



**US Army Corps
of Engineers**
Waterways Experiment
Station

Technical Report ITL-94-1
February 1994

②

AD-A277 896



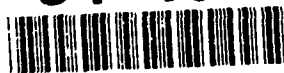
Theory and Application of Image Enhancement

*by Michael G. Ellis, Sr., Roy L. Campbell, Jr.
Information Technology Laboratory*

DTIC
S **E** **D**
ELECTE
APR 06 1994

Approved For Public Release; Distribution Is Unlimited

94-10259



DTIC QUALITY CONTROLLED 3

94 4 4 151

Prepared for Headquarters, U.S. Army Corps of Engineers

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.



PRINTED ON RECYCLED PAPER

Theory and Application of Image Enhancement

by Michael G. Ellis, Sr., Roy L. Campbell, Jr.
Information Technology Laboratory

U.S. Army Corps of Engineers
Waterways Experiment Station
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

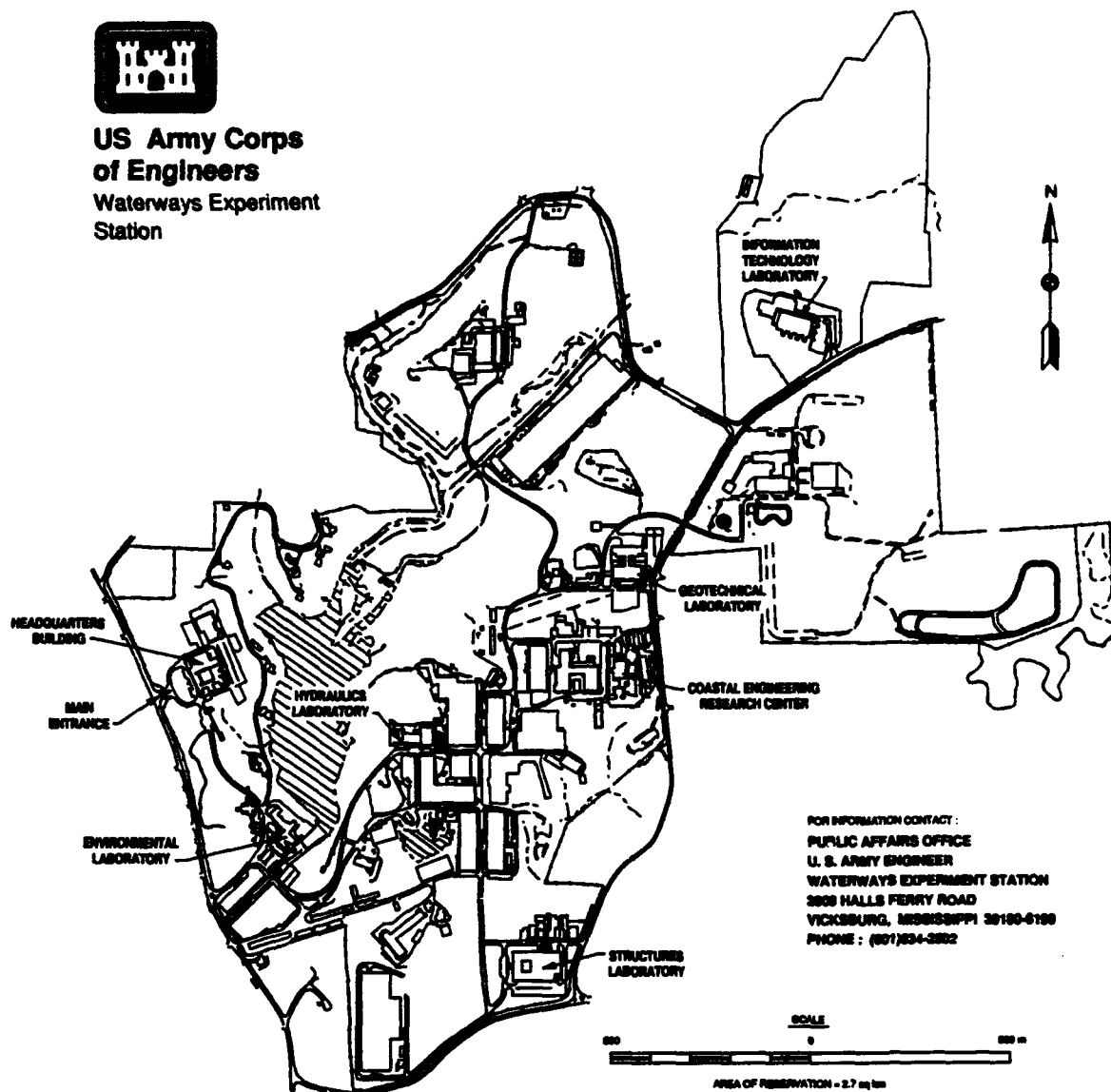
Approved for public release; distribution is unlimited

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED 3



**US Army Corps
of Engineers**
Waterways Experiment
Station



Waterways Experiment Station Cataloging-in-Publication Data

Ellis, Michael G.

Theory and application of image enhancement / by Michael G. Ellis, Sr., Roy L. Campbell, Jr. ; prepared for U.S. Army Corps of Engineers.

1 v. in various pagings : ill. ; 28 cm. — (Technical report ; ITL-94-1)

Includes bibliographical references.

1. Image compression. 2. Transformations (Mathematics) 3. Image processing. 4. Data transmission systems. I. Campbell, Roy L. II. United States. Army. Corps of Engineers. III. U.S. Army Engineer Waterways Experiment Station. IV. Title. V. Series: Technical report (U.S. Army Engineer Waterways Experiment Station) ; ITL-94-1. TA7 W34 no.ITL-94-1

Contents

Preface	vii
1—The Image Enhancement Project	1
2—Using Singular Value Decomposition to Remove Image Blur	4
Abstract	4
Introduction	4
Theory	5
Implementation	7
Results	9
Conclusions	14
3—Using the Discrete Fourier Transform to Remove Interference Patterns	15
Theory	15
Implementation	21
Results	21
Conclusions	26
4—Processing Images Morphologically	27
Abstract	27
Introduction	27
Theory	27
Implementation	31
Results	31
Conclusions	32
5—Detecting Edges	53
Abstract	53
Introduction	53
Theory	53
Implementation	56
Results	57
Conclusions	57

6—Crispening Images	67
Abstract	67
Introduction	67
Theory	67
Implementation	68
Results	68
Conclusions	68
7—Noise Cleaning	71
Abstract	71
Introduction	71
Theory	71
Implementation	72
Results	72
Conclusions	72
8—Image Thresholding	75
Abstract	75
Introduction	75
Theory	74
Implementation	76
Results	76
Conclusions	76
9—Contrast Manipulation	80
Abstract	80
Introduction	80
Theory	80
Implementation	81
Results	81
Conclusions	82
10—Median Filtering	93
Abstract	93
Introduction	93
Theory	93
Implementation	94
Results	94
Conclusions	94
11—Video Conferencing Using Personal Computers	97
Abstract	97
Introduction	97
Implementation	98
Future Improvements	100

12—IMAGE93 Software	102
Abstract	102
Introduction	102
Implementation	102
13—The Clear Speech Algorithm	109
References	113
Appendix A: The SVD Processor	A1
Appendix B: The Image Factory	B1
Appendix C: Vision	C1
Appendix D: The DFT Processor	D1
Appendix E: The Inverse DFT Processor	E1
Appendix F: The FFT Processor	F1
Appendix G: The Inverse FFT Processor	G1
Appendix H: The DFT Editor	H1
Appendix I: The Dilation Processor	I1
Appendix J: The Erosion Processor	J1
Appendix K: The Closure Processor	K1
Appendix L: The Opening Processor	L1
Appendix M: The Windowed Convolution Processor	M1
Appendix N: The Value Thresholding Processor	N1
Appendix O: The Occurrence Thresholding Processor	O1
Appendix P: The Square Contrast Manipulation Processor	P1
Appendix Q: The Cube Contrast Manipulation Processor	Q1
Appendix R: The Square Root Contrast Manipulation Processor	R1
Appendix S: The Cube Root Contrast Manipulation Processor	S1
Appendix T: The 1/x (Inverse) Manipulation Processor	T1
Appendix U: The Inverting Processor	U1
Appendix V: The Image Extraction Processor	V1
Appendix W: The Median Filter Processor	W1
Appendix X: The Transmit Processor	X1

Appendix Y:	The Receive Processor	Y1
Appendix Z:	The Virtual Receive Processor	Z1
Appendix AA:	The IMAGE93 Host Processor	AA1
Appendix BB:	The VESA Initializer	BB1
Appendix CC:	The Display Processor	CC1
Appendix DD:	The PCX Loader	DD1
Appendix EE:	The PCX to Binary Converter	EE1
Appendix FF:	The Image Printer	FF1
Appendix GG:	The Image Chopper	GG1
Appendix HH:	The Bitplane Processor	HH1
Appendix II:	The File Subtractor	II1
Appendix JJ:	The File Adder	JJ1
Appendix KK:	The Single File Examiner	KK1
Appendix LL:	The Two File Examiner	LL1
Appendix MM:	The Image Paster	MM1
Appendix NN:	The Huffman Encoder	NN1
Appendix OO:	The Adaptive Huffman Encoder	OO1
Appendix PP:	The Arithmetic Order 0 Encoder	PP1
Appendix QQ:	The Arithmetic Order 1 Encoder	QQ1
Appendix RR:	The Arithmetic Order 2 Encoder	RR1
Appendix SS:	The Huffman Decoder	SS1
Appendix TT:	The Adaptive Huffman Decoder	TT1
Appendix UU:	The Arithmetic Order 0 Decoder	UU1
Appendix VV:	The Arithmetic Order 1 Decoder	VV1
Appendix WW:	The Arithmetic Order 2 Decoder	WW1
Appendix XX:	The Discrete Transform Processor	XX1
Appendix YY:	The Entropy Calculator	YY1
Appendix ZZ:	The Mean Squared Error Calculator	ZZ1
Appendix AAA:	The Histogram Processor	AAA1
Appendix BBB:	The Packet Driver Specifications	BBB1
Appendix CCC:	The VESA Graphics Interface	CCC1

Preface

This report is the result of research done in fiscal year 1993 (FY93) on the theory and application of image enhancement techniques performed in the Computer Science Division (CSD), Information Technology Laboratory (ITL), U.S. Army Engineer Waterways Experiment Station (WES), Vicksburg, MS. It is the continuation of the FY92 research on image compression (Ellis 1992).

The report describes the evaluation of image processing techniques including singular value decomposition (SVD), the two-dimensional Fourier transform, median filters, morphological transforms, edge detection, image crispening, noise cleaning, and other enhancement techniques. The FY92 program, IMAGE92, was completely rewritten by Mr. Michael G. Ellis, Sr., and Mr. Roy L. Campbell, Jr. (the authors), to include both the FY92 compression programs and the FY93 enhancement programs. The complete listing for IMAGE93 is given in the Appendixes.

Both the FY92 and FY93 research are preludes for developing three-dimensional image processing techniques adaptable to use in real-time applications. The prototype desktop conferencing system developed by the authors as part of this research effort is intended to be a test application for the demonstration of these algorithms in FY94. An image processing minilab, established during this research effort, has been equipped with the hardware digital signal processing (DSP) tools needed for real-time implementation of image compression and enhancement algorithms.

The work was accomplished at WES under the supervision of Dr. N. Radhakrishnan, Director, ITL, and Dr. Wendell F. Ingram, Chief, CSD.

At the time of publication of this report, the Director of WES was Dr. Robert W. Whalin. The Commander was COL Bruce K. Howard, EN.

1 The Image Enhancement Project

The image enhancement project reported herein is a continuation of Fiscal Year 1992 (FY92) research at the U.S. Army Engineer Waterways Experiment Station (WES) on image compression (Ellis 1992). The FY92 work developed new techniques for image compression yielding very high compression ratios. Research conducted in FY93 focused on image enhancement techniques, and the results of that research are fully described in this report. The software IMAGE92 was rewritten to include both the compression techniques studied and developed in FY92 and the enhancement techniques researched in FY93. The new software, IMAGE93, incorporates many algorithms not normally attempted on personal computers (PC's) because of memory limitations, including singular value decomposition (SVD) and two-dimensional (2-D) Fourier transforms.

An offshoot of the FY93 project was the establishment of an image-processing minilab equipped with the hardware and software necessary to perform image analysis research. Part of the minilab is shown in Figure 1. The equipment consists largely of PC's, dedicated digital signal processors (DSP), video capture boards, network programming utilities, and various types of scientific and imaging libraries. A complete list of the resources of the minilab is given in Table 1.

Suggestions for research into FY94 will include the extension of the image compression and enhancement algorithms into 3-D and modification for real-time applications. A prototype desktop conferencing system has been developed as part of the



Figure 1. A portion of the image processing minilab at WES

FY93 project as a possible application to extend and test the FY92 and FY93 compression and enhancement algorithms. This desktop video conferencing system currently transmits uncompressed voice and video over the WES ethernet. The incorporation of the compression and enhancement algorithms into this prototype system is intended to provide a suitable demonstration of the utility and feasibility of video collaboration tools over local and wide area networks.

Table 1
Capabilities of WES Image Analysis Minilab

Computers and Equipment	
Four 80486 PCs One SGI IRIS 4D/25G workstation Access to local SUN and CRAY Macintosh IIfx computer with CDROM Novell 2.15C PC Fileserver	Laserprinter Bernoulli Drive CDROM 3-1/2-in. (8.9 cm) 21 Mbyte Floptical Drive Courier 9600 Baud (V.32) modem
Digital Signal Processor Development Stations	
Texas Instruments TMS320 Digital Signal Processor Evaluation module Texas Instruments ELF-31 (TMS32031) Digital Signal Processor Development module Motorola DSP56002 Development Station	
Programming Languages	
Microsoft Assembly Visual Basic	Microsoft C++7.0
Programming Enhancement Tools	
C-Scape software library TSRific (TSR development software) Builder Software Toolkit C Windows Toolchest	Zinc Programming Library Video for Windows Programming Library Windows Programming Library
Network Programming Tools	
Netlib (for Novell) Packet Driver Software Development Specifications	FTP Socket Library
Video and Sound Capture Boards	
CompuAdd Multimedia TV Board Super Video-SL Video Capture Board with Developers Kit 16-bit Targa Bravado Board with Developers Kit SuperVia Video Capture Board MicroKey Digiview Video Capture Board	Action Media II Video Capture Board Pro DigiTV Video Capture Board Video ProMovie Spectrum Video Capture Board Two Sound Blaster Pro 16 Boards with Developers Kit
Video Equipment	
Three NTSC cameras VHS VCR	TV Spectrum Analyzer, 2 Mhz to 1 Ghz
Speech Recognition and Synthesis	
Dragon Dictate voice recognition Speech Secretary II voice recognition	Phonetic Engine voice recognition Echo PCII Speech Synthesis Board
Image Processing Software Libraries	
ImageLab Software (FY92 version) Image93 Software (FY93 version) SuperVIA Image Processing software	Victor image processing library Imaging objects software library
Math and Scientific Libraries for the PC	
Unpack scientific software library Scientific subroutines by Wiley	Touchstone/DOS & Linecalc software MATLAB software
CDROM Library	
Simtel20 CICA MS Windows USENET Source CDROM INFO-MAC	GIFS OS/2 X11R/GNU CUG Library

A Fiber Distributed Data Interface (FDDI) network, operating at 100 Mbps, is currently used as the backbone for a local area network serving over 2,000 nodes within the WES 685-acre (277-hectare) facility. Without image compression, the operation of just a few video collaboration devices on the WES network would be sufficient to exhaust the capacity of the entire FDDI backbone. Even the installation of a high-speed SONET backbone would not suffice to allow all WES users to concurrently participate in video collaboration and desktop conferencing. The ability to perform real-time image compression and enhancement will be vital in the creation of video collaboration tools to support future work.

The remainder of this report will concentrate on the research performed during FY93. Complete source code listings are included in the appendixes. A combination of BASIC, C, and 8086 assembly language was used in writing IMAGE93. The main menu is a Visual Basic program that shells to DOS to run a C program, depending on what options are chosen. Each individual menu item shells to a different C program. The C programs often include embedded assembly language routines when processing speed is required. Although most tasks can be performed in 640K memory, the SVD and 2-D Fourier transforms require at least 8 Mbytes of RAM in the computer. A full set of source and compiled code occupies four high-density 3-1/2-in. floppy disks. A super video graphics adapter (SVGA) is recommended using an appropriate VESA graphics interface to support the graphic modes required by the program.

The prototype desktop conferencing software is independent of IMAGE93 and consists of three network applications that use the TCP/IP protocol as a transport mechanism for voice and video. This set of three programs will also be described later in this report.

2 Using Singular Value Decomposition to Remove Image Blur

Abstract

According to Scharf (1991), SVD is an acceptable method for decomposing a noisy transmission matrix into a noise matrix and a refined approximation of the true signal. This chapter extends this concept to blurred images, defining the blur as noise and the sharpened image as the refined signal. The theory as well as the implementation of the SVD are discussed in detail.

Introduction

Blur is considered to be the result of a system's transfer function. If the system is analogous to a stationary camera, the transfer function will likely cause nominal blurring; if the system is more compatible to a moving camera, the transfer function will probably contain notable blurring. If an impulse response can be obtained, an SVD can be applied to separate the noise or blur tendencies from the image-producing abilities of the transfer function (Huang 1979). The SVD is a numerical analysis method used to convert a matrix into three corresponding components: two eigenvector sets and one eigenvalue set (Pratt 1991). The resulting sets are ordered such that the majority of the noise or blur can be extracted by removing the last few values of each set. Construction of the image with the noise members removed eliminates the majority of the blur (Huang 1979).

Theory

Any linear system can be modeled by the following equation (Ziemer, Tranter, and Fannin 1989):

$$G = H \otimes F \quad (1)$$

where

G = output response

H = impulse response or transfer function

F = system input

$\otimes$ = convolution

With respect to images, G is the blurred image and F is the original or ideal image; H is the transfer function or impulse response of the image-processing system. The following equation describes the ideal image F (Huang 1979):

$$F = [H]^{-1} G \quad (2)$$

where

F = ideal image

G = blurred image

$[H]^{-1}$ = Moore-Penrose pseudoinverse of H

This equation can be manipulated into a form for which the SVD is useful (Huang 1979):

$$F = \sum_{i=1}^R \lambda_i^{-1/2} v_i u_i^t g \quad (3)$$

where

F = ideal image

R = arbitrary integer

λ_i = eigenvalue of H

v_i = eigenvector from the V set of H

u_i^t = eigenvector from the transpose of the U set of H

g = column of the blurred image

To understand the variables involved, a discussion of SVD is needed.

Of all the other algorithms, SVD is the most efficient energy-packing technique in the least square sense (Jain 1989). The SVD is based primarily on the following equation (Pratt 1991):

$$U^T H V = \sqrt{\Lambda} \quad (4)$$

where

U^T = transpose of the U eigenvector set

H = impulse response matrix

V = V eigenvector set

Λ = eigenvalue set

Since only the impulse response H is known, there are three unknowns: the U and V eigenvector sets and the eigenvalue set. To obtain these three unknowns, an iterative method, the SVD, must be applied. The SVD employs a Householder reduction as well as other complicated manipulations, which are all beyond the scope of this report (Press et al. 1988).

The dimensions of the impulse response matrix are important to the usefulness of the SVD. If the width is less than or equal to the height, a unique solution can be expected; otherwise, an infinite set of solutions can be expected (Press et al. 1988). To simplify the theory, only cases with unique solutions will be used.

Once the matrix is decomposed, it can be reconstructed with the following equation (Pratt 1991):

$$H = U \sqrt{\Lambda} V^T \quad (5)$$

The inverse matrix or Moore-Penrose pseudoinverse can then be described by the following equation:

$$[H]^{-1} = V \sqrt{\lambda^{-1}} U^T \quad (6)$$

Noise can be removed by zeroing out the last few columns in the V matrix, the last few rows in the U transpose matrix, and the last few diagonal elements in the eigenvalue matrix, as shown in the following equation (Pratt 1991):

$$[H]^{-1} = \begin{bmatrix} V_{11} & V_{1R} & 0 & 0 & \sqrt{\lambda_1^{-1}} & 0 & \dots & 0 & U_{11} & U_{21} & \dots & U_{N1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \sqrt{\lambda_2^{-1}} & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \sqrt{\lambda_R^{-1}} & \vdots & U_{1R} & U_{2R} & \dots & U_{MR} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 & \vdots & \vdots & 0 \\ V_{N1} & V_{NR} & 0 & 0 & 0 & \vdots & \vdots & \vdots & 0 & \vdots & \vdots & 0 \end{bmatrix} \quad (7)$$

where

M = width of impulse response matrix

N = height of impulse response matrix

Although the subscripts on the U matrix seem to be backwards, they signify the transpose of U . The subscript R is an arbitrary value and is dependent on the specific impulse response.

Construction of the ideal image can be performed according to Equation 2. However, this construction can be done one eigen-set at a time (where one eigen-set consists of one U eigenvector, one V eigenvector, and one eigenvalue) as described in Equation 3 and as repeated in Equation 8 (Pratt 1991, Huang 1979):

$$F = \sum_{i=1}^R \lambda_i^{-1/2} v_i u_i^t g \quad (8)$$

where

F = ideal image

R = arbitrary integer

λ_i = eigenvalue of H

v_i = eigenvector from the V set of H

u_i^t = eigenvector from the transpose of the U set of H

g = column of the blurred image

This method provides versatility in adding and removing a given eigen-set's effect on the composite image.

Implementation

For still images, deducing an impulse response is unlikely without full control of the image-capturing system: the camera. Therefore, no blur removal could be performed; however, the SVD's efficient energy-packing ability could be demonstrated by replacing the impulse response matrix

with an image I in Equations 4 and 5. Amounts of energy could then be associated with the eigen-set's placement in the series.

The C code in Appendix A was written to perform SVD on an entire image. The dynamic memory allocation and deallocation as well as the SVD computation subroutines were found in *Numerical Recipes in C* (Press et al. 1988); however, the SVD computation was heavily modified. All other codes pertain to data management, file management, and image reconstruction.

With regard to data management, a large memory pool was needed since the SVD was expected to handle an entire image. This memory pool needed to be approximately 7 Mbytes in size; therefore, extended PC memory had to be used. This task was handled with the Victor Library (Catenary Systems 1992); the Victor Library is capable of interfacing between C-based code and the extended memory windowing system of the PC.

For file management, the major issue was the internal and external file formats. A standard 8-bit palette definition was chosen as the standard external file format so that the input and the reconstructed output could be viewed. In order to preserve the exactness of the SVD's results, internal files, used for eigenvector and eigenvalue storage, were created as a list of floating point numbers. These internal files were necessary to allow future reconstruction without requiring SVD computation because the SVD's operation time can be lengthy.

Image reconstruction was performed according to the following equation, which is based on Equation 5 (Pratt 1991):

$$I \approx \sum_{i=1}^R \sqrt{\lambda(i)} u_i v_i^T \quad (9)$$

where

I = image matrix

R = arbitrary integer

λ = eigenvalue of the image

u_i = eigenvector from the image's U set

v_i^T = eigenvector from the image's transpose of the V set

The display routine was designed to demonstrate the reconstruction one eigen-set at a time, giving visual evidence that the first few eigen-sets contain the majority of the image energy. The demonstration could be recreated at any time with the second program, located in Appendix B.

The code in Appendix B was written for reconstruction and image-manipulation purposes. The program, first, reconstructs the image one

eigen-set at a time. Afterwards, the user has the opportunity to remove or add eigen-sets. All display images can be saved within the program in the external file format.

The code in Appendix C was written to display external files. One VGA resolution mode (320X200) and three SVGA resolution modes (640X480, 800X600, and 1024X768) are available.

Results

Figures 2-8 demonstrate the inverse proportionality of the image energy to the eigen-set number. Figure 2 is the original image. Figure 3 is the original image absent the effects of the first eigen-set; as can be seen, the image is heavily degraded because the first eigen-set contains the most energy of all the eigen-sets. The removal of eigen-set 2 from the original image, as shown in Figure 4, has some minor effects. All other removals have only subtle effects (Figures 5-8).

Figures 9-22 show a reconstruction progression as individual eigenvalue influences are added. Dramatic change is evident in the first seven additions; however, the effects of further additions severely taper as the eigen-set number increases.



Figure 2. Original image



Figure 3. Original with eigen-set 1 removed



Figure 4. Original with eigen-set 2 removed



Figure 5. Original with eigen-set 8 removed



Figure 6. Original with eigen-set 16 removed

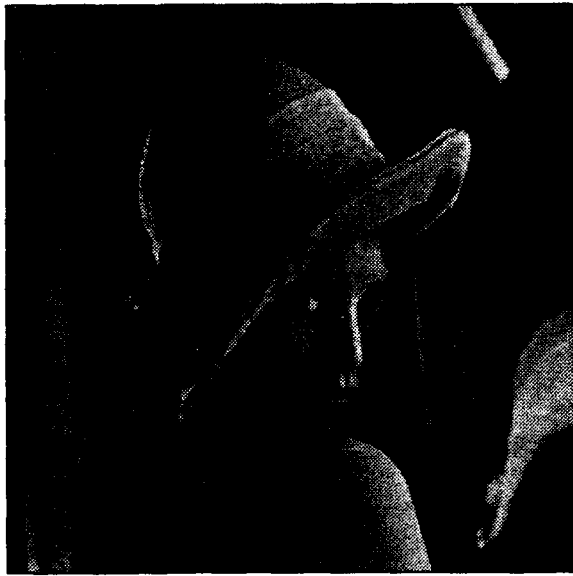


Figure 7. Original with eigen-set 64 removed



Figure 8. Original with eigen-set 256 removed

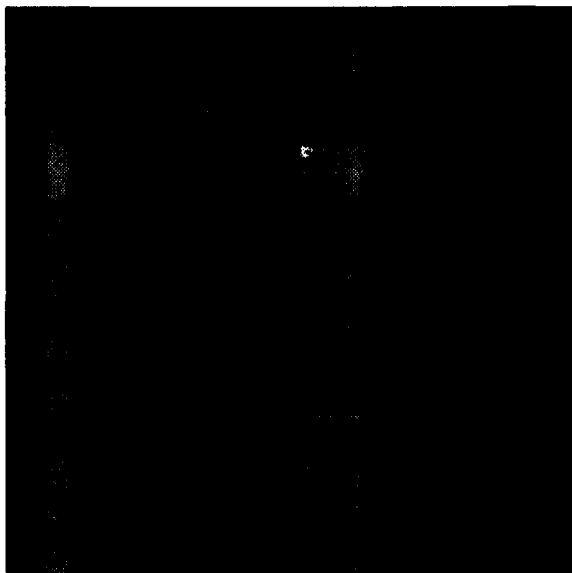


Figure 9. Reconstruction using the first eigen-set

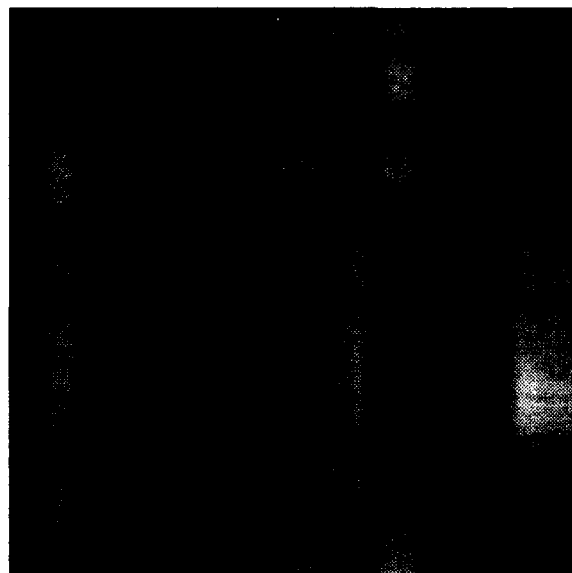


Figure 10. Reconstruction using the first two eigen-sets

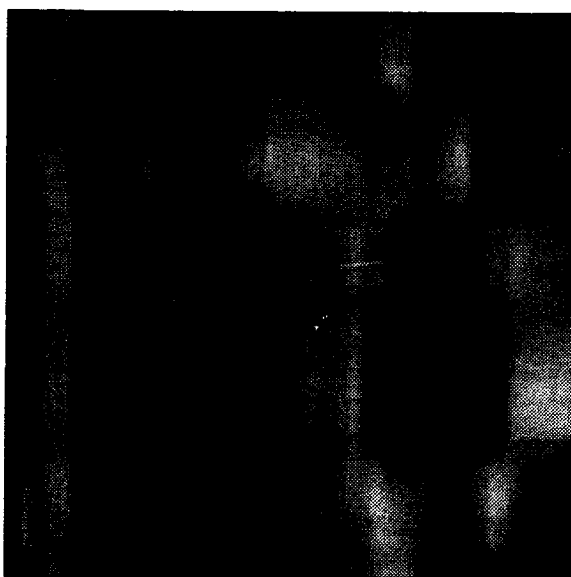


Figure 11. Reconstruction using the first three eigen-sets

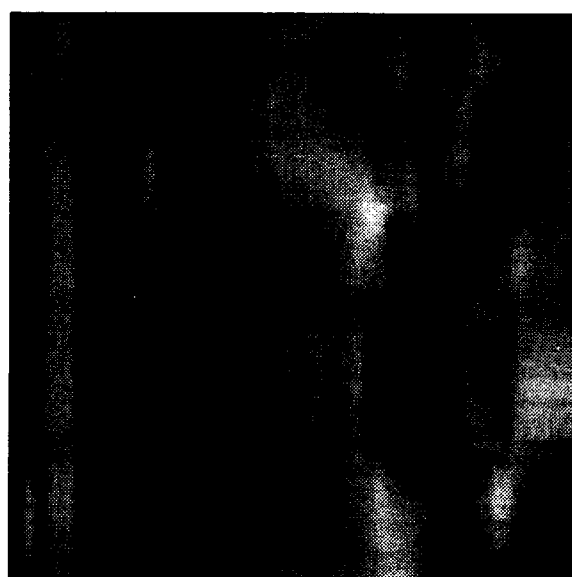


Figure 12. Reconstruction using the first four eigen-sets

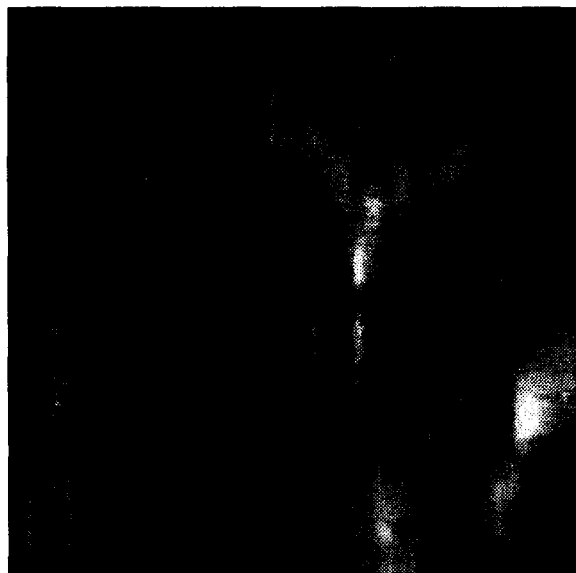


Figure 13. Reconstruction using the first five eigen-sets

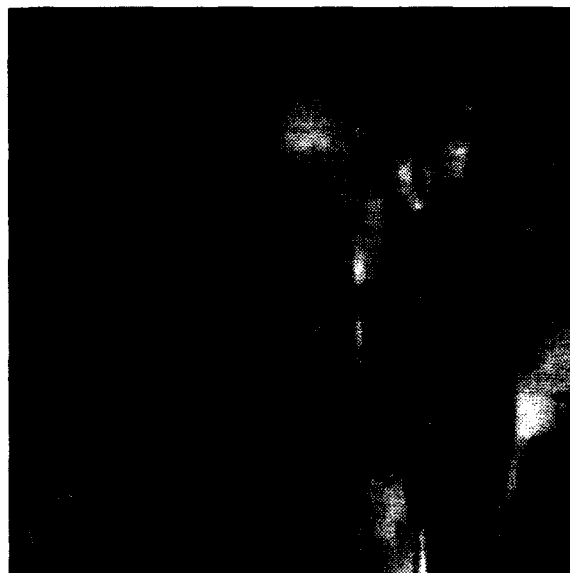


Figure 14. Reconstruction using the first seven eigen-sets



Figure 15. Reconstruction using the first nine eigen-sets



Figure 16. Reconstruction using the first 12 eigen-sets



Figure 17. Reconstruction using the first 15 eigen-sets



Figure 18. Reconstruction using the first 18 eigen-sets



Figure 19. Reconstruction using the first 21 eigen-sets



Figure 20. Reconstruction using the first 24 eigen-sets



Figure 21. Reconstruction using the first 27 eigen-sets



Figure 22. Reconstruction using the first 30 eigen-sets

Conclusions

For cases in which the image-capturing system cannot be modeled, SVD is not a feasible method for deblurring. However, if a system's impulse response or transfer function is known, SVD is capable of separating the parts of the transfer function that cause noise from those that transfer the image.

3 Using the Discrete Fourier Transform to Remove Interference Patterns

Theory

The Discrete Fourier Transform (DFT) is a direct result of the Fourier Transform (FT). The following equation describes the 1-D FT over a continuous space (Ziemer, Tranter, and Fannin 1989):

$$\begin{aligned} g(x) &= \int_{-\infty}^{\infty} G(f) e^{j2\pi fx} df \\ G(f) &= \int_{-\infty}^{\infty} g(x) e^{-j2\pi fx} dx \end{aligned} \tag{10}$$

where

$g(x)$ = continuous space representation

$G(f)$ = continuous frequency representation

For an image, the space is more accurately represented in a discrete form as shown:

$$g_s(x) = \sum_{n=-\infty}^{\infty} g(x) \delta(x - nX) \tag{11}$$

where

$x = cX$

c = arbitrary interger

X = spacing between samples

$g_s(x)$ = function of discrete space
 $g(x)$ = function of continuous space

Using the following property of the impulse function, Equation 13 can be deduced (Ziemer, Tranter, and Fannin 1989):

$$g(x) \delta(x - x_o) = g(x_o) \delta(x - x_o) \quad (12)$$

$$g_s(cX) = \sum_{n=-\infty}^{\infty} g(nX) \delta(cX - nX) \quad (13)$$

Substituting the $g_s(cX)$ for $g(x)$ in the second equation of Equation 10, the following equations result:

$$\begin{aligned} G(f) &= \int_{-\infty}^{\infty} \left[\sum_{n=-\infty}^{\infty} g(nX) \delta(x - nX) \right] e^{-j2\pi f x} dx \\ G(f) &= \sum_{n=-\infty}^{\infty} g(nX) \left[\int_{-\infty}^{\infty} \delta(x - nX) e^{-j2\pi f x} dx \right] \\ G(f) &= \sum_{n=-\infty}^{\infty} g(nX) e^{-j2\pi f nX} \end{aligned} \quad (14)$$

The last equation is a direct result of the sifting property of the impulse function shown in the following equation (Ziemer, Tranter, and Fannin 1989):

$$\int_{-\infty}^{\infty} g(x) \delta(x - x_o) dx = g(x_o) \quad (15)$$

Since the sample space is discrete, the frequency representation will be discrete:

$$f = kF$$

where

k = arbitrary integer
 F = base frequency

$$\begin{aligned}
g(kF) &= \sum_{n=-\infty}^{\infty} g(nX) e^{-j2\pi(kF)nX} \\
g(k) &= \sum_{n=-\infty}^{\infty} g(nX) e^{-j2\pi knX}
\end{aligned} \tag{16}$$

Assuming the nonzero region of $g(nX)$ is observed for $\{n: 0 \leq n \leq N-1\}$, X represents a fraction of some unit length such that $NX = 1$ unit; also, the limits of n are reduced to 0 and $N-1$:

$$G(k) = \sum_{n=0}^{N-1} g(n/N) e^{-j2\pi kn/N} \tag{17}$$

With the following definitions, Equation 18 can be deduced:

$u(n) \triangleq g(n/N)$, $v(k) \triangleq G(k)$, and $W_n \triangleq e^{-j2\pi n/N}$ (Jain 1989).

$$v(k) = \sum_{n=0}^{N-1} u(n) W_N^{kn}, \text{ for } 0 \leq k \leq N-1, k \in I \tag{18}$$

where

$v(k)$ = function of discrete frequency

$u(n)$ = function of discrete space

This 1-D result can be extended to the following 2-D equation, which represents the 2-D DFT (Jain 1989):

$$v(k, l) = \sum_{m=0}^{M-1} \left[\sum_{n=0}^{N-1} u(m, n) W_N^{ln} \right] W_M^{km}, \tag{19}$$

for $0 \leq k \leq M-1$ and $0 \leq l \leq N-1$, $k, l \in I$

where

$v(k, l)$ = function of 2-D discrete frequency

$u(m, n)$ = function of 2-D discrete space

k = vertical frequency index (row)

l = horizontal frequency index (column)

m = vertical sapce index (row)

n = horizontal space index (column)

M = image height

N = image width

The 2-D inverse DFT, shown in Equation 20, can be developed in a similar manner (Jain 1989).

$$u(m, n) = \frac{1}{MN} \sum_{k=0}^{M-1} \left[\sum_{l=0}^{N-1} v(k, l) W_N^{-ln} \right] W_M^{-km} \quad (20)$$

Although the DFT is easy to implement, it is extremely time-consuming; therefore, the Fast Fourier Transform (FFT) needs to be investigated. The FFT provides exactly the same domain translation that the DFT does, but it is exceedingly faster. When the 1-D case described in Equation 18 is analyzed, this method takes advantage of the periodicity of W_N^{nk} (Rabiner and Gold 1975):

$$W_N^{nk} = W_N^{(n+cN)(k+dN)} \quad (21)$$

where c, d = arbitrary integers

To develop the method, a discrete function $u(n)$ is assumed to describe a row in an image such that its length is N , where N is a power of two. This row can be broken into two subrows as shown:

$$\begin{aligned} u_1(n) &= u(2n) \\ u_2(n) &= u(2n+1) \end{aligned} \quad (22)$$

where $n = 0, 1, 2, \dots, N/2 - 1$

Combining Equations 18 and 22, the following results:

$$v(k) = \sum_{\substack{n=0 \\ n \text{ even}}}^{N-1} u(n) W_N^{nk} + \sum_{\substack{n=0 \\ n \text{ odd}}}^{N-1} u(n) W_N^{nk} \quad (23)$$

Adjusting the summation indices to match those in Equation 22 and compensating this change by adjusting the argument of the summation, the following equations are achieved:

$$\begin{aligned}
v(k) &= \sum_{n=0}^{\frac{N}{2}-1} u(2n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} u(2n+1) W_N^{(2n+1)k} \\
v(k) &= \sum_{n=0}^{\frac{N}{2}-1} u_1(n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} u_2(n) W_N^{(2n+1)k} \\
v(k) &= \sum_{n=0}^{\frac{N}{2}-1} u_1(n) W_N^{2nk} + W_N^k \cdot \sum_{n=0}^{\frac{N}{2}-1} u_2(n) W_N^{2nk}
\end{aligned} \tag{24}$$

$$v(k) = U_1(k) + W_N^k \cdot U_2(k)$$

Since Equation 24 only describes the first half of the frequency components, it must be extended according to the general definition's periodic nature (Rabiner and Gold 1975):

$$\begin{aligned}
v(k) &= U_1(k) + W_N^k \cdot U_2(k) & \text{for } 0 \leq k \leq N/2 - 1 \\
v(k) &= U_1(k - \frac{N}{2}) + W_N^k \cdot U_2(k - \frac{N}{2}) & \text{for } N/2 \leq k \leq N - 1
\end{aligned} \tag{25}$$

To provide the FFT with a standard operation scheme, the row $u(n)$ is halved until all subrows are two pixels long such that each subrow can be transformed by the following:

$$\begin{aligned}
V_1 &= U_1 + W_N^k \cdot U_2 \\
V_2 &= U_1 - W_N^k \cdot U_2
\end{aligned} \tag{26}$$

This equation describes the FFT butterfly shown in Figure 23 (Rabiner and Gold 1975). This initial set of transformations is only the first stage in the FFT process; there are $\log_2 N$ stages. Figure 24 shows complete FFT schematic for a row eight pixels long (Rabiner and Gold 1975).

