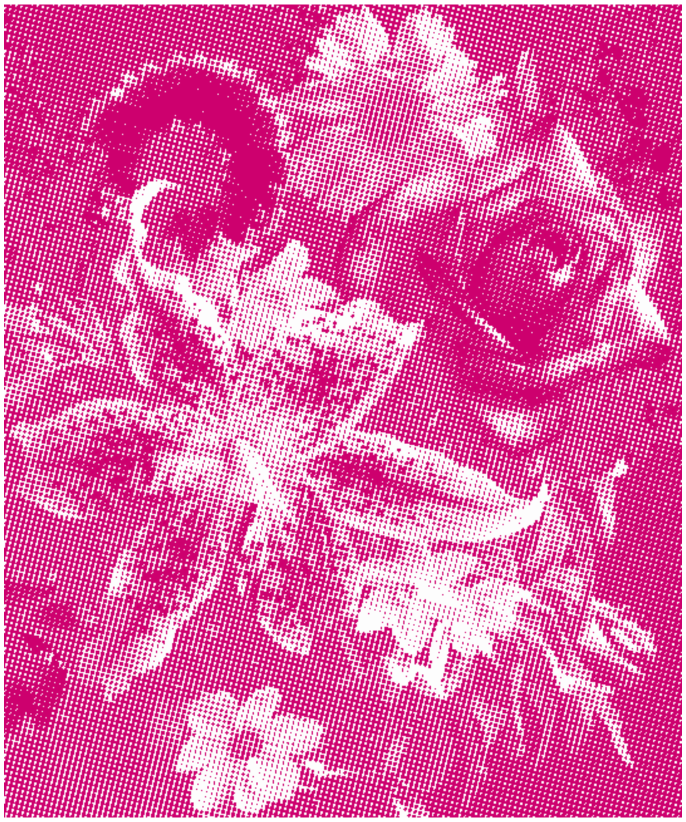


Figure 5.10: AM halftoned Magenta (flcmyk_M_amc)

Figure 5.11: AM halftoned Yellow (flcmyk_Y_amc)

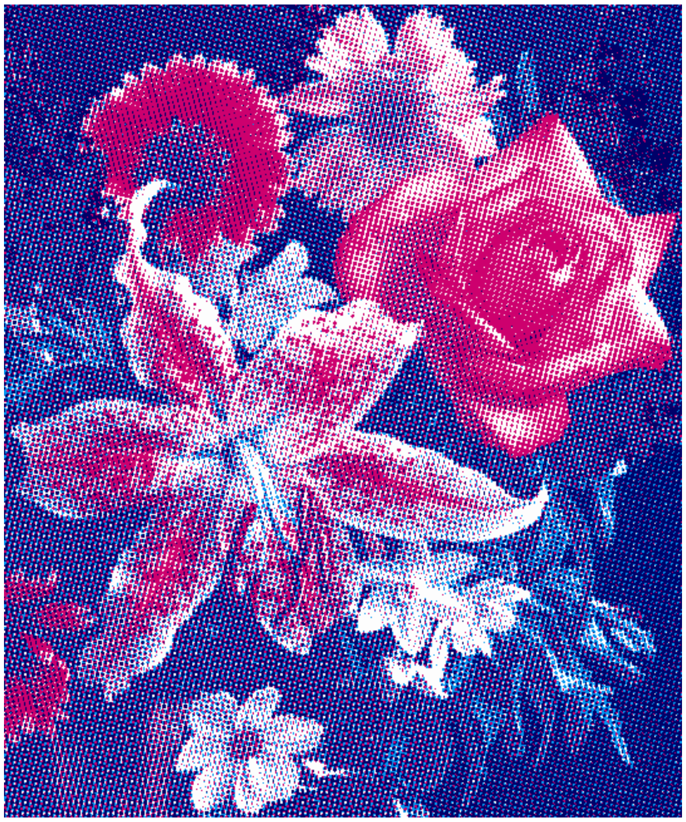


Figure 5.12: Progressive Proof: Cyan and Magenta (flcmyk_CM_amc)

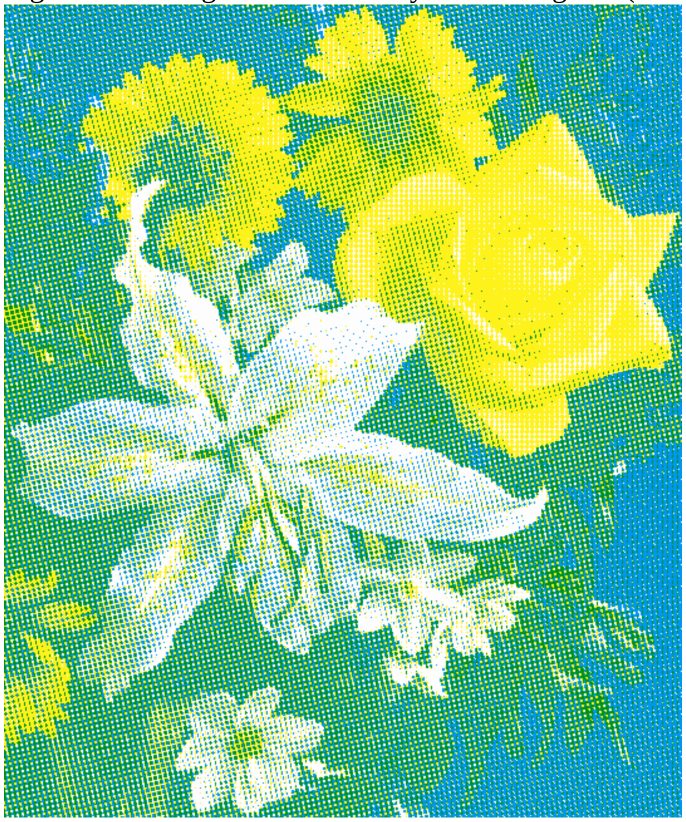


Figure 5.13: Progressive Proof: Cyan and Yellow (flcmyk_CY_amc)

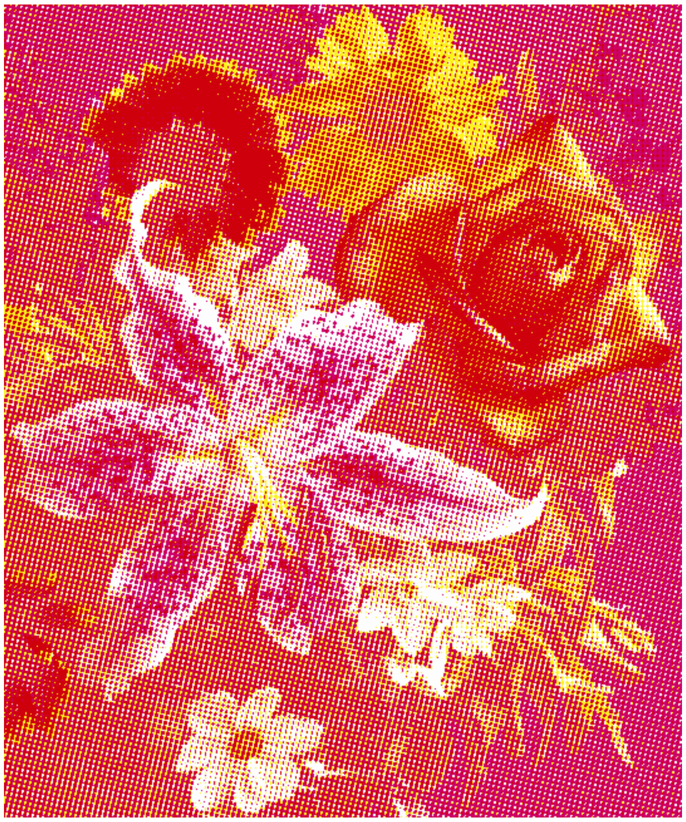


Figure 5.14: Progressive Proof: Magenta and Yellow (flcmyk_MY_amc)