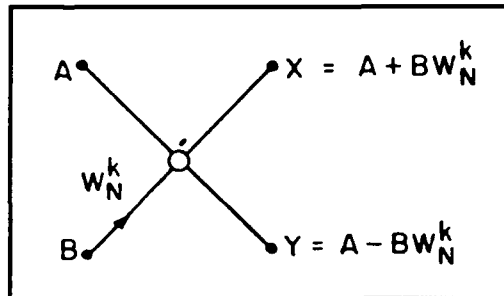


Figure 23. FFT butterfly

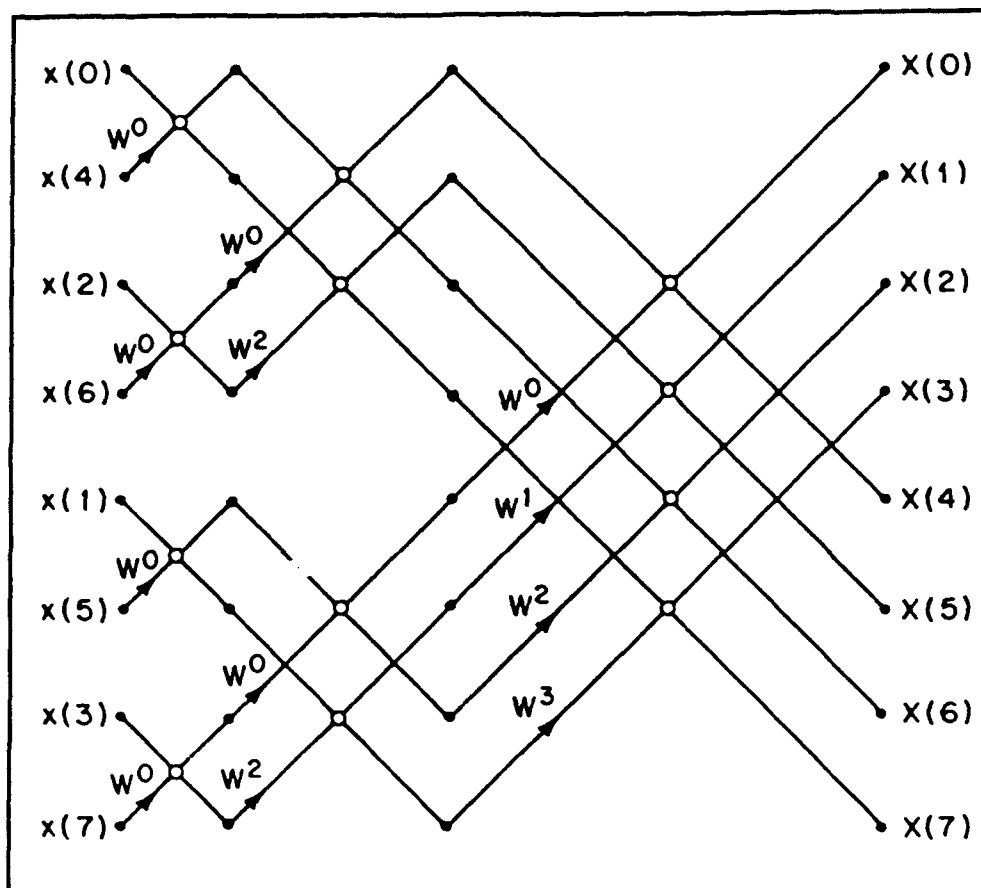


Figure 24. FFT schematic for row with eight pixels

Table 2
Example of Bit Reversal

Index	Binary	Bit-Reversed	New Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

The row has to be reordered as implied in Equation 22 and as shown in Figure 24; a simple technique can be used to perform this shuffling: index bit reversal. This concept is illustrated in Table 2 (Rabiner and Gold 1975).

The inverse FFT can be performed in three basic steps (Rabiner and Gold 1975): (1) All values of $v(k)$ are conjugated. (2) The FFT is applied to $v(k)$. (3) The resulting image row is divided by N .

For 2-D images, the 1-D FFT can be used. First, each row is processed; using these results, each column is processed. The outcome is the conjugate of the results obtained if a 2-D DFT were applied. To inverse the process, all values of $v(k,l)$ are first conjugated; then, the FFT is applied to each row. The results are divided by N . Next, the FFT is applied to each column of this outcome. The results are divided by N , providing a restored 2-D image.

Implementation

The C code in Appendix D performs the DFT on an image; this code is based on Equation 19. Extended memory management was handled with the Victor Library (Catenary Systems 1992). The external file format, used for reading the original image and storing the reconstructed image, is binary or standard RGB; internal files, used for storing the real and imaginary planes of the DFT, are a list of floating point numbers.

The code in Appendix E performs the inverse DFT; it is based on Equation 20 and also uses extended memory.

Appendix F contains the code for the FFT. The FFT subroutine is a modified version of the FFT routine found in a publication by Rabiner and Gold (1975). This program loads an image into extended memory, computes the real and imaginary planes of the DFT, and stores these results to hard disk. The real and imaginary DFT planes are displayed.

Appendix G contains the code for the inverse FFT. This program loads the real and imaginary planes of the DFT into extended memory; then, the image is reconstructed, stored to hard disk, and displayed.

The code in Appendix H is a program used for editing both planes of the DFT. First, the two planes are loaded into extended memory. Next, three editing options are given: (1) display a section of the DFT graphically, (2) edit a DFT entry, (3) show a section of the DFT textually, and (4) zero out a block. The graphical display allows the user to spot interference flares in the DFT. Autoscaling is performed for each zoomed-in section so that peaks and valleys become evident. Interference usually appears as a tumor-like figure on the DFT. When the user zooms in on this "tumor," its existence becomes evident; its topology is analogous to a mountain. The fourth option can be used to level this mountain. Upon exiting, the user can choose to save the modified DFT file.

Results

Due to the lengthy processing time, the DFT was limited in use; however, it did give interesting insight into the symmetries of its real and imaginary planes. Figure 25 shows a 4x4 image and both of its DFT planes. Disregarding the first row and first column on both the planes, there is an obvious symmetry; however, this symmetry only holds when the height and width are even. Figure 26 shows a 5x5 image and both of its DFT planes. Absent the first row and column on both

IMAGE				REAL DFT				IMAGINARY DFT			
1	2	5	1	270	-72	6	-72	0	24	0	-24
9	24	7	0	-99	-1	-99	143	73	1	-83	3
15	1	90	2	-36	-86	204	-86	0	-24	0	24
8	27	3	75	-99	143	-99	-1	-73	-5	83	-1

Figure 25. Summary of 4x4 image

planes, a definite symmetry exists, which is different from the even height and even width case. This symmetry is unique only to images with odd width and height.

IMAGE					REAL DFT					IMAGINARY DFT				
1	2	5	1	10	358	-81.1	29.6	29.6	-81.1	0	-27	26.7	-26.7	27
9	24	7	0	9	-151.3	36.3	62.9	-127.1	132.9	-10.6	4.2	20.3	22.2	-13.8
15	1	90	2	20	19.8	-1.9	-172.8	173.1	-86.9	6.6	4.5	48.6	30.9	-30.8
8	27	3	75	5	19.8	-86.9	173.1	-172.8	-1.9	-6.6	30.8	-30.9	-48.6	-4.5
18	2	13	7	4	-151.3	132.9	-127.1	62.9	36.3	10.6	13.8	-22.2	-20.3	-4.2

Figure 26. Summary of 5x5 image

The results of the DFT code matched hand calculations; therefore, this program was used to aid in troubleshooting the FFT code. Once the FFT program was perfected, it was used thereafter, since it provided such enormous time savings. A time test was conducted on a 64x64 image. The processing time for the DFT was 40 min, and the time for the FFT was 10 sec, giving a time ratio of 240:1. The theoretical time ratio is approximately 341:1 as shown:

$$\text{2-D DFT No. of operations} = N^4$$

$$\text{2-D FFT No. of operations} = 2N^2 \log_2 N \quad (27)$$

$$\text{RATIO} = \frac{N^2}{2 \cdot \log_2 N} = \frac{64^2}{2 \cdot \log_2 64} \approx 341$$

where $N = \text{width} = \text{height}$.

A herringbone-type of interference occurs in television when the video carrier is not spaced exactly 3.579545 MHz below the chrominance carrier as illustrated in Figure 27 (Grob 1975). The FFT was applied to this herringbone-type image (Figure 28) and edited with the FFT Editor. Then, the inverse FFT was applied; the results are shown in Figure 29. Although traces of interference are still present, the majority of the interference has been removed.

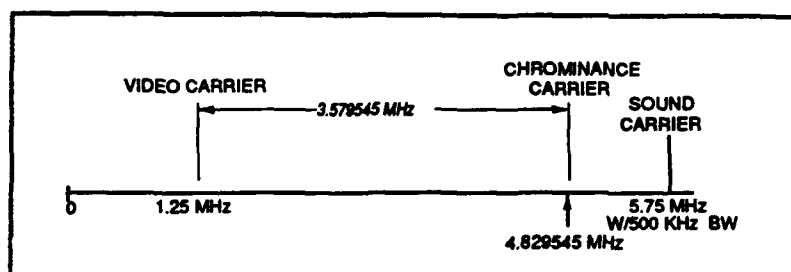


Figure 27. Standard NTSC

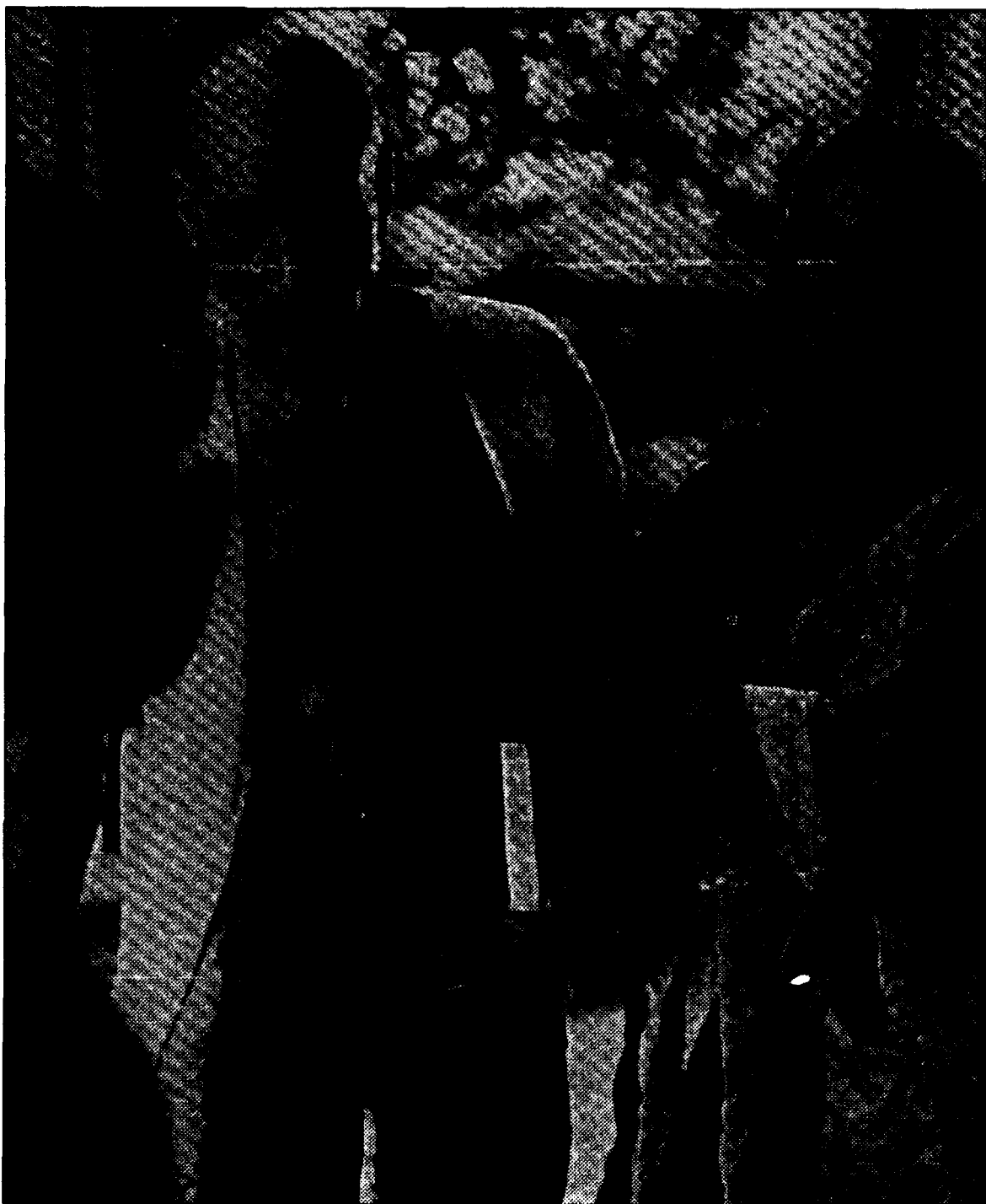


Figure 28. Image with heavy interference pattern

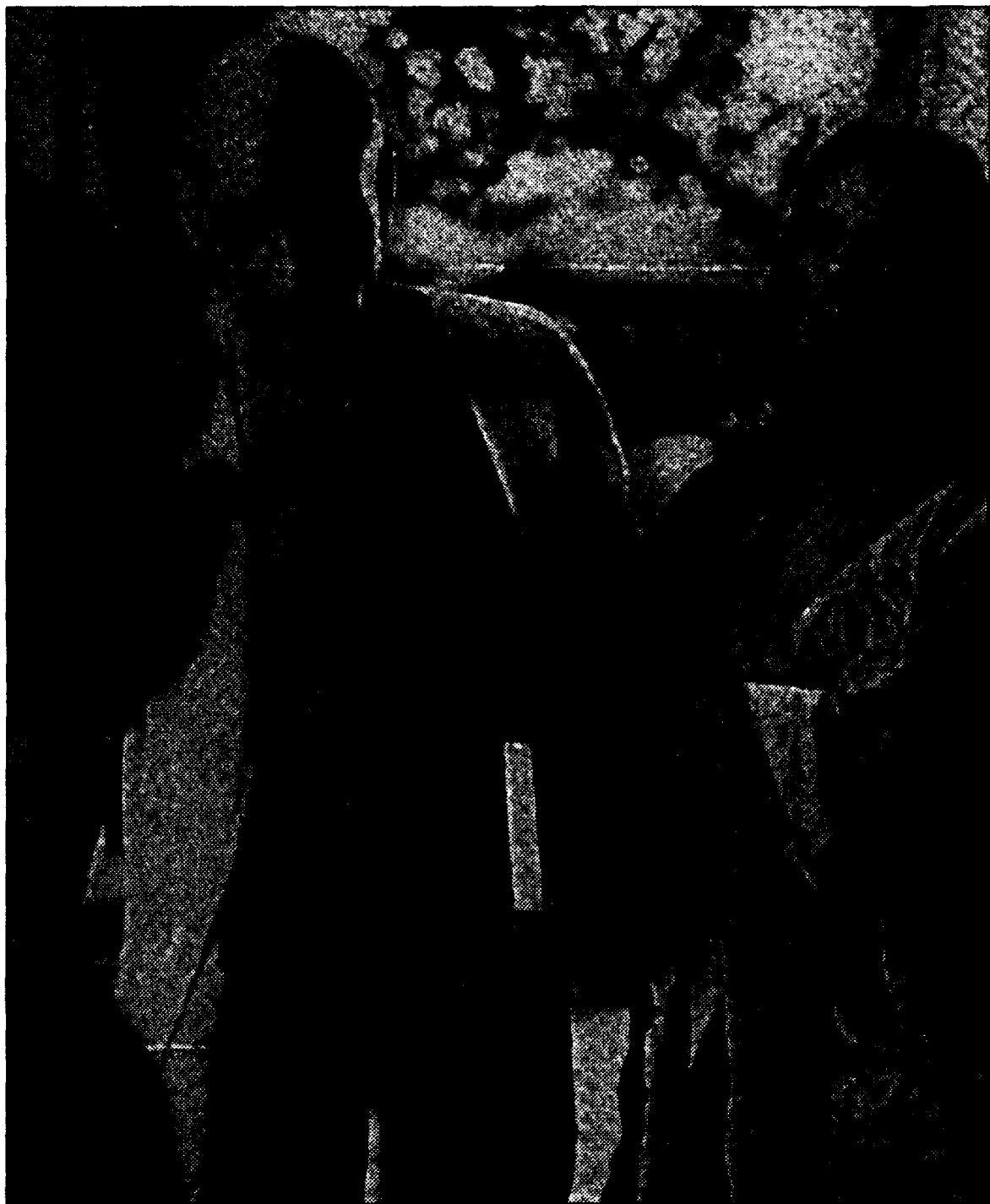


Figure 29. Image with interference reduced

While debugging the inverse FFT code, an interesting peculiarity was discovered. For example, an original image is provided in Figure 30. If the FFT is not conjugated before row operations are performed, the reconstructed image is flipped both vertically and horizontally, as shown in Figure 31. If the FFT is conjugated before row operations and again just before column operations, the image is flipped vertically, as shown in Figure 32. Finally, if the FFT is conjugated only previously to the column operations, the image is flipped horizontally, as shown in Figure 33.



Figure 30. Original image before conjugating



Figure 31. Image flipped both vertically and horizontally

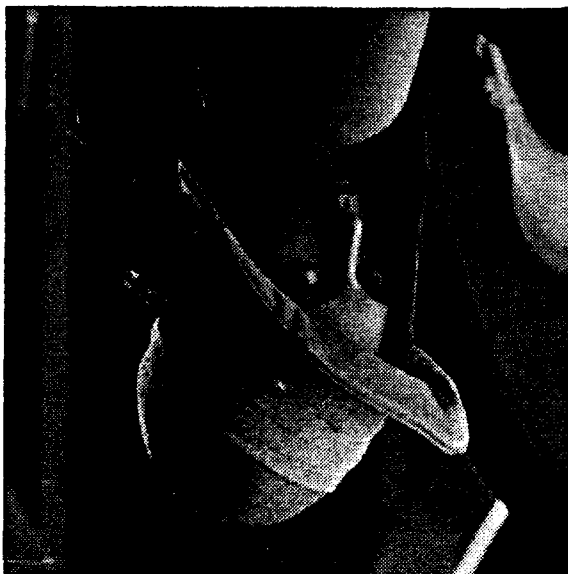


Figure 32. Image flipped vertically

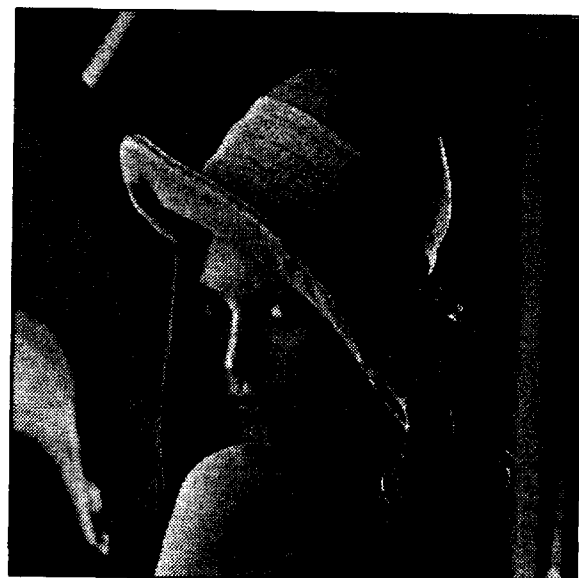


Figure 33. Image flipped horizontally

Conclusions

The DFT is a competent method for isolating image interference; however, this method is slow. The FFT performs the same task as the DFT, but the FFT is exceptionally faster. Therefore, the FFT is the prescribed method for interference removal.

4 Processing Images Morphologically

Abstract

Morphological techniques can be used to subjectively improve or distort images. This chapter discusses the theory and implementation of the following operations: dilation, erosion, closing, and opening.

Introduction

Morphological processes enhance images in a manner that preserves the general structure of the image but manipulates a given aspect. Dilation swells dark regions often causing an image to appear thicker or fuller. Erosion does the opposite; it expands light regions, usually thinning the image. Closing consists of a dilation then an erosion. The net effect can occasionally be undesirable; however, when a given image has small gaps between its dark regions, the closing procedure joins these regions together. Opening consists of an erosion then a dilation. If the dark regions are thin, this method erases them.

All four methods are based on image set algebra, which is commonly viewed as tedious. Fortunately, shortcuts exist such that the processes can be simplified to mere magnitude comparisons (Pratt 1991).

Theory

To discuss morphological methods mathematically, potential images must be assumed to be binary, in essence, restricting each pixel to the values 0 or 1. From the mathematics, a generalization can be made for all images.

The first morphological method to be discussed is dilation; dilation is based on the following equation (Jain 1989, Pratt 1991):

$$G(j, k) = F(j, k) \oplus H(j, k) \quad (28)$$

where

$G(j, k)$ = resulting image

$F(j, k)$ = original image

$H(j, k)$ = structuring element

$\oplus$ = Minkowski addition

The structuring element is a small masklike matrix that is chosen by the user. Equation 29 contains an equivalent expression of Equation 28 (Pratt 1991):

$$G(j, k) = \bigcup_{(r, c) \in H} T_{r, c} [F(j, k)] \quad (29)$$

where

$\bigcup$ = union of a matrix series

$T_{r, c}$ = matrix translation function

The translation function spatially shifts a matrix r rows down and c columns to the right. Figure 34 contains an example of dilation by means of Equation 29 (Pratt 1991).

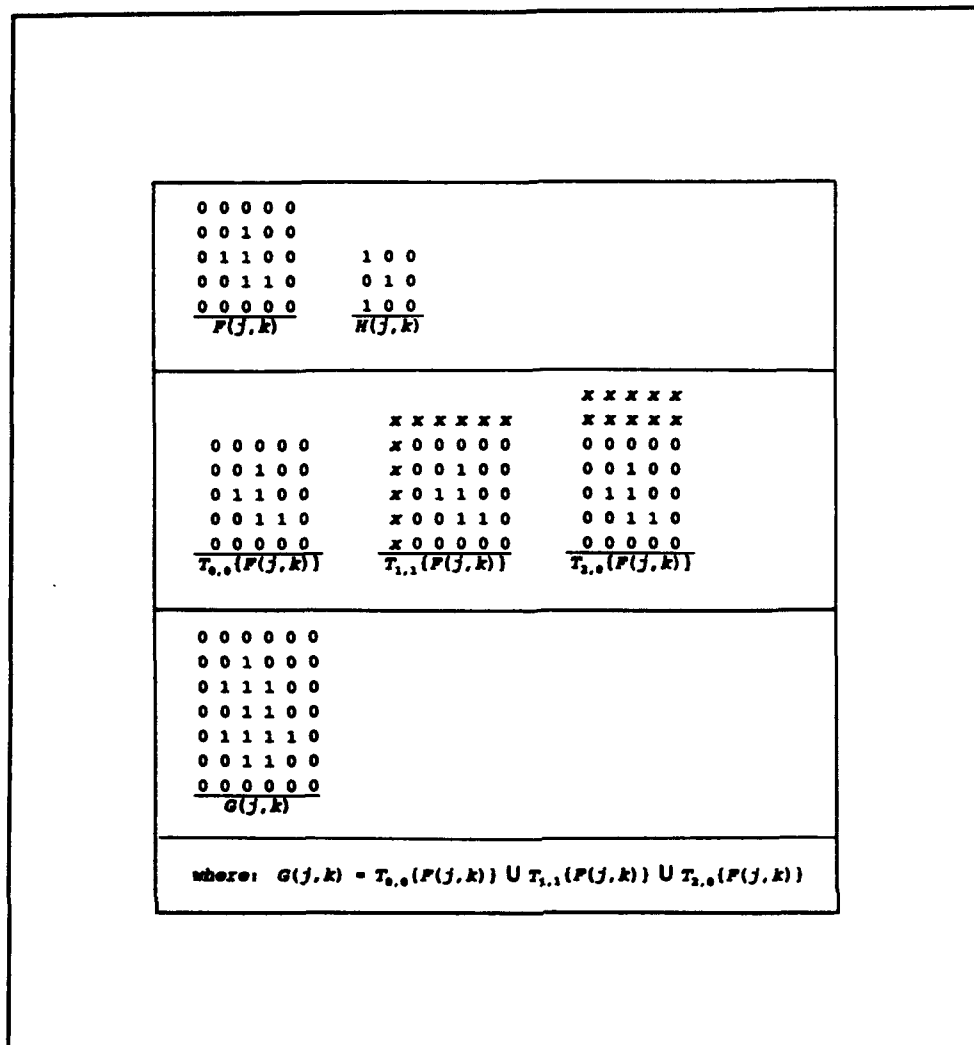


Figure 34. Example of binary image dilation

Generalizing, dilation can be performed on all images with the following equation (Pratt 1991):

$$G(j, k) = \text{MAX}[F(j+1, k+1), F(j+1, k), F(j+1, k-1), F(j, k+1), F(j, k), F(j, k-1), F(j-1, k+1), F(j-1, k), F(j-1, k-1)] \quad (30)$$

In essence, this equation finds the maximum value among a pixel's neighbors and itself. The new pixel is equal to this maximum. The neighborhood size can be increased to strengthen the dilation.

The second method is erosion; erosion is based on the following equation (Jain 1989, Pratt 1991):

$$G(j, k) = F(j, k) \ominus H(j, k) \quad (31)$$

where

- $G(j, k)$ = resulting image
- $F(j, k)$ = original image
- $H(j, k)$ = structuring element
- $\ominus$ = Minkowski subtraction

Equation 32 contains an equivalent expression of Equation 31 (Pratt 1991):

$$G(j, k) = \bigcap_{(r, c) \in H} \bigcap T_{r,c} [F(j, k)]$$

where

- $\bigcap$ = intersection of a matrix series
- $T_{r,c}$ = matrix translation function

Erosion can be performed, in general, with the following equation (Pratt 1991):

$$G(j, k) = \text{MIN}[F(j+1, k+1), F(j+1, k), F(j+1, k-1), \\ F(j, k+1), F(j, k), F(j, k-1), \\ F(j-1, k+1), F(j-1, k), F(j-1, k-1)] \quad (33)$$

In opposition to the dilation equation, this equation finds the minimum value among a pixel's neighbors and itself. The new pixel is equal to this minimum. The neighborhood size can be increased to strengthen the erosion.

The third method, closing, is described in the following equation:

$$G(j, k) = F(j, k) \bullet H(j, k) \quad (34)$$

where

- $G(j, k)$ = resulting image
- $F(j, k)$ = original image
- $H(j, k)$ = structuring element
- $\bullet$ = Minkowski closure

The Minkowski closure operation is achieved by first applying a Minkowski addition and then applying a Minkowski subtraction. Closing can be performed on all images by first using a general dilation and then using a general erosion.

The fourth method, opening, is governed by the following:

$$G(j, k) = F(j, k) \circ H(j, k) \quad (35)$$

where

$G(j, k)$ = resulting image

$F(j, k)$ = original image

$H(j, k)$ = structuring element

$\circ$ = Minkowski opening

Minkowski opening is performed by applying a Minkowski subtraction and then applying a Minkowski addition. In general, opening can be accomplished by using a general erosion then using a general dilation.

Implementation

Four programs were written for this chapter. Each uses the Victor Library to provide extended memory management (Catenary Systems 1992). The internal and external file formats are binary or standard RGB.

The code in Appendix I dilates an image according to Equation 30. Appendix J contains the code to erode an image, as described in Equation 33. The code in Appendix K performs the closure operation on an image, according to Equation 34. Appendix L contains the code to open an image, as described in Equation 35.

For each program, after the given operation is performed, the results are displayed and stored to hard disk. The neighborhood size is adjustable on each.

Results

First, the four morphological methods were used on a binary image. Figure 35 shows the original. The results of dilation, erosion, closing, and opening are contained in Figures 36-39, respectively. The closing and opening neighborhoods were increased to emphasize the effects of these two methods and to force the desired effect.

Next, five gray scale images were tested. Figures 40-64 contain the original images and the results. All four methods operated correctly for each case, but the open and close methods showed limited capabilities as described in the theory. For closing, the light regions were often too big to consume. For opening, the light regions were usually separated by

large dark regions. By increasing the neighborhood size, both the opening and closing methods could have been forced to work, but the neighborhood size is directly proportional to the resolution degradation of the image.

Conclusions

Dilation, erosion, closing, and opening are four morphological processes that can be used to manipulate an image subjectively. The end products of these methods can be beneficial although their use reduces the image resolution in proportion with the neighborhood size.