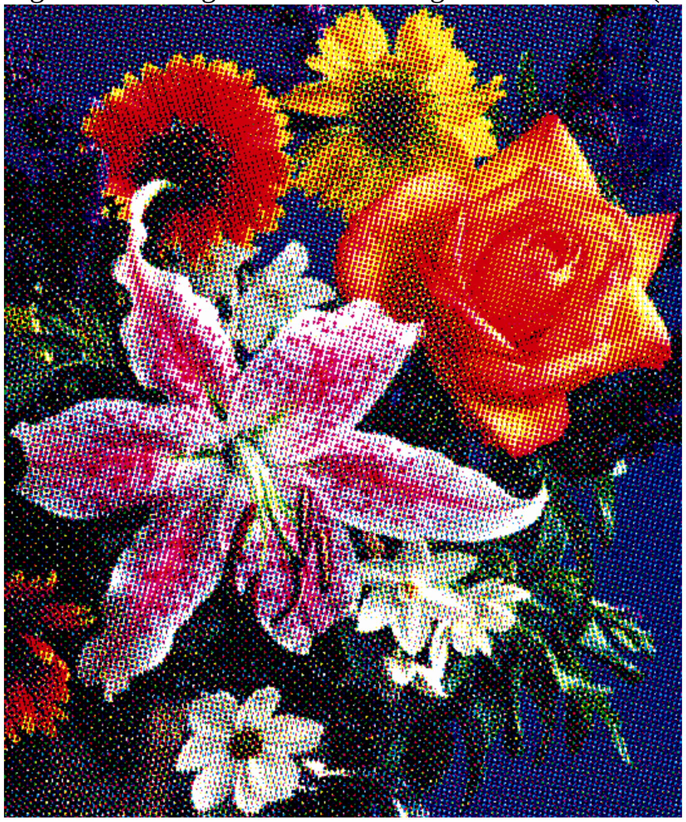


Figure 5.15: AM halftoned Proof (flcmyk_am_proof)

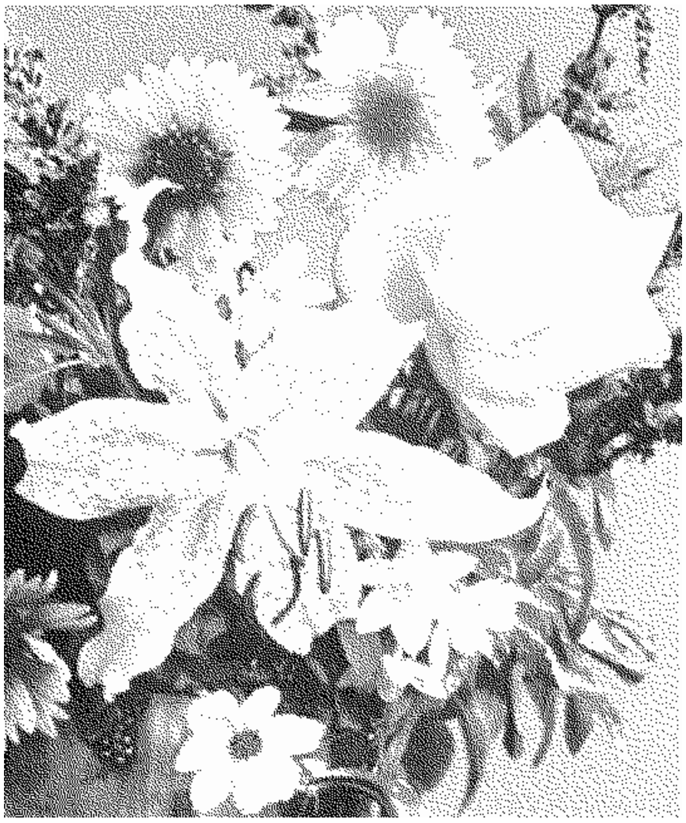


Figure 5.16: FM halftoned Black (flcmyk_K_fmc)

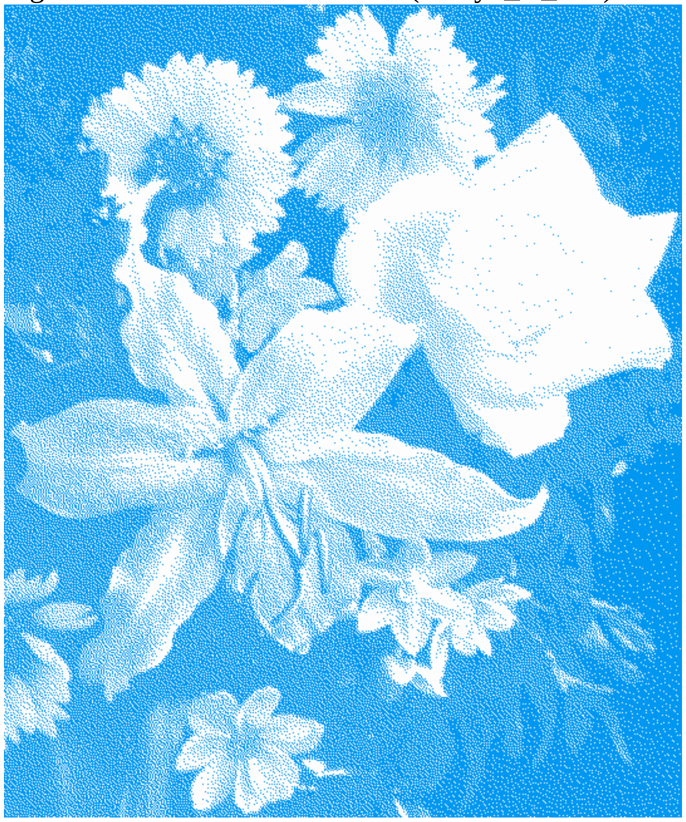


Figure 5.17: FM halftoned Cyan (flcmyk_C_fmc)

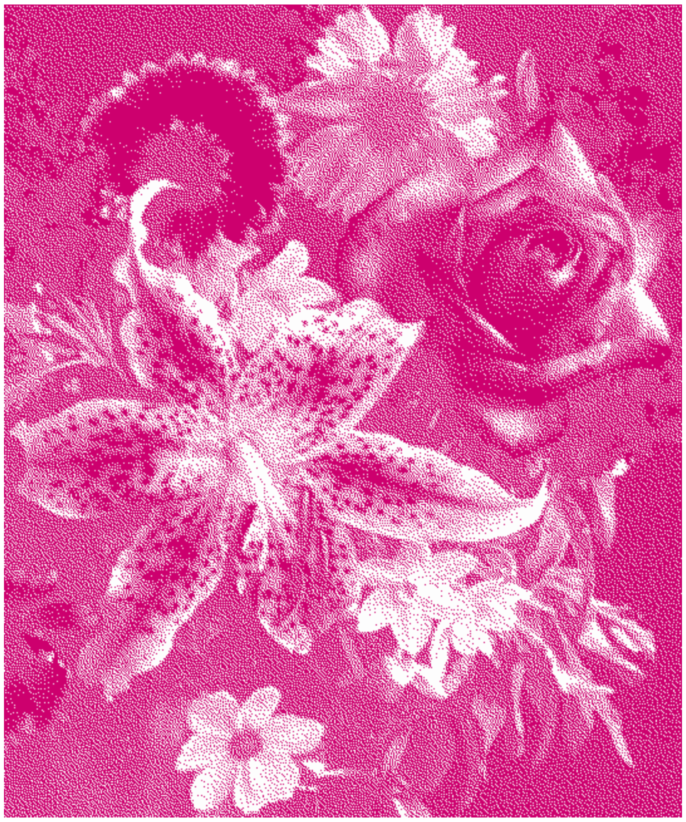


Figure 5.18: FM halftoned Magenta (flcmyk_M_fmc)

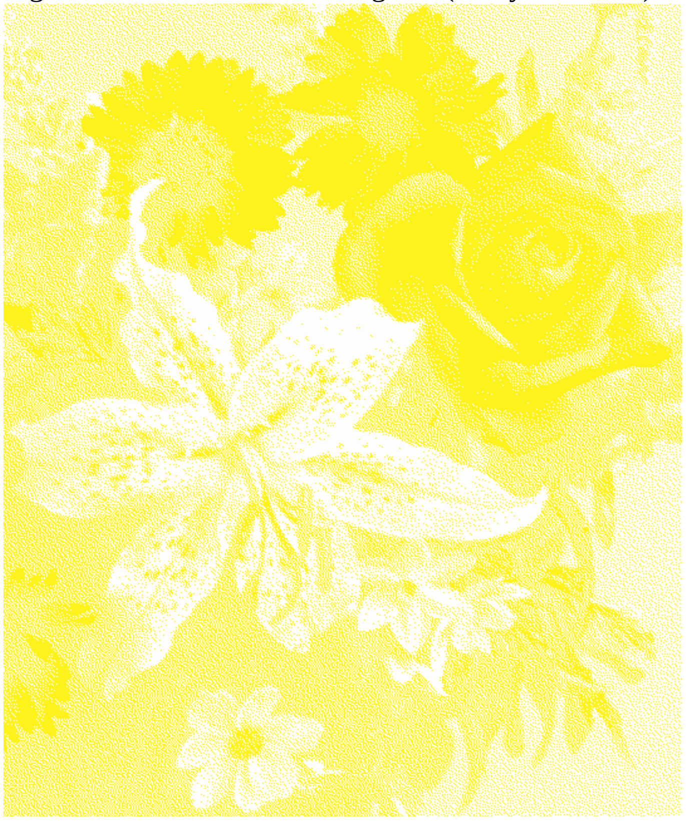


Figure 5.19: FM halftoned Yellow (flcmyk_Y_fmc)

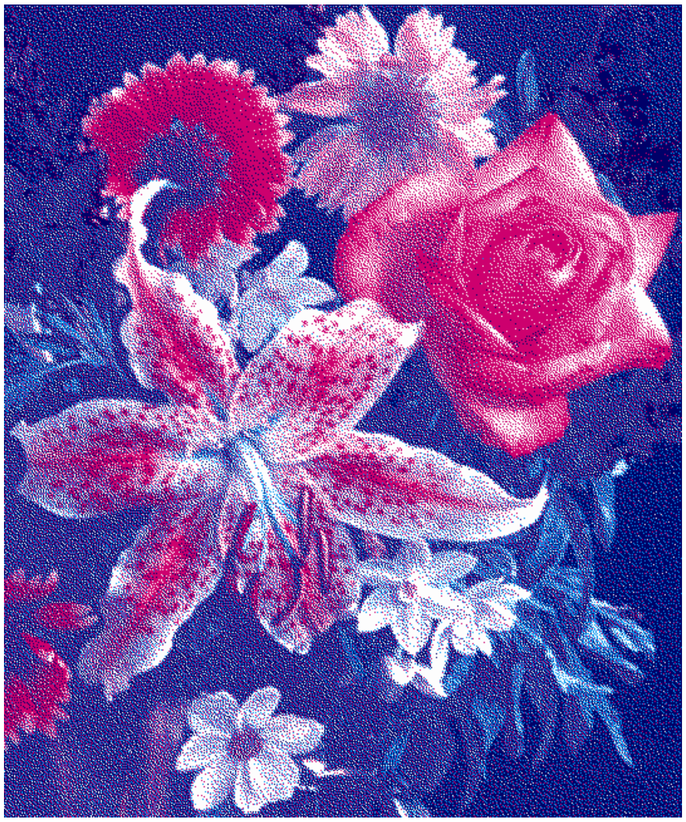


Figure 5.20: Progressive Proof: Cyan and Magenta (flcmyk_CM_amc)

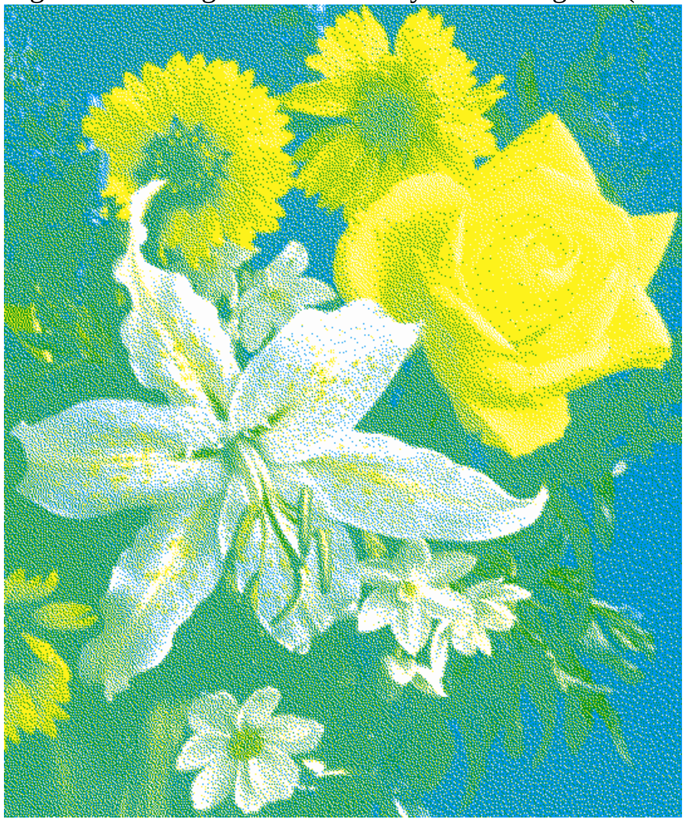


Figure 5.21: Progressive Proof: Cyan and Yellow (flcmyk_CY_amc)