Figure 35. Original binary image



Figure 36. Dilated binary image



Figure 37. Eroded binary image



Figure 38. Closed binary image



Figure 39. Opened binary image

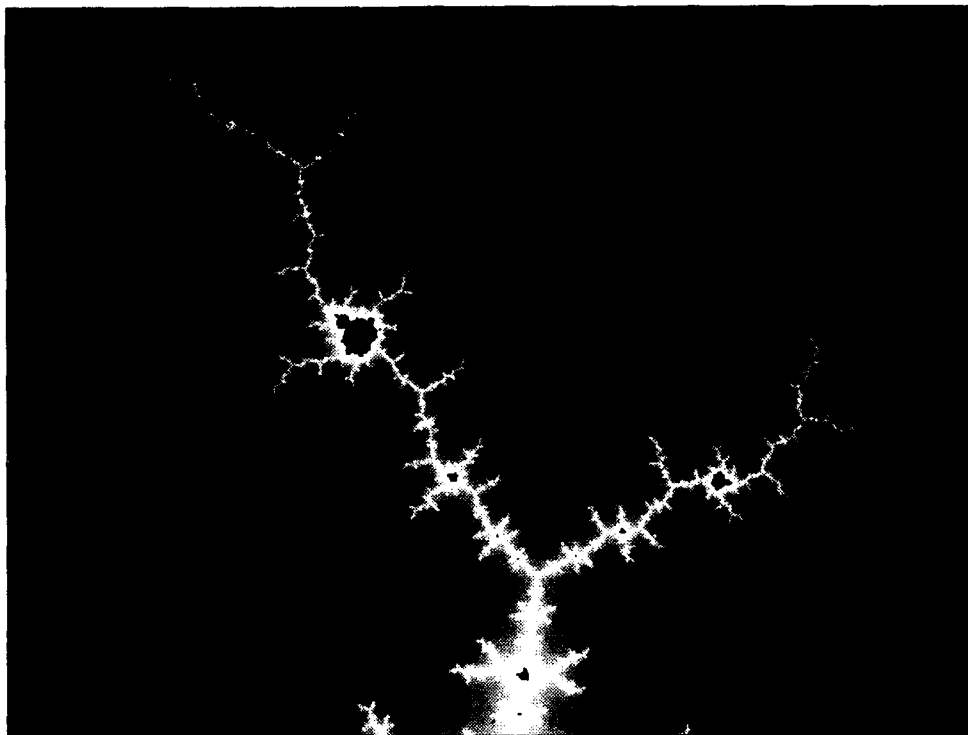


Figure 40. Example 1, original gray scale image

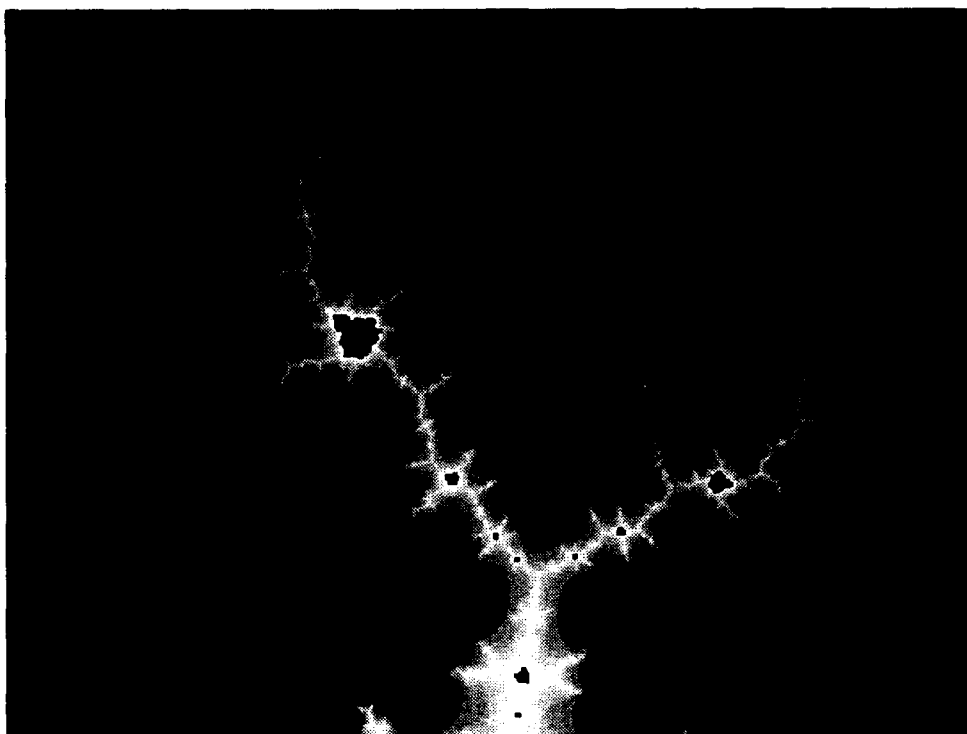


Figure 41. Example 1, dilated gray scale image

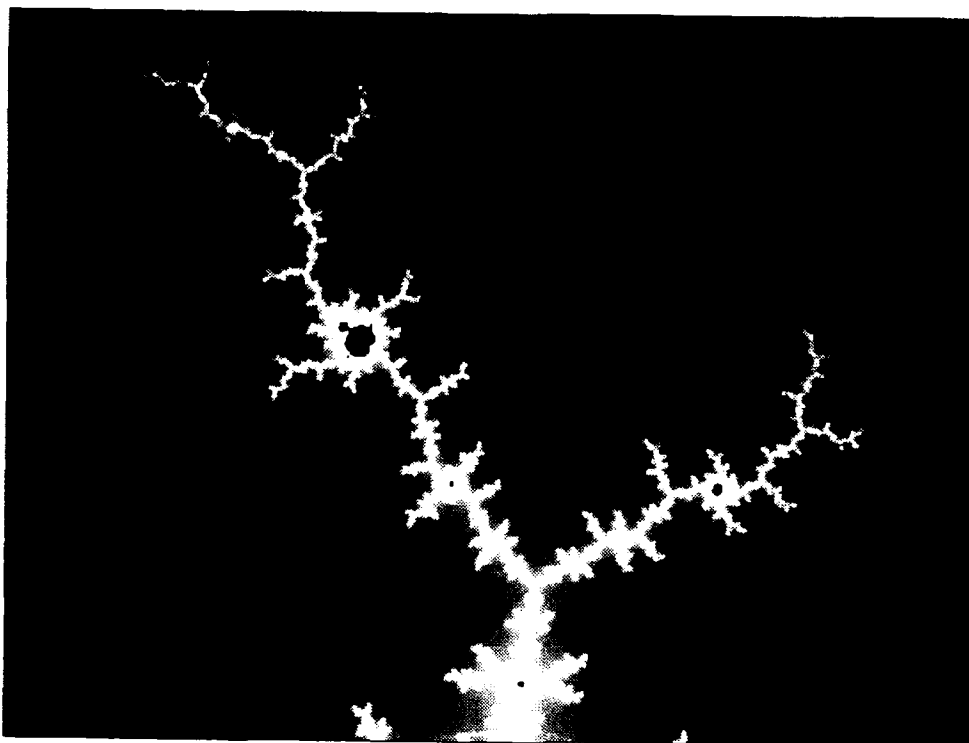


Figure 42. Example 1, eroded gray scale image

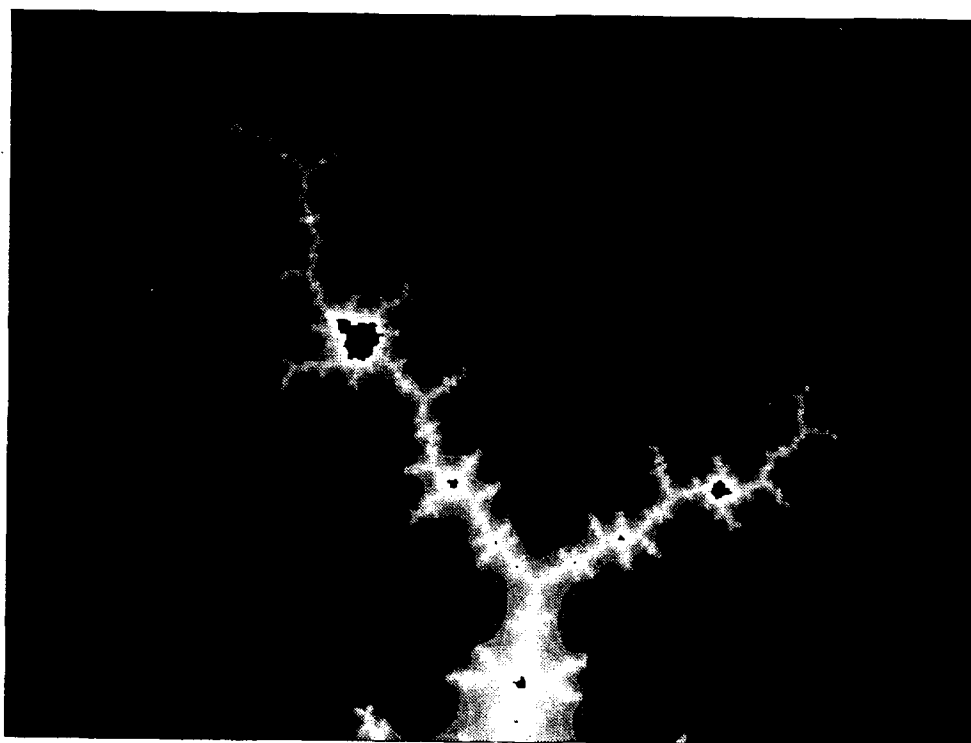


Figure 43. Example 1, closed gray scale image

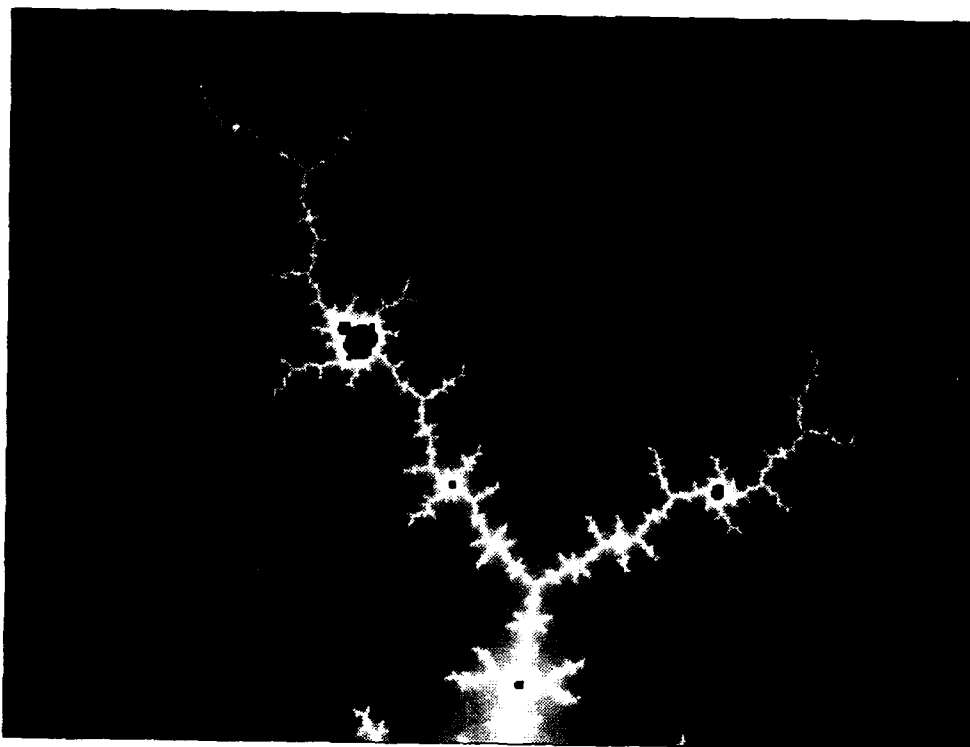


Figure 44. Example 1, opened gray scale image



Figure 45. Example 2, original gray scale image



Figure 46. Example 2, dilated gray scale image

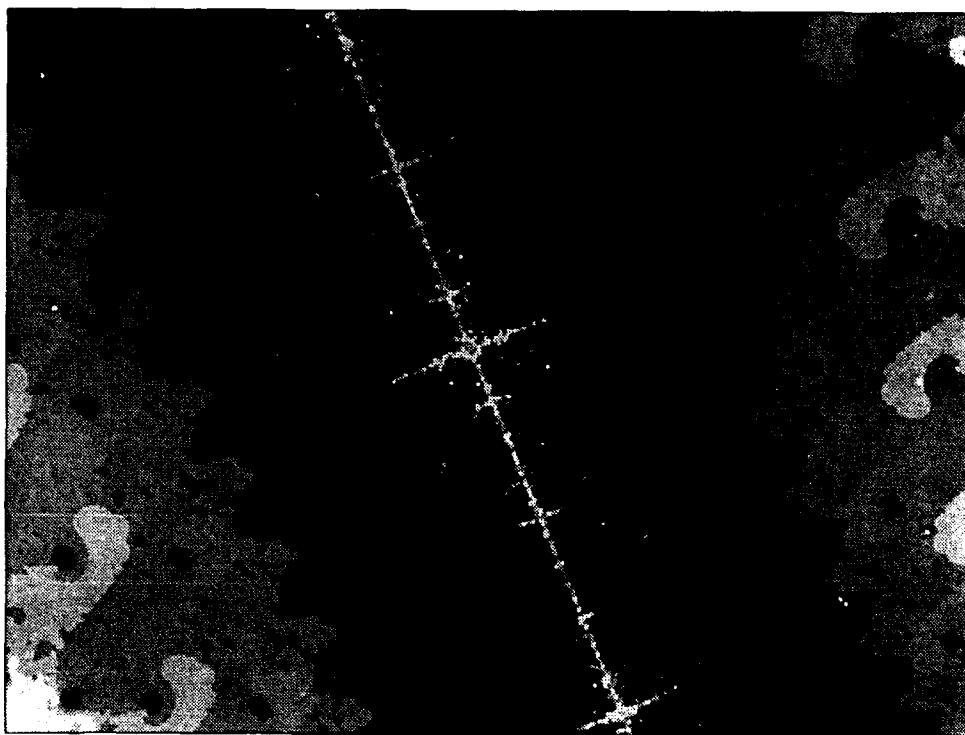


Figure 47. Example 2, eroded gray scale image

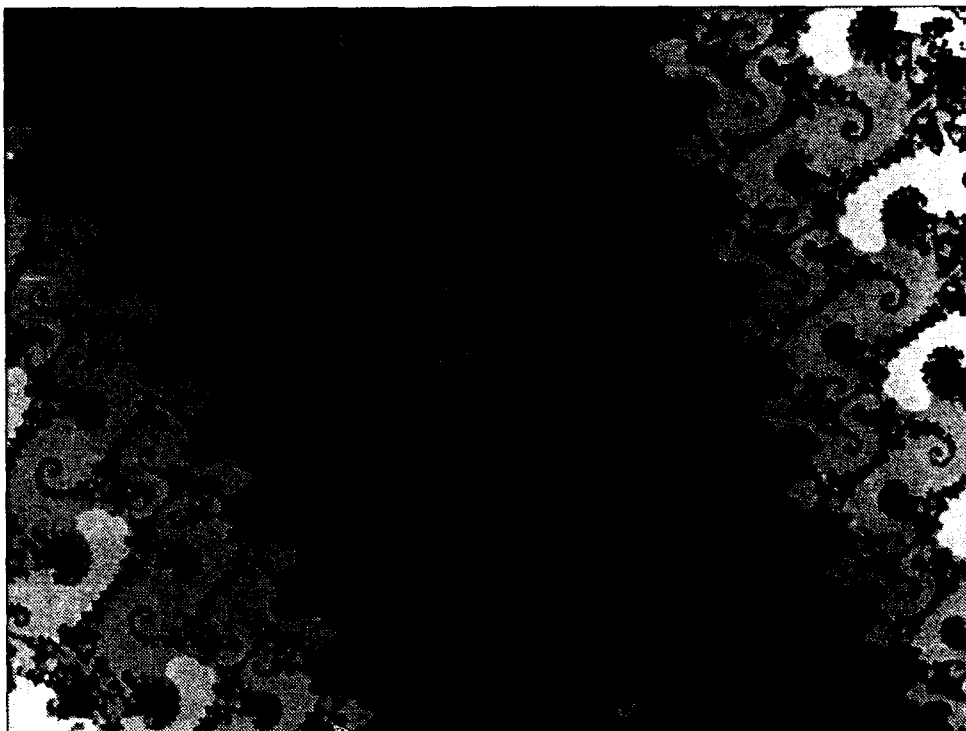


Figure 48. Example 2, closed gray scale image

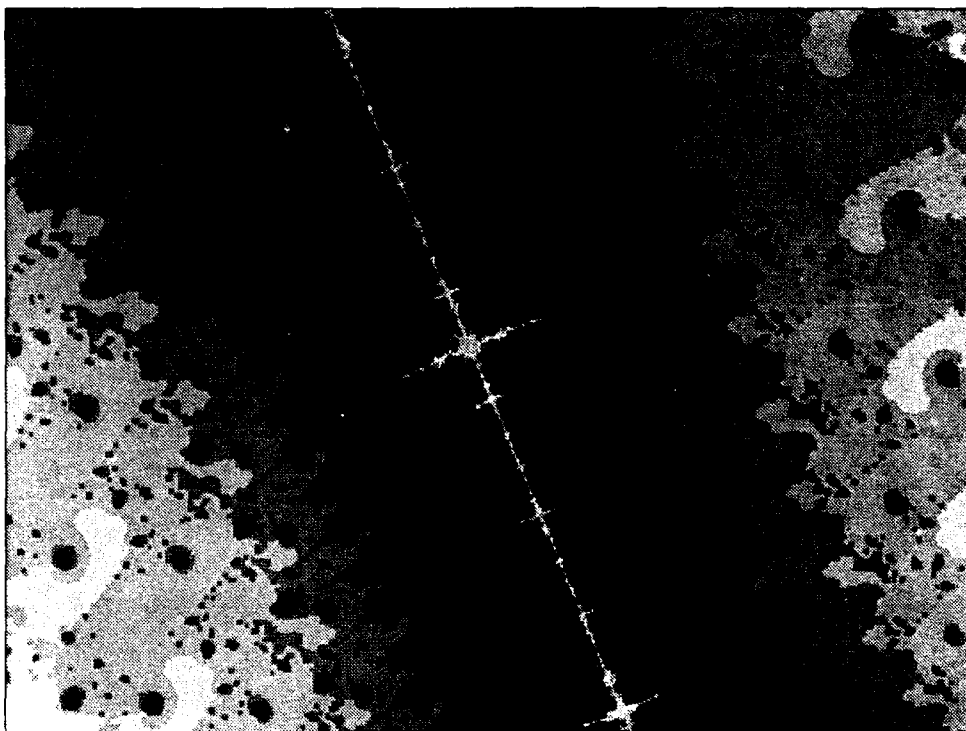


Figure 49. Example 2, opened gray scale image

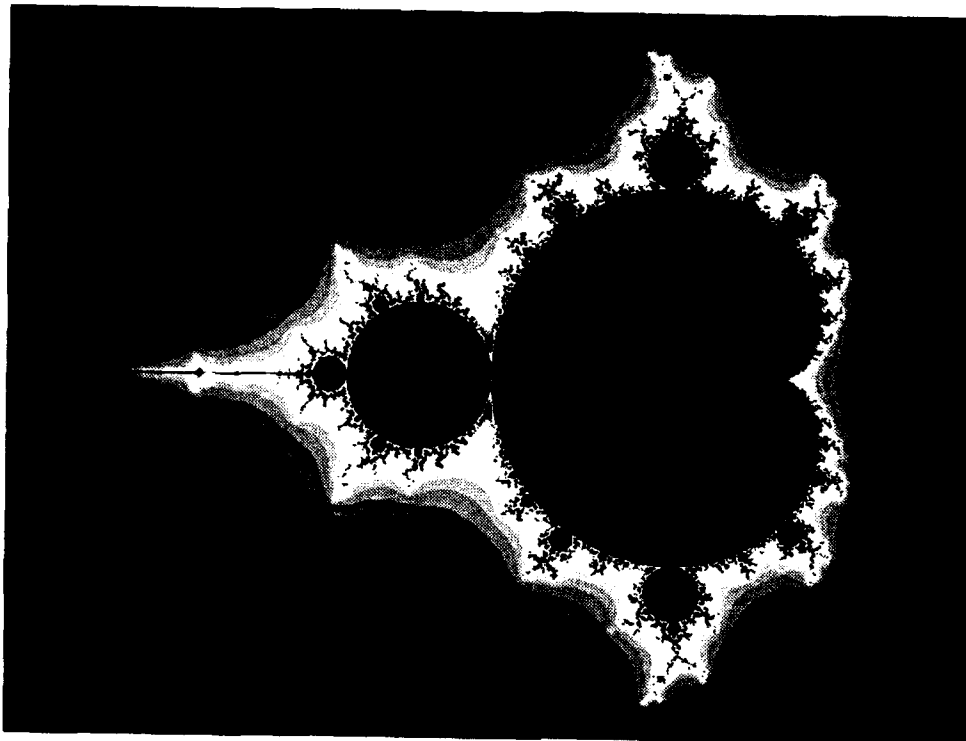


Figure 50. Example 3, original gray scale image

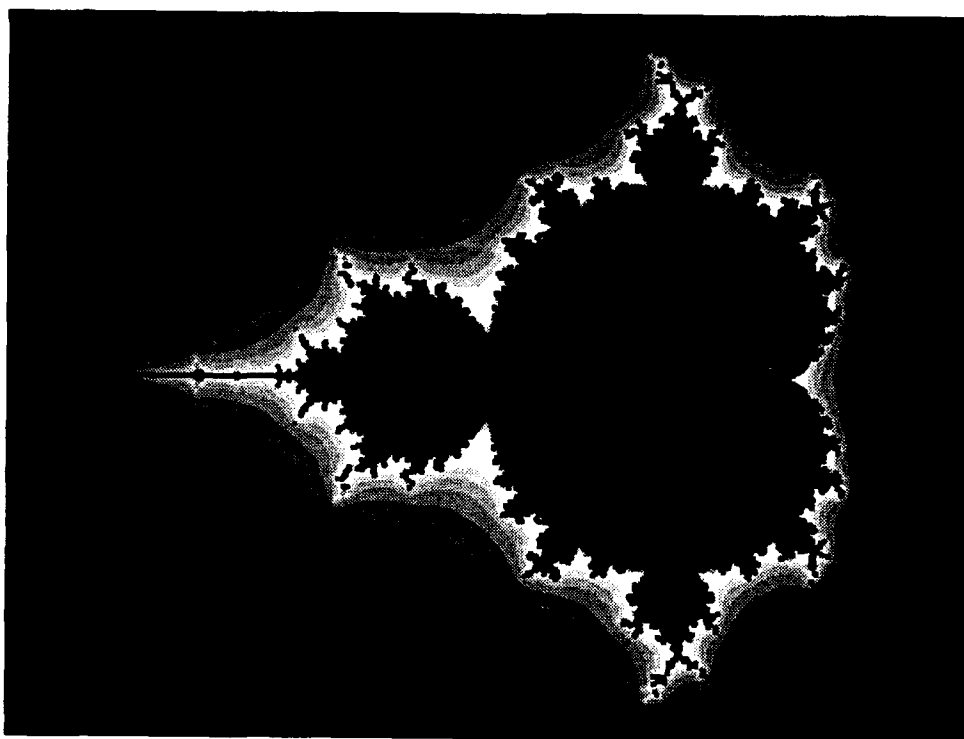


Figure 51. Example 3, dilated gray scale image

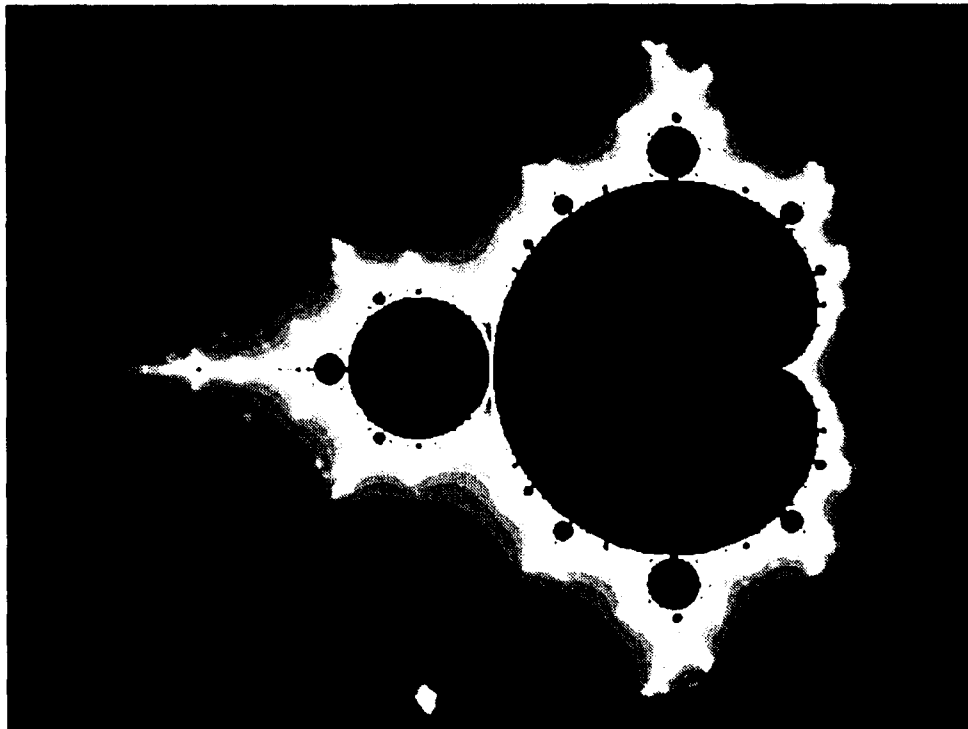


Figure 52. Example 3, eroded gray scale image

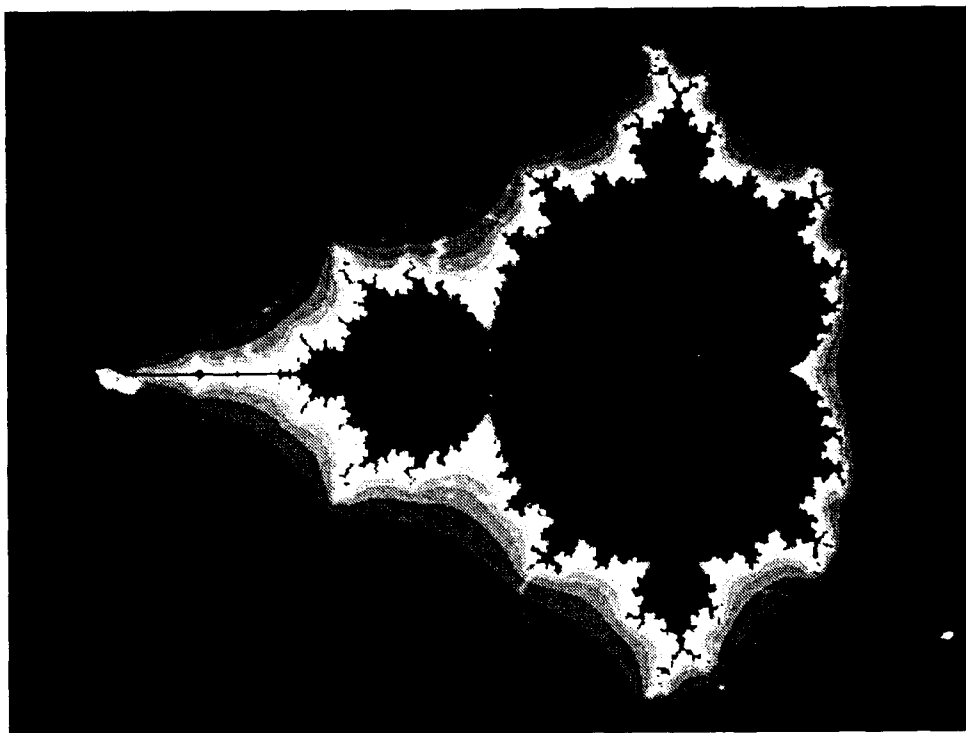


Figure 53. Example 3, closed gray scale image

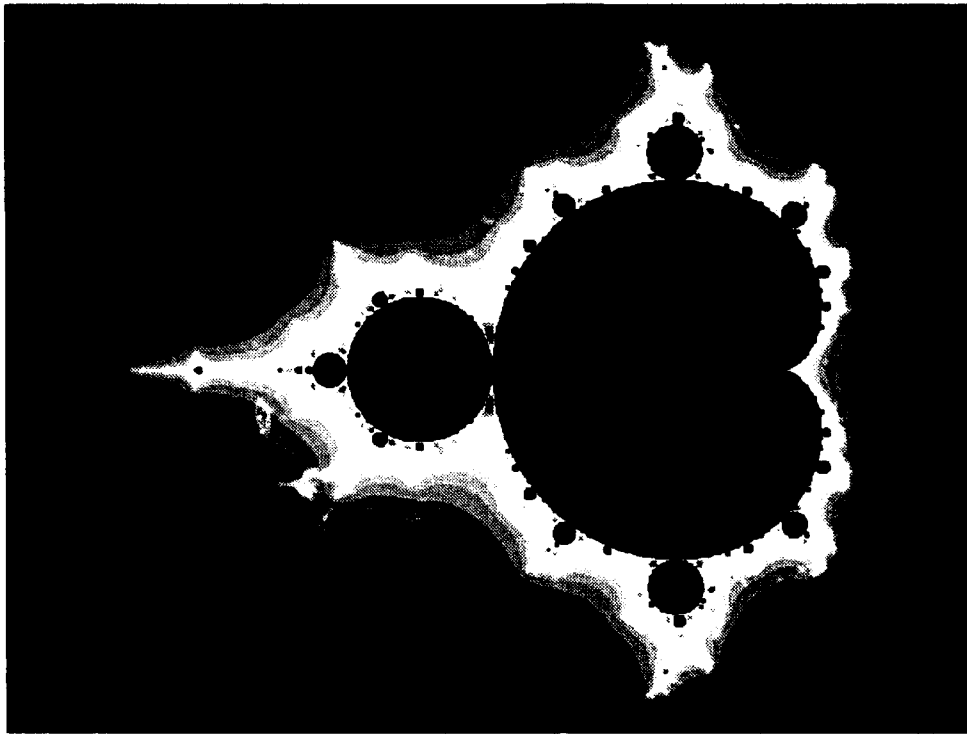


Figure 54. Example 3, opened gray scale image

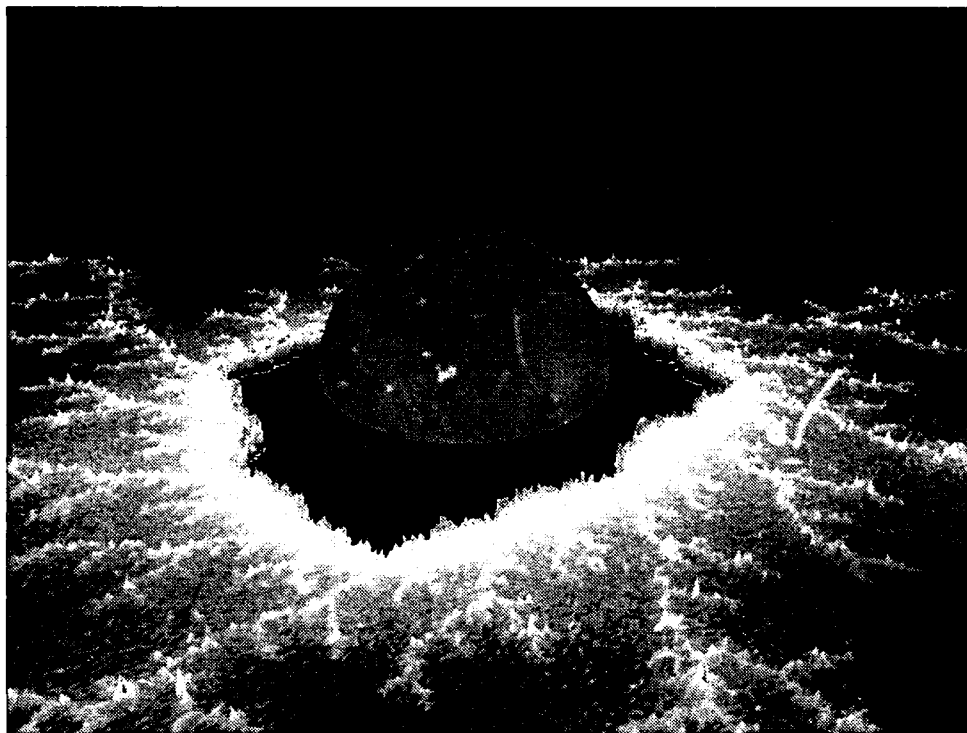


Figure 55. Example 4, original gray scale image



Figure 56. Example 4, dilated gray scale image



Figure 57. Example 4, eroded gray scale image



Figure 58. Example 4, closed gray scale image

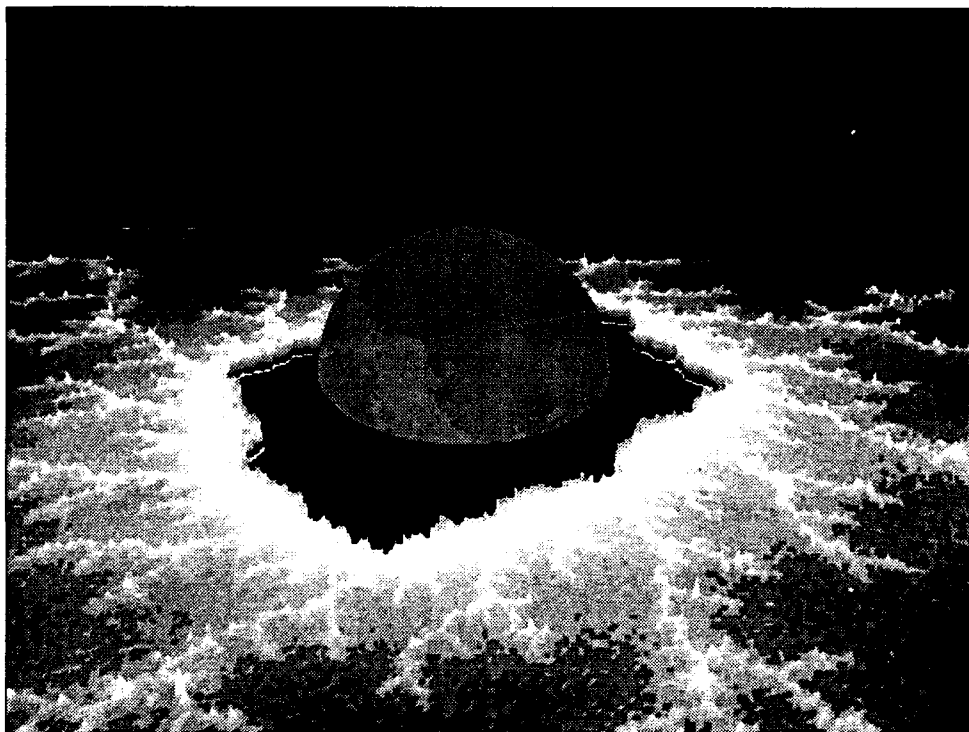


Figure 59. Example 4, opened gray scale image

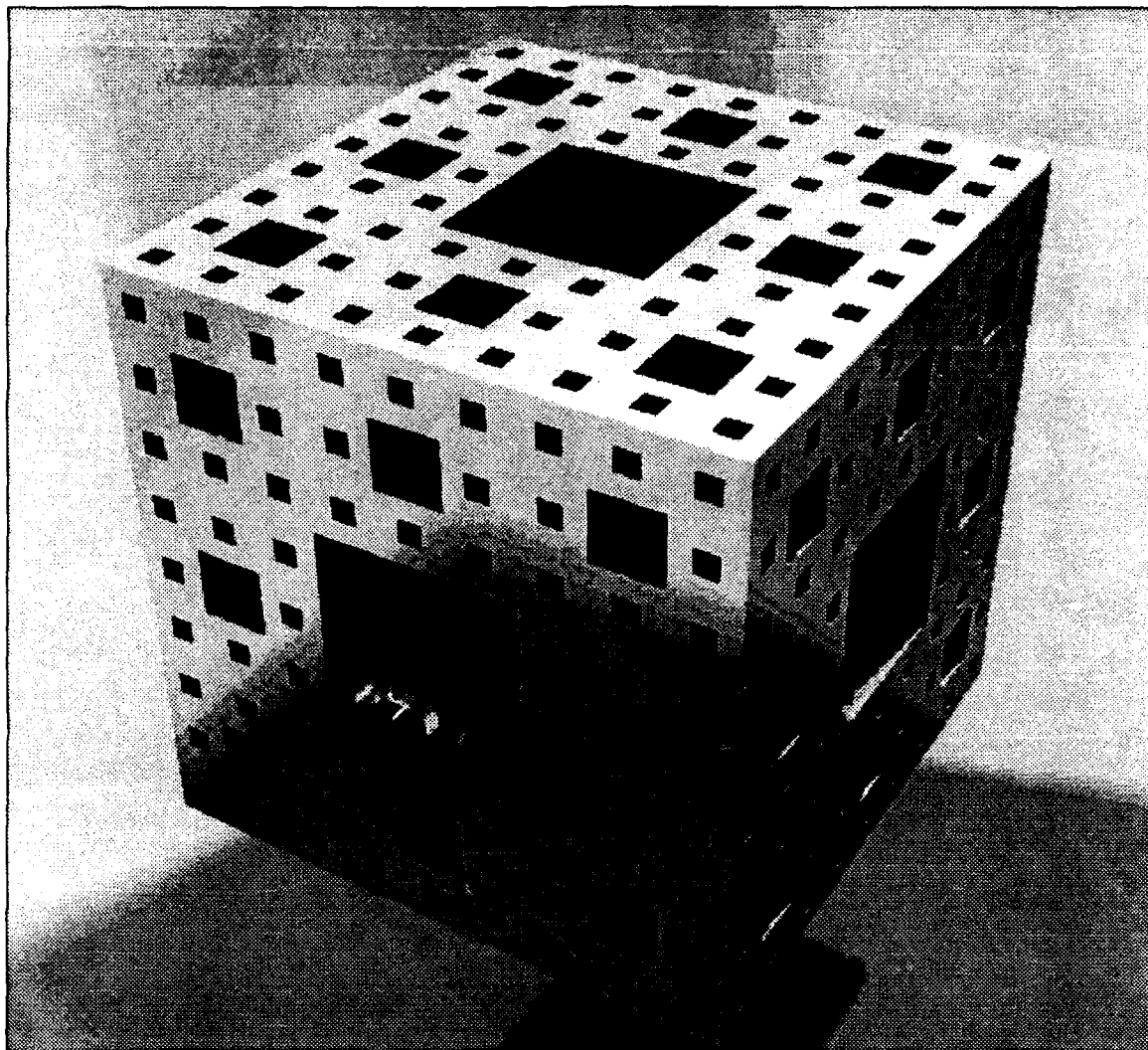


Figure 60. Example 5, original gray scale image

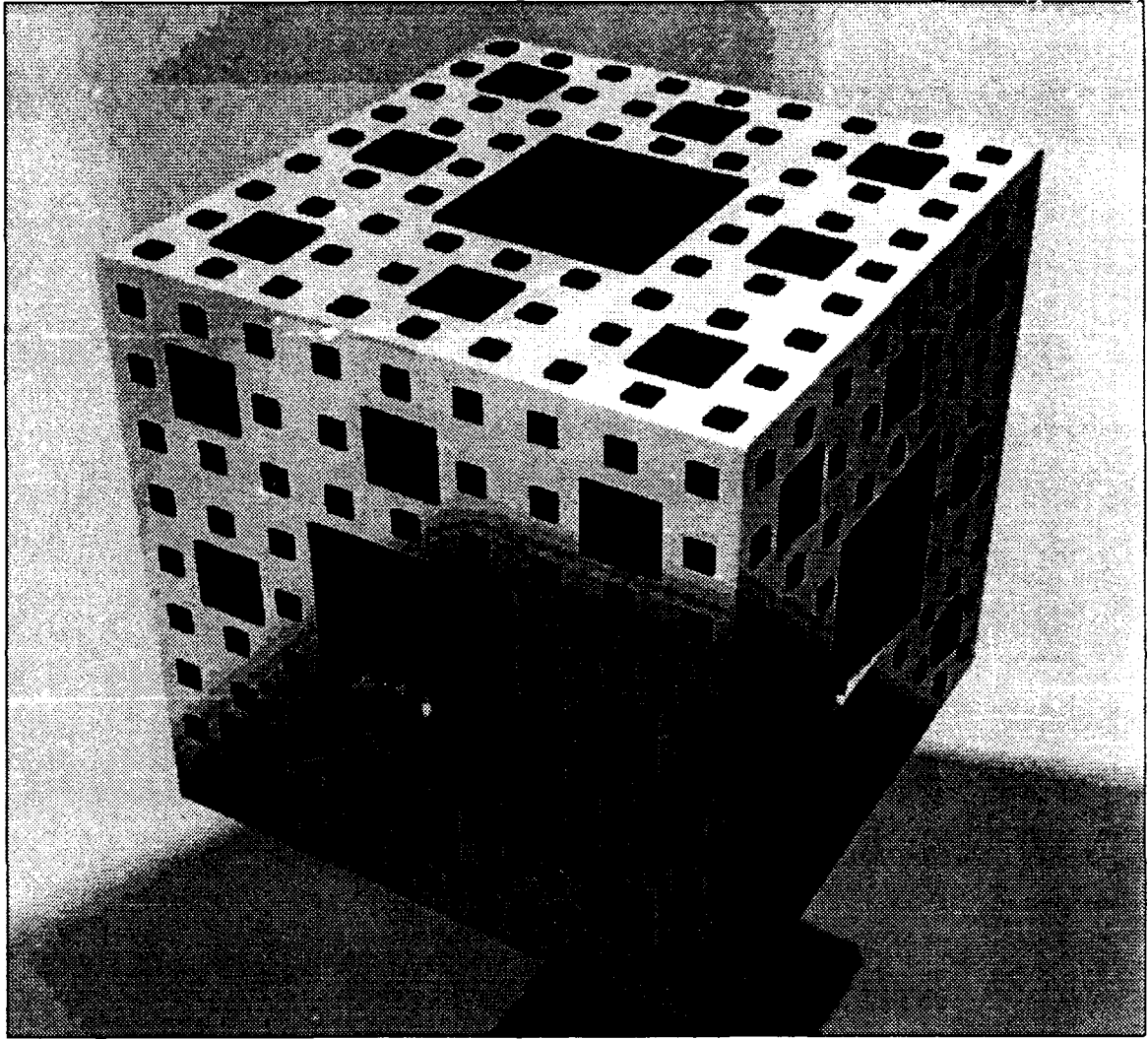


Figure 61. Example 5, dilated gray scale image

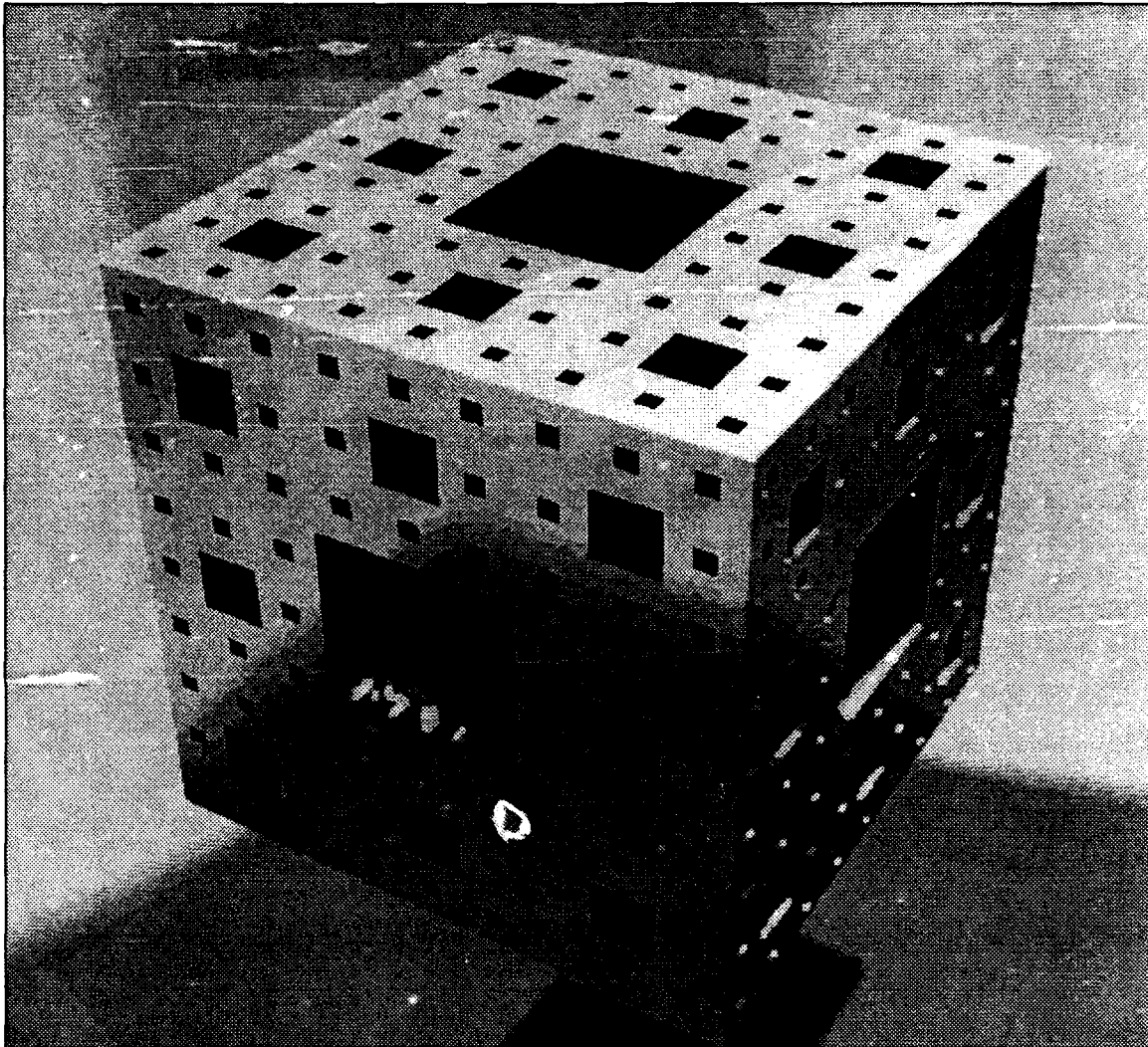


Figure 62. Example 5, eroded gray scale image

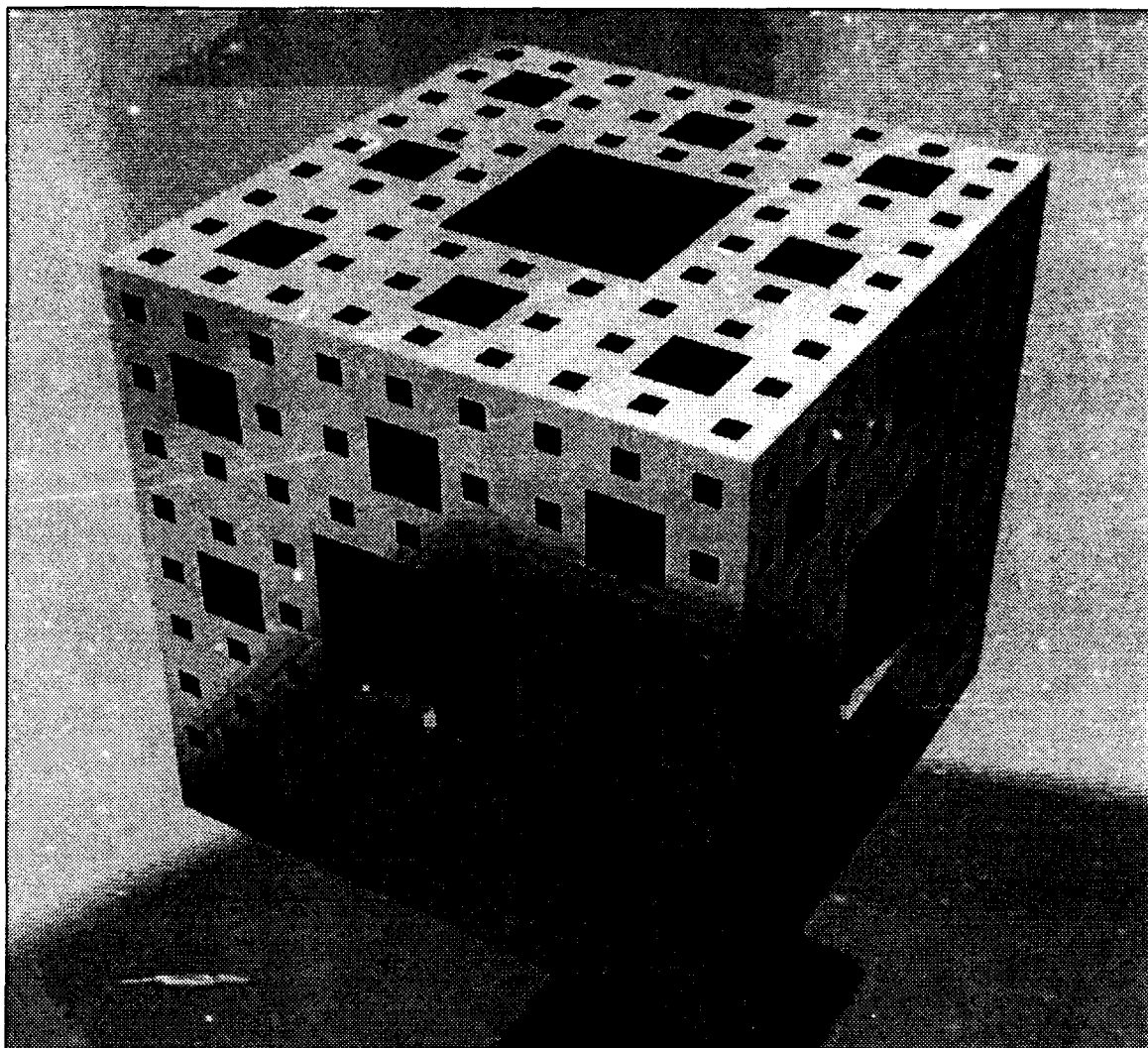


Figure 63. Example 5, closed gray scale image

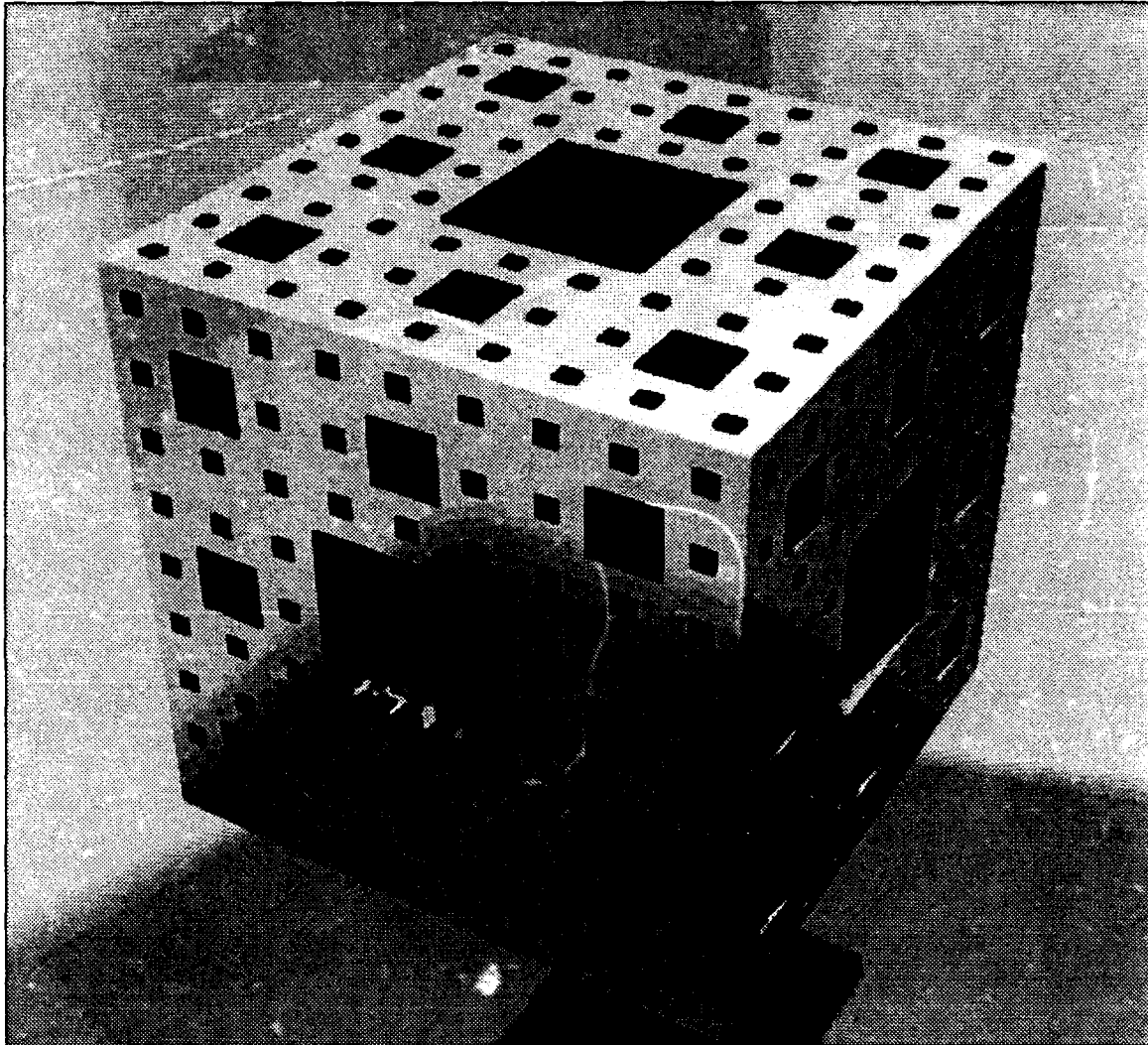


Figure 64. Example 5, opened gray scale image

5 Detecting Edges

Abstract

Edge detection is based on the concept of gradients. This chapter discusses numerous edge detection methods and their implementation.

Introduction

The importance of edge detection is best seen in the digitization of optical images. The real or optical image tends to have more edge distinction (according to luminance) than the sampled image because a low pass filtering must be applied during the digitization process to avert aliasing (Pratt 1991). Therefore, if edges can be detected, other algorithms can be used to strengthen the luminance contrast at these edges.

Edge detection is an indirect product of the continuous space gradient, where rows are in the x direction, columns are in the y direction, and pixel values are represented in the z direction. Due to the spatially discrete nature of images, this process can be performed by a windowing convolution algorithm.

Theory

The following equation describes the continuous gradient of an optical image (Pratt 1991):

$$G(x, y) = \frac{\partial F(x, y)}{\partial x} \cos(\theta) + \frac{\partial F(x, y)}{\partial y} \sin(\theta) \quad (36)$$

where

θ = angle of approach

$F(x, y)$ = continuous 2-D gradient

$G(x, y)$ = continuous gradient

In a discrete environment like a digitized image, the gradient must be described discretely (Pratt 1991):

$$\begin{aligned} G(j, k) &= \sqrt{[G_R(j, k)]^2 + [G_C(j, k)]^2} \\ \Theta(j, k) &= \tan^{-1} \frac{G_C(j, k)}{G_R(j, k)} \end{aligned} \quad (37)$$

where

$G(j, k)$ = discrete gradient

$G_R(j, k)$ = discrete row gradient

$G_C(j, k)$ = discrete column gradient

$\Theta(j, k)$ = spatial orientation of discrete gradient with respect to the row axis

A simple method would be to compare two adjacent pixels according to magnitude (Pratt 1991):

$$\begin{aligned} G_R(j, k) &= F(j, k) - F(j, k+1) \\ G_C(j, k) &= F(j, k) - F(j+1, k) \end{aligned} \quad (38)$$

However, to provide versatility, a convolution method can be used (Pratt 1991):

$$\begin{aligned} G_R(j, k) &= F(j, k) \otimes H_R(j, k) \\ G_C(j, k) &= F(j, k) \otimes H_C(j, k) \end{aligned} \quad (39)$$

where

$\otimes$ = windowed convolution

$H_R(j, k)$ = row impulse response

$H_C(j, k)$ = column impulse response

The row and column impulses responses are chosen by the user; Table 3 is a list of some typical 3X3 impulse responses (Pratt 1991):

Table 3 Typical First-Order 3x3 Impulse Responses						
Name	Row Response			Column Response		
Pixel difference	0	0	0	0	-1	0
	0	1	-1	0	1	0
	0	0	0	0	0	0
Separated difference	0	0	0	0	-1	0
	1	0	-1	0	0	0
	0	0	0	0	1	0
Sobel	0.25	0	-0.25	-0.25	-0.5	-0.25
	0.5	0	-0.5	0	0	0
	0.25	0	-0.25	0.25	0.5	0.25
Roberts	0	0	-1	-1	0	0
	0	1	0	0	1	0
	0	0	0	0	0	0
Prewitt	0.33	0	-0.33	-0.33	-0.33	-0.33
	0.33	0	-0.33	0	0	0
	0.33	0	-0.33	0.33	0.33	0.33
Frei-Chen	0.293	0	-0.293	-0.293	-0.414	-0.293
	0.414	0	-0.414	0	0	0
	0.293	0	-0.293	0.293	0.414	0.293

The windowed 3x3 convolution is performed as follows:

$$\begin{aligned}
 G(j, k) = & H(0, 0) * F(j-1, k-1) + H(0, 1) * F(j-1, k) \\
 & + H(0, 2) * F(j-1, k+1) + H(1, 0) * F(j, k-1) \\
 & + H(1, 1) * F(j, k) + H(1, 2) * F(j, k+1) \\
 & + H(2, 0) * F(j+1, k-1) + H(2, 1) * F(j+1, k) \\
 & + H(2, 2) * F(j+1, k+1)
 \end{aligned} \tag{40}$$

For pixels located at the border of the image, reflection is used to replace missing pixels that are needed to process the windowed 3x3 convolution. Table 4 contains a description of a typical 5x5 impulse response (Pratt 1991):

Table 4 Typical First-Order 5x5 Impulse Response										
Name	Row Response					Column Response				
Nevatia-Babu	0.1	0.1	0	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
	0.1	0.1	0	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
	0.1	0.1	0	-0.1	-0.1	0	0	0	0	0
	0.1	0.1	0	-0.1	-0.1	0.1	0.1	0.1	0.1	0.1
	0.1	0.1	0	-0.1	-0.1	0.1	0.1	0.1	0.1	0.1

The windowed 5x5 convolution is an extension of Equation 40.

The following equation is the basis for second-order edge detection (Pratt 1991):

$$G(x, y) = -\nabla^2 [F(x, y)] = \frac{-\partial^2 F(x, y)}{\partial x^2} + \frac{-\partial^2 F(x, y)}{\partial y^2} \quad (41)$$

Locations where the second derivative of the image function is non-zero are considered to be members of an edge (Pratt 1991). The following is a simple second-order method (Pratt 1991):

$$G(j, k) = [F(j, k) - F(j, k-1)] - [F(j, k+1) - F(j, k)] + [F(j, k) - F(j+1, k)] - [F(j-1, k) - F(j, k)] \quad (42)$$

The convolution method is described as follows (Pratt 1991):

$$G(j, k) = F(j, k) \otimes H(j, k) \quad (43)$$

Table 5
Typical Second-Order 3x3 Impulse Responses

Name	Impulse Response		
Laplacian 4 Neighbor	0 -0.25 0	-0.25 1 -0.25	0 -0.25 0
Prewitt 8 Neighbor	-0.125 -0.125 -0.125	-0.125 1 -0.125	-0.125 -0.125 -0.125
Separable 8 Neighbor	-0.25 0.125 -0.25	0.125 0.5 0.125	-0.25 0.125 -0.25

Table 5 lists some 3x3 impulse responses for use in second-order edge detection. An impulse response is considered to be separable when it can be broken into a row second derivative and a column second derivative, as shown (Pratt 1991):

$$H(j, k) = \begin{bmatrix} -0.25 & 0.125 & -0.25 \\ 0.125 & 0.5 & 0.125 \\ -0.25 & 0.125 & -0.25 \end{bmatrix} = \frac{1}{8} * \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} + \frac{1}{8} * \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad (44)$$

Implementation

The code in Appendix M performs the windowed convolution with any odd order square impulse response. A "bias_flag" of 1 is needed to enable representation for both positive and negative gradients. Positive gradients appear bright (with values above 128); negative gradients appear dark (with values below 128). All computation is done in conventional memory. Internal and external files are binary or standard RGB. A unique feature of this program is that as the windowed convolution is performed, the screen is updated.

Results

All first- and second-order impulse responses were applied in a windowed convolution, as shown in Figures 66-82. The original is shown in Figure 65.

The pixel difference impulse responses are weaker than the other impulse responses, due to the pixel difference's lack of numerical sophistication. The separated difference impulse responses are powerful and simple. The Sobel impulse responses are comparable to the separated difference in strength; however, the Sobel impulse responses are unnecessarily more complex. The Roberts impulse responses are keen in extracting finer details, as can be seen in Figures 72 and 73; the individual stones can easily be located. Also, the Roberts impulse responses can detect both horizontal and vertical edges by using either the row or column impulse response, unlike the previous types which typically detect vertical edges with a row response and horizontal edges with a column response. The Prewitt set is much like the Sobel set, the only difference being magnitude. For its complexity, the Frei-Chen set has only average edge detection capabilities. The Nevatia-Babu gives low resolution results.

Little difference exists between the results of the windowed convolutions of different second-order impulse responses. Although the graphical representations in Figures 80-82 seem to indicate weak edge detection, they actually prove that second-order impulse responses are superior to first-order responses. *Careful examination is required because the images are represented in halftones.*

Conclusions

The Roberts set of impulse responses has proved to be the most desirable first-order edge-detection tool because of its simplicity, its high resolution results, and its ability to use only one of its impulse responses to extract the majority of the edges. The second-order set of impulse responses is deemed superior to the first-order responses because of the second order's high resolution edge extraction.



Figure 65. Original image used for edge detection example

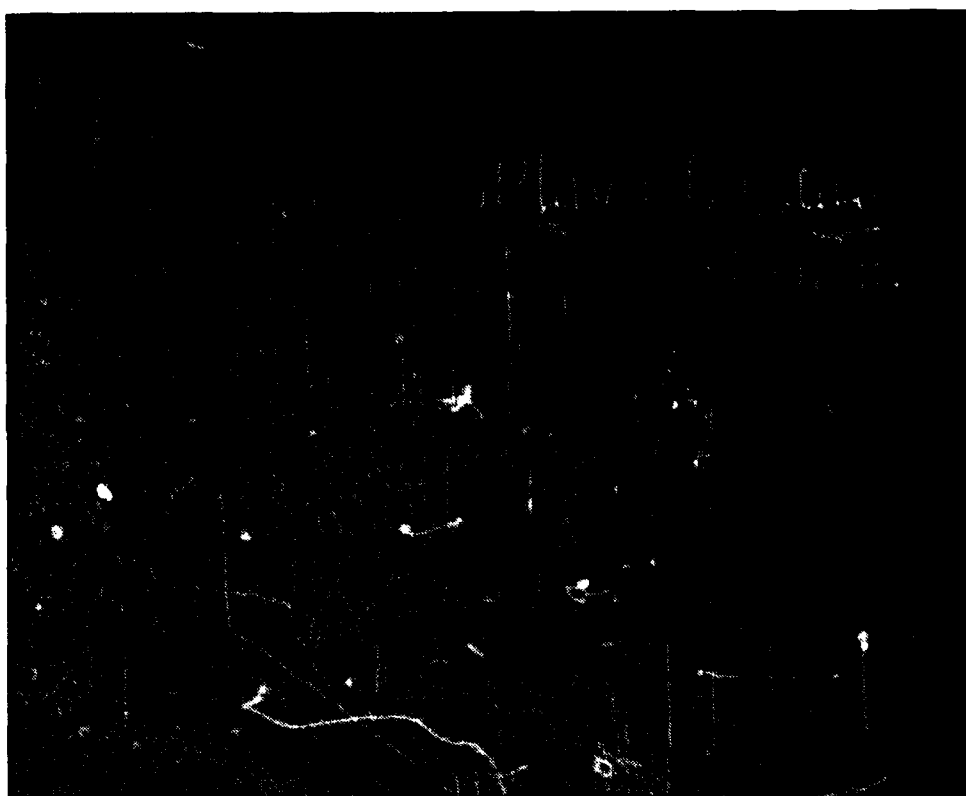


Figure 66. Result of pixel difference row windowed convolution



Figure 67. Result of pixel difference column windowed convolution

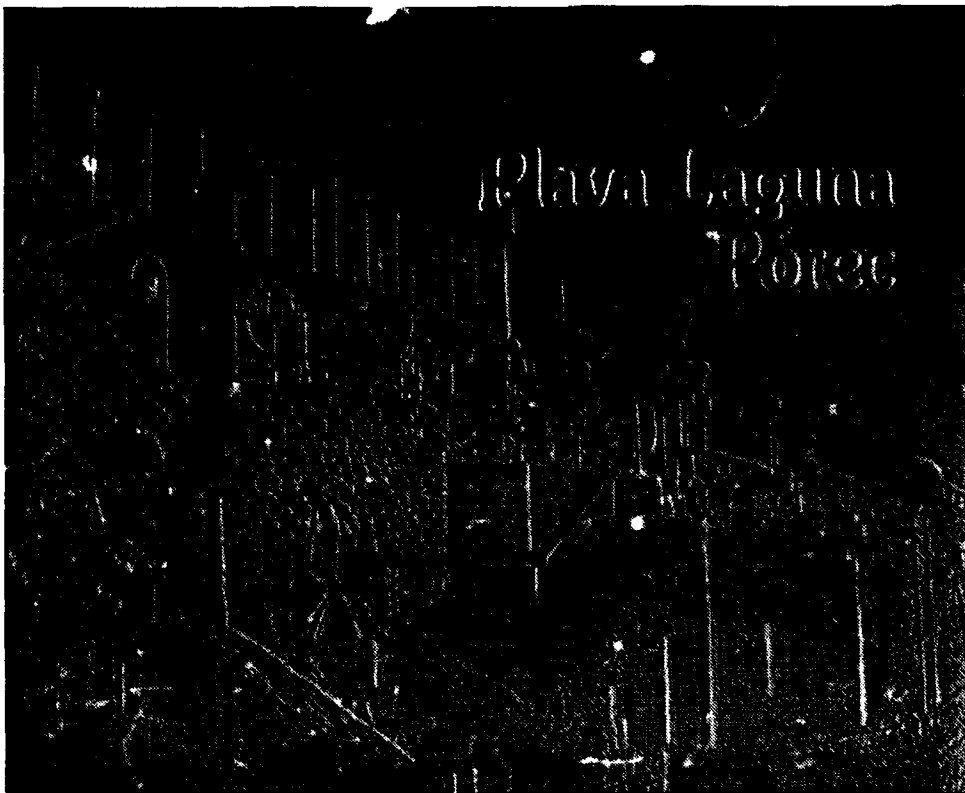


Figure 68. Result of separated difference row windowed convolution



Figure 69. Result of separated difference column windowed convolution

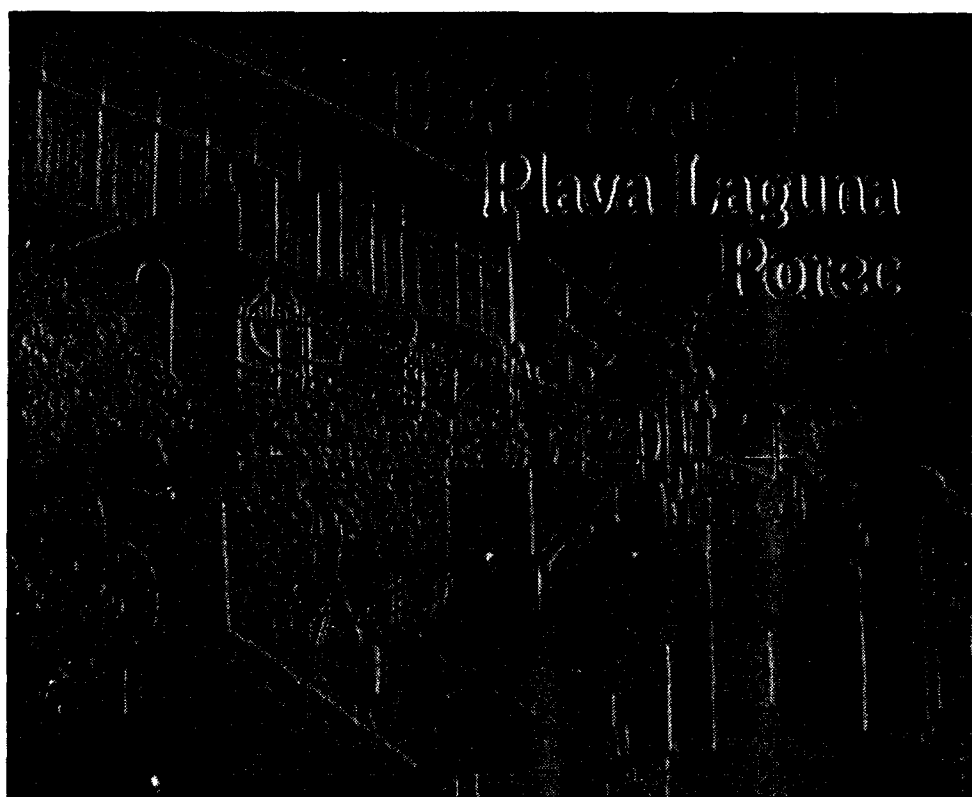


Figure 70. Result of Sobel row windowed convolution

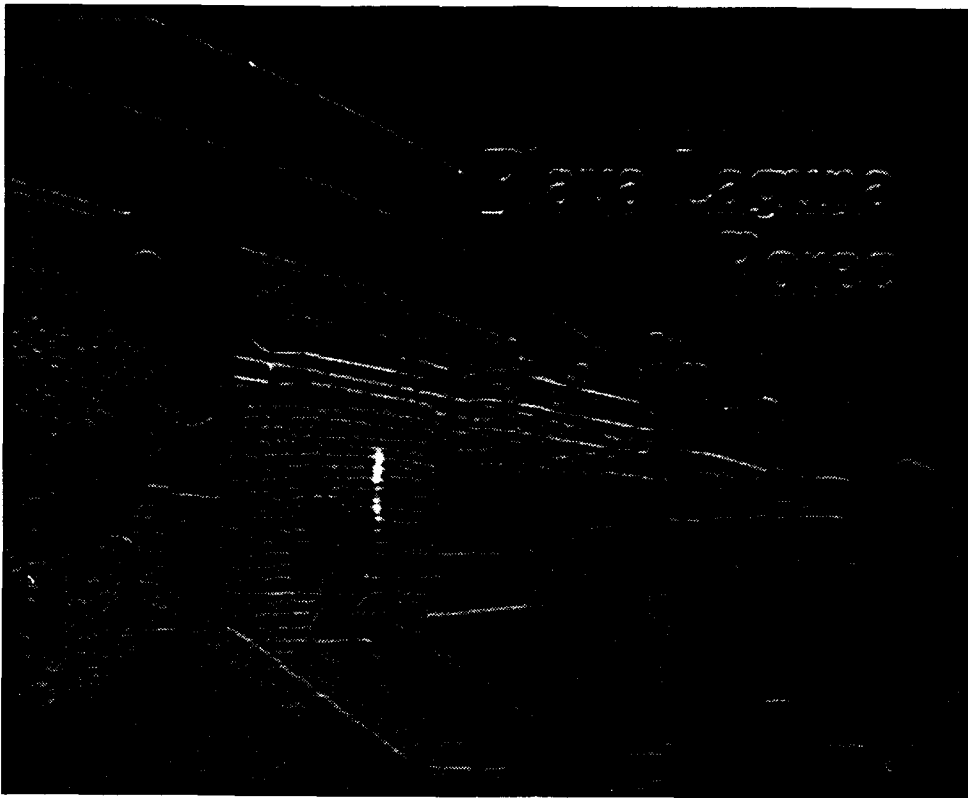


Figure 71. Result of Sobel column windowed convolution

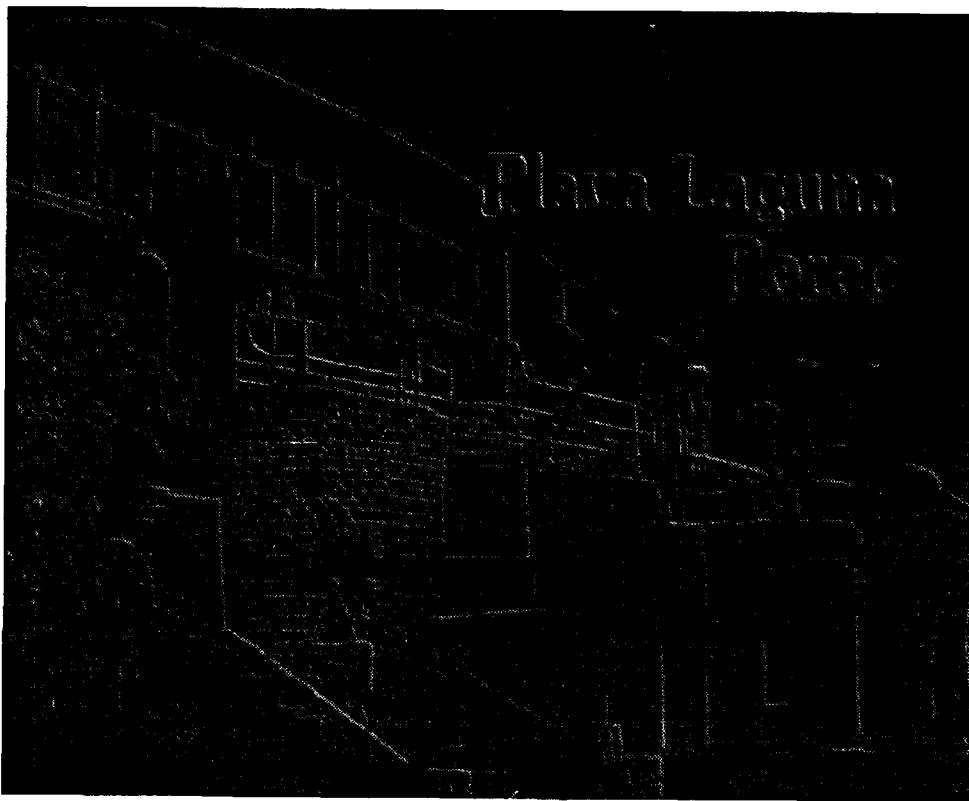


Figure 72. Result of Roberts row windowed convolution

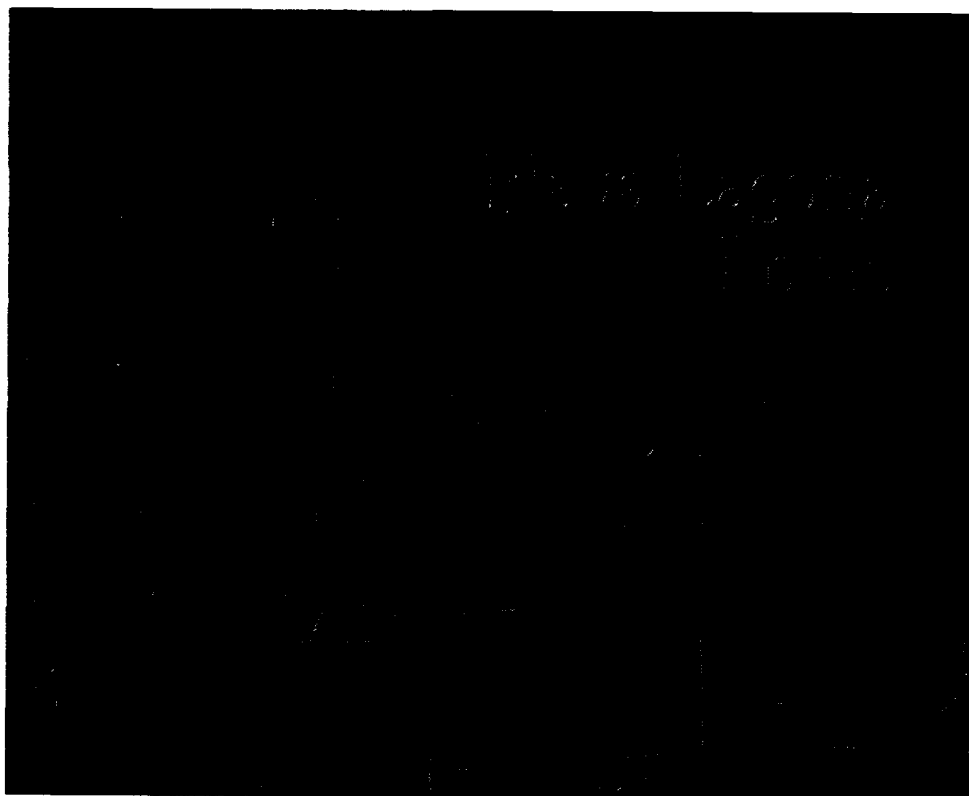


Figure 73. Result of Roberts column windowed convolution

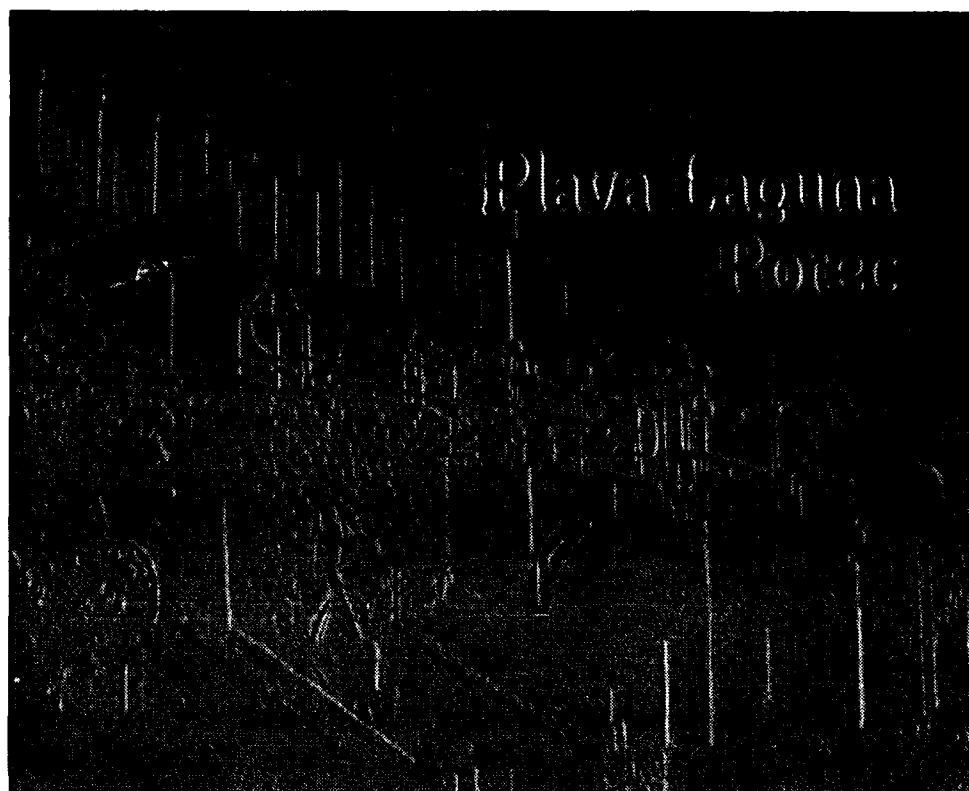


Figure 74. Result of Prewitt row windowed convolution

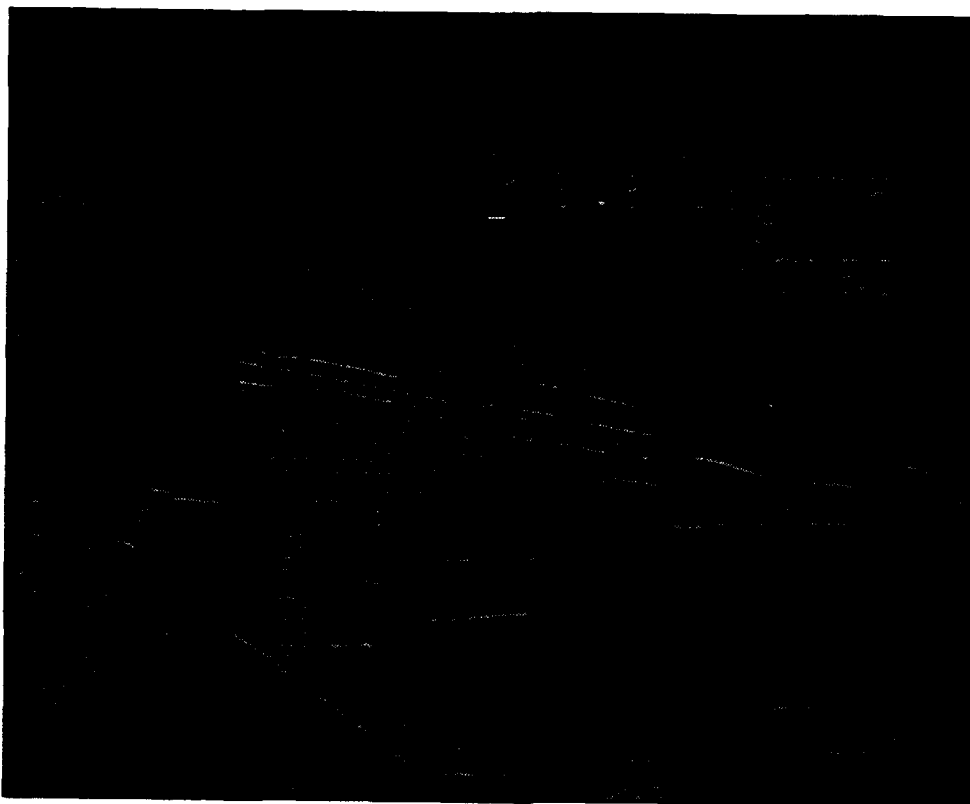


Figure 75. Result of Prewitt column windowed convolution

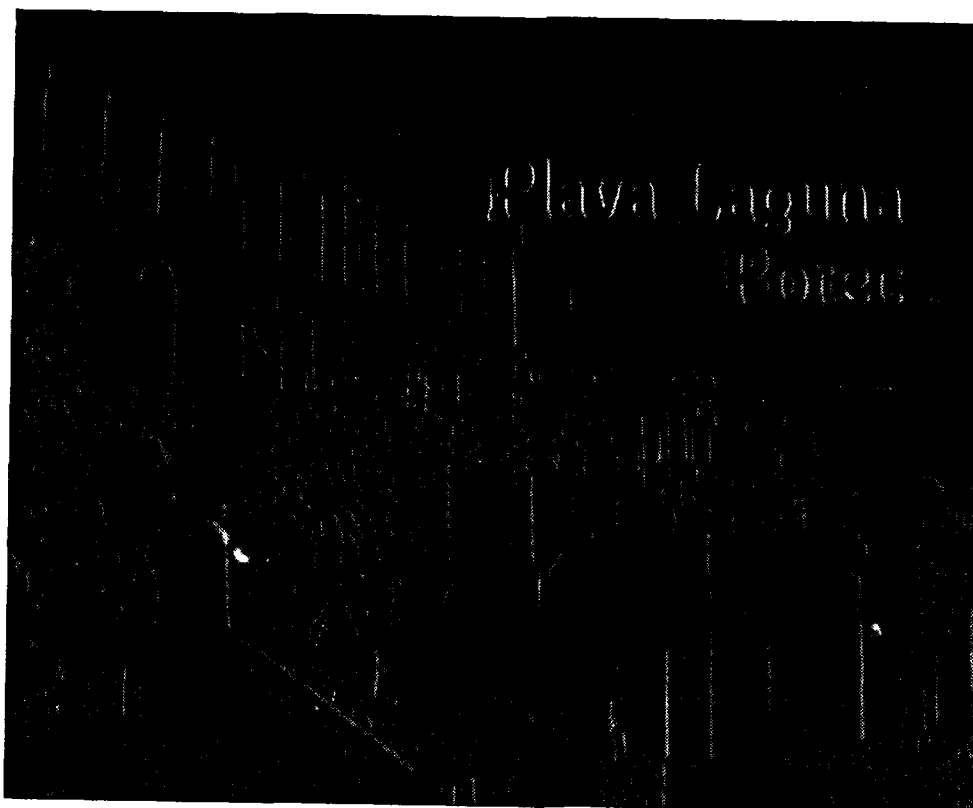


Figure 76. Result of Frei-Chen row windowed convolution

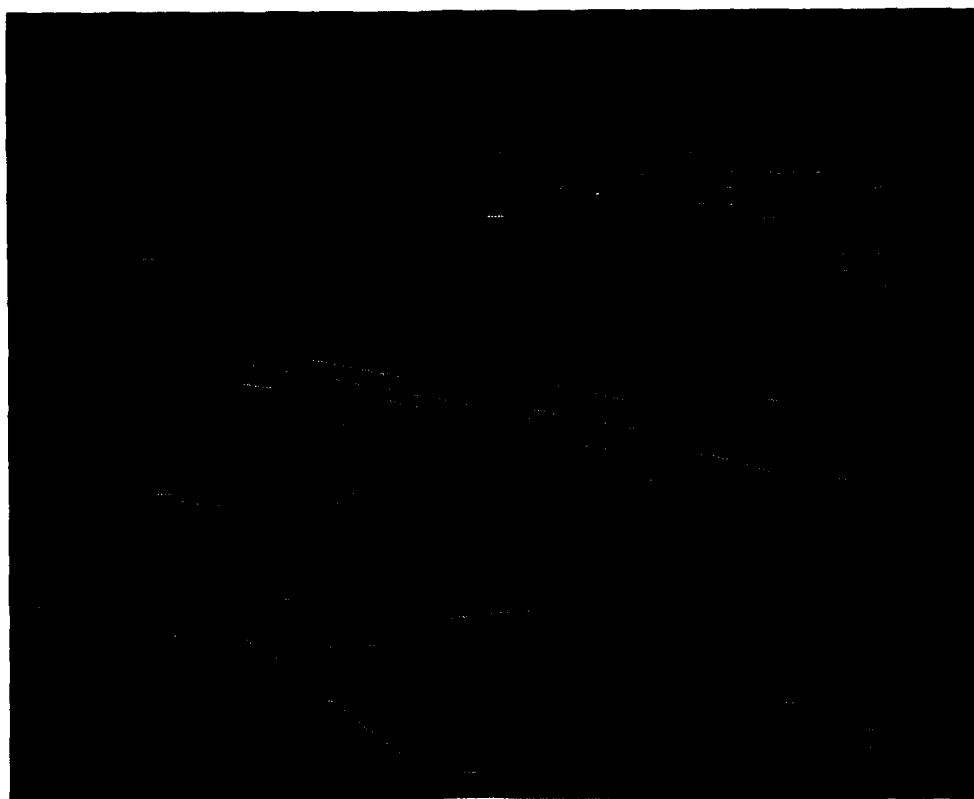


Figure 77. Result of Frei-Chen column windowed convolution

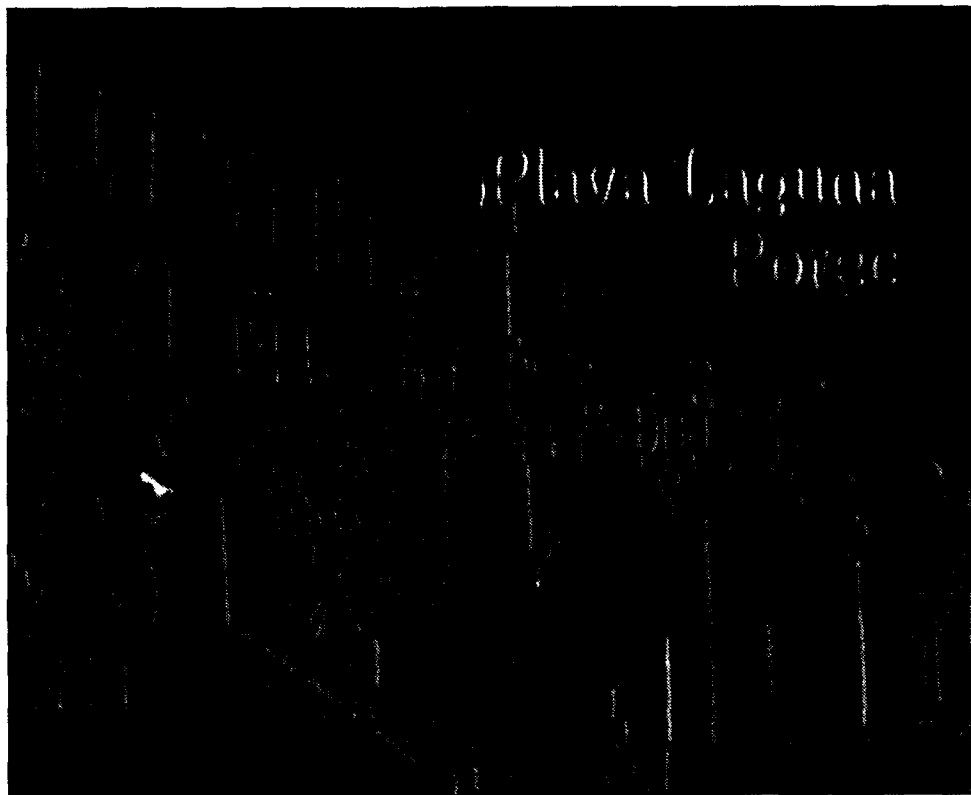


Figure 78. Result of Nevatia-Babu row windowed convolution

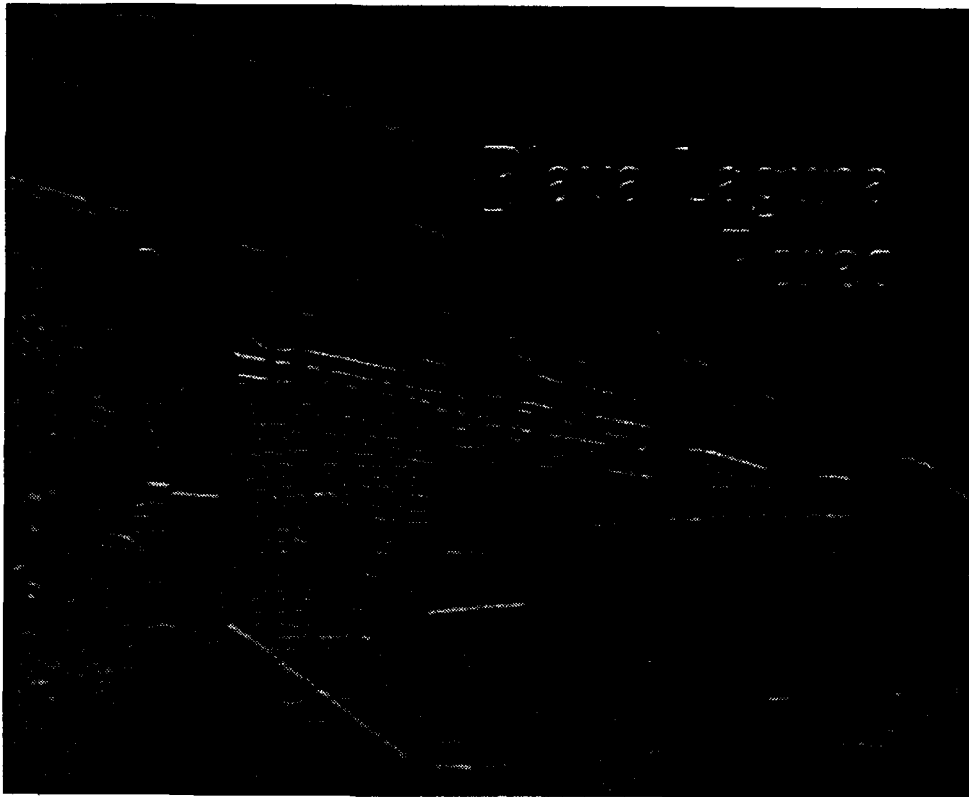


Figure 79. Result of Nevatia-Babu column windowed convolution

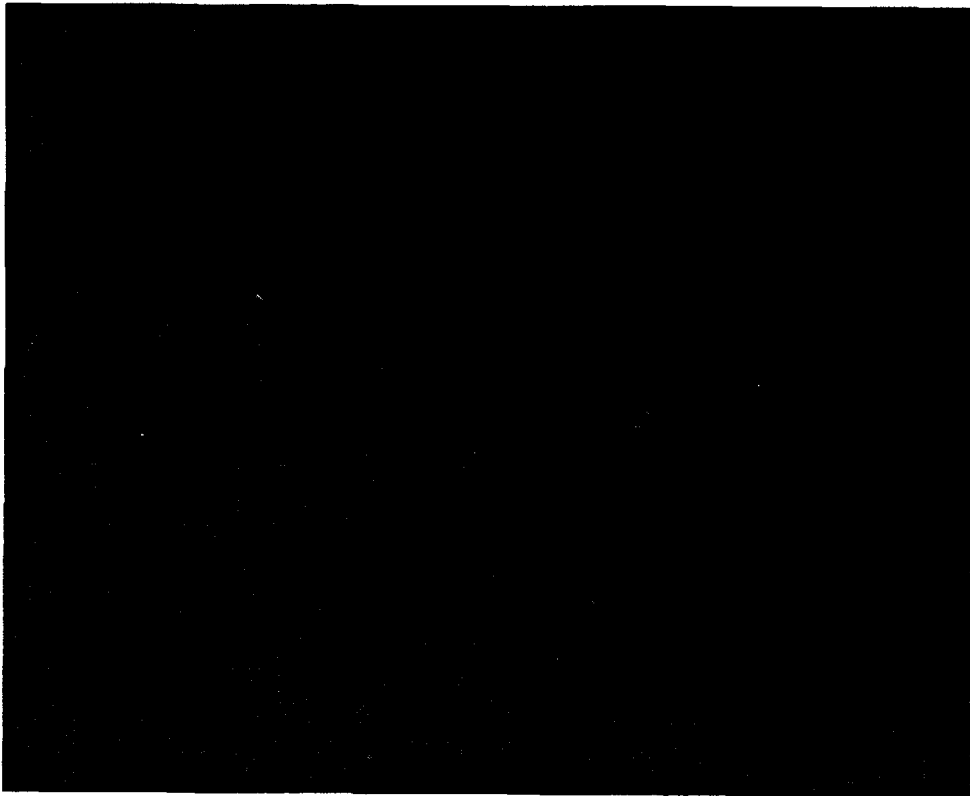


Figure 80. Result of Laplace four neighbor windowed convolution



Figure 81. Result of Prewitt eight neighbor windowed convolution



Figure 82. Result of separable eight neighbor windowed convolution

6 Crispening Images

Abstract

Image “crispening” is a subjective enhancement technique. This chapter discusses three crispening masks and their implementation.

Introduction

Crispening is useful in sharpening images that are slightly blurred. The implementation relies on a windowed convolution, as discussed in Chapter 5.