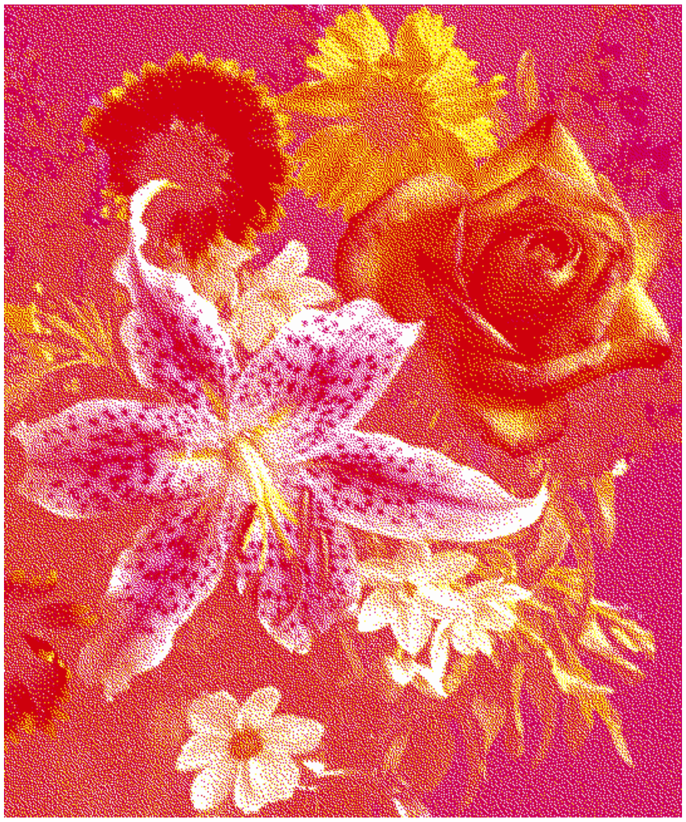


Figure 5.22: Progressive Proof: Magenta and Yellow (flcmyk_MY_amc)

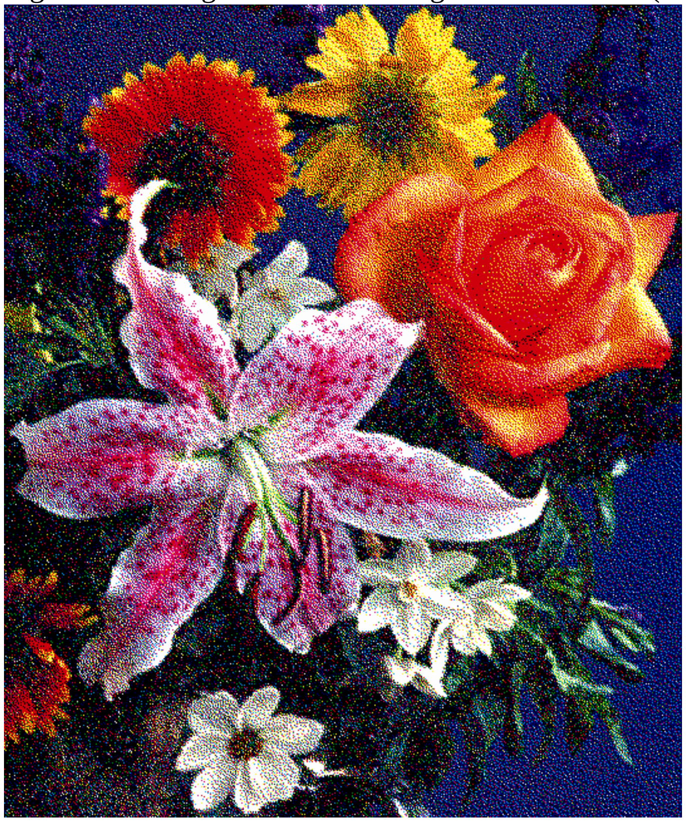


Figure 5.23: FM halftoned Proof (flcmyk_fm_proof)

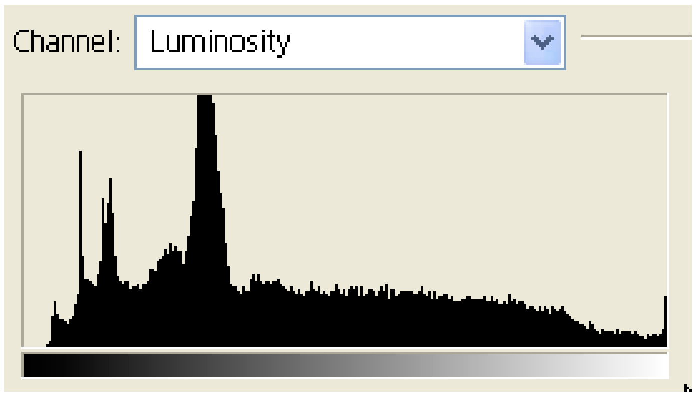


Figure 5.24: Histogram of figure 5.2 (flrgb_hg)

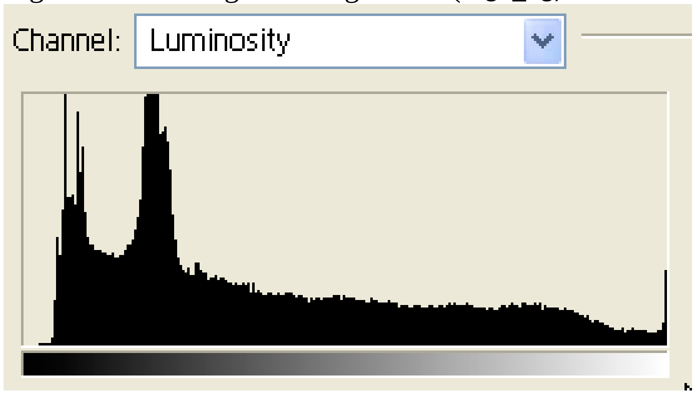


Figure 5.25: Histogram of the figure 5.3 (flcmyk_hg)

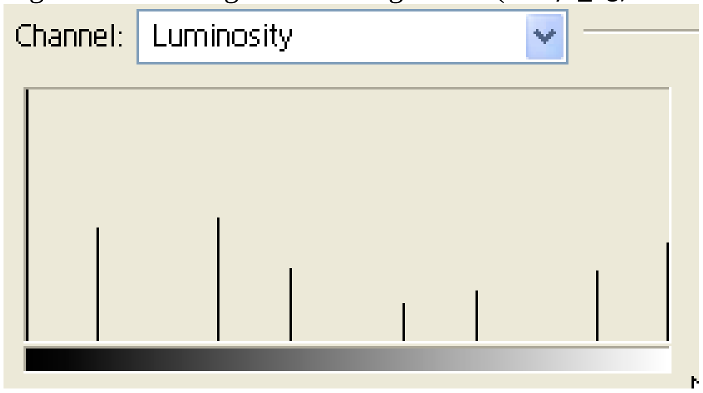


Figure 5.26: Luminosity histogram of the figure 5.15 (am_hg_2)

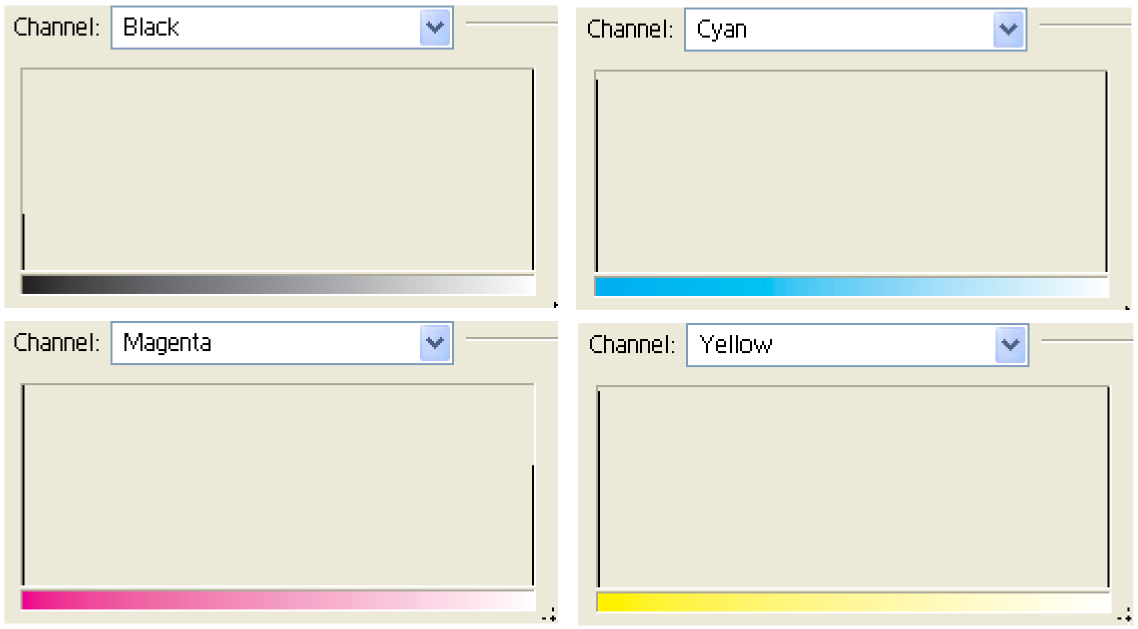


Figure 5.27: K, C, M and Y histograms of the figure 5.15 (am_hg)

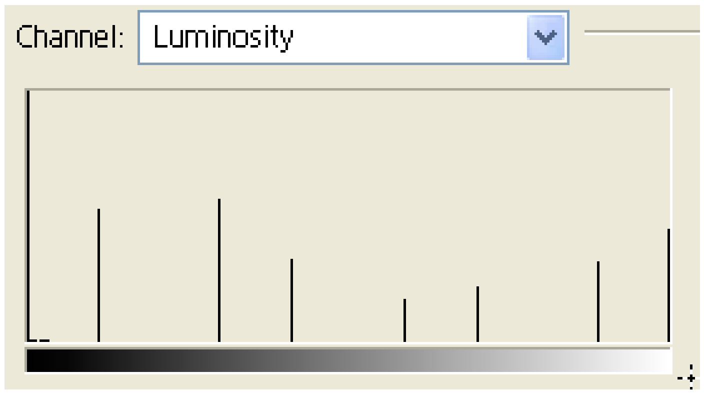


Figure 5.28: Luminosity histogram of the figure 5.23 (fm_hg2)

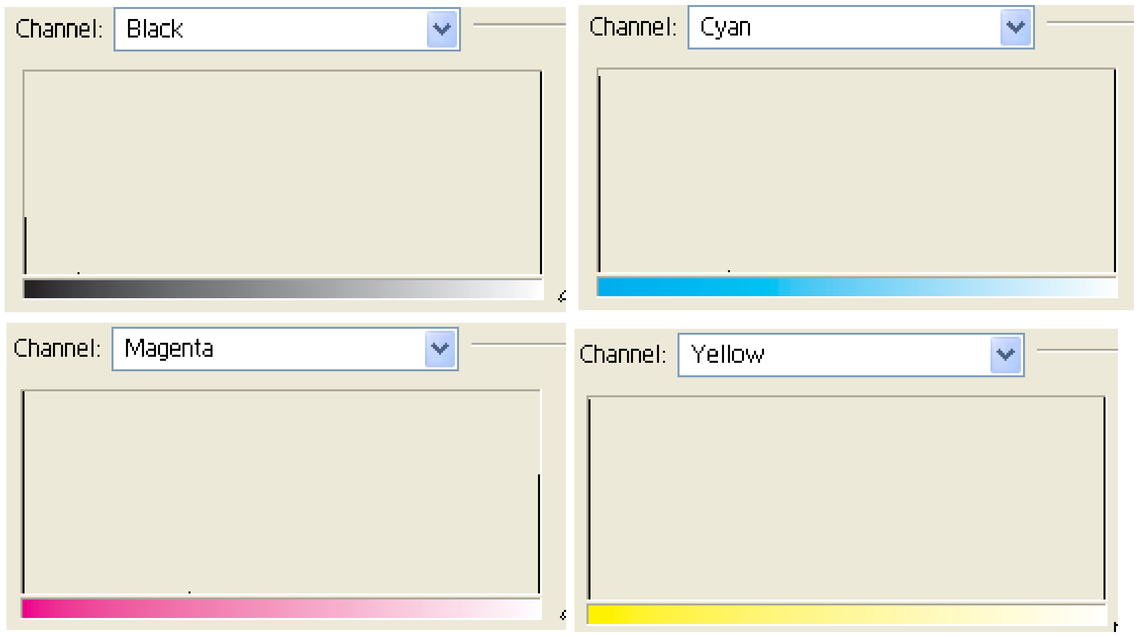


Figure 5.29: K, C, M and Y histograms of the figure 5.23 (fm_hg)