Theory

Table 6 contains three crispening masks (Pratt 1991):

Table 6 Crispening 3x3 Impulse Responses									
Mask 1	0	-1	0	Mask 2	-1	-1	-1	Mask 3	1
	-1	5	-1		-1	9	-1		-2
	0	-1	0		-1	-1	-1		1

Again, the windowed 3x3 convolution is performed as follows:

$$\begin{aligned} G(j, k) = & H(0, 0) * F(j - 1, k - 1) + H(0, 1) * F(j - 1, k) \\ & + H(0, 2) * F(j - 1, k + 1) + H(1, 0) * F(j, k - 1) \\ & + H(1, 1) * F(j, k) + H(1, 2) * F(j, k + 1) \\ & + H(2, 0) * F(j + 1, k - 1) + H(2, 1) * F(j + 1, k) \\ & + H(2, 2) * F(j + 1, k + 1) \end{aligned} \quad (45)$$

Reflection is used to replace missing pixels at image borders.

Implementation

The code in Appendix M performs the windowed convolution with any odd order square impulse response. A "bias_flag" of 0 is needed to disable histogram biasing. All computation is done in conventional memory. Internal and external files are binary or standard RGB. As the windowed convolution is performed, the screen is updated.

Results

Figure 83 contains the original image. Figures 84-86 contain the results of applying masks 1-3, respectively. The second mask proved to be the strongest, and the third mask proved to be the weakest.

Conclusions

Image crispening is purely a subjective adjustment. Limited blur removal can be performed with the following masks, ordered according to strength: mask 2, mask 1, and mask 3.



Figure 83. Example 1, original image before mask application

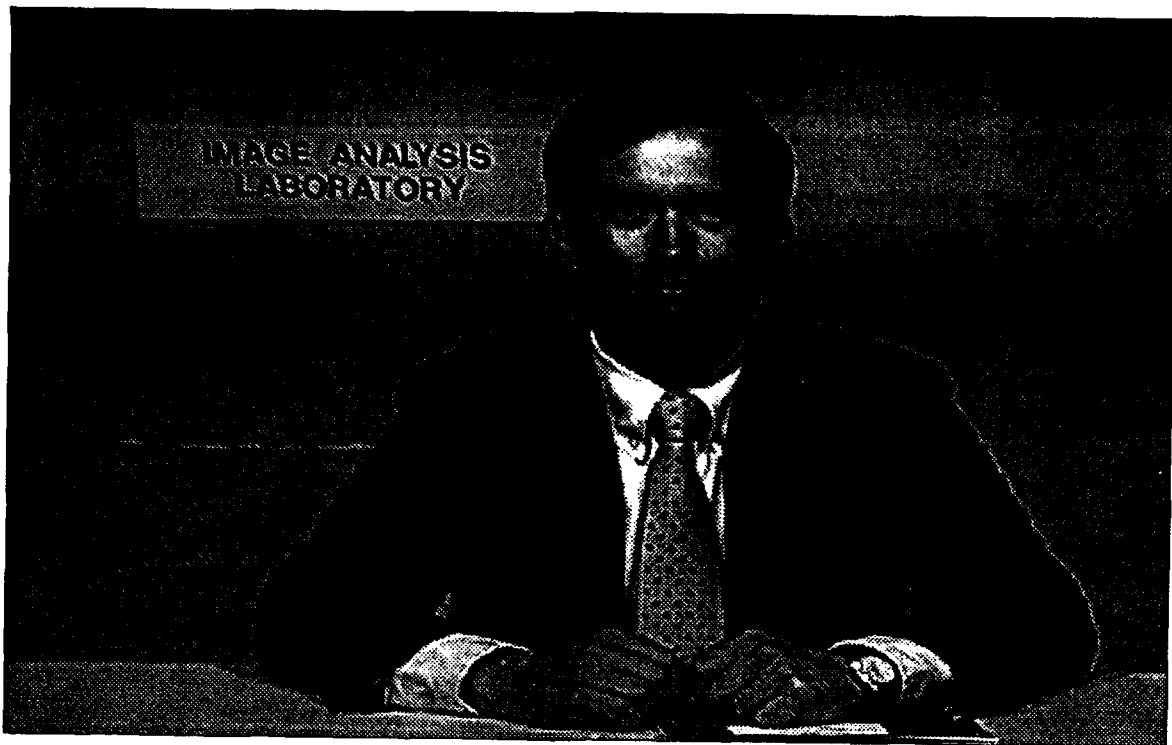


Figure 84. Example 1, result of mask 1 application

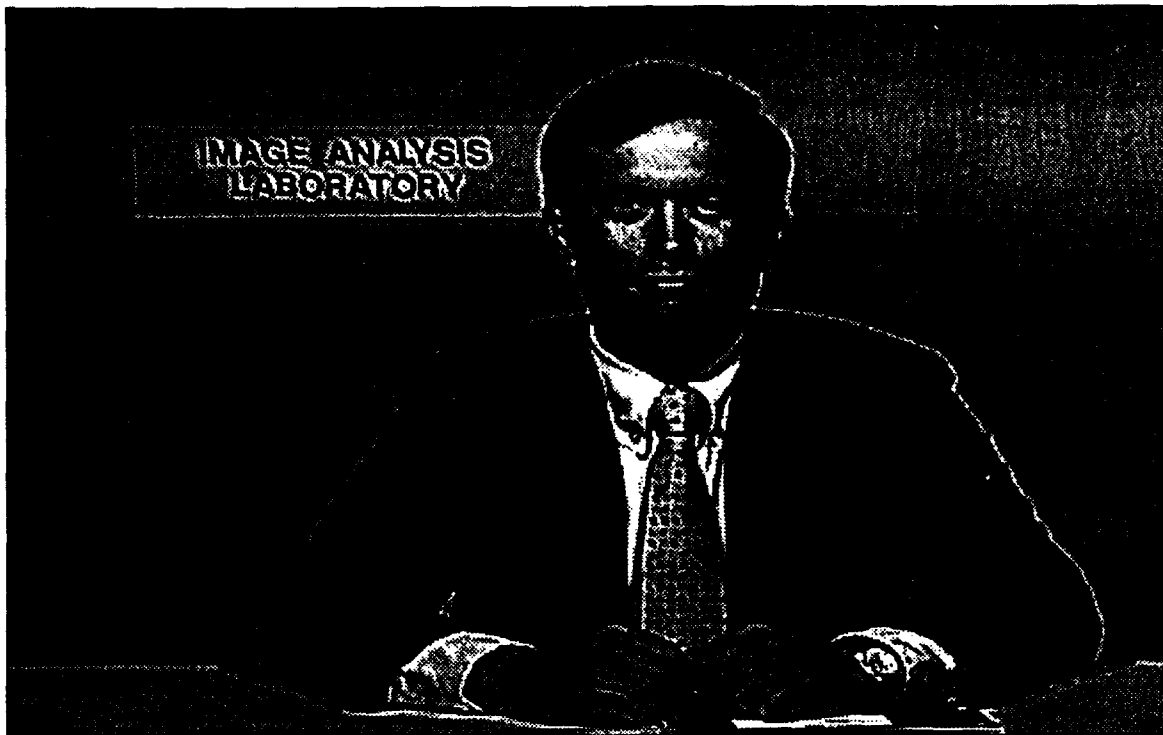


Figure 85. Example 1, result of mask 2 application



Figure 86. Example 1, result of mask 3 application

7 Noise Cleaning

Abstract

Noise cleaning is a simple method used to isolate and remove noise from an image. This chapter discusses three noise-cleaning masks and their implementation.

Introduction

Noise commonly consists of high 2-D frequency components because it is not correlated with the image it is preying upon. The noise cleaning takes advantage of this fact by means of a windowed convolution.

Theory

Table 7 shows three noise-cleaning masks (Pratt 1991):

Table 7 Noise Cleaning 3x3 Impulse Responses											
Mask 1	1/9	1/9	1/9	Mask 2	1/10	1/10	1/10	Mask 3	1/16	1/8	1/16
	1/9	1/9	1/9		1/10	1/5	1/10		1/8	1/4	1/8
	1/9	1/9	1/9		1/10	1/10	1/10		1/16	1/8	1/16

Again, the windowed 3x3 convolution is performed as follows:

$$\begin{aligned} G(j, k) = & H(0, 0) * F(j - 1, k - 1) + H(0, 1) * F(j - 1, k) \\ & + H(0, 2) * F(j - 1, k + 1) + H(1, 0) * F(j, k - 1) \\ & + H(1, 1) * F(j, k) + H(1, 2) * F(j, k + 1) \\ & + H(2, 0) * F(j + 1, k - 1) + H(2, 1) * F(j + 1, k) \\ & + H(2, 2) * F(j + 1, k + 1) \end{aligned} \quad (46)$$

Reflection is used to replace missing pixels at image borders.

Implementation

The code in Appendix M performs the windowed convolution with any odd order square impulse response. A "bias_flag" of 0 is needed to disable histogram biasing. All computation is done in conventional memory. Internal and external files are binary or standard RGB. As the windowed convolution is performed, the screen is updated.

Results

Figure 87 contains the original image. Figures 88-90 contain the results of applying masks 1-3, respectively. All three masks seem to have approximately the same strength.

Conclusions

In some cases, noise cleaning can be used to remove noise from an image. When applicable, this method is more desirable than a Fourier routine because of its simplicity.

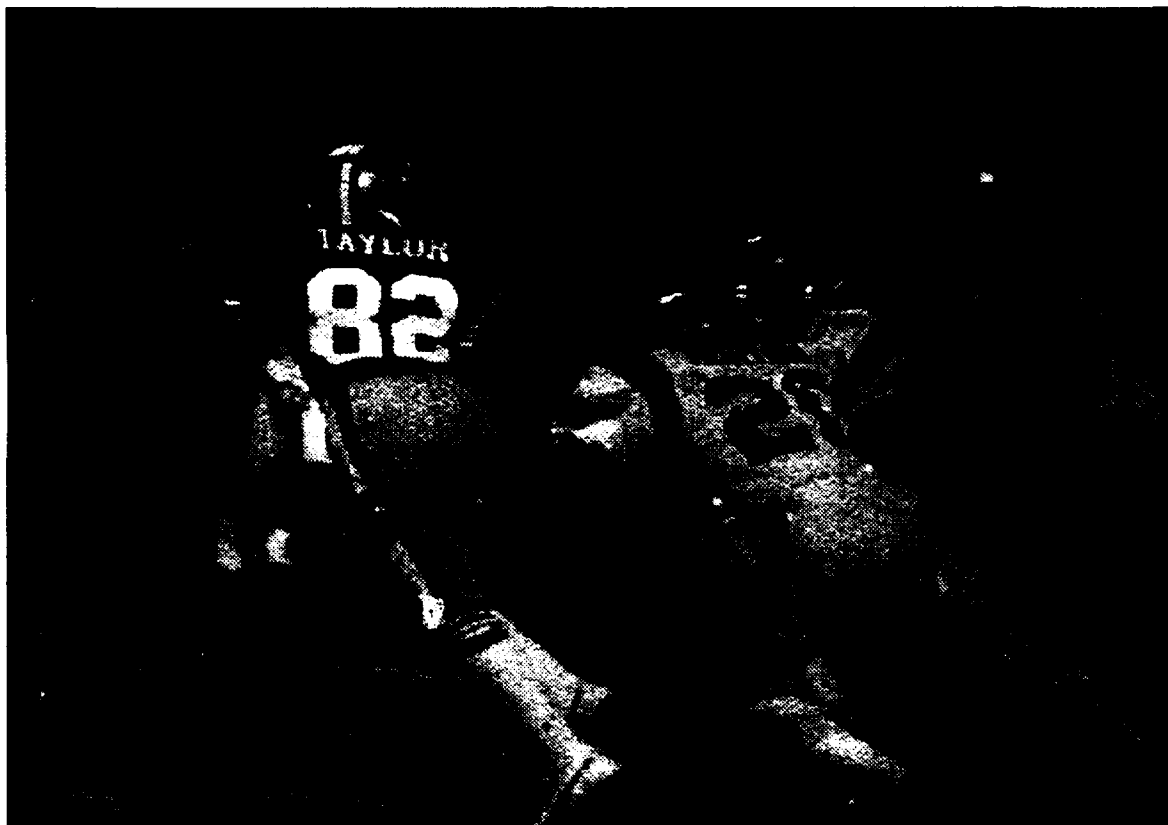


Figure 87. Example 2, original image before mask application

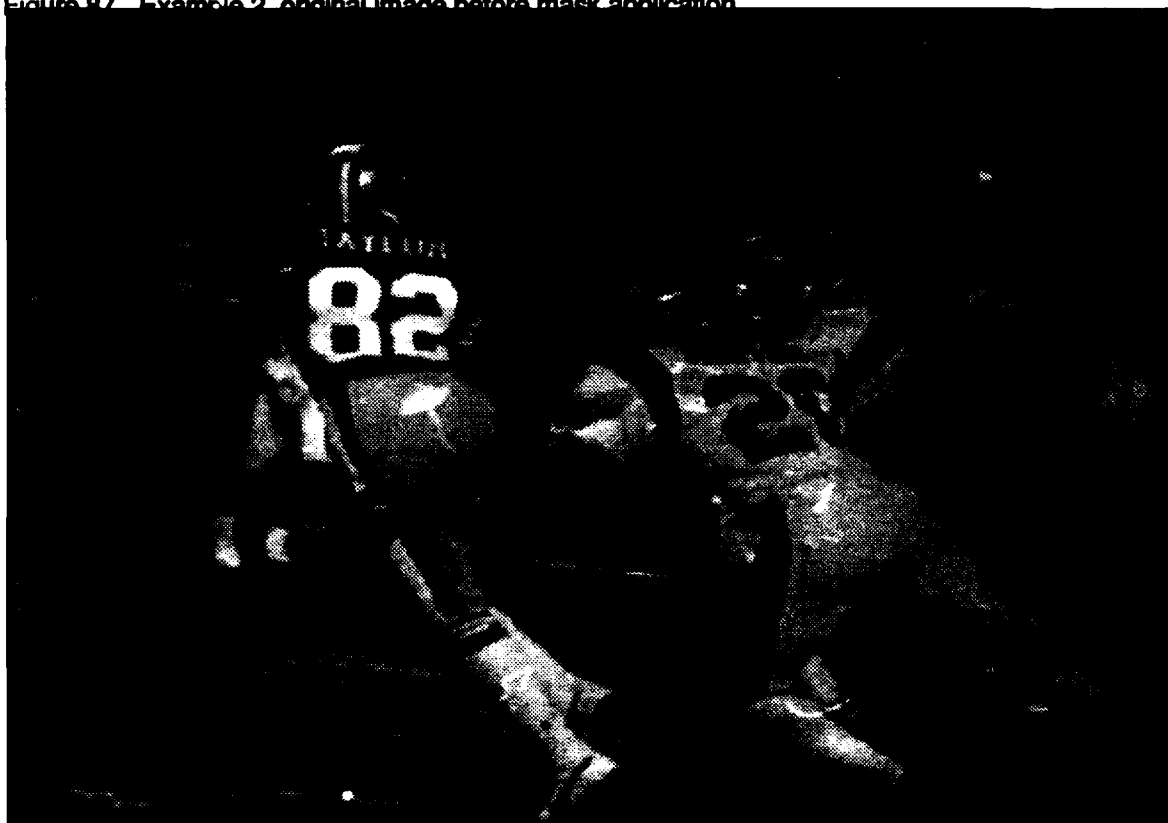


Figure 88. Example 2, result of mask 1 application

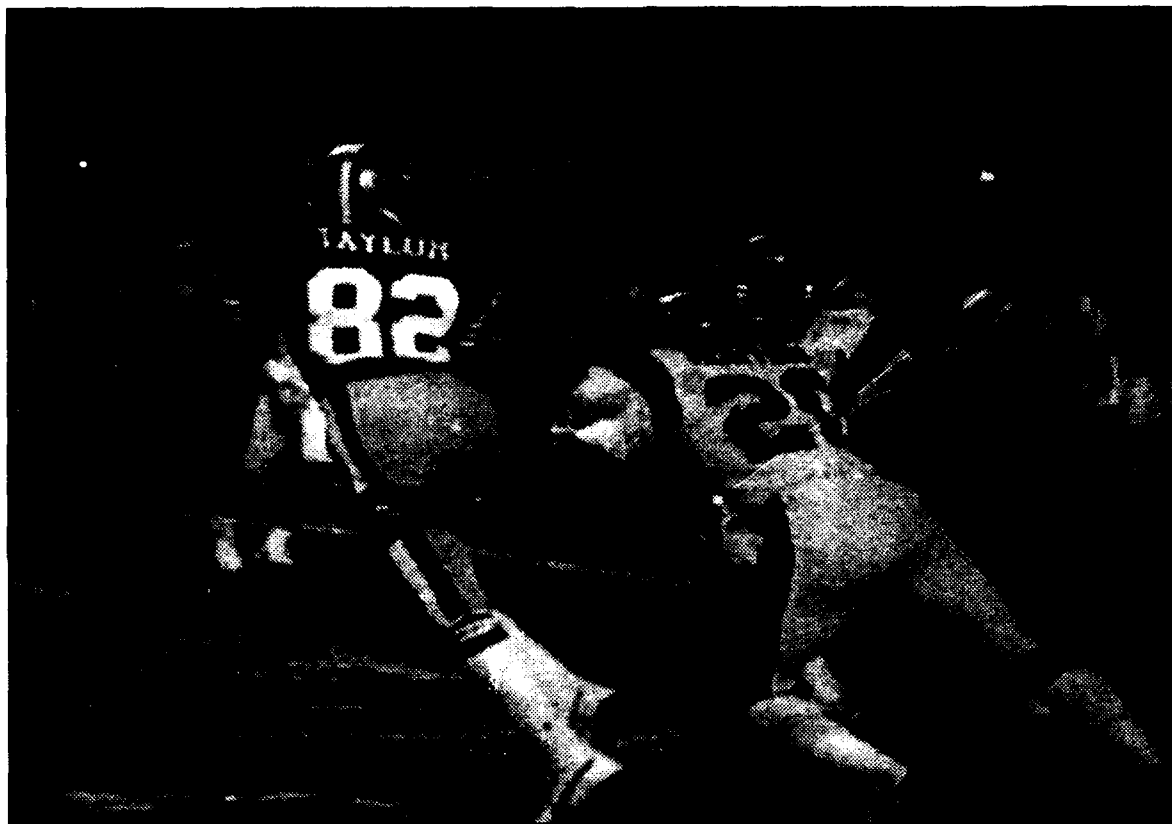


Figure 89. Example 2, result of mask 2 application

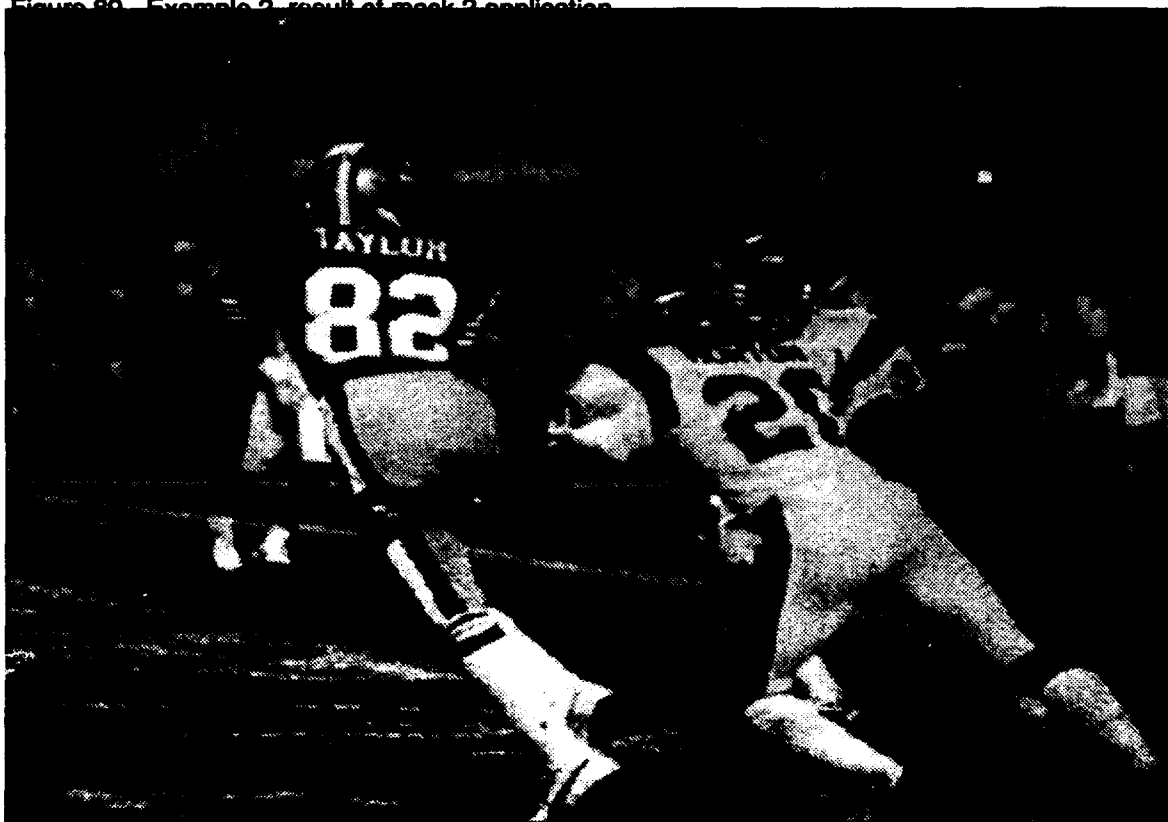


Figure 90. Example 2, result of mask 3 application

8 Image Thresholding

Abstract

Image thresholding in some cases can be used to enhance images and in other cases can be used to preprocess an image in order to increase the effectiveness of other enhancement techniques. This chapter discusses two thresholding techniques and their implementation.

Introduction

For some images, scratches and spots are isolated in value from the rest of the pixels. In these instances, thresholding by value can remove the blemishes. For other images, there may be a certain pixel value that has a low occurrence rate that hinders the full effect of some enhancement technique; thresholding by occurrence serves as a remedy.

Theory

Value thresholding allows all pixels within a certain value range to retain their value; pixels above this range are set to the high end value of the range, and pixels below the range are set to the low end value.

Occurrence thresholding, first, requires a histogram of the image. With this histogram, all pixel values with an occurrence below a certain threshold value can be remapped to one that has an occurrence above the threshold value. The new pixel value will always be the nearest pixel value (in the direction of the maximum occurrence pixel) that complies with the threshold.

Implementation

The code in Appendix N performs thresholding by value. All computation is done in conventional memory. Internal and external files are binary or standard RGB.

The code in Appendix O performs thresholding by occurrence; it uses the Victor Library to provide extended memory management (Catenary Systems 1992). Internal and external files are binary or standard RGB.

Results

First, value thresholding was tested. Figures 91 and 92 contain the original image and its histogram, respectively; Figures 93 and 94 contain the resulting image along with its histogram.

Next, occurrence thresholding was tested. Figures 95 and 96 show the resulting image as well as its histogram.

For both cases, the histogram proves that the techniques work as described in the theory. The displayed percentage on the histogram tells the percentage of pixels that possess the maximum occurring value.

Conclusions

Image thresholding is a simple process that can be used to enhance images as well as preprocess images to increase the effectiveness of other enhancement techniques.

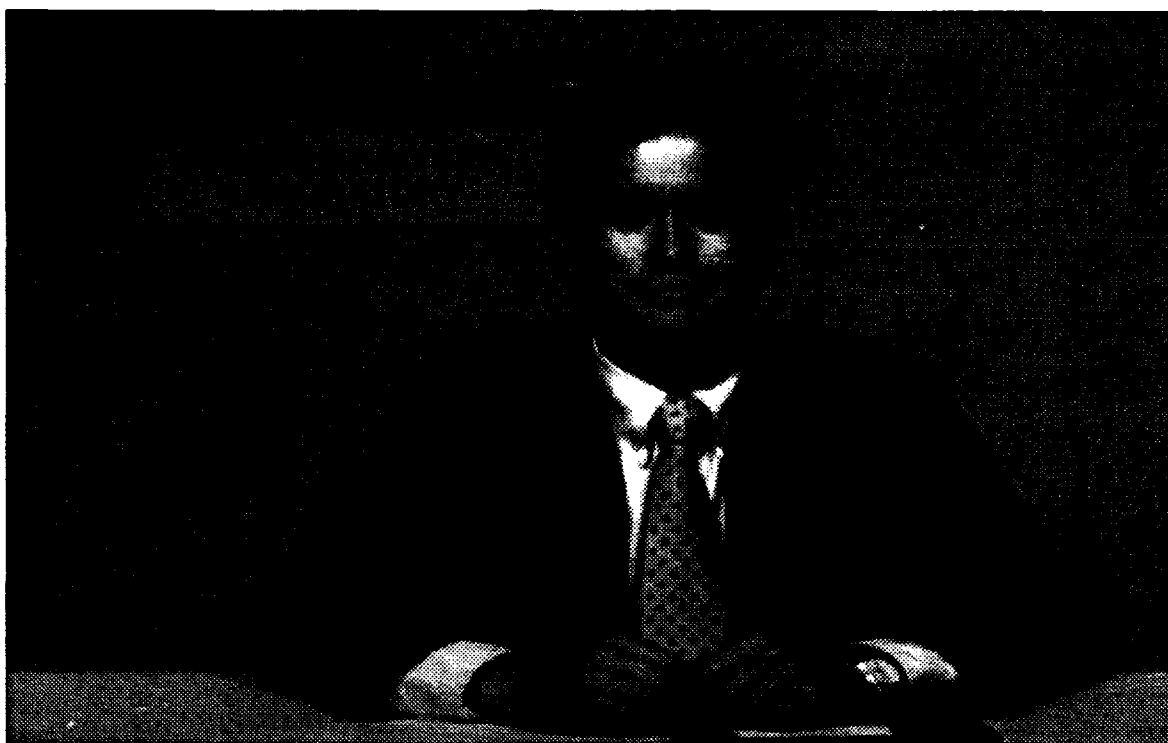


Figure 91. Original image before thresholding

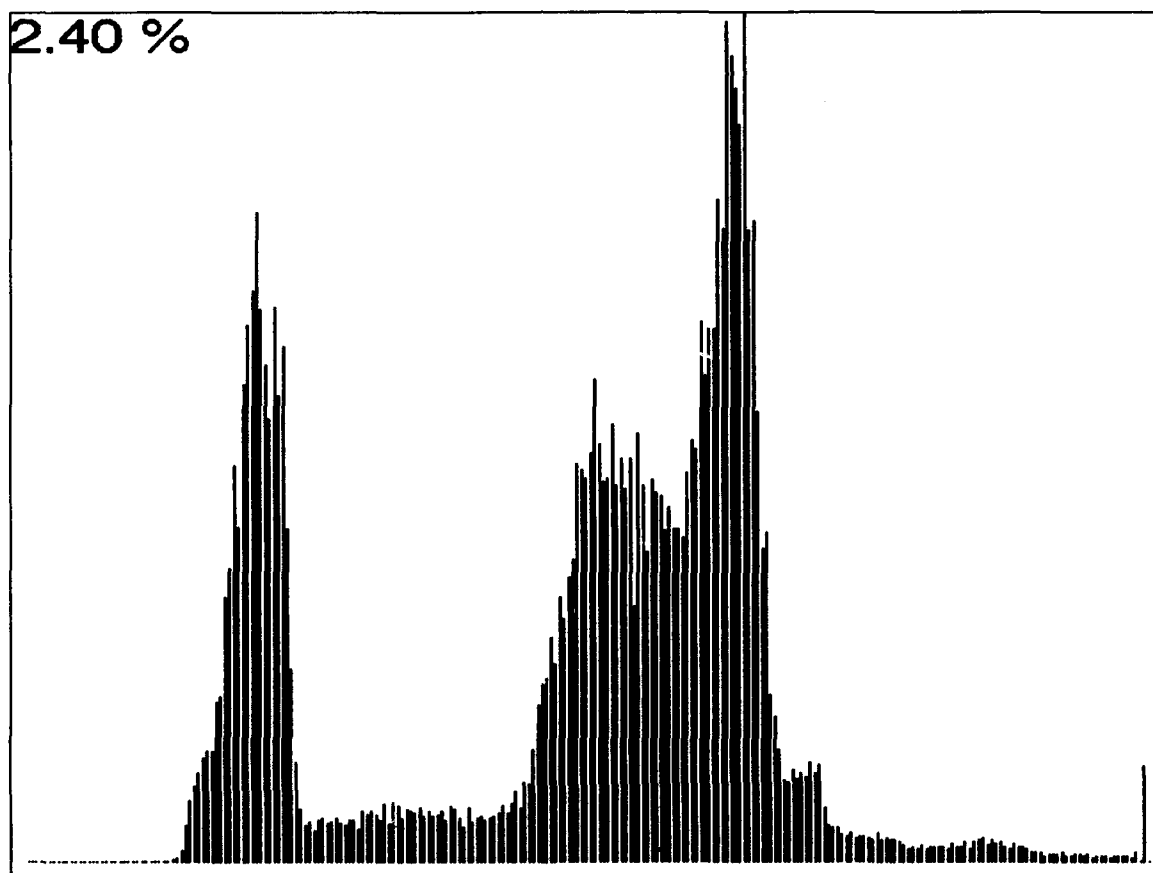


Figure 92. Histogram of original image before thresholding

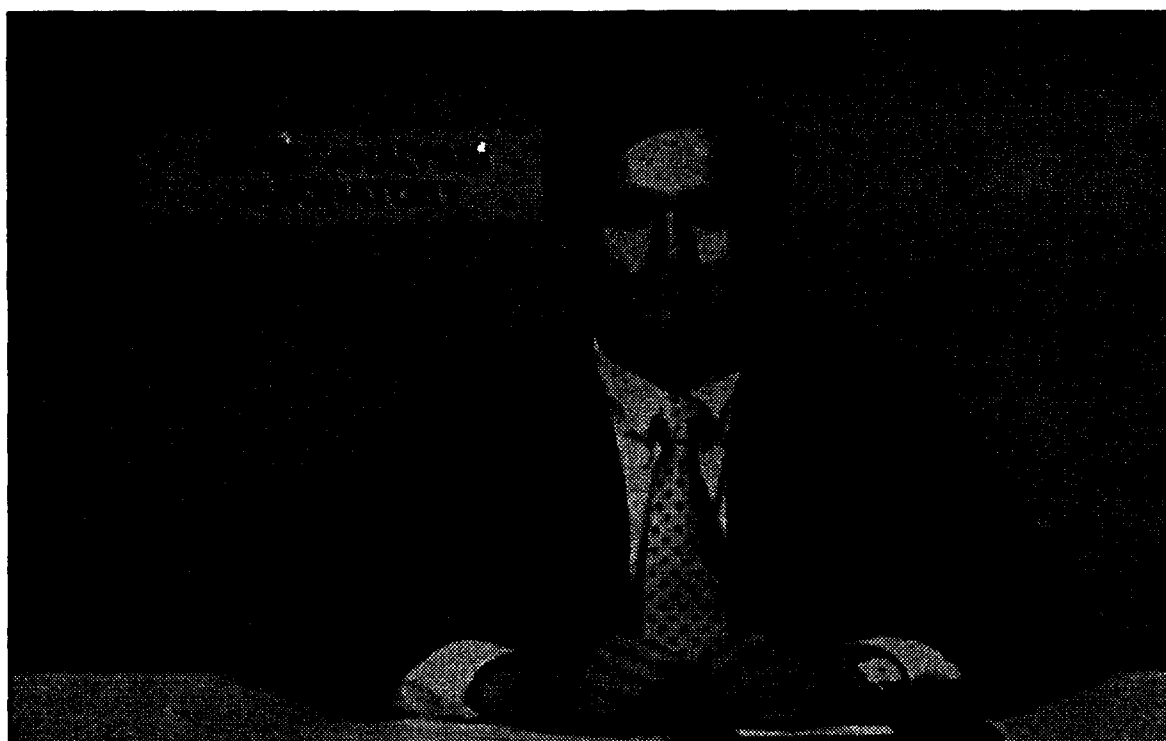


Figure 93. Result of value thresholding

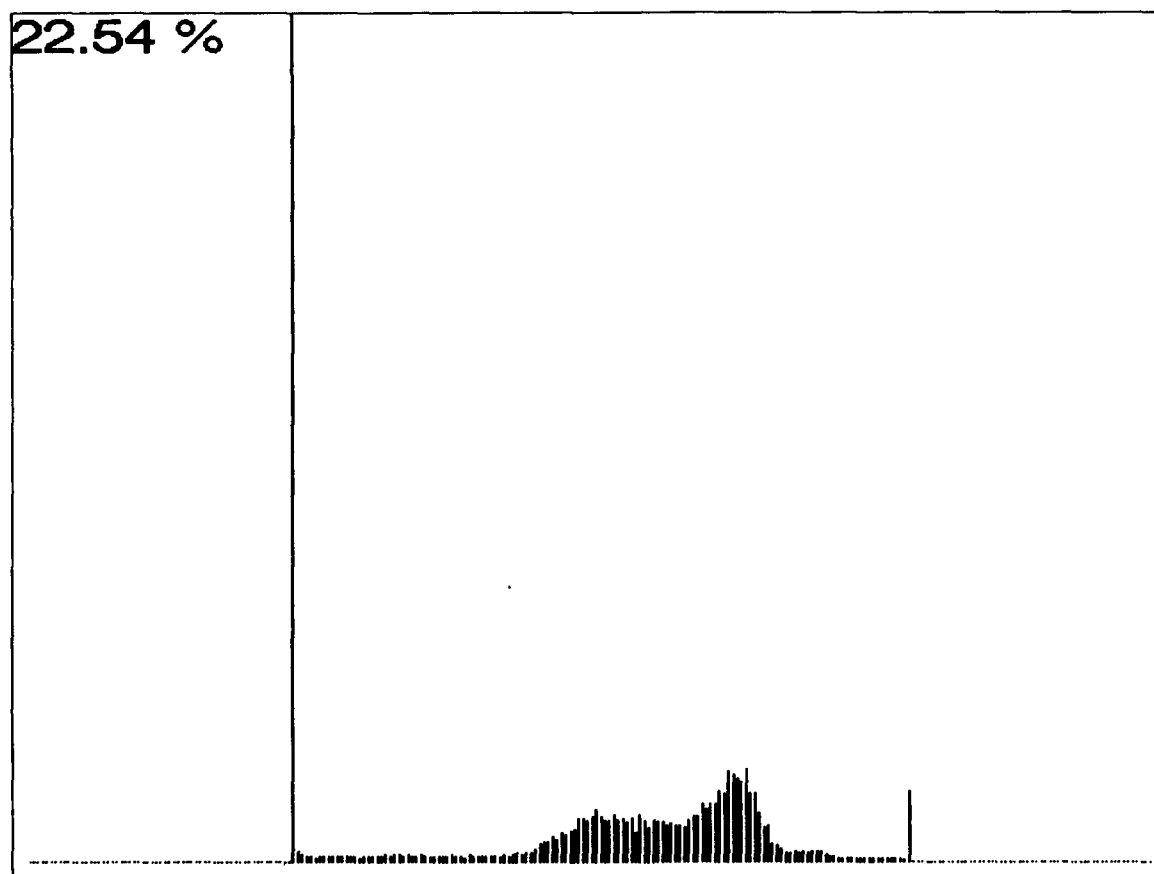


Figure 94. Histogram of value thresholding result

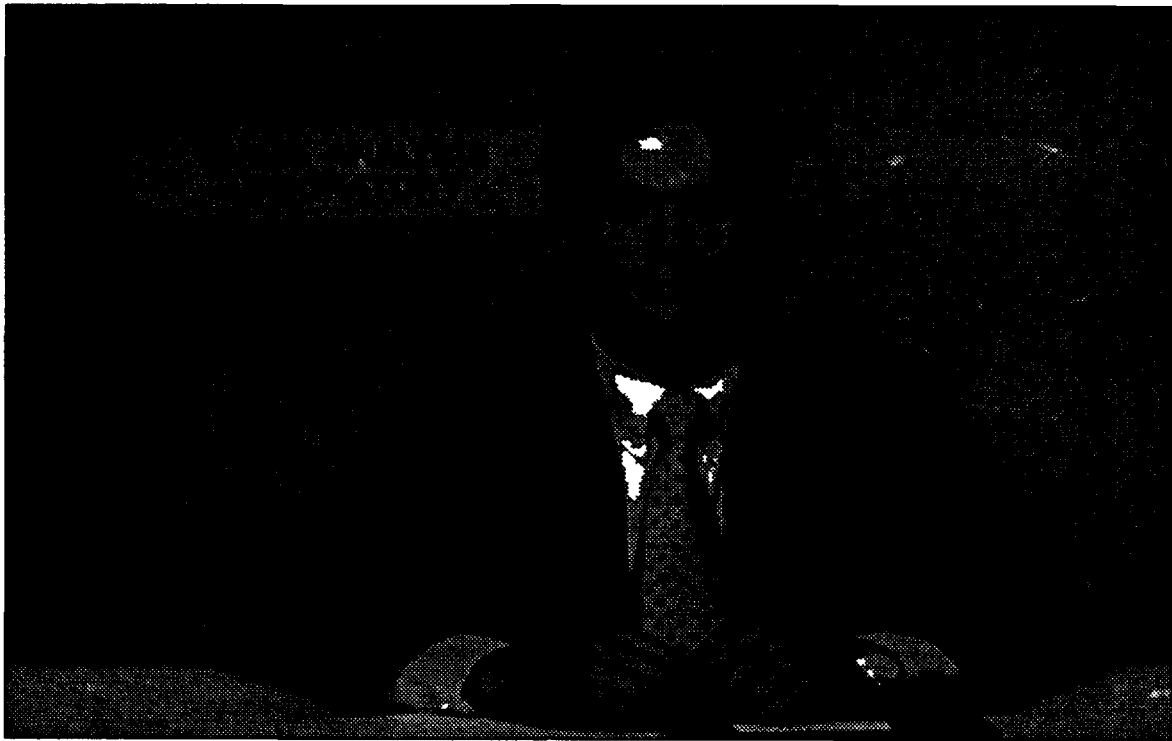


Figure 95. Result of occurrence thresholding

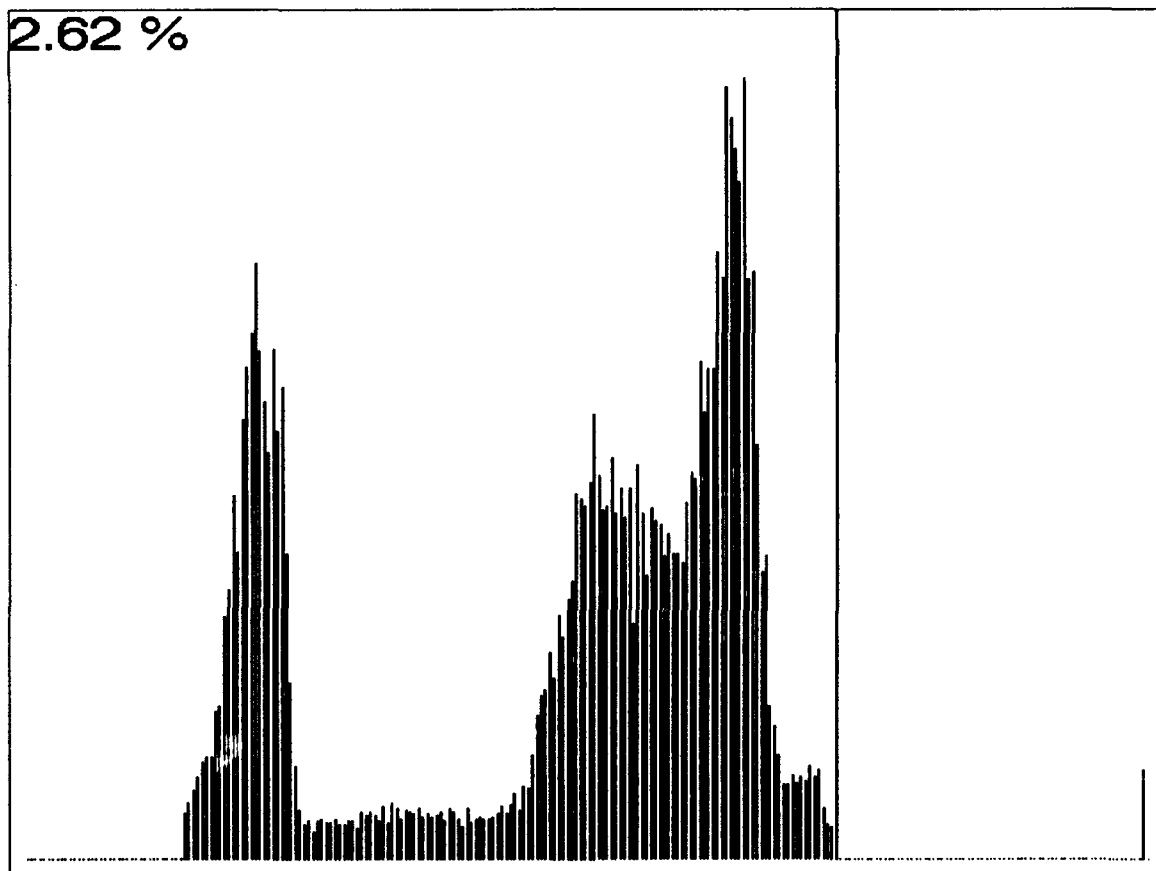


Figure 96. Histogram of occurrence thresholding result

9 Contrast Manipulation

Abstract

Contrast can be manipulated both subjectively and objectively. This chapter discusses both manipulations and their implementation.