5.3 Results and Discussions

Fig.5.8 to fig.5.11 are the amplitude modulated screened black, cyan, magenta and yellow plates respectively. Dot shape, screen angle, screen frequency (Lines per Inch) and resolution (DPI) of the original images and filename details are given in the Table-5.1. Fig. 5.12 is cyan-magenta, fig 5.13 is cyan-yellow and fig.5.14 is magenta-yellow, progressive proofs. Fig.5.15 is the amplitude modulated halftone proof. Fig.5.16 to fig.5.19 are the frequency modulated screened black, cyan, magenta and yellow plates respectively. Dots per inch of the original images, type of screening used and filename details are given in the Table-5.2. Fig. 5.20 is cyan-magenta, fig 5.21 is cyan-yellow and fig.5.22 is magenta-yellow, progressive proofs. Fig.5.23 is frequency modulated halftone proof. Fig.5.24 and fig.5.25 are the luminosity histograms of the fig.5.2 and fig.5.3 respectively and their image-wise difference is evident in the histograms especially in darker tones. Fig.5.26 is the luminosity histogram of the fig. 5.15 which shows few histogram tones because of using pure and single tone color. Fig. 5.27 contains histograms of black, cyan, magenta and yellow channels of the fig. 5.15 which show pure and single tone K, C, M and Y colors (as intermediate tones are not available in the histograms of K, C, M and Y) are used to make AM proof. Like that fig.5.28 is the luminosity histogram of the FM proof (fig 5.23) which shows few histogram tones because of using pure and single tone color. Fig.5.29 contains histograms of K, C, M and Y channels of the fig.5.23 which show pure and single tone K, C, M and Y colors (as intermediate tones are not available in the histograms of K, C, M and Y) are used to generate the FM proof also.

5.4 Conclusions

The method presented here is more versatile i.e. having wider applications than the existing methods as shown in Chapter 2 (Section 2.4), as both amplitude modulated and frequency modulated halftone proofs can be generated. Proofs of good quality (having more image details and C, M, Y, K plates colorized with pure colors, i.e. one color not mixed with other color) can be generated by this method for both types of halftoning. Color gamut might change if different colors are selected at the time of colorizing each plate.

5.5 Comparison with the Other Techniques Developed

In this chapter a novel method is developed in proofing which could produce both amplitude modulated and frequency modulated soft halftone proofs within a short period of time. This method produces better quality soft halftone proofs than the existing systems. Fig. 5.15 and fig. 5.23 are amplitude modulated and frequency modulated soft halftone proofs respectively, which are of better quality (obtained by histogram analysis e.g. fig2.2, fig.2.4, fig.2.6, fig.2.8, fig.5.27 and fig.5.29 and visual comparison of the image details) than fig. 2.1 and fig.2.3, the proofs produced by Photoshop 7 and Corel Photo-Paint 11 respectively (Refer Section 2.4 and Section 5.3).

Chapter 6

Discussions and Concluding Remarks

Chapter 6

Discussions and Concluding Remarks

6.1 Discussions

The present research work has been subdivided into three sections. 1. Non-Halftone Binary Image Transformations. (Chapter 3) 2. Digital Halftoning (Chapter 4) and 3. Soft Halftone Proofing (Chapter 5).

In the first section, the author has presented various novel non-halftone binary image transformation methods which might create new avenues of image transformations in digital imaging and image processing.

In the second section, the introduced two methods of novel digital halftoning would generate halftone for various application and would fuel further improvement in digital halftoning.

In the third section, the method presented in soft halftone proofing would solve the existing problems of soft halftone proofing.

6.2 Conclusions

The research work has been subdivided into three sections. 1. Non-Halftone Binary Image Transformations. (Chapter 3) 2. Digital Halftoning (Chapter 4) And 3. Soft Halftone Proofing (Chapter 5).

The following conclusions may be drawn from carried out research work.

We have achieved the goal in non-halftone binarization section i.e. we have produced non-halftone binary images through various methods. From the image processing point of view few methods take more time to produce result and few methods take less time to process images. Example of one such method which takes less time is “Row-wise and column-wise processing”. Example of one such method which involves more mathematical calculation and takes more time is “Processing the image diagonally”. Quality non-halftone output for special application can be produced from methods such as “Processing the image diagonally” and other methods of “Processing based on pixel location and path based on location” like “Processing the image columnwise”.

In digital halftoning (ordered dither) section two proposed novel methods are based on relatively faster algorithms. Method 1 which is based on pre-embedded pattern produced halftone of considerable good quality. Method 2 which is based on character writing pattern which has simulated character writing in halftone generation.

In the section soft halftone proofing, methodology proposed to generate soft halftone proofs is more versatile with wider application.

6.3 Scope of Future Investigations

The algorithms proposed for some methods e.g. “Processing the image diagonally” in non halftone binarization section, might require further investigations to design faster algorithms.

In the similar way new avenues might come on further research in improving the image quality and pattern embedding methods in the digital halftoning section (ordered dither).

Likely we might get new methods of soft halftone proofing with further research in digital soft halftone proofing section. The method of soft halftone proofing requires faster algorithm to be designed and implemented through software environment.

Appendix A

Program A.1

The following program function (in matlab 6.1) generates grayscale ramp

```
function gs=Grayscale(mng,mxg,sj,noc,nor,ori,res,nows,method)
%---------------------------------------------------------
%---------------------------------------------------------
% gs=Grayscale(mng,mxg,sj,noc,nor,ori,res,nows,method)
%
% GRAYSCALE (function)
%
% This function generates grayscale
%
% INPUT ARGUMENTS:
% mng: Minimum gray value mng>=0
% mxg: Maximum gray value mxg<=255
% sj: Grayscale step to step jump required
% noc: No of columns required for each grayscale step
% nor: No of rows required for each grayscale step
% ori: Orientation (string)
%    't2b': Top to bottom (mng to mxg) or any string
%    'l2r': Left to right (mng to mxg)
% res: Resolution required for output file. This is essential
%     to reduce the physical size of the grayscale image,
%     rather than filesize. if res=0 is given,
%     default 72 is taken.
% nows: No of whitespaces given in between each steps
% method: Methods 1, 2 and 3
%
% EXAMPLE:
```

```
% gs=Grayscale(0,255,1,800,8,'t2b',300,0,1)
%--------------------------------------------------------
%--------------------------------------------------------
if method==1

% Method: 1
g=[(mng:sj:mxg).'];
gv=[];
gvh=[];

% White space to be given in between gray steps
gw=[255];
gw2=[];

nost=length(g);  % no of steps

% Vertical repeat loop
for i=1:nost
   gw2=[];
   for j=1:nor % vertical steps
      gv=[gv;g(i)];
   end

   % Insertion of white spaces in between gray steps
   if nows>0
      for w=1:nows
         gw2=[gw2;gw];
      end
      gv=[gv;gw2];
   else
      gv=[gv;gw2];
   end
```

```
end

% Horizontal repeat loop
for i=1:noc % horizontal steps
   gvh=[gvh gv];
end
gs=(gvh);

% Orientation
if ori=='l2r'
   gs=rot90(gs);
else % or=='t2b'
   gs=gs;
end
gs=uint8(gs);

% Output
imwrite(gs,'gs1.tif','resolution',res)
end
%------------------------------------------------------------
if method==2

% Method: 2
gs=[];
gw2=[];
for i=mng:sj:mxg
   gsi=zeros(nor,noc)+i;

   % Insertion of white spaces in between gray steps
   if nows>0
      gw2=[];
```

```
        gw=zeros(1,noc)*255;
        for w=1:nows
           gw2=[gw2;gw];
        end
        gsi=[gsi;gw2];
    else
        gsi=[gsi;gw2];
    end

    gs=[gs; gsi];
end

% Orientation
if ori=='l2r'
    gs=rot90(gs);
else % or=='t2b'
    gs=gs;
end

gs=uint8(gs);

% Output
imwrite(gs,'gs2.tif','resolution',res);
end
%----------------------------------------------------------
if method==3

% Method: 3
nost=length(mng:sj:mxg);
nd=ndgrid(mng:sj:mxg,1:noc);

row=(nor*(nost+nows));
```

```
gs=zeros(row,noc);


gs2=[];
gw2=[];
for i=1:nost
   gw2=[];
   nd(i,1:noc);
   for j=1:nor
     gs2=[gs2; gs(j,1:noc)+nd(i,1:noc)];
   end


   % Insertion of white spaces in between gray steps
   if nows>0
      gw2=[];
      gw=zeros(1,noc)*255;
      for w=1:nows
         gw2=[gw2;gw];
      end
      gs2=[gs2;gw2];
   else
      gs2=[gs2;gw2];
   end


end
gs2;


% Orientation
if ori=='l2r'
   gs2=rot90(gs2);
else % or=='t2b'
   gs2=gs2;
end
```

```
gs2=uint8(gs2);

% Output
gs=gs2;
imwrite(gs,'gs3.tif','resolution',res);
end
%------------------------------------------------------------
%------------------------------------------------------------
```