Introduction

Subjective contrast manipulation consists of shading adjustments by means of linear or nonlinear re-orderings of a given histogram. Outcomes can be more or less appealing than the original.

Objective contrast manipulation involves respacing a histogram such that maximum equalized contrast can be achieved. This method can be described as an image extraction.

Theory

Table 8 contains equations to describe six types of subjective contrast manipulations (Pratt 1991). In each procedure, the pixel must be converted

Table 8 Subjective Contrast Manipulation Equations	
Square	$\text{new\_pixel} = (\text{pixel}/255.0)^2 * 255.0$
Cube	$\text{new\_pixel} = (\text{pixel}/255.0)^3 * 255.0$
Square root	$\text{new\_pixel} = (\text{pixel}/255.0)^{1/2} * 255.0$
Cube root	$\text{new\_pixel} = (\text{pixel}/255.0)^{1/3} * 255.0$
Inverse	$\text{new\_pixel} = (\text{pixel}/255.0)^{-1}$, $\text{pixel} \neq 0$ $\text{new\_pixel} = 255.0$, $\text{pixel} = 0$
Invert	$\text{new\_pixel} = (1.0 - \text{pixel}/255.0) * 255.0$

to a fraction between 0 and 1; then, the result is manipulated and finally reweighted.

Objective contrast manipulation takes an image that has an upper pixel value below 255 and/or a lower pixel value above zero and spreads out the histogram as evenly as possible to create an image with an upper value of 255 and a lower value of 0 (Pratt 1991). This procedure is commonly used to extract information from satellite and airplane reconnaissance photos.

Implementation

The code in Appendixes P, Q, R, S, T, and U performs the following subjective contrast manipulations, respectively: square, cube, square root, cube root, inverse, and invert. All computation is done in conventional memory. Internal and external files are binary or standard RGB.

The code in Appendix V performs image extraction; it uses the Victor Library to provide extended memory management (Catenary Systems 1992). Internal and external files are binary or standard RGB.

Results

First, the subjective contrast manipulation techniques were tested. Figures 97 and 98 contain the original image and its histogram. Figures 99 and 100 contain the results of the squaring method; as can be seen, the histogram is shifted nonlinearly toward pixel value 0. The results of the cubing method are shown in Figures 101 and 102; the histogram is shifted toward 0 more dramatically than the square method's. Figures 103 and 104 contain the results of the square root method; the histogram is shifted nonlinearly toward pixel value 255. The cube root results have a more pronounced shift toward 255, as shown in Figures 105 and 106. Figures 107 and 108 show the results of the inverse ($1/x$) method; the results show that this method has a strong tendency to darken the image. The results of the inverting method are shown in Figures 109 and 110; the corresponding histogram is a mirror image of the original.

Next, image extraction was tested. To demonstrate the usefulness of this method, the original image was chosen to be the Prewitt Eight Neighbor second-order edge-detection results. Figures 111 and 112 show the original and its histogram. Before performing the extraction, occurrence thresholding was performed, as shown in Figures 113 and 114. The final extraction results are shown in Figures 115 and 116.

Conclusions

The most powerful contrast manipulation tool is image extraction; it can increase the resource value of an image that has extremely dull contrasts. Other contrast manipulation methods are purely subjective adjustments.

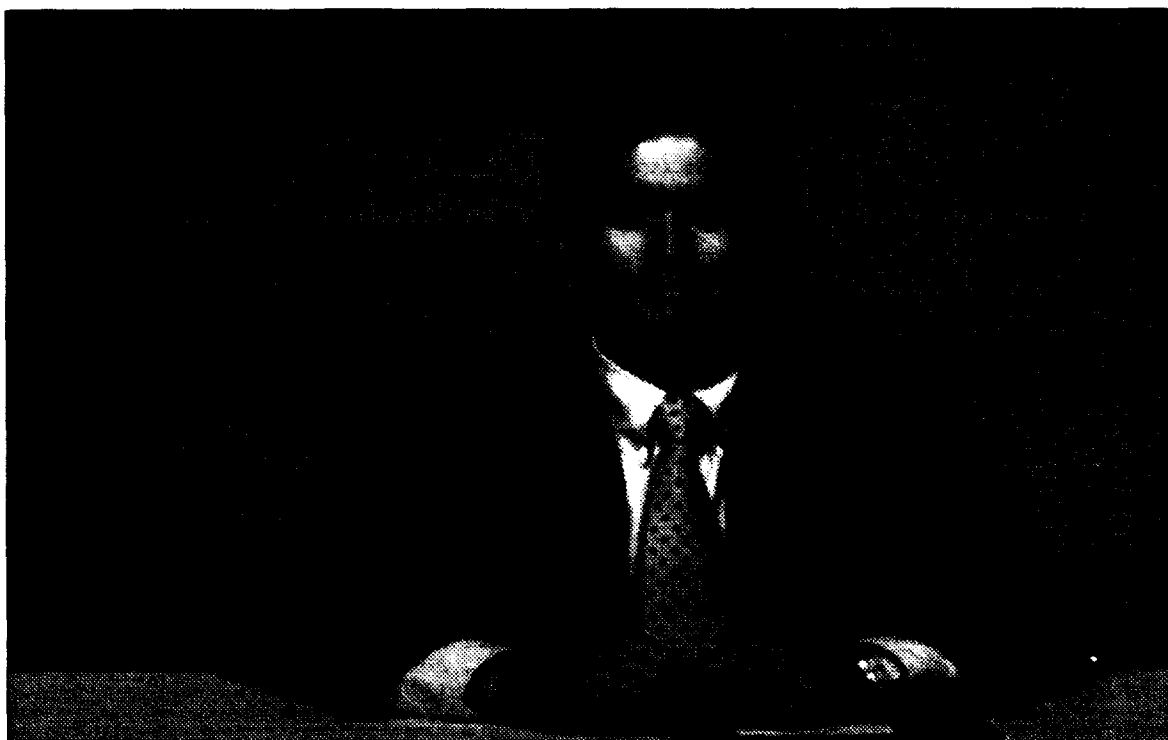


Figure 97. Original image before contrast manipulation

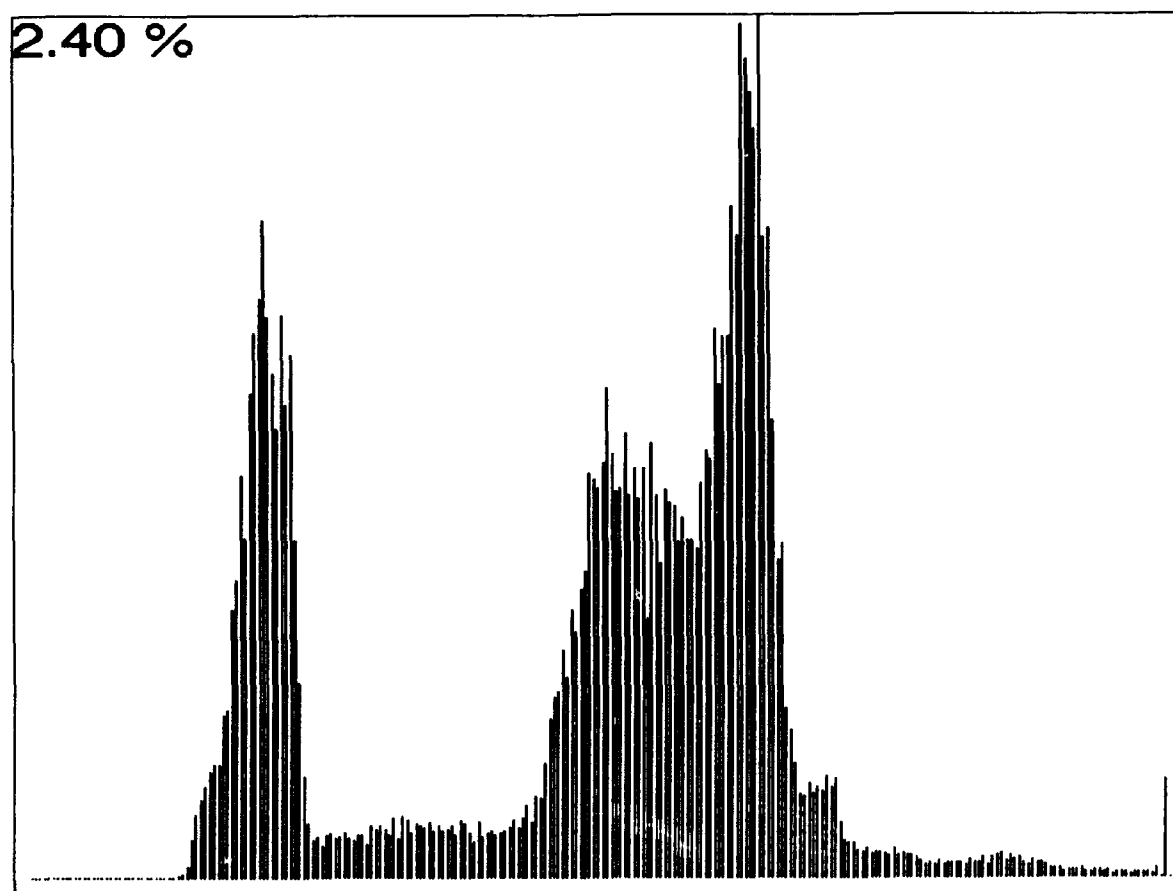


Figure 98. Histogram of original image before contrast manipulation



Figure 99. Result of square method

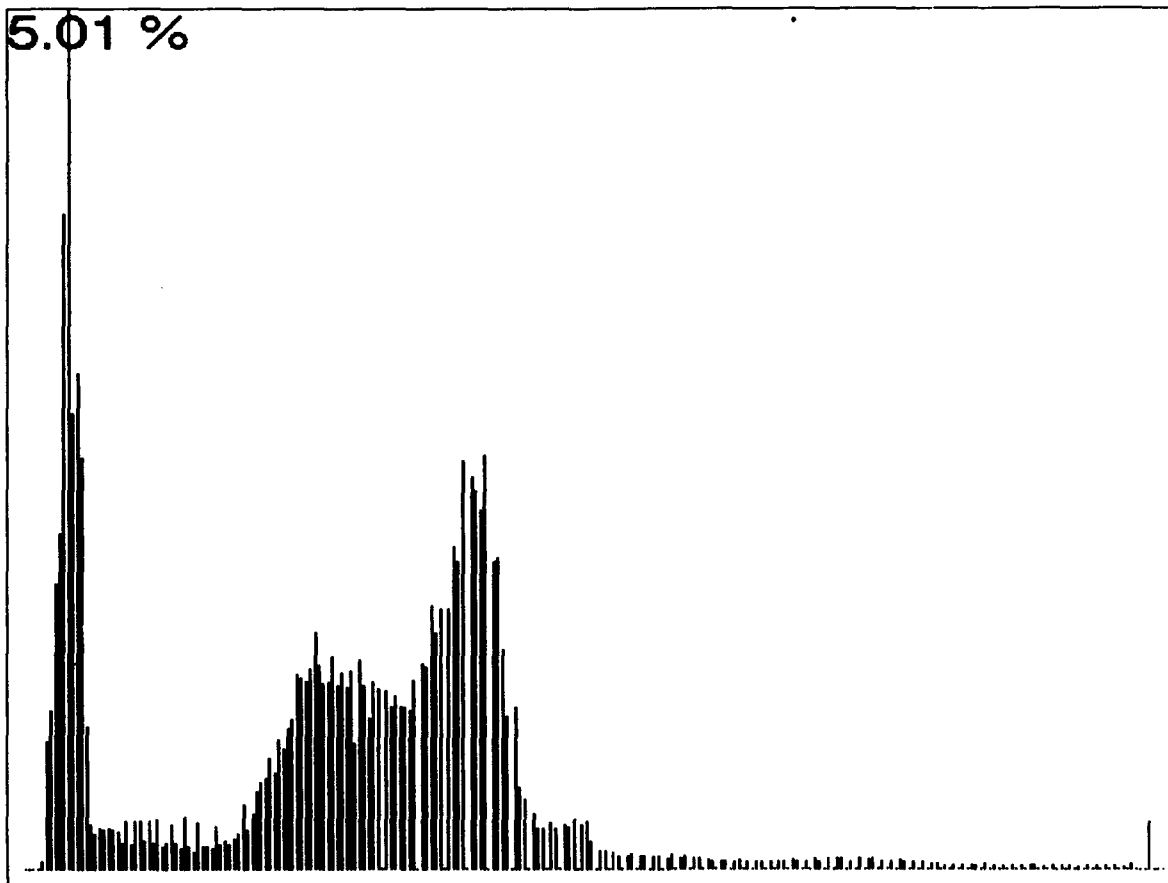


Figure 100. Histogram of square method result



Figure 101. Result of cube method

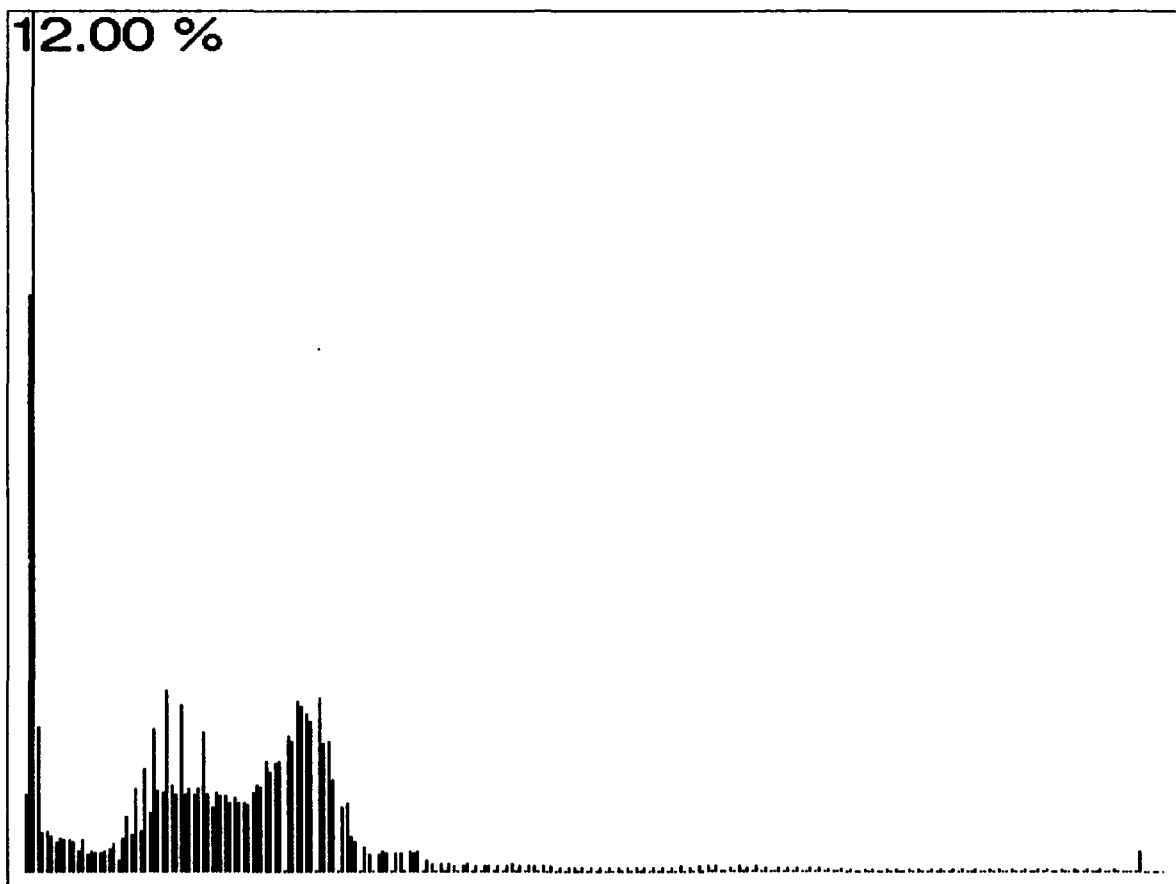


Figure 102. Histogram of cube method result

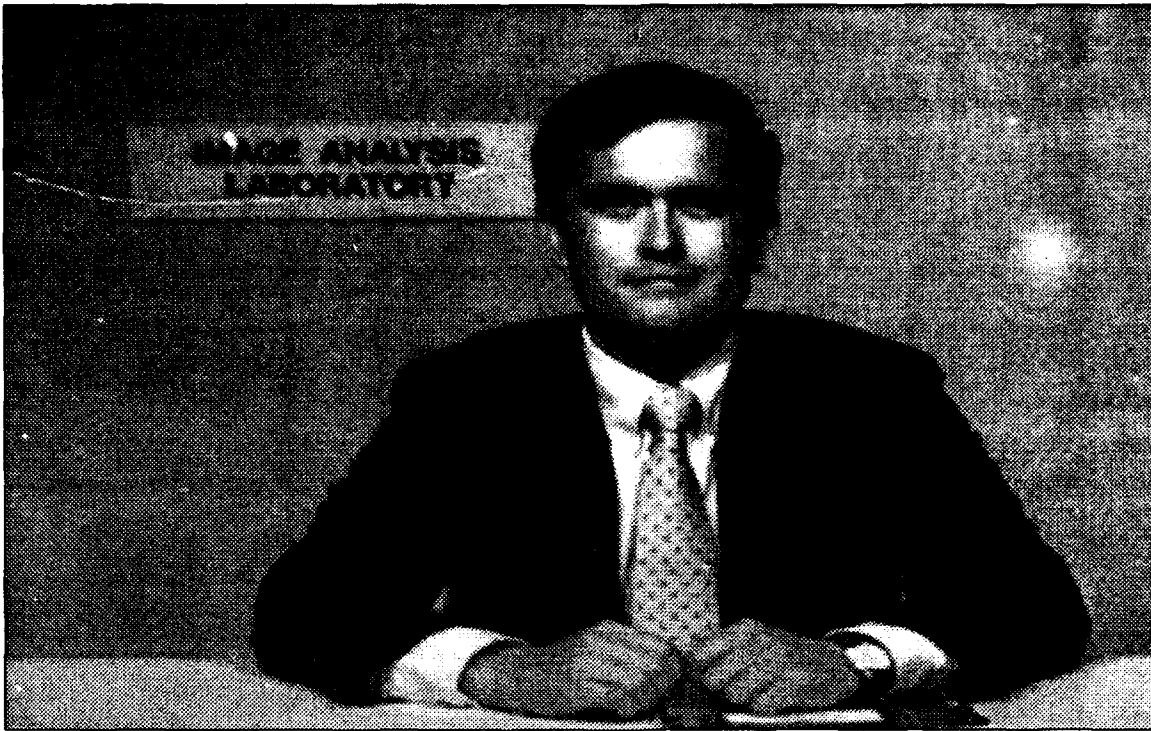


Figure 103. Result of square root method

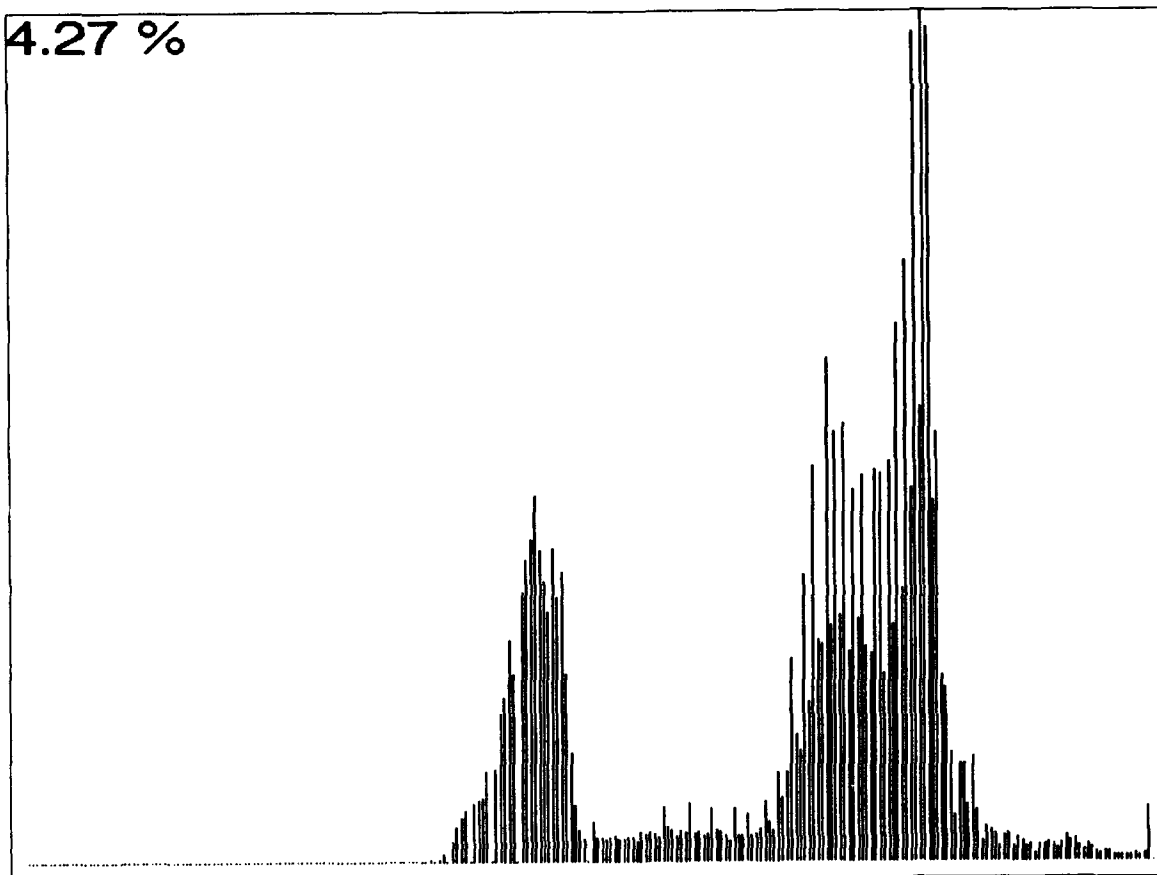


Figure 104. Histogram of square root method result

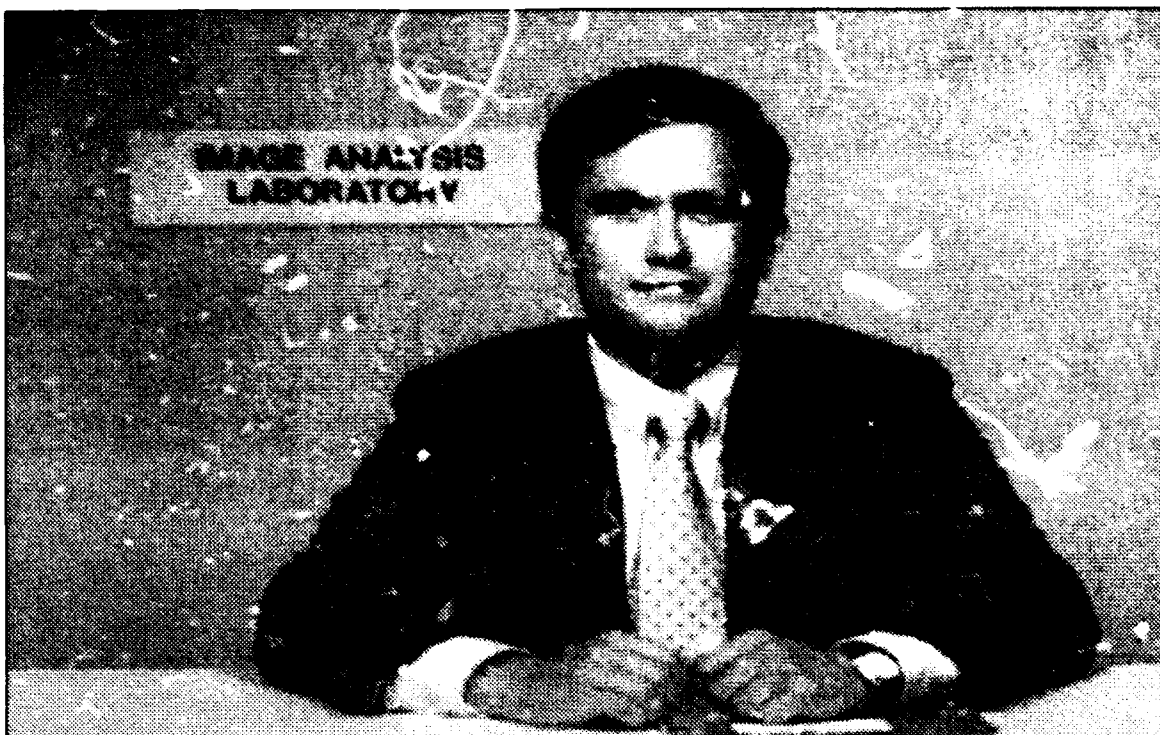


Figure 105. Result of cube root method

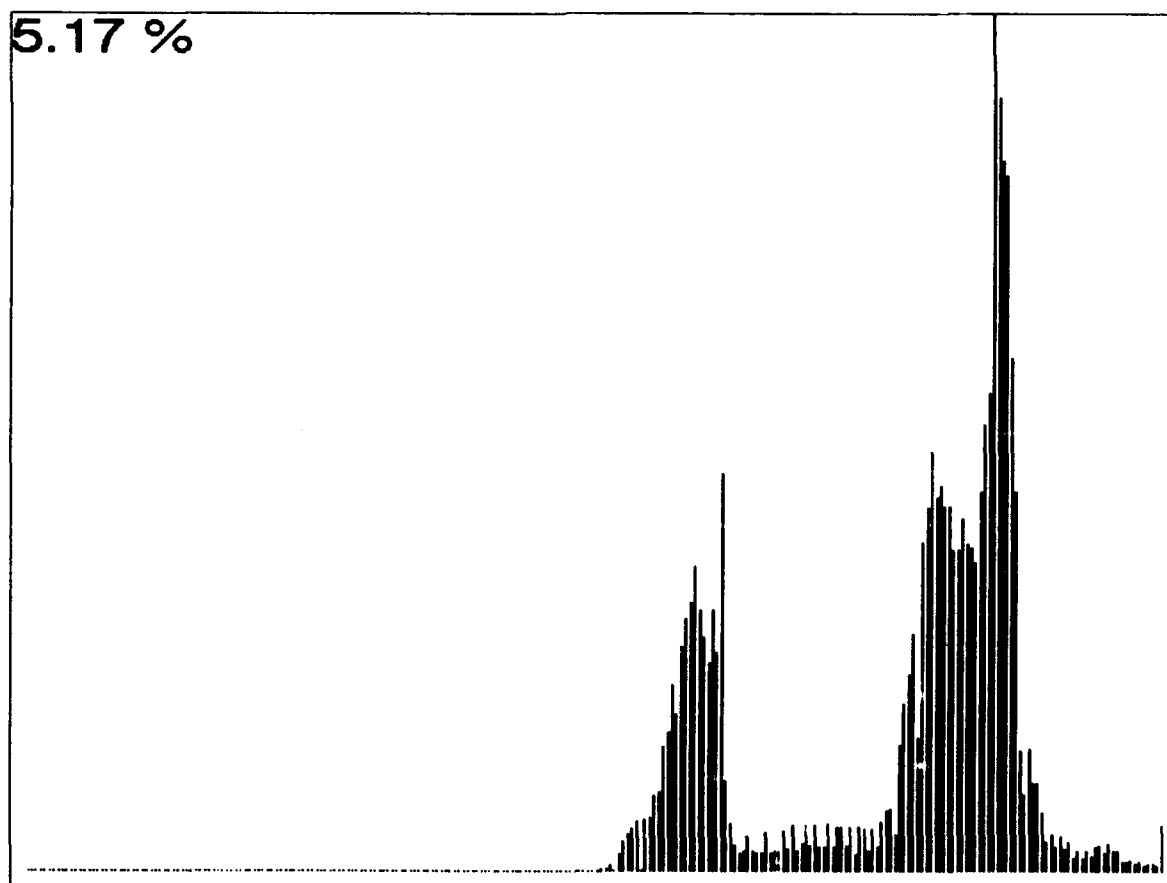


Figure 106. Histogram of cube root method result



Figure 107. Result of inverse ($1/x$) method

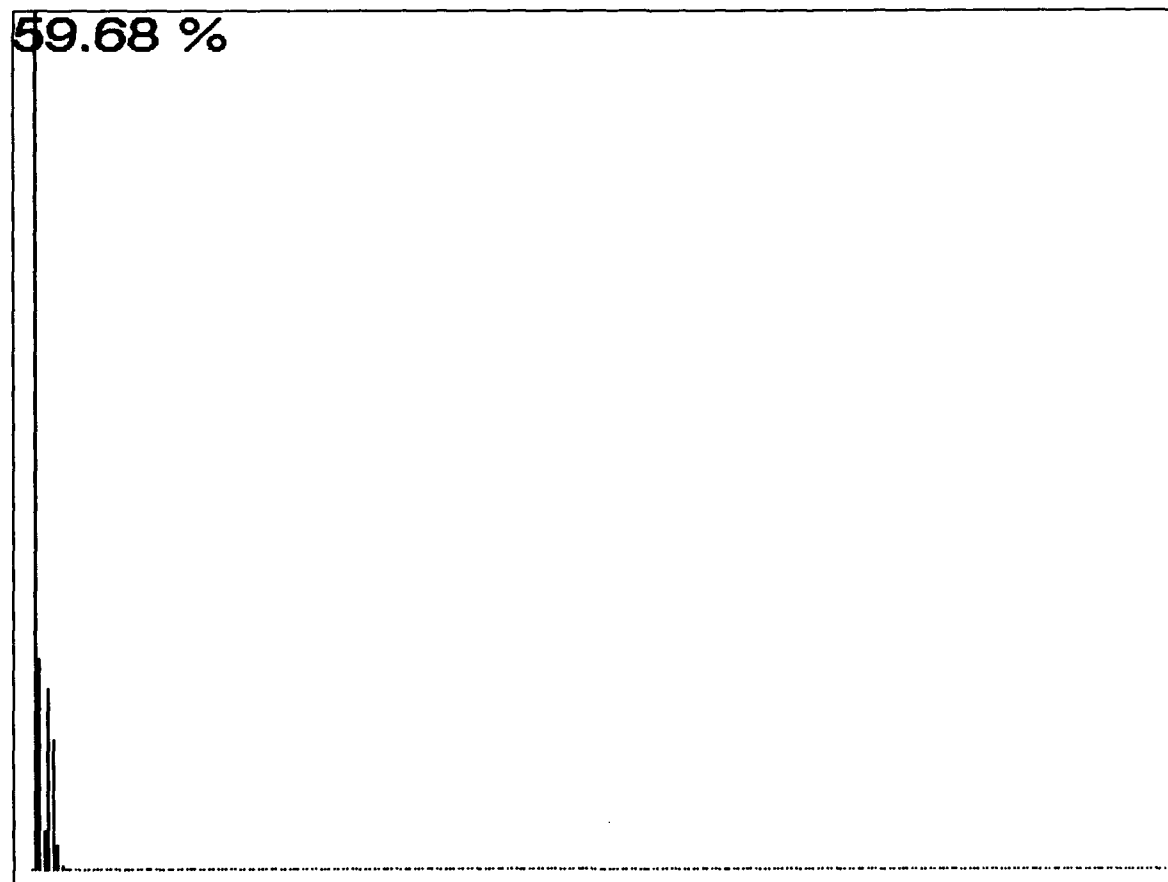


Figure 108. Histogram of inverse ($1/x$) method result

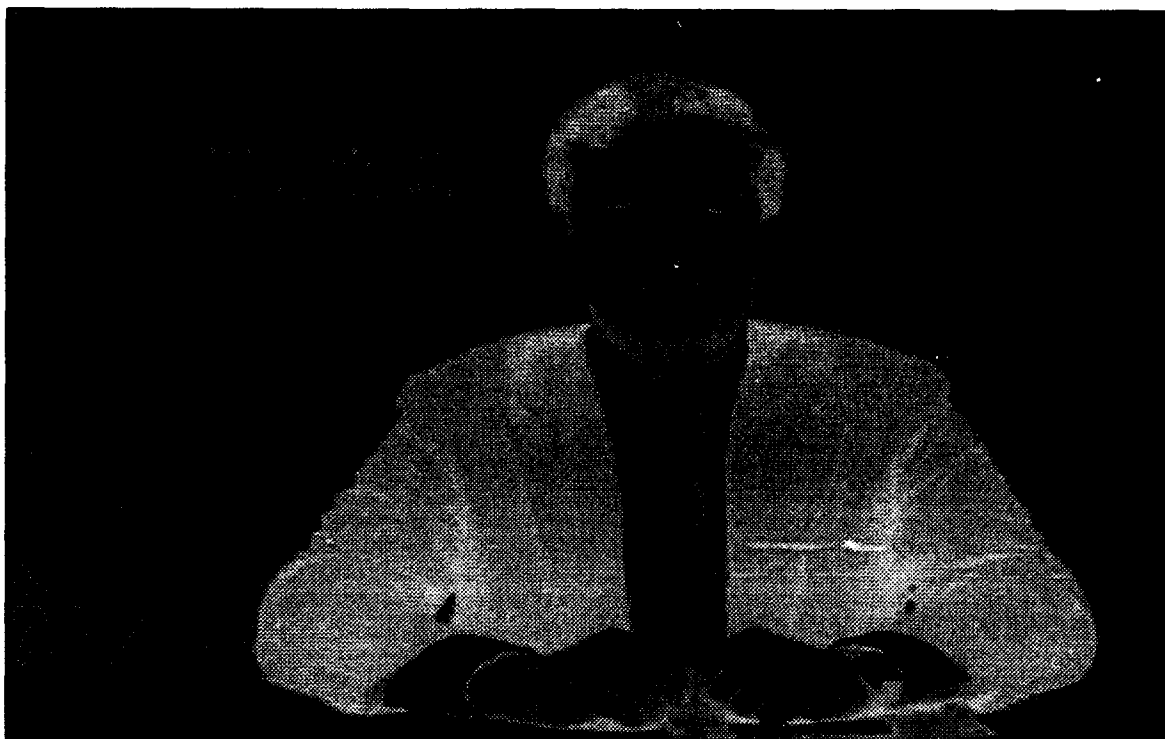


Figure 109. Result of inverting method

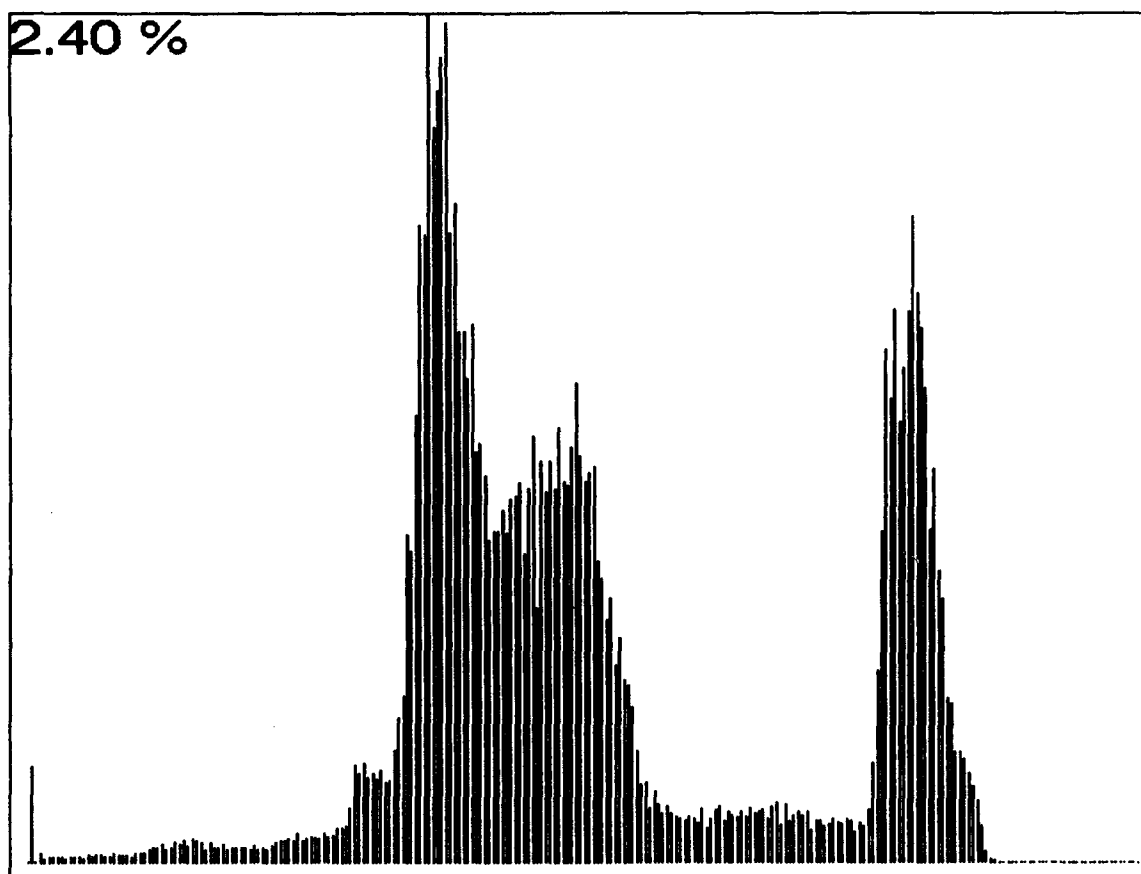


Figure 110. Histogram of inverting method result

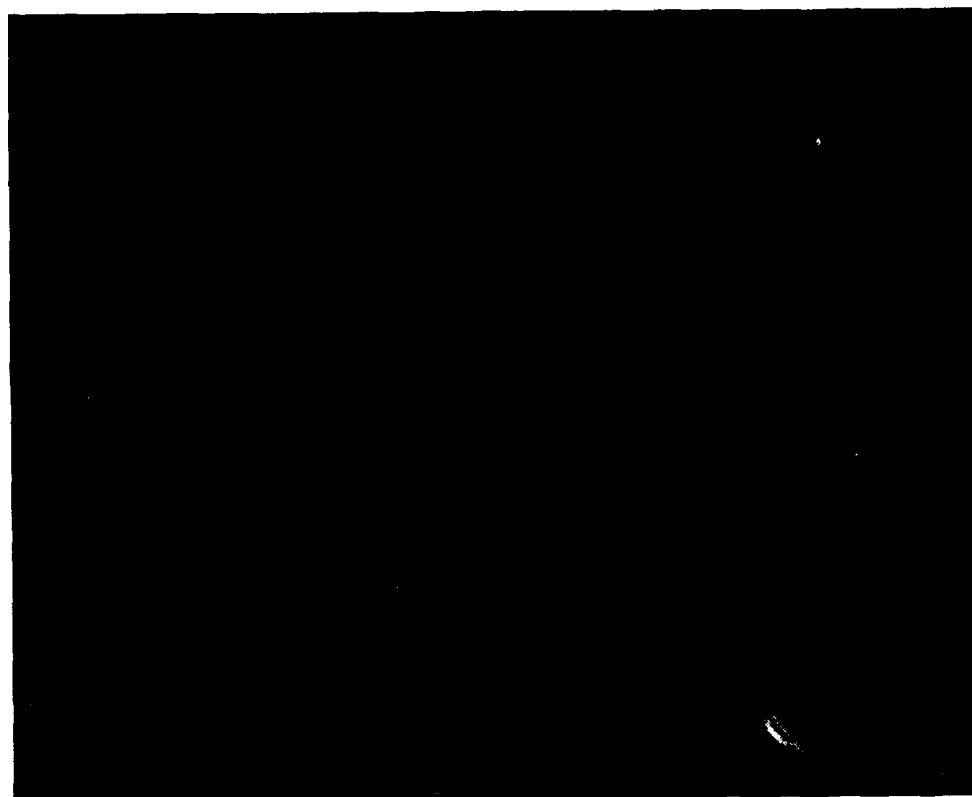


Figure 111. Original image before image extraction

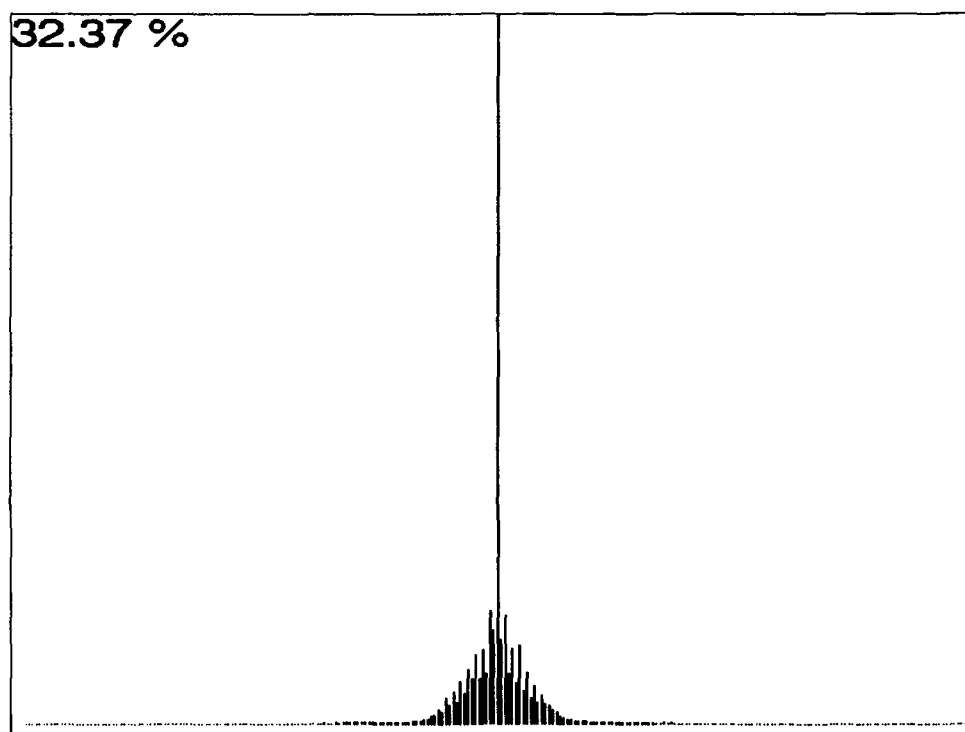


Figure 112. Histogram of original image before image extraction

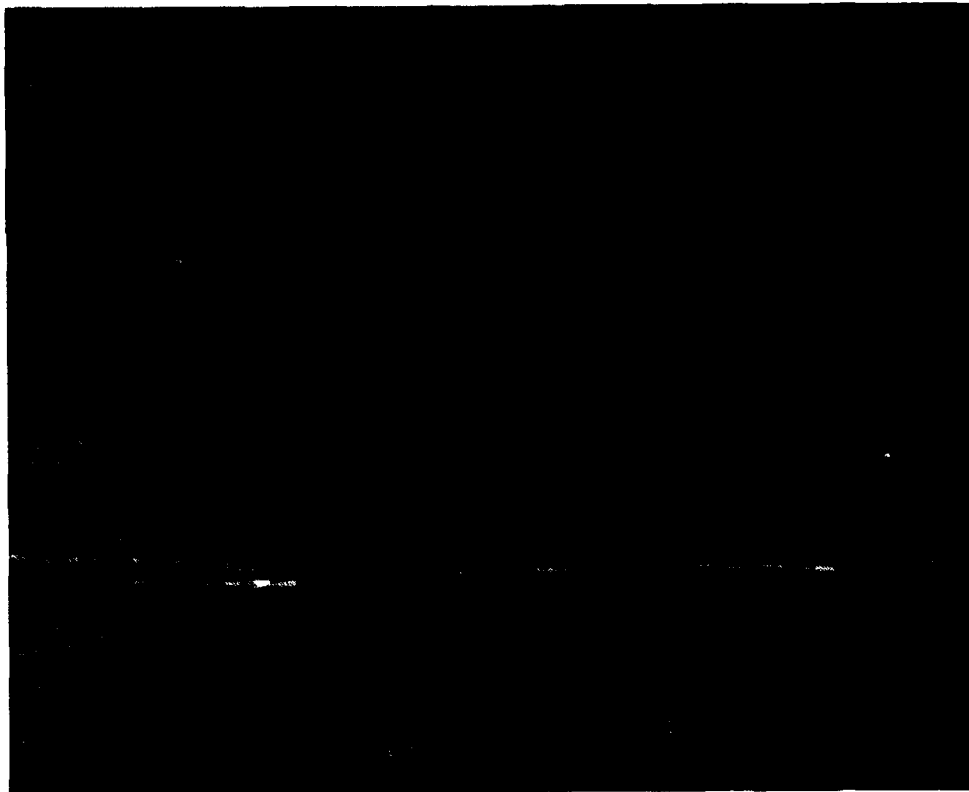


Figure 113. Result of occurrence thresholding

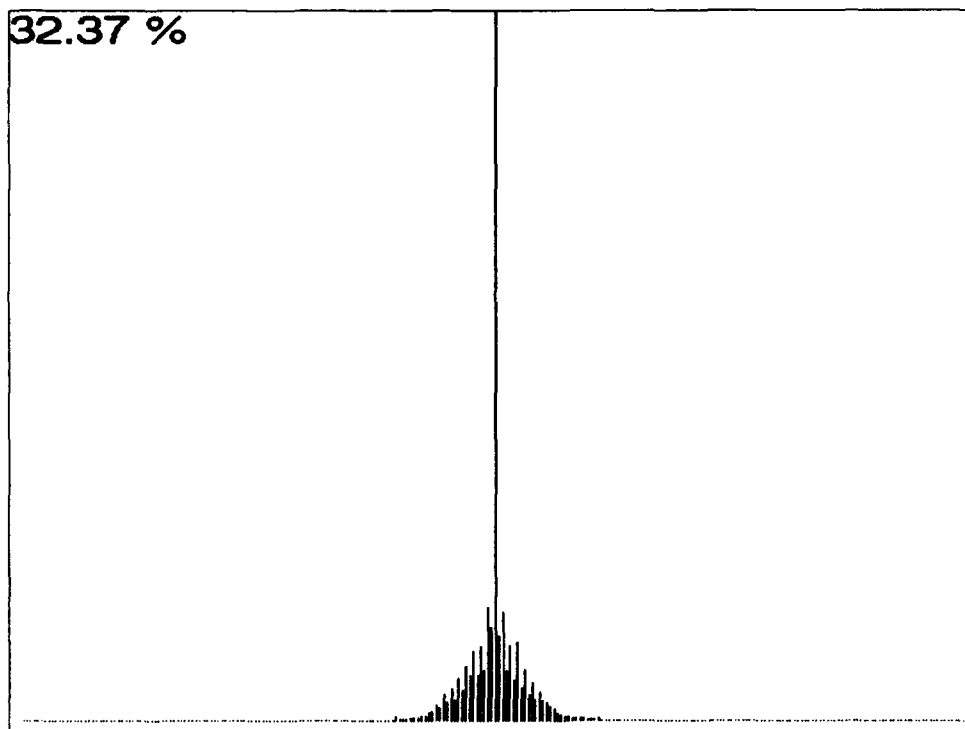


Figure 114. Histogram of occurrence thresholding result

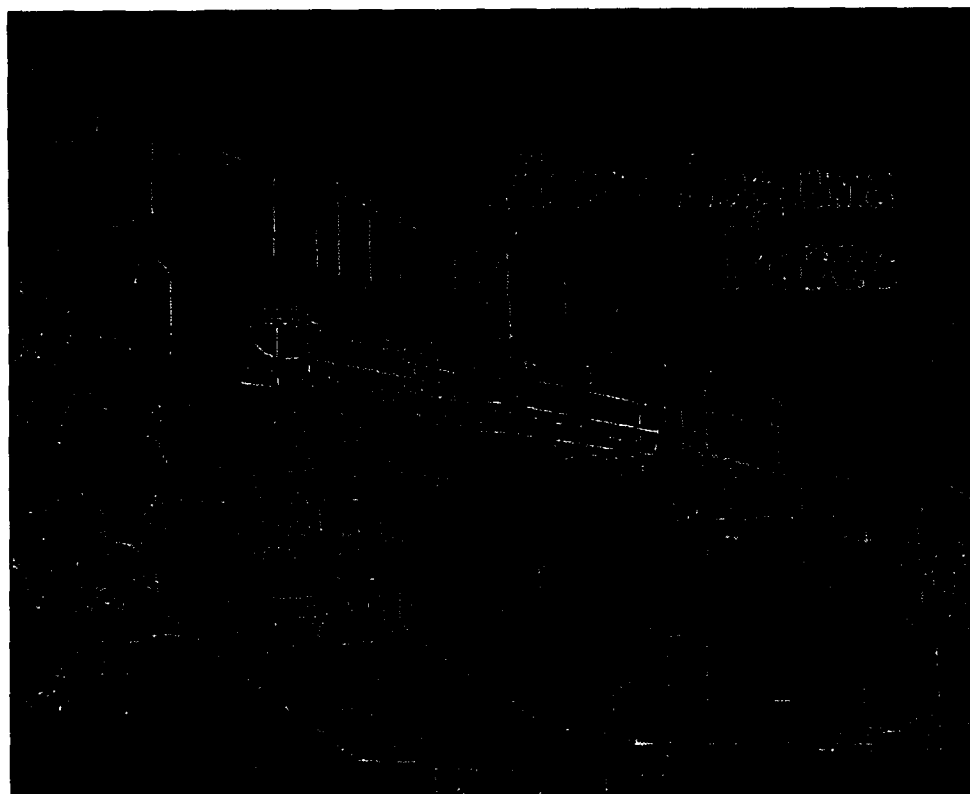


Figure 115. Result of image extraction

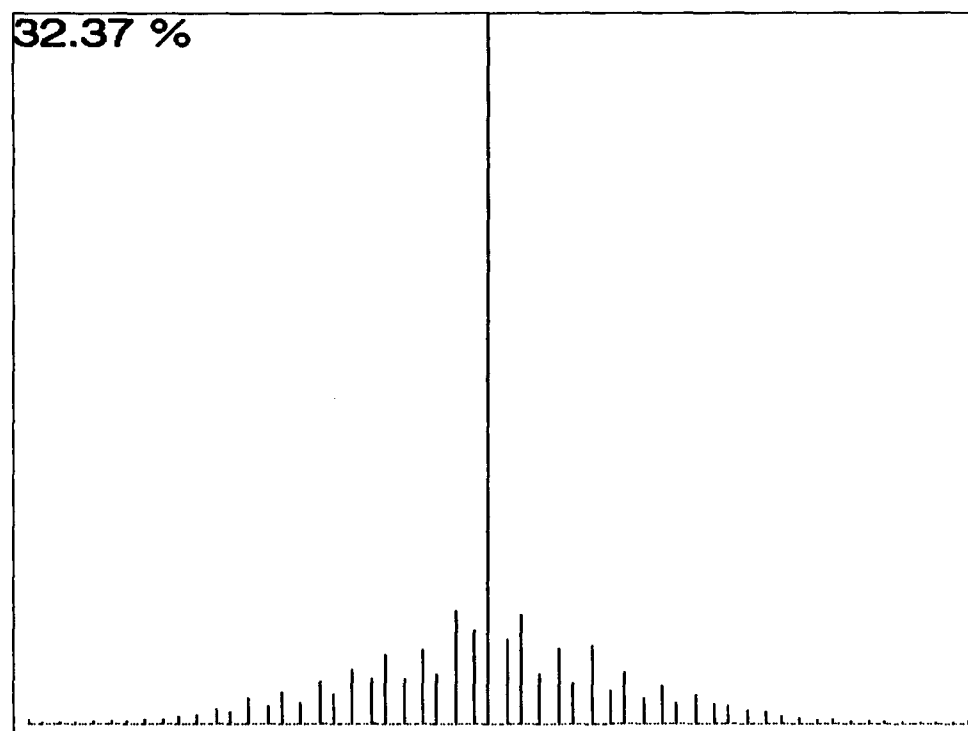


Figure 116. Histogram of image extraction result

10 Median Filtering

Abstract

The Median Filter is unlike other noise-cleaning tools in that it is not a true low-pass filter. This chapter discusses both the theory and implementation of this filter.

Introduction

The Median Filter was developed by J. W. Turkey, as a noise-suppression tool (Pratt 1991); however, it is not a general purpose filter because it has no distinct frequency or correlation basis. This method is merely a spatial manipulation that uses the mathematical median in an attempt to attenuate noise. In some cases, the ideal image becomes suppressed, especially when the image has low correlation from pixel to pixel (Pratt 1991). Also, resolution is usually diminished as a result of using the Median Filter (Pratt 1991).

Theory

The mathematical median is defined as the value of the middle member of an odd-membered sorted set, as shown:

$$S = X_1, X_2, X_3, \dots, X_N$$

such that $N \in \text{odd integers}$ and S is sorted (47)

$$\text{MEDIAN}\{S\} = X_{(N+1)/2}$$

For an image, a windowing algorithm, much like the windowed convolution, is used. However, instead of performing the convolution, the median is found for each window.

Implementation

The code in Appendix W performs a windowed implementation of the Median Filter. All computation is done in conventional memory. Internal and external files are binary or standard RGB. Sorting is performed with a standard bubble sort.

Results

For comparison, the Median Filter was tested on the same image that the noise-cleaning masks were tested. The original image and its histogram are located in Figures 117 and 118; the results of the Median Filter and the corresponding histogram are shown in Figures 119 and 120. The resulting image is noticeably blurred; however, the noise has been reduced. In comparison with the noise-cleaning masks, the Median Filter is inferior because of its heavy loss of resolution.

Conclusions

The Median Filter in most cases is an undesirable method for noise removal, since noise-cleaning masks provide a reliable tool.

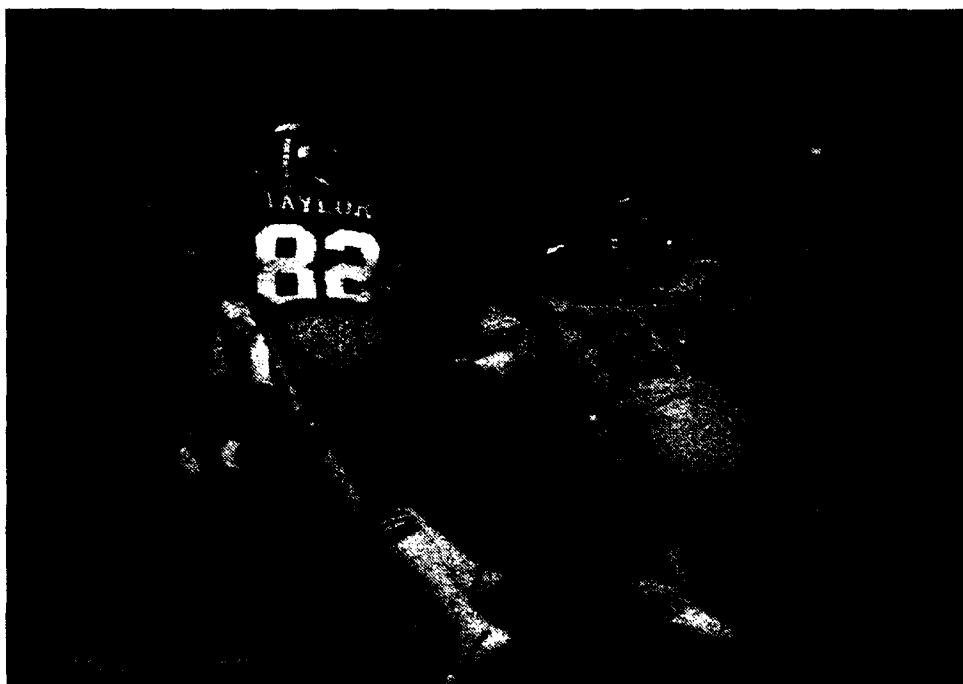


Figure 117. Original image before median filtering

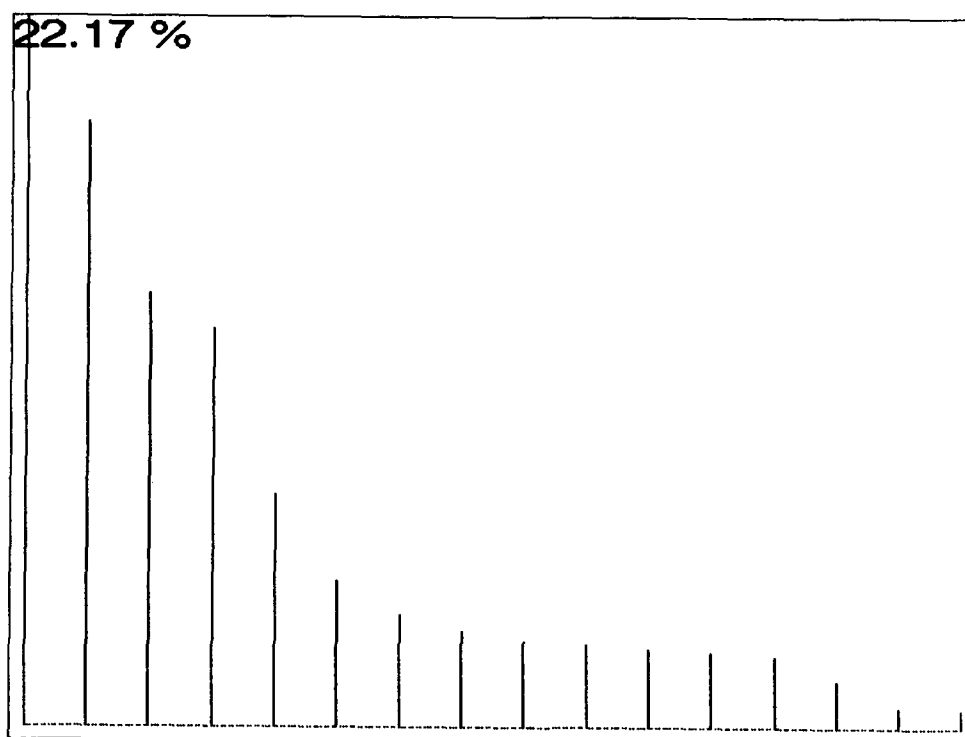


Figure 118. Histogram of original image before median filtering

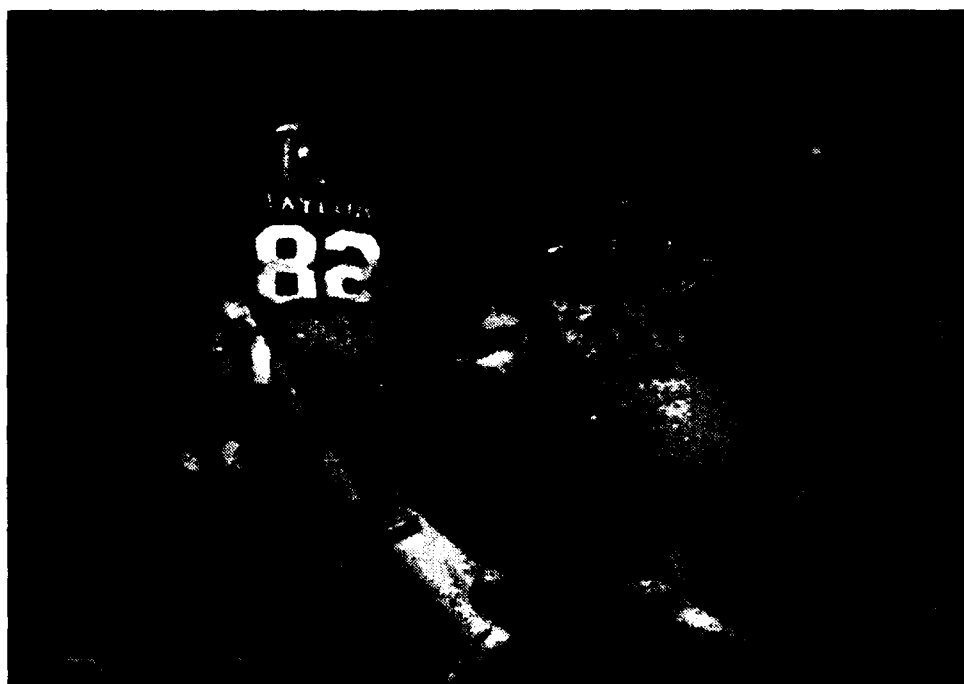


Figure 119. Result of median filter

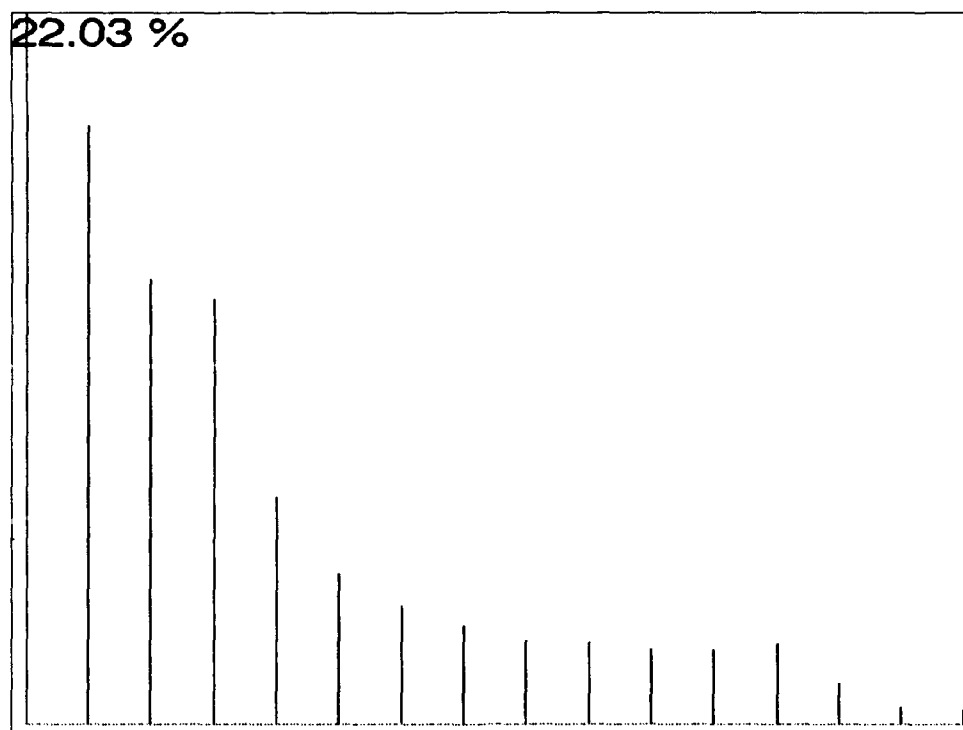


Figure 120. Histogram of median filter result

11 Video Conferencing Using Personal Computers

Abstract

Video conferencing using PCs requires digital control of a video signal, an audio signal, and a connection network. This chapter discusses the implementation and application of all three controls.

Introduction

For the past few years, interest in video communication has swelled, influencing the invention of videophones. Although these phones allow users to see as well as hear others, they cost approximately \$2,500 per phone and take several seconds to transmit a single still frame image. Systems that send and receive continuous video usually require, as a minimum, a dedicated PBX as a central control unit, costing approximately \$200,000, in addition to the cost of each desktop station at approximately \$18,000, or more.

During the summer of 1993, a design team at WES developed a prototype PC video-conferencing system with a \$1,000 price tag per node, requiring no overhead control unit such as a PBX. The system provides one-way video and audio communication over the WES ethernet network and was developed as a prototype system for incorporation and demonstration of the high level image compression and image restoration techniques developed within the WES Information Technology Laboratory (ITL) during FY92 and FY93. It is anticipated that the use of these sophisticated imaging algorithms will allow the prototype video-conferencing system to be developed into a fully functional system, complete with higher resolution, lower bandwidth requirements, multiway conferencing, split screens, secure video-conference sessions, video mail, and many other features. The video frame is currently 160x100 pixels in size and is refreshed at 12 times per second, demanding a 1.5 Mbits/s transfer rate. The audio is sampled

at 8,000 times per second and sent at 64 Kbits/s. All pixels and audio samples are 8 bits long.

The sample output in Figure 121 shows the video conferencing system used in a split screen configuration.



Figure 121. Typical video conferencing screen used for split mode operation

Implementation

Two 486/33MHz machines were used, one as a transmitter, the other as a receiver. Table 9 lists the critical system components for each machine.

Table 9 List of Components	
Transmit PC	Receive PC
3C503 Ethernet Card FTP, Inc., TCP/IP Kernel Sound Blaster Pro 16 Card Video Graphics Adapter Card New Media Graphics Video Card NTSC Home Video Camera Microphone	Western Digital Ethernet Card FTP, Inc., TCP/IP Kernel Sound Blaster Pro 16 Card Video Graphics Adapter Card Speaker

Appendix X contains the code for the transmit PC, and Appendix Y contains the code for the receive PC. The FTP kernel, by FTP, Inc., was used as the communication driver to the ethernet board. Appendix Z contains an alternate receiver algorithm that uses a packet driver interface rather than the FTP kernel software (the packet driver specifications are located in Appendix BBB.) The code in Appendix Z can be run on multiple PCs simultaneously to view the images sent by a single transmitting station. The alternate receiver algorithm temporarily reprograms the local ethernet card to trick it into accepting packets sent to the main receiving station.

Together, all hardware and software components provide one-way video and audio communication. The transmit program sends video packets captured from the camera by the New Media Graphics Video Card and audio captured from the microphone by the Sound Blaster Pro 16 Card; the audio is transmitted every 1/8th second, which is the time required to accumulate 1,000 bytes of audio. The video is transmitted, two scan lines at a time, between audio samples. The time delay for both the audio and the video is approximately 1/8th.

The function *AVgrabToBitmap*, in Appendix X, captures NTSC frames from a home video camera and records them in *hbuf*, a buffer located in PC RAM. Since the color format is 4:1:1 YUV, every other byte of *hbuf* is copied to *buffer* in order to isolate the luminance. Once *buffer* is filled with 320 bytes or two scan lines, the function *net_write* sends *buffer*'s contents in a UDP packet over the ethernet. A single byte header is provided in the packet to tell the receiving PC the row position of the two scan lines.

Currently, this transmitting method uses subsampling to reduce the required transmission rate. However, a DSP chip may be included to perform various Fourier compression methods to allow enhanced resolution and increased frame rates.

The function *ctvm_input* initializes digitizing sound capture from an analog microphone to a 1,024-byte digital PC RAM buffer *superbuf*. The first 16 bytes and last 4 bytes are used for header and trailer storage. Every 0.125 sec, *superbuf* is filled and transmitted. Due to a design flaw in the Sound Blaster Card, the sound information must be processed with a raised-cosine filter, which is optimal. The flaw is exposed in the receive Sound Blaster. With an oscilloscope, a 7-msec pasting delay between consecutive sound packets was measured. This delay causes a sharp clicking sound, which is attenuated with the raised-cosine filter. Equation 48 describes the filter's transfer function, and Equation 49 describes the filter's impulse response (Couch 1983).

$$H(f) = \begin{cases} 1 & , |f| < f_1 \\ \frac{1}{2} \left[1 + \cos \frac{\pi(|f| - f_1)}{2f_\Delta} \right] & , f_1 < |f| < B \\ 0 & , |f| > B \end{cases} \quad (48)$$

where

f_0 = 6dB bandwidth of filter

B = absolute bandwidth

$f_\Delta = B - f_0$

$f_1 = f_0 - f_\Delta$

$$h(t) = 2f_0 \left[\frac{\sin(2\pi f_0 t)}{2\pi f_0 t} \right] \left[\frac{\cos(2\pi f_\Delta t)}{1 - (4f_\Delta t)^2} \right] \quad (49)$$

The receive program displays video packets in a standard VGA mode (320x200 resolution, 256 colors). Since only the luminance is received, the display is limited to 64 shades of gray. All packets with a length less than 700 bytes are considered to be video packets and are written to the first 65,536-byte video memory bank, which begins at address 0A000h. Packets longer than 700 bytes are deemed sound packets and are sent to the Sound Blaster Board, which in turn drives the speaker. (A 125-msec delay exists between the transmit and receive sound.) A triple buffer system is used to prevent sound and video packet overwrites. Video images can be saved to hard drive and displayed by pressing '2', '3', or '4'. 'Z' exits the program.

Table 10
Enhancement Goals

1	2-way conference
2	24-bit SVGA color (VESA standard)
3	Multicast network connectivity
4	Implementation of MPEG compression
5	Frame rate increase to 30 frames/sec
6	Lower required transmission rate

Future Improvements

Enhancement goals are listed in Table 10. Two-way conferencing will require both the transmit and receive programs to be modified into a common unit. A 24-bit SVGA card as well as code enhancements are needed to provide full display color. Multicast connectivity will allow multiple users to participate in a single conference.

Using a DSP chip for real time processing, the high-level compression and enhancement techniques developed at WES during FY92 and FY93 can be implemented; these techniques can be extended to three-dimensions to take advantage of video frame redundancies. Such data reduction will allow the frame rate to increase to perhaps 30 frames/second. Also, the required transmission rate should reduce.

12 IMAGE93 Software

Abstract

This chapter details the attributes of the IMAGE93 Software, engineered at WES ITL.

Introduction

During FY92, "Image Lab" was written to facilitate the construction of image-compression algorithms. As an extension, in FY93, IMAGE93 was created to preserve the capabilities of Image Lab and to exploit numerous combinations of image-enhancement algorithms.

Implementation

Figure 122 contains an outline of the IMAGE93 software. Executables end in *exe*, and source files end in either *bas* or *c*. All other files are ASCII storage files. Tables 11 and 12 map sources files with the corresponding appendixes (A, B, and D-AAA). Appendix CCC lists the specifications for VESA Graphics Interface.

OUTLINE OF IMAGE 93

Host file: image.exe (box.bas)

Overhead files:

- gray.pal
- color.pal
- vesa.ini
- vesal.exe (vesal.c)

FILE

Load Binary Image: display.exe (display.c)

Display BMP: display.exe (display.c)

Display PCX: pcxload.exe (pcxload.c)

Load Palette: (internal to image.exe)

Save Palette: (internal to image.exe)

Convert PCX to Binary: pcxtobin.exe (pcxtobin.c)

Print Image: printb.exe (printb.c)

EDIT

Chop Image: chop.exe (chop.c)

View Bit Plane(s): bitplane.exe (bitplane.c)

Show Red Palette: bitplane.exe (bitplane.c)

Show Green Palette: bitplane.exe (bitplane.c)

Show Blue Palette: bitplane.exe (bitplane.c)

Show as Grayscale: bitplane.exe (bitplane.c)

Subtract Files Byte by Byte: differ.exe (differ.c)

Add Files Byte by Byte: adder.exe (adder.c)

Examine an Image: lookone.exe (lookone.c)

Examine Two Images: looktwo.exe (looktwo.c)

Paste Top-to-Bottom: paste.exe (paste.c)

Paste Side-to-Side: paste.exe (paste.c)

Edit DFT File: editdft.exe (editdft.c)

Compress

Huffman: huff.exe (huff.c)

Adaptive Huffman: ahuff.exe (ahuff.c)

Arithmetic Order 0: arith.exe (arith.c)

Figure 122. Outline of IMAGE93 (Sheet 1 of 4)

Arithmetic Order 1:	arith1.exe (arith1.c)
Arithmetic Order 2:	arithn.exe (arithn.c)
LSW (SIP):	pksip.exe (no source available)
LSW (LEARC):	lharc.exe (no source available)
Cosine:	dct.exe (dct.c)
Sine:	dct.exe (dct.c)
Hadamard:	dct.exe (dct.c)

Decompress

Huffman:	unbuff.exe (unbuff.c)
Adaptive Huffman:	aunbuff.exe (aunbuff.c)
Arithmetic Order 0:	unarith.exe (unarith.c)
Arithmetic Order 1:	unarith1.exe (unarith1.c)
Arithmetic Order 2:	unarithn.exe (unarithn.c)
LSW (SIP):	pkunsip.exe (no source available)
LSW (LEARC):	lharc.exe (no source available)
Cosine:	dct.exe (dct.c)
Sine:	dct.exe (dct.c)

Enhancement

2-D Fourier:	dft.exe (dft.c)
	fft.exe (fft.c)
	invdft.exe (invdft.c)
	invfft.exe (invfft.c)
Singular Valued Decomposition:	svd.exe (svd.c)
	invsvd.exe (invsvd.c)
Detect Edges:	mask.exe (mask.c)

First Order:

Pixel Difference Row:	pdr
Pixel Difference Column:	pdc
Separated Difference Row:	sdr
Separated Difference Column:	sdc
Sobel Row:	sobelr
Sobel Column:	sobelc
Roberts Row:	robertsr
Roberts Column:	robertsc
Prewitt Row:	prewitr

Figure 122. (Sheet 2 of 4)

Prewitt Column:	prewittc
Frei-Chen Row:	fcr
Frei-Chen Column:	fcc
Nevatia-Rabu 0:	neva0
Nevatia-Rabu 90:	neva90
Second Order:	
Laplace 4 Neighbor:	laplace4
Prewitt 8 Neighbor:	prewitt8
Separable 8 Neighbor:	sep8
Noise Cleaning:	mask.exe (mask.c)
Noise Mask 1:	noise1
Noise Mask 2:	noise2
Noise Mask 3:	noise3
Crispen Image:	mask.exe (mask.c)
Crispening Mask 1:	crisp1
Crispening Mask 2:	crisp2
Crispening Mask 3:	crisp3
Crispening Mask 4:	crisp4
Crispening Mask 5:	crisp5
Contrast Manipulation:	
Extract Image:	extract.exe (extract.c)
Square Normalization:	squareh.exe (squareh.c)
Cube Normalization:	cubeh.exe (cubeh.c)
Square Root Normalization:	sqrth.exe (sqrth.c)
Cube Root Normalization:	cuberth.exe (cuberth.c)
Inverse Normalization:	invh.exe (invh.c)
Invert:	revh.exe (revh.c)
Morphological:	
Dilating:	dil.exe (dil.c)
Eroding:	erode.exe (erode.c)
Opening:	open.exe (open.c)
Closing:	close.exe (close.c)
Median Filter:	median.exe (median.c)

Figure 122. (Sheet 3 of 4)

Image Thresholding:

By Occurrence:

thresbyo.exe (thresbyo.c)

By Value:

thresbyv.exe (thresbyv.c)

Tools

Calculate Entropy:

entropy.exe (entropy.c)

Mean-Square Error:

mse.exe (mse.c)

Histogram:

histo.exe (histo.c)

Load Macros:

(part of host), macro.lst

Figure 122. (Sheet 4 of 4)

Table 11
Location of Source Files According to Appendix

Appendix A	svd.c	Appendix DD	pcxload.c
Appendix B	invsvd.c	Appendix EE	pcxtobin.c
Appendix D	dft.c	Appendix FF	printb.c
Appendix E	invdft.c	Appendix GG	chop.c
Appendix F	fft.c	Appendix HH	bitplane.c
Appendix G	invfft.c	Appendix II	differ.c
Appendix H	editdft.c	Appendix JJ	adder.c
Appendix I	dil.c	Appendix KK	lookone.c
Appendix J	erode.c	Appendix LL	looktwo.c
Appendix K	close.c	Appendix MM	paste.c
Appendix L	open.c	Appendix NN	huff.c
Appendix M	mask.c	Appendix OO	ahuff.c
Appendix N	thresbyv.c	Appendix PP	arith.c
Appendix O	thresbyo.c	Appendix QQ	arithl.c
Appendix P	squareh.c	Appendix RR	arithn.c
Appendix Q	cubeh.c	Appendix SS	unhuff.c
Appendix R	sqth.c	Appendix TT	aunhuff.c
Appendix S	cuberth.c	Appendix UU	unarith.c
Appendix T	invh.c	Appendix VV	unarithl.c
Appendix U	revh.c	Appendix WW	unarithn.c
Appendix V	extract.c	Appendix XX	dct.c
Appendix W	median.c	Appendix YY	entropy.c
Appendix AA	box.bas	Appendix ZZ	mse.c
Appendix BB	veasl.c	Appendix AAA	histo.c
Appendix CC	display.c		

Table 12
Location of Source Files According to Filename

adder.c	Appendix JJ	invfft.c	Appendix G
ahuff.c	Appendix OO	invh.c	Appendix T
arith.c	Appendix PP	invsvd.c	Appendix B
arithl.c	Appendix QQ	lookone.c	Appendix KK
arithn.c	Appendix RR	looktwo.c	Appendix LL
aunhuff.c	Appendix TT	mask.c	Appendix M
bitplane.c	Appendix HH	median.c	Appendix W
box.bas	Appendix AA	mse.c	Appendix ZZ
chop.c	Appendix GG	open.c	Appendix L
close.c	Appendix K	paste.c	Appendix MM
cubeh.c	Appendix Q	pcxload.c	Appendix DD
cuberth.c	Appendix S	pcxtobin.c	Appendix EE
dct.c	Appendix XX	printb.c	Appendix FF
dft.c	Appendix D	revh.c	Appendix U
differ.c	Appendix II	squareh.c	Appendix P
dil.c	Appendix I	sqtrh.c	Appendix R
ldisplay.c	Appendix CC	svd.c	Appendix A
editdft.c	Appendix H	thresbyv.c	Appendix N
entropy.c	Appendix YY	thresbyo.c	Appendix O
erode.c	Appendix J	unarith.c	Appendix UU
extract.c	Appendix V	unarithl.c	Appendix VV
fft.c	Appendix F	unarithn.c	Appendix WW
histo.c	Appendix AAA	unhuff.c	Appendix SS
huff.c	Appendix NN	vesal.c	Appendix BB
invdft.c	Appendix E		

13 The Clear Speech Algorithm

A clear speech algorithm was developed around sigma-delta technology to provide a means of using the PC speaker to reproduce intelligible speech and music. This problem results from the fact that the PC speaker can be driven only from binary high and low pulses. The sigma-delta modulation technique has been recently used in the electronics industry for high resolution analog-to-digital converters and can also help improve the quality of sound from a PC speaker on an IBM compatible machine.

A model of the sigma-delta modulator using two digital integrator stages is shown below. For the PC application, the input voltage, V_{in} , is 8-bit per byte digitized sound which is either played back from a hard disk, or received over the network and pumped into the sigma-delta modulator. V_5 is the output voltage of the sigma-delta modulator and is either a binary low or high. The other elements shown in Figure 123 are summers and delays. The following explanation will assume an extensive familiarity with network theory.

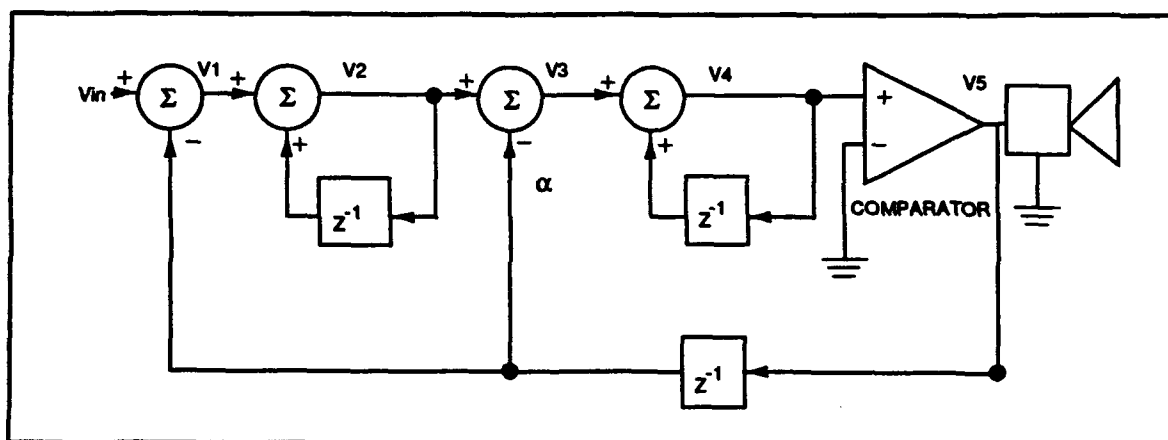


Figure 123. Sigma-delta modulator

By using the Fourier transform, it can be shown that the binary output voltage, V_5 , has the same exact spectral makeup as the input voltage, V_{in} , except for the addition of some high frequency spectral noise. In practice,

the sampling rate on a PC should be chosen so that this quantization noise is pushed above the audible region. In this case, the perceived audio sound from the PC speaker will be a high-fidelity reproduction of the original. For analysis, the equations that describe the operation of the sigma-delta modulator in Figure 123 can be written as shown in Table 13. By simulating and collecting hundreds of samples of the output voltage $V5$, a Fourier transform analysis can be performed to examine the spectral purity of the output signal $V5$ driving the PC speaker. It is a given that the input signal is a pure sine wave at 5,000-Hz frequency. Figure 124 shows typical sigma-delta spectral output and can be reproduced by using the Fourier transform equations given by

Table 13
Sigma-Delta Modulator Operations

Operation	Equation
V_{in}	$\sin(2\pi \cdot 3.14159 \cdot 5000 \cdot i / f_s)$, $i = 0, 1, \dots, 4999$; $f_s = 5000$
$V1$	$V_{in} - V5$
$V2$	$V2 + V1$
$V3$	$V2 - V5$
$V4$	$V3 + V4$
$V5$	2 if $V4$ is greater than or equal to 0. -2 if $V4$ is less than 0.