Program A.2

The following program function (in Matlab 6.1) creates tile arrangement of photos and with separate condition gives black borders to a single photo

```
function []=PhotoTileBSP(fname,nohp,novp,wp)

%PHOTOTILEBSP : Creates tile arrangement of photos
%----------------------------------------------------------
%----------------------------------------------------------
% []=PhotoTileBSP(fname,nohp,novp,wp)
% PHOTOTILEBSP (function) : PHOTO TILE Black SPace
%
%
%
% EXAMPLES:
% PhotoTileBSP('pout.tif',5,4,10)
% PhotoTileBSP('pout.tif',1,1,10) Give black borders to a single photo
%
% Tested using Matlab 6.1.0.450 Release 12.1
%----------------------------------------------------------
%----------------------------------------------------------

i=imread(fname);
[r c dim]=size(i);
wp; % black spaces 10 rows/columns

% Generating vertical black spaces (width=wp, height=r)
z=zeros(r,wp);
z=z*255;

if dim==3
  z=zeros(r,wp,3);
  z=z*255;
```

```
end

% Creating horizontal strip of tiling with black spaces in between
i2=[];
for j=1:nohp % nohp: no of horizontal photos
    i2=[i2,z,i];
end

[r2 c2 dim2]=size(i2);
% Last vetical strip of black spaces
zc=zeros(r2,wp);
zc=zc*255;

if dim==3
   zc=zeros(r2,wp,3);
   zc=zc*255;
end

% Appending white spaces to the right of last column
i2=[i2,zc];
[r3 c3 dim3]=size(i2);

% Generating horizontal black spaces (width=c3, height=wp)
z2=zeros(wp,c3);
z2=z2*255;

if dim==3
   z2=zeros(wp,c3,3);
   z2=z2*255;
end

i3=[];
```

```
for k=1:novp % novp: no of vertical photos
    i3=[i3;z2;i2];
end

[r4 c4 dim4]=size(i3);

% Last horizontal strip of black spaces
z3=zeros(wp,c4);
z3=z3*255;

if dim==3
   z3=zeros(wp,c4,3);
   z3=z3*255;
end

% Adding it to the bottom of tiled photos.
i3=[i3;z3];

% i3=logical(i3); % only to get binary images, not valid for
% graycale image (uncomment this line for grayscale images)

imwrite(i3,'phototile.tif')
imshow(i3)
%----------------------------------------------------------
```

References

1. Allebach, J.P., Liu, B., “Analysis of halftone dot profile and aliasing in the discrete binary representation of images”, J. Optical Society America, Vol. 67, no.9, pp.1147-1154, 1977.
2. Analoui , M. and Allebach , J. P., “Model-based Halftoning using Direct Binary Search”, Proceedings of the 1992 SPIE/IS&T Symposium on Electronic Imaging Science and Technology, Vol. 1666, San Jose, CA, pp. 96-108, 1992.
3. Bandyopadhyay, Dr. Swati, “Effects of Screen Ruling and Screen Shape on Image Quality”, Proceedings of IS&T’s Image Processing, Image Quality, Image Capture Systems Conference, Pages 311-314, Savannah, GE, April 25-28 1999. IS&T.
4. Bayer, B.E., “An Optimum Method for Two Level Rendition of Continuous-Tone Pictures”, Proc. IEEE Int. Conf. Commun., Conference Record, pp.(26-11)-(26-15)., 1973.
5. Billmeyer, F.W. & Saltzman, M., “Principles of Color Technology”, John Wiley & Sons, USA, 1981.
6. Eschbach, R. and R. Hauck, “Binarization using a two dimentional pulse-density modulation”, J.Opt. Soc. of America, Vol. 4, no.10, pp.1873-1878, 1987
7. Eschbach, Reiner, “Pseudo-Vector Error Diffusion Using Cross Separation Threshold Imprints”, Proceedings of IS&T’s Image Processing, Image Quality, Image Capture Systems Conference, Pages 321-323, Savannah, GE, April 25-28 1999. IS&T.

8. Fan, Zhigang, "Error Diffusion for CMYK Color Images", Proceedings of IS&T's Image Processing, Image Quality, Image Capture Systems Conference, Pages 324-326, Savannah, GE, April 25-28 1999. IS&T.

9. Field, Gary G., Color and its Reproduction, GATF Press, USA, 1999

10. Floyd, R. W. and Steinberg, L, "Adaptive algorithm for spatial grey scale", Proc. Society Information Display, Vol. 17/2, pp. 36-37, 1976.

11. Gonzalez, R.C., Woods, R.E., "Digital Image Processing", 2nd ed., Prentice Hall, Uper Saddle River, NJ, 2002

12. Gonzalez, R.C., Woods, R.E., Eddins, Steven L., "Digital Image Processing Using MATLAB", Pearson Education, Delhi, India, 2004

13. Gooran S, "Context Dependent Color Halftoning in Digital Printing", proceedings of IS & T's 2000 PICS Conference, pp. 242-246, March 2000, Portland, USA.

14. Gooran, S and Kruse, B, "Color Halftoning in Digital Printing", IARIGAI 26th Research Conference, Advances in Digital Printing, Sept. 1999, Munich, Germany.

15. Gooran, S., "High Quality Frequency Modulated Halftoning". Thesis, Linköping University, Linköping, Sweden, 2001

16. Gooran, S., and Kruse, B., "Near-optimal model-based halftoning technique with dot gain", SPIE, Human Vision and Digital Display III, San Jose, 1998.

17. Gooran, S., Österberg, M., and Kruse, B., "Hybrid halftoning –A Novel Algorithm for Using Multiple Halftoning Technologies", Proc. IS&T Int. Conf. On Digital Printing Technologies (NIP12), pp. 79-86, 1996.

18. Gusev, D. A., “Anti-Correlation Digital Halftoning by Generalized Russian Roulette”, Proceedings of IS&T’s Image Processing, Image Quality, Image Capture Systems Conference, Pages 327-332, Savannah, GE, April 25-28 1999. IS&T.

19. Hearn D., and Baker M.P., “Computer Graphics, C Version”, (Pearson Education (Singapore) Pte. Ltd., Delhi 110 092, India), 2005.

20. Herniter, Marc E., “Programming in MATLAB”, Thomson Asia, Vikas Publishing House Pvt. Ltd., New Delhi, 2003

21. Hunt, R.W.G., “The Reproduction of Colour in Photography, Printing & Television”, Fountain Press, England, 1987

22. Jarvis, J. F.; Judice, C. N.; Ninke, W. H., “A Survey of Techniques for the Display of Continuous-tone Pictures on Bilevel Displays.” Computer Graphics and Image Processing, Vol. 5, 13-40, 1976

23. Kang, H. R., “Digital Color Halftoning” New York: IEEE Press, 1999.

24. Knuth, D.E., “Digital halftones by dot-diffusion”, ACM Transactions on Graphics, 6(4):245-273, October 1987.

25. Kundu, Pradeep, and Pal, Arun Kiran, “Some Methods of Digital Halftoning”, TAGA Conference, March 15-18, 2009, New Orleans, Louisiana, USA, 2009.

26. Kundu, Pradeep, Pal, Arun Kiran; “A Novel Versatile Method Of Generating Soft Halftone Proofs”, TAGA Conference, March 12-21, 2012, Jacksonville, Florida, USA.

27. Kundu, Pradeep, Pal, Arun Kiran; “Some Methods of Non-halftone Binary Image Transformations”, International Journal of Intelligent Information Processing, ISSN: 09-3892, Vol. 4 Number 2, July-December 2010, Page Nos: 167-172

28. Lau, D.L., Arce G.R., “Modern Digital Halftoning” Marcel Dekker, Inc, New York, 2001.

29. Lau, D.L., Arce G.R., and Gallagher, N.C., “ Digital color halftoning via generalized error-diffusion and vector green-noise masks”, IEEE Transactions on Image Processing, Vol.9, no.5, May, 2000.

30. Lau, D.L., Arce G.R., and Gallagher, N.C., “Digital color halftone with generalized error diffusion and multichannel green-noise masks”, IEEE Trans. On Image Processing, Vol.9, no.5, pp.923-935, 1998

31. Lau, D.L., Arce G.R., and Gallagher, N.C., “Digital halftoning green-noise masks”, Journal of the Optical Society of America, Vol. 16, no.7, pp.1575-1586, July 1999.

32. Lau, D.L., Arce G.R., and Gallagher, N.C., “Green-noise digital halftoning”, Proc of the IEEE, Vol. 86, no.12, pp.2424-2444, December 1998.

33. Lau, D.L., Arce, G.R., “Robust halftoning with green noise”, Proceedings of the IS&T’s Image Processing, Image Quality, Image Capture Systems Conference, Pages 315-320, Savannah, GE, April 25-28 1999. IS&T.

34. McCann, John J., “Color Theory and Color Imaging Systems: Past, Present and Future”, Journal of Imaging Science and Technology, Vol.42, Number 1, January/February 1998.

35. Mertle, J.S. and Monsen, G.L. “Photomechanics and Printing: Practical Information on Platemaking and Presswork by Recognized Procedures”, Oxford & IBH Publishing Co., New Delhi, India, 1969.

36. Mese, M., and Vaidyanathan, P.P., Improved Dot Diffusion for Image Halftoning, Proceedings of NIP 15: International Conference of Digital Printing Technologies, pages 350-353, Orlando, FL, October 17-22, 1999, IS&T.

37. Nilsson, F., “Pre-computed frequency modulated halftoning maps that meet the continuity criterion”, Proceedings of the IS&T International Conference on Digital Printing Technologies (NIP12), pp. 72-77, 1996.

38. Otsu, N., “A Threshold Selection Method from Gray-Level Histograms” IEEE Transactions on Systems, Man, and Cybernetics., Vol.9, no. 1, pp. 62-66, 1979.

39. Pratt, William K., “Digital Image Processing”, Wiley-Interscience, New York, 2001.

40. Russ, J.C, “The Imageprocessing Handbook”, 3rd ed., CRC Press, Boca Raton, FL, 1999.

41. Shapiro, “The Lithographers Manual”, GATF, USA, 1981

42. Shiau, J.N., Fan, Z. “A set of easily implementable coefficients in error-diffusion with reduced worm artifacts” In J.Bares, editor, Color Imaging: Device-Independent Color, Color Hard Copy, and Graphics Arts, Volume 2658, pages 222-225, SPIE, March 1996.

43. Stevenson, R.L. and G.R.Arce, “Binary display of hexagonally sampled continuous-tone images”, J.Opt. Soc. America, Vol.2, no.7, pp.1009-1013, 1985.

44. Stucki, P., “MECCA –A Multiple-Error Correcting Computation Algorithm for Bilevel Image Hardcopy Reproduction”, Research Report RZ1060, IBM Research Laboratory, Zurich, Switzerland, 1981.

45. Sullivan, J.; Miller, R.; Pios, G., ”Image Halftoning Using a Visual Modeling Error Diffusion” Journal of Optical Society of America, Vol. 10, 1714-1724, 1993.

46. Tritton, K., “Colour Control in Lithography”, PIRA International, U.K., 1993.

47. Ulichney, R., “Digital Halftoning”, MIT Press, 1987.

48. Ulichney, R., “Dithering with Blue Noise”, Proc. IEEE, Vol.76, no.1, 1988.

49. Ulichney, R., “Frequency Analysis of Ordered dither”, Proc. SPIE, Hardcopy Output, Vol.1079, pp.361-373, 1989.

50. Ulichney, R., “The Void-and-Cluster Method for Generating Dither Arrays”, IS&T/SPIE Symposium on Electronic Imaging Science & Technology, San Jose, CA, Vol. 1913, Feb. 1-5, pp. 332-343, 1993

51. Velho, L., Gomes, J.M., “Stochastic screening dithering with adaptive clustering”, Computer Graphics, pp.273-276, 1995.

52. Wyszecki, G.; Stiles, W. S., “Color Science”, New York (et al.): John Wiley & Sons, INC, 2000.

53. Yule, J. A. C,. “Principle of Color Reproduction”, New York: John Wiley and Sons, 1967.