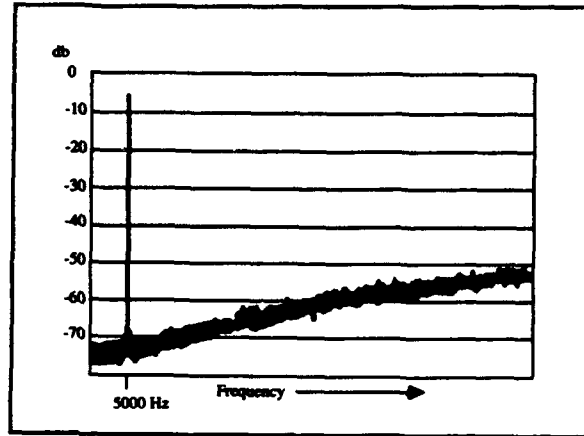


Figure 124. Typical $\Sigma\text{-}\Delta$ spectral output at location $V5$ before lowpass filtering, for a single sine wave input of 5,000 Hz

$$m\text{th harmonic energy in db} = 10 \log_{10} (am^2 + bm^2)$$

where

$$am = \frac{1}{\pi m} \sum_{i=1}^N V5(i) \left\{ \sin\left(\frac{2\pi m i}{N}\right) - \sin\left[\frac{2\pi m (i-1)}{N}\right] \right\} \quad (50)$$

$$bm = -\frac{1}{\pi m} \sum_{i=1}^N V5(i) \left\{ \cos\left(\frac{2\pi m i}{N}\right) - \cos\left[\frac{2\pi m (i-1)}{N}\right] \right\}$$

When tested on an IBM-compatible 486/33Mhz PC, the sampling rate was allowed to run as fast as possible and was timed at 200,000 samples per second. The equations in Table 13 were executed 200,000 times each second. The sound reproduction was highly intelligible but, as expected, the PC speaker volume was low. As part of an actual application, the clear speech algorithm might also be handicapped by the need for the microprocessor to perform other tasks as well. The significance of this technique is that any networked PC with a VGA monitor can be set up to receive real-time video and audio using software only, since the VGA monitor can be written in real time to update the video image, and the built-in PC speaker can be made to reproduce intelligible sound. The addition of a dedicated sound board, such as a sound blaster, in the PC would, of course, offset the sound processing and provide volume control. The addition of a video capture card in the same PC would also allow that PC to originate video and audio transmission over the network for full participation in a video-conference session.

The demonstration program shown in Figure 125 reads a sound blaster .VOC file and plays it using the clear-speech algorithm.

```
#include <stdio.h>

char d[400000];          /*Up to 25 seconds of sound*/
FILE *infile;

main()
{
    int b,e,h,v0=c,v1,v2=0,v3,v4=0;
    long int j,k,n=0;
    unsigned char cp;

    infile=fopen("TEMP.VOC","r+b");    /*Open Sound Blaster VOC file*/
    for (j=1;j<=66;j++)                /*Skip header info*/
    {
        fscanf(infile,"%c",&cp);
    }
    while (!feof(infile))
    {
        fscanf(infile,"%c",&cp);
        n++;
        d[n] = cp - 128;                /*Sound files should be*/
                                        /*Sampled at 8000 bytes per second*/
                                        /*Subtract 128 to convert VOC files*/
    }
    b = inp(0x61);                      /*Get Byte at location 61*/
    b = b | 2;                          /*Set Sound bit high*/
    e = b & 0xFD;                       /*Set Sound bit low*/
    for (j=0;j<=n;j++)                  /*Start of Sigma-Delta Loop*/
    {
        for (k=1;k<=48;k++)             /*Max value of K may vary*/
        {
            v1 = d[j] - v0;
            v2 = v1 + v2;
            v3 = v2 - v0;
            v4 = v3 + v4;
            if (v4>0)
            {
                v0 = -150;
                outp(0x61,b);            /*150 is twice highest expected*/
            }
            else
            {
                v0 = 150;
                outp(0x61,e);
            }
        }
    }
    fcloseall();
}
```

Figure 125. Demonstration program using the clear-speech algorithm

References

- Catenary Systems. (1992). *Victor image processing library manual*. Catenary Systems, St. Louis.
- Couch, L. W. (1993). *Digital and analog communications systems*. Macmillan, New York.
- Ellis, M. G. (1992). "An image processing technique for achieving lossy compression of data at ratios in excess of 100:1," Technical Report ITL-92-10, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.
- Grob, B. (1975). *Basic television*. 4th ed., McGraw-Hill, New York.
- Huang, T. S. (1979). *Picture processing and digital filtering*. 2nd ed., Springer-Verlag, Berlin, Heidelberg, Germany.
- Jain, A. K. (1989). *Fundamentals of digital image processing*. Prentice-Hall, Englewood Cliffs, NJ.
- Pratt, W. K. (1991). *Digital image processing*. 2nd ed., John Wiley & Sons, New York.
- Press, W. H., et al. (1988). *Numerical recipes in C*. Cambridge, New York.
- Rabiner, L. R., and Gold, B. (1975). *Theory and application of digital signal processing*. Prentice-Hall, Englewood Cliffs, NJ.
- Scharf, L. L. (1991). "The SVD and reduced-rank signal processing."
- Ziemer, R. E., Tranter, W. H., and Fannin, D. R. (1989). *Signals and systems: Continuous and discrete*. 2nd ed., Macmillan, New York.

Appendix A

The SVD Processor

The SVD Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\malloc.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,sqrt_eigenvals,V,result,mode,width,height;
int ourseq,WIDTH1,HEIGHT1,height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile;

static float at,bt,ct;
#define PYTHAG(a,b) ((at=fabs(a)) > (bt=fabs(b)) ? \
(ct=bt/at,at*sqrt(1.0+ct*ct)) : (bt ? (ct=at/bt,bt*sqrt(1.0+ct*ct)) : 0.0))

static float maxarg1,maxarg2;
#define MAX(a,b) (maxarg1=(a),maxarg2=(b), (maxarg1) > (maxarg2) ? \
(maxarg1) : (maxarg2))
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
/*Globals end*/

/*Error handling routine*/
void nrerror(error_text)
char error_text[];
{
    void _exit();

    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    xfree(image);
    xfree(sqrt_eigenvals);
    xfree(V);
    xfree(result);
    fclose(infile);
    _exit(1);
}

/*Dynamic memory allocation*/
float *vector(nl,nh)
int nl,nh;
/* allocate a float vector with subscript range v[nl..nh] */
{
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float))-nl;
    if (!v) nrerror("allocation failure in vector()");
    return v;
}

/*Dynamic memory deallocation*/
void free_vector(v,nl,nh)
float *v;
int nl,nh;
/* free a float vector allocated with vector() */
{
    free((char*) (v+nl));
}
```

```

/*Extended memory data retrieval*/
float dget(handle,index_x,index_y)
int handle,index_x,index_y;
{
float data;
int test;

test=xmgetrow(&data,handle,(index_x-1)*4,(index_y-1),4,2880);
if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
int handle,index_x,index_y;
float data;
{
float dummy;
int test;

dummy = data;
test=xmputrow(&dummy,handle,(index_x-1)*4,(index_y-1),4,2880);
if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
getchar();}
}

/*Modified Singular Value Decomposition Algorithm*/
void svdcmp(a,m,n,w,v)
int a,w,v,m,n;
{
int flag,i,its,j,jj,k,l,nm;
float c,f,h,s,x,y,z,data,datal;
float anorm=0.0,g=0.0,scale=0.0;
float *rvl,*vector();
void nrerror(),free_vector();

if (m < n) nrerror("SVDcmp: You must augment A with extra zero rows");
rvl=vector(1,n);
for (i=1;i<=n;i++) {
l=i+1;
rvl[i]=scale*g;
g=s=scale=0.0;
if (i <= m) {
for (k=i;k<=m;k++) scale += fabs(dget(a,k,i));
if (scale) {
for (k=i;k<=m;k++) {
data = dget(a,k,i)/scale;
dput(a,k,i,data);
s += dget(a,k,i)*dget(a,k,i);
}

f=dget(a,i,i);
g = -SIGN(sqrt(s),f);
h=f*g-s;
data=f-g;
dput(a,i,i,data);
if (i != n) {
for (j=1;j<=n;j++) {
for (s=0.0,k=i;k<=m;k++)
{
s += dget(a,k,i)*dget(a,k,j);
}
f=s/h;
}
}
}
}
}
}

```



```

        for (k=1;k<=m;k++)
        {
            data = dget(a,k,j)+f*dget(a,k,i);
            dput(a,k,j,data);
        }
    }
    for (k=1;k<=m;k++)
    {
        data = dget(a,k,i) * scale;
        dput(a,k,i,data);
    }
}
data = scale*g;
dput(w,i,1,data);
g=s*scale=0.0;
if (i <= m && i != n) {
    for (k=1;k<=n;k++) scale += fabs(dget(a,i,k));
    if (scale) {
        for (k=1;k<=n;k++) {
            data = dget(a,i,k)/scale;
            dput(a,i,k,data);
            s += dget(a,i,k)*dget(a,i,k);
        }
        f=dget(a,i,1);
        g = -SIGN(sqrt(s), f);
        h=f*g-s;
        data=f-g;
        dput(a,i,1,data);
        for (k=1;k<=n;k++) rv1[k]=dget(a,i,k)/h;
        if (i != m) {
            for (j=1;j<=m;j++) {
                for (s=0.0,k=1;k<=n;k++) s += dget(a,j,k)*dget(a,i,k);
                for (k=1;k<=n;k++)
                {
                    data = dget(a,j,k) + s*rv1[k];
                    dput(a,j,k,data);
                }
            }
        }
        for (k=1;k<=n;k++)
        {
            data = dget(a,i,k)*scale;
            dput(a,i,k,data);
        }
    }
    anorm=MAX(anorm, (fabs(dget(w,i,1))+fabs(rv1[i])));
}
for (i=n;i>=1;i--) {
    if (i < n) {
        if (g) {
            for (j=1;j<=n;j++)
            {
                datal = (dget(a,i,j)/dget(a,i,1))/g;
                dput(v,j,i,datal);
            }
            for (j=1;j<=n;j++) {
                for (s=0.0,k=1;k<=n;k++) s += dget(a,i,k)*dget(v,k,j);
                for (k=1;k<=n;k++)
                {
                    datal = dget(v,k,j) + s*dget(v,k,i);
                    dput(v,k,j,datal);
                }
            }
        }
    }
}

```

```

        }
        for (j=1; j<=n; j++)
        {
            datal = 0.0;
            dput(v, i, j, datal);
            dput(v, j, i, datal);
        }
    }
    datal=1.0;
    dput(v, i, i, datal);
    g=rvl[i];
    l=i;
}
for (i=n; i>=1; i--) {
    l=i+1;
    g=dget(w, i, l);
    if (i < n)
        for (j=1; j<=n; j++)
        {
            data = 0.0;
            dput(a, i, j, data);
        }
    if (g) {
        g=1.0/g;
        if (i != n) {
            for (j=1; j<=n; j++) {
                for (s=0.0, k=1; k<=m; k++) s += dget(a, k, i)*dget(a, k, j);
                f=(s/dget(a, i, i))*g;
                for (k=1; k<=m; k++)
                {
                    data = dget(a, k, j) + f*dget(a, k, i);
                    dput(a, k, j, data);
                }
            }
        }
        for (j=i; j<=m; j++)
        {
            data = dget(a, j, i)*g;
            dput(a, j, i, data);
        }
    } else {
        for (j=i; j<=m; j++)
        {
            data = 0.0;
            dput(a, j, i, data);
        }
    }

    data = dget(a, i, i) + 1.0;
    dput(a, i, i, data);
}
for (k=n; k>=1; k--) {
    for (its=1; its<=100; its++) {
        flag=1;
        for (l=k; l>=1; l--) {
            nm=l-1;
            if ((float)(fabs(rvl[l])+anorm) == anorm) {
                flag=0;
                break;
            }
            if ((float)(fabs(dget(w, nm, l))+anorm) == anorm) break;
        }
        if (flag) {
            c=0.0;

```

```

s=1.0;
for (i=1;i<=k;i++) {
    f=s*rvl[i];
    rvl[i]=c*rvl[i];
    if ((float)(fabs(f)+anorm) == anorm) break;
    g=dget(w,i,1);
    h=PYTHAG(f,g);
    data=h;
    dput(w,i,1,data);
    h=1.0/h;
    c=g*h;
    s=(-f*h);
    for (j=1;j<=n;j++) {
        y=dget(a,j,nm);
        z=dget(a,j,1);
        data = y*c+z*s;
        dput(a,j,nm,data);
        data = z*c-y*s;
        dput(a,j,1,data);
    }
}
z=dget(w,k,1);
if (1 == k) {
    if (z < 0.0) {
        data = -z;
        dput(w,k,1,data);
        for (j=1;j<=n;j++) {
            {
                datal = (-dget(v,j,k));
                dput(v,j,k,datal);
            }
        }
        break;
    }
}
if (its == 100) nrerror("No convergence in 100 SVDcmp iterations");
x=dget(w,1,1);
nm=k-1;
y=dget(w,nm,1);
g=rvl[nm];
h=rvl[k];
f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
g=PYTHAG(f,1.0);
f=((x-z)*(x+z)+h*((y/(f+SIGN(g,f)))-h))/x;
c=s=1.0;
for (j=1;j<=nm;j++) {
    i=j+1;
    g=rvl[i];
    y=dget(w,i,1);
    h=s*g;
    g=c*g;
    z=PYTHAG(f,h);
    rvl[j]=z;
    c=f/z;
    s=h/z;
    f=x*c+g*s;
    g=g*c-x*s;
    h=y*s;
    y=y*c;

    for (jj=1;jj<=n;jj++) {
        x=dget(v,jj,j);
        z=dget(v,jj,1);
        datal = x*c+z*s;
        dput(v,jj,j,datal);
    }
}

```

```

        data1 = z*c-x*s;
        dput(v,jj,1,data1);
    }
    z=PYTHAG(f,h);
    data=z;
    dput(w,j,1,data);
    if (z) {
        z=1.0/z;
        c=f*z;
        s=h*z;
    }
    f=(c*g)+(s*y);
    x=(c*y)-(s*g);
    for (jj=1;jj<=m;jj++) {
        y=dget(a,jj,j);
        z=dget(a,jj,1);
        data = y*c+z*s;
        dput(a,jj,j,data);
        data = z*c-y*s;
        dput(a,jj,1,data);
    }
    }
    rv1[1]=0.0;
    rv1[k]=f;
    data=x;
    dput(w,k,1,data);
}
}
free_vector(rv1,1,n);
}

```

```

#undef SIGN
#undef MAX
#undef PYTHAG

```

/*Video mode control*/

```

void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
    }
}

```

```

    _asm int 10h;
    WIDTH1=1024;
    HEIGHT1=768;
}

)

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    int i,j,jj,ii,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    FILE *uprime;
    FILE *vmatrix;
    /*Read in image info*/
    infile = fopen("input","r");
    fscanf(infile,"%d %d %d",&mode,&width,&height);
    fscanf(infile,"%s",string);

```

```

fclose(infile);
if (width>720 && height>720)
{
    printf ("Image is too large.");
    exit(1);
}

/*Read in image and initialize extended memory*/
infile = fopen(string,"r+b");
xmallocate(&image,2048);
xmallocate(&sqrt_eigenvals,3);
xmallocate(&V,2048);
xmallocate(&result,2048);
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fscanf(infile,"%c", &dummy);
        data = (float) (dummy);
        xputrow(&data,image,(j-1)*4,(i-1),4,2880);
    }
}

/*Process image*/
svdcmp(image,height,width,sqrt_eigenvals,V);

/*Save U prime and V matrices*/
uprime=fopen("uprime.out","w");
vmatrix=fopen("v.out","w");
for (i=width+1;i<=height;i++)
{
    data=0.0;
    dput(sqrt_eigenvals,i,1,data);
}
for (j=1;j<=height;j++)
{
    for (i=1;i<=width;i++)
    {
        data=dget(image,j,i)*dget(sqrt_eigenvals,i,1);
        dput(image,j,i,data);
        fprintf(uprime,"%f ",data);
    }
}
for (j=1;j<=width;j++)
{
    for (jj=1;jj<=height;jj++)
    {
        data=dget(V,j,jj);
        fprintf(vmatrix,"%f ",data);
    }
}
fclose(uprime);
fclose(vmatrix);

printf("READY - PRESS A KEY"); getchar();

/*Reconstruct image*/
//Set display mode
Set_Mode();

//Setup palette
Set_Palette();

//Setup VGA memory

```

```

Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Reconstruct and display image with bank switching
if (height>HEIGHT1)
    height=HEIGHT1;
else
    height=height;
for (ii=1; ii<=width; ii++)
{
    for (j=1; j<=height; j++)
    {
        for (jj=1; jj<=width; jj++)
        {
            data=dget(image, j, ii)*dget(V, jj, ii);
            if (ii != 1) data=data+dget(result, j, jj);
            dput(result, j, jj, data);
            pixel = (unsigned int) data;
            x=jj-1;
            y=j-1;
            pix_no= (float) y*WIDTH1+x+1;
            new_bank=pix_no/65536;
            if (new_bank!=bank)
            {
                bank=new_bank;
                Set_Bank(bank);
            }
            position=pix_no-new_bank*65536;
            Display_Pixel(pixel, position);
        }
    }
}

getchar();

//Reset text mode
_asm mov ax, 3;
_asm int 10h;

/*Free extended memory and close files*/
xfree(image);
xfree(sqrt_eigenvals);
xfree(V);
xfree(result);
fclose(infile);
}
/*Main program ends*/

```

Appendix B

The Image Factory

The Image Factory

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\malloc.h>
#include <c:\c700\include\videfs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image, sqrt_eigenvals, V, result, mode, width, height;
int ourseg, WIDTH1, HEIGHT1, _height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile;
FILE *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle, index_x, index_y)
    int handle, index_x, index_y;
{
    float data;
    int test;

    test=xmgetrow(&data, handle, (index_x-1)*4, (index_y-1), 4, 2880);
    if (test!=NO_ERROR) {printf("dget %i, %i, %i", handle, index_x, index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle, index_x, index_y, data)
    int handle, index_x, index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy, handle, (index_x-1)*4, (index_y-1), 4, 2880);
    if (test!=NO_ERROR) {printf("dput %i, %i, %i", handle, index_x, index_y);
getchar();}
}

/*Vide mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax, 13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax, 4f02h;
        _asm mov bx, 101h;
        _asm int 10h;
        WIDTH1=640;
    }
}
```

```

        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{

```

```

float data;
float pix_no;
int i, j, jj, ii, x, y, set_no, add_rem, save_continue;
unsigned int bank=0, pixel, new_bank, position;
char string[15];

FILE *uprime;
FILE *vmatrix;

/*Read in image info*/
infile = fopen("input", "r");
fscanf(infile, "%d %d %d", &mode, &width, &height);
fclose(infile);
if (width>720 && height>720)
{
    printf ("Image is too large.");
    exit(1);
}

/*Read in image and initialize extended memory*/
xmallocate(&image, 2048);
xmallocate(&V, 2048);
xmallocate(&result, 2048);

/*Retrieve U prime and V matrices*/
uprime=fopen("uprime.out", "r");
vmatrix=fopen("v.out", "r");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fscanf(uprime, "%f", &data);
        dput(image, j, i, data);
    }
}
for (j=1; j<=width; j++)
{
    for (jj=1; jj<=height; jj++)
    {
        fscanf(vmatrix, "%f", &data);
        dput(V, j, jj, data);
    }
}
fclose(uprime);
fclose(vmatrix);

printf("READY - PRESS A KEY"); getchar();

/*Reconstruct image*/
//Set display mode
Set_Mode();

//Setup palette
Set_Palette();

//Setup VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Reconstruct and display image with bank switching
if (height>HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (ii=1; ii<=width; ii++)

```

```

    {
        for (j=1;j<=_height;j++)
        {
            for (jj=1;jj<=width;jj++)
            {
                data=dget(image,j,ii)*dget(V,jj,ii);
                if (ii != 1) data=data+dget(result,j,jj);
                dput(result,j,jj,data);
                pixel = (unsigned int) data;
                x=jj-1;
                y=j-1;
                pix_no= (float) y*WIDTH1+x+1;
                new_bank=(pix_no)/65536;
                if (new_bank!=bank)
                {
                    bank=new_bank;
                    Set_Bank(bank);
                }
                position= pix_no-new_bank*65536;
                Display_Pixel(pixel,position);
            }
        }
    }

    getchar();

    /*Allow eigen-set removal*/
    while(1)
    {
        //Reset text mode
        _asm mov ax,3;
        _asm int 10h;

        //Offer to save image
        printf("(0)-Save Current File (1)-Continue");
        printf("\n?: ");
        scanf("%i",&save_continue);
        getchar();
        if (!save_continue)
        {
            printf("\nEnter filename: ");
            scanf("%s",string);
            getchar();
            outfile=fopen(string,"w+b");
            for (j=1;j<=_height;j++)
            {
                for (jj=1;jj<=width;jj++)
                {
                    data=dget(result,j,jj);
                    pixel = (unsigned int) data;
                    fprintf(outfile,"%c",pixel);
                }
            }
            fclose(outfile);
        }

        //Obtain set number and give choice of adding or removing
        printf("\nEnter eigen-set number.");
        printf("\n1<=x<=41 quit=999",width);
        printf("\nx: ");
        scanf("%i",&set_no);
        getchar();
        if (set_no==999) break;
    }

```

```

printf("\n(0)-Add (1)-Remove");
printf("\n?: ");
scanf("%i", &add_rem);
getchar();

/*Reconstruct image*/
//Set display mode
Set_Mode();

//Setup palette
Set_Palette();

//Setup VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Reconstruct and display image with bank switching
for (ii=set_no; ii<=set_no; ii++)
{
    for (j=1; j<=_height; j++)
    {
        for (jj=1; jj<=width; jj++)
        {
            data=dget(image, j, ii)*dget(V, jj, ii);
            if (add_rem) data=dget(result, j, jj)-data;
            else data=dget(result, j, jj)+data;
            dput(result, j, jj, data);
            pixel = (unsigned int) data;
            x=jj-1;
            y=j-1;
            pix_no= (float) y*WIDTH1+x+1;
            new_bank=(pix_no)/65536;
            if (new_bank!=bank)
            {
                bank=new_bank;
                Set_Bank(bank);
            }
            position= pix_no-new_bank*65536;
            Display_Pixel(pixel, position);
        }
    }
}

getchar();
}

/*Free extended memory and close files*/
xmfree(image);
xmfree(V);
xmfree(result);
}
/*Main program ends*/

```

Appendix C

Vision

Vision

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\malloc.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int mode,width,height,WIDTH1,HEIGHT1,ourseg;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile;
/*Globals end*/

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Load Palette*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
```

```

        palbuf[i][0]=1/4;
        palbuf[i][1]=1/4;
        palbuf[i][2]=1/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,jj,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    unsigned char data;
    char string[15];

    /*Read in image info*/
    printf("Enter filename:      ");
    scanf("%s",string);
    getchar();
    infile=fopen(string,"r+b");
    printf("Enter mode:          ");
    scanf("%d",&mode);
    getchar();
    printf("Enter width,height:  ");
    scanf("%d,%d",&width,&height);
    getchar();

    //Set display mode
    Set_Mode();

    //Setup palette
    Set_Palette();

    //Setup VGA memory
    Set_Bank(bank);

```



```

_asm mov [ourseg], 0A000h;

//Reconstruct and display image with bank switching
for (j=1; j<=height; j++)
{
    for (jj=1; jj<=width; jj++)
    {
        fscanf(infile, "%c", &data);
        pixel = (unsigned int) data;
        x=jj-1;
        y=j-1;
        pix_no = (float) y*WIDTH1+x+1;
        new_bank=(pix_no)/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position= pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}
getchar();

//Reset text mode
_asm mov ax, 3;
_asm int 10h;

fclose(infile);
}
/*Main program ends*/

```

Appendix D

The DFT Processor

The DFT Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image, real, imaginary, mode, width, height;
int ourseg, WIDTH1, HEIGHT1, _height;
unsigned char palbuf[256][3];
float biasi;
void __far *p;

FILE *infile, *real_outfile, *imaginary_outfile;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle, index_x, index_y)
    int handle, index_x, index_y;
{
    float data;
    int test;

    test=xmgetrow(&data, handle, (index_x-1)*4, (index_y-1), 4, 2880);
    if (test!=NO_ERROR) {printf("dget %i, %i, %i", handle, index_x, index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle, index_x, index_y, data)
    int handle, index_x, index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy, handle, (index_x-1)*4, (index_y-1), 4, 2880);
    if (test!=NO_ERROR) {printf("dput %i, %i, %i", handle, index_x, index_y);
getchar();}
}

/*Discrete Fourier Transform*/
void dft(image, height, width, real, imaginary)
    int image, height, width, real, imaginary;
{
    int m, n, l, k;
    float mag, phase, tot_real, tot_imaginary;
    float sav_real, sav_imaginary;
    float fwidth, fheight;

    fwidth = (float) width;
    fheight = (float) height;

    for (k=0; k<height; k++)
    {
        for (l=0; l<width; l++)
        {
            sav_real=sav_imaginary=0;
            for (m=0; m<height; m++)
```

```

    {
        tot_real=tot_imaginary=0;
        for (n=0;n<width;n++)
        {
            mag=dget(image,m+1,n+1);
            phase= (float) -2.0*3.1415926535/fwidth*1*n;
            tot_real+=mag*cos(phase);
            tot_imaginary+=mag*sin(phase);
        }
        phase= (float) -2.0*3.1415926535/fheight*k*m;

        sav_real+=tot_real*cos(phase)-tot_imaginary*sin(phase);
        sav_imaginary+=tot_imaginary*cos(phase)+tot_real*sin(phase);
    }
    if (sav_imaginary>biasi) biasi=sav_imaginary;
    dput(real,k+1,l+1,sav_real);
    dput(imaginary,k+1,l+1,sav_imaginary);
}
printf(".");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {

```

```

    palbuf[i][0]=1/4;
    palbuf[i][1]=1/4;
    palbuf[i][2]=1/4;
}
j=256;
k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    float bias;
    int i,j,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("input","r");
    fscanf(infile, "%d %d %d", &mode, &width, &height);
    fscanf(infile, "%s", string);
    fclose(infile);
    /*Read in image and initialize extended memory*/
    infile=fopen(string,"r+b");
    xmallocate(&image,2048);
    xmallocate(&real,2048);
    xmallocate(&imaginary,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile,"%c",&dummy);
            data = (float) dummy;
            xputrow(&data,image,(j-1)*4,(i-1),4,2880);
        }
    }
}

```

```

    }

    /*Process image*/
    dft(image,height,width,real,imaginary);

    /*Save DFT results to hard disk*/
    real_outfile=fopen("real.out","w");
    imaginary_outfile=fopen("imag.out","w");
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            data = dget(real,j,i);
            fprintf(real_outfile,"%f ",data);
            data = dget(imaginary,j,i);
            fprintf(imaginary_outfile,"%f ",data);
        }
    }

    /*Display real and imaginary planes*/
    //Set display mode
    Set_Mode();

    //Set up palette
    Set_Palette();

    //Set-up VGA memory
    Set_Bank(bank);
    _asm mov [ourseg],0A000h;

    //Display real plane
    bias=dget(real,1,1);
    if (height> HEIGHT1)
        _height=HEIGHT1;
    else
        _height=height;
    for (j=1;j<=_height;j++)
    {
        for (i=1;i<=width;i++)
        {
            pixel = (unsigned int) (dget(real,j,i)*128.0/bias+127.0);
            x=i-1;
            y=j-1;
            pix_no = (float) y*WIDTH1+x+1;
            new_bank=pix_no/65536;
            if (new_bank!=bank)
            {
                bank=new_bank;
                Set_Bank(bank);
            }
            position=pix_no-new_bank*65536;
            Display_Pixel(pixel,position);
        }
    }

    getchar();

    //Display imaginary plane
    for (j=1;j<=_height;j++)
    {
        for (i=1;i<=width;i++)
        {
            pixel = (unsigned int) (dget(imaginary,j,i)*128.0/bias+127.0);
            x=i-1;
            y=j-1;

```

```

    pix_no= (float) y*WIDTH1+x+1;
    new_bank=pix_no/65536;
    if (new_bank!=bank)
    {
        bank=new_bank;
        Set_Bank(bank);
    }
    position=pix_no-new_bank*65536;
    Display_Pixel(pixel,position);
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(real);
xmfree(imaginary);
}
/*Main program ends*/

```

Appendix E

The Inverse DFT Processor

The Inverse DFT Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,real,imaginary,mode,width,height;
int ourseg,WIDTH1,HEIGHT1,height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile1, *infile2, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    float data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

/*Inverse Discrete Fourier Transform*/
void invdft(image,height,width,real,imaginary)
    int image,height,width,real,imaginary;
{
    int m,n,l,k;
    float mag,phase,tot_real,tot_imaginary;
    float sav_real,sav_imaginary;
    float new_real,new_imaginary;
    float fwidth, fheight;

    fwidth = (float) width;
    fheight = (float) height;

    for (m=0;m<height;m++)
    {
        for (n=0;n<width;n++)
        {
            sav_real=sav_imaginary=0;
            for (k=0;k<height;k++)
```

```

    {
        tot_real=tot_imaginary=0;
        for (l=0;l<width;l++)
        {
            new_real=dget(real,k+1,l+1);
            new_imaginary=dget(imaginary,k+1,l+1);
            phase= (float) 2.0*3.1415926535/fwidth*l*n;
            tot_real+=new_real*cos(phase)-new_imaginary*sin(phase);
            tot_imaginary+=new_imaginary*cos(phase)+new_real*sin(phase);
        }
        phase= (float) 2.0*3.1415926535/fheight*k*m;
        sav_real+=tot_real*cos(phase)-tot_imaginary*sin(phase);
        sav_imaginary+=tot_imaginary*cos(phase)+tot_real*sin(phase);
    }
    mag=sqrt(sav_real*sav_real+sav_imaginary*sav_imaginary);
    mag/= (float) (width*height);
    dput(image,m+1,n+1,mag);
}
printf(".");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {

```

```

        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    int i,j,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    /*Read in image info*/
    infile1=fopen("input","r");
    fscanf(infile1, "%d %d %d", &mode, &width, &height);
    fscanf(infile1, "%s", string);
    fclose(infile1);

    /*Read in real and imaginary matrices and initialize extended memory*/
    infile1=fopen("real.out","r");
    infile2=fopen("imag.out","r");
    xmallocate(&image,2048);
    xmallocate(&real,2048);
    xmallocate(&imaginary,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile1,"%f ",&data);
            xputrow(&data,real,(j-1)*4,(i-1),4,2880);
            fscanf (infile2,"%f ",&data);

```

```

        xmputrow(&data, imaginary, (j-1)*4, (i-1), 4, 2880);
    }
}

/*Process image*/
invdft(image, height, width, real, imaginary);

/*Save DFT results to hard disk*/
outfile=fopen("image.out", "w+b");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data = dget(image, j, i);
        dummy = (unsigned char) data;
        fprintf(outfile, "%c", dummy);
    }
}

/*Display image*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Display image
if (height > HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(image, j, i));
        x=i-1;
        y=j-1;
        pix_no = (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}

getchar();

//Reset text mode
_asm mov ax, 3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(real);
xmfree(imaginary);

```

```
}  
/*Main program ends*/
```

Appendix F

The FFT Processor

The FFT Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,real,imaginary,mode,width,height;
int ourseg,WIDTH1,HEIGHT1,height;
unsigned char palbuf[256][3];
float biasi=0.0;
void __far *p;

FILE *infile, *real_outfile, *imaginary_outfile;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    float data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
getchar();}
}

float a[2][1025];

/*1-D Fast Fourier Transform*/
void fft_ld()
{
    int n,nv2,nml,i,j,k,l,le,ip,m;
    float t[2],pi,u[2],w[2],lel,temp;

    n=width;
    m=log10((float) n)/log10(2.0)+.5;
    nv2=n/2;
    nml=n-1;
    j=1;
    for (i=1;i<=nml;i++)
    {
        if (i>=j) goto label5;
        t[0]=a[0][j];
        t[1]=a[1][j];
label5:
    }
```

$?$
 $n(l=1;$
 $l \leq m;$
 $l++)$

```

a[0][j]=a[0][i];
a[1][j]=a[1][i];
a[0][i]=t[0];
a[1][i]=t[1];
label15: k=nv2;
label16: if (k>=j) goto label17;
j=j-k;
k=k/2;
goto label16;
label17: j=j+k;
}
pi=3.141592653589793;
for (l=1; l<=m; l++) ← ?
{
le=pow(2,l);
lel=(float)le/2;
u[0]=1.0;
u[1]=0.0;
w[0]=cos(pi/lel);
if (w[0]<0.0) w[0]*=-1;
w[1]=sin(pi/lel);
for (j=1; j<=lel; j++)
{
for (i=j; i<=n; i+=le)
{
ip=i+lel;
t[0]=a[0][ip]*u[0]-a[1][ip]*u[1];
t[1]=a[1][ip]*u[0]+a[0][ip]*u[1];
a[0][ip]=a[0][i]-t[0];
a[1][ip]=a[1][i]-t[1];
a[0][i]+=t[0];
a[1][i]+=t[1];
}
temp=u[0];
u[0]=u[0]*w[0]-u[1]*w[1];
u[1]=u[1]*w[0]+temp*w[1];
}
}
}

```

```

/*2-D control of Fast Fourier Transform*/
void fft()
{
int k,l;

for (k=1;k<=height;k++)
{
for (l=1;l<=width;l++)
{
a[0][l]=dget(image,k,l);
a[1][l]=0.0;
}
fft_1d();
for (l=1;l<=width;l++)
{
dput(real,k,l,a[0][l]);
dput(imaginary,k,l,a[1][l]);
}
printf("%.~");
}
for (l=1;l<=width;l++)
{
for (k=1;k<=height;k++)
{
a[0][k]=dget(real,k,l);

```



```

        a[1][k]=dget(imaginary,k,1);
    }
    fft_ld();
    for (k=1;k<=height;k++)
    {
        dput(real,k,1,a[0][k]);
        dput(imaginary,k,1,-a[1][k]);
        if (-a[1][k]>bias1) bias1=-a[1][k];
    }
    printf("%.7");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;

```

```

    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseq];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    float bias;
    int i,j,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("input","r");
    fscanf(infile, "%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fclose(infile);

    /*Read in image and initialize extended memory*/
    infile=fopen(string,"r+b");
    xmallocate(&image,2048);
    xmallocate(&real,2048);
    xmallocate(&imaginary,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile,"%c",&dummy);
            data = (float) dummy;
            xputrow(&data,image,(j-1)*4,(i-1),4,2880);
        }
    }

    /*Process image*/
    fft();

    /*Save FFT results to hard disk*/
    real_outfile=fopen("real.out","w");
    imaginary_outfile=fopen("imag.out","w");

```

```

for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data = dget(real, j, i);
        fprintf(real_outfile, "%f ", data);
        data = dget(imaginary, j, i);
        fprintf(imaginary_outfile, "%f ", data);
    }
}

/*Display real and imaginary planes*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Display real plane
bias=dget(real, 1, 1);
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(real, j, i)*128.0/bias+127.0);
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}

getchar();

//Display imaginary plane
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(imaginary, j, i)*128.0/bias+127.0);
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
    }
}

```

```

        Display_Pixel(pixel,position);
    }

    getchar();

    //Reset text mode
    _asm mov ax,3;
    _asm int 10h;

    /*Close files and free extended memory*/
    _fcloseall();
    xfree(image);
    xfree(real);
    xfree(imaginary);
}
/*Main program ends*/

```

Appendix G

The Inverse FFT Processor

The Inverse FFT Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\videfs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,real,imaginary,mode,width,height;
int ourseg,WIDTH1,HEIGHT1,_height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile1, *infile2, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    float data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

float a[2][1025];

/*1-D Inverse Fast Fourier Transform*/
void invfft_ld()
{
    int n,nv2,nml,i,j,k,l,le,ip,m;
    float t[2],p1,u[2],w[2],le1,temp;

    n=width;
    m=log10((float) n)/log10(2.0)+.5;
    nv2=n/2;
    nml=n-1;
    j=1;
    for (i=1;i<=nml;i++)
    {
        if (i>=j) goto label5;
        t[0]=a[0][j];
        t[1]=a[1][j];
        a[0][j]=a[0][i];
        label5:
    }
```

`a[1][j]=a[1][i];
a[0][i]=t[0];
a[1][i]=t[1];
label5: k=nv2;
label6: if (k>=j) goto label7;
j=j-k;
k=k/2;
goto label6;
label7: j=j+k;
}
pi=3.141592653589793;
for (l=1; l<=m; l++)
{
le=pow(2, l);
le1=(float)le/2;
u[0]=1.0;
u[1]=0.0;
w[0]=cos(pi/le1);
if (w[0]<0.0) w[0]*=-1;
w[1]=sin(pi/le1);
for (j=1; j<=le1; j++)
{
for (i=j; i<=n; i+=le)
{
ip=i+le1;
t[0]=a[0][ip]*u[0]-a[1][ip]*u[1];
t[1]=a[1][ip]*u[0]+a[0][ip]*u[1];
a[0][ip]=a[0][i]-t[0];
a[1][ip]=a[1][i]-t[1];
a[0][i]+=t[0];
a[1][i]+=t[1];
}
temp=u[0];
u[0]=u[0]*w[0]-u[1]*w[1];
u[1]=u[1]*w[0]+temp*w[1];
}
}`

`/*2-D control of Inverse Fast Fourier Transform*/
void invfft()`

`{
int k, l;
float n;

n = (float) width;

for (k=1; k<=height; k++)
{
for (l=1; l<=width; l++)
{
a[0][l]=dget(real, k, l);
a[1][l]=dget(imaginary, k, l);
}
invfft_1d();
for (l=1; l<=width; l++)
{
dput(real, k, l, a[0][l]/n);
dput(imaginary, k, l, a[1][l]/n);
}
printf(".");
}
for (l=1; l<=width; l++)
{
for (k=1; k<=height; k++)`

```

        {
            a[0][k]=dget(real,k,1);
            a[1][k]=dget(imaginary,k,1);
        }
    invfft_ld();
    for (k=1;k<=height;k++)
    {
        a[0][k]=sqrt(a[0][k]*a[0][k]+a[1][k]*a[1][k])/n;
        dput(image,k,1,a[0][k]);
    }
    printf("%.~");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;

```



```

    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    float bias;
    int i,j,x,y;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    /*Read in image info*/
    infile1=fopen("input","r");
    fscanf(infile1, "%d %d %d", &mode, &width, &height);
    fscanf(infile1, "%s", string);
    fclose(infile1);

    /*Read in real and imaginary matrices and initialize extended memory*/
    infile1=fopen("real.out","r");
    infile2=fopen("imag.out","r");
    xmallocate(&image,2048);
    xmallocate(&real,2048);
    xmallocate(&imaginary,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile1,"%f ",&data);
            xmputrow(&data,real,(j-1)*4,(i-1),4,2880);
            fscanf (infile2,"%f ",&data);
            xmputrow(&data,imaginary,(j-1)*4,(i-1),4,2880);
        }
    }

    /*Process image*/
    invfft();

    /*Save FFT results to hard disk*/
    outfile=fopen("image.out","w+b");

```

```

for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data = dget(image,j,i);
        dummy = (unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display image*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Display image
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(image,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(real);
xmfree(imaginary);
}
/*Main program ends*/

```

Appendix H

The DFT Editor

The DFT Editor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,real,imaginary,mode,width,height;
int ourseg,WIDTH1,HEIGHT1,height,x1,x2,y1,y2;
unsigned char palbuf[256][3];
float bias1=0.0,bias=0.0,biasl=0.0,biasil=0.0;
void __far *p;
char ch;

FILE *infile1,*infile2,*outfile1,*outfile2;
/*Globals end*/

/*Extended memory data retrieval*/
float dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    float data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    float data;
{
    float dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*4,(index_y-1),4,2880);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
getchar();}
}

/*DFT Editor*/
void editdft()
{
    float data;
    int chl,i,j,x,y;

start: printf("\n(0)-Display section (1)-Edit entry (2)-Show section (3)-Zero
block");
    printf("\n?: ");
    scanf("%i",&chl);
    getchar();
    if (!chl)
    {
        printf("\nEnter mode: ");
        scanf("%i",&mode);
        getchar();
        printf("Enter start x: ");
        scanf("%i",&x1);
    }
}
```

```

    getchar();
    printf("Enter end x:  ");
    scanf("%i",&x2);
    getchar();
    printf("Enter start y:  ");
    scanf("%i",&y1);
    getchar();
    printf("Enter end y:  ");
    scanf("%i",&y2);
    getchar();
    if (x1==1 && y1==1 && x2==width && y2==height)
    {
        bias=bias1;
        biasi=bias1;
    }
    else
    {
        bias=0.0;
        biasi=0.0;
        for (j=y1;j<=y2;j++)
        {
            for (i=x1;i<=x2;i++)
            {
                data=dget(real,j,i);
                if (data>bias) bias=data;
                data=dget(imaginary,j,i);
                if (data>biasi) biasi=data;
            }
        }
    }
else
{
    if (chl==1)
    {
        printf("\nEnter x:  ");
        scanf("%i",&x);
        getchar();
        printf("Enter y:  ");
        scanf("%i",&y);
        getchar();
        printf("Enter new real:  ");
        scanf("%f",&data);
        getchar();
        dput(real,y,x,data);
        printf("Enter new imaginary:  ");
        scanf("%f",&data);
        getchar();
        dput(imaginary,y,x,data);
        goto start;
    }
    if (chl==2)
    {
        printf("\nEnter start x:  ");
        scanf("%i",&x1);
        getchar();
        printf("Enter end x:  ");
        scanf("%i",&x2);
        getchar();
        printf("Enter start y:  ");
        scanf("%i",&y1);
        getchar();
        printf("Enter end y:  ");
        scanf("%i",&y2);
        getchar();
    }
}

```

```

        for (j=y1;j<=y2;j++)
        {
            for (i=x1;i<=x2;i++)
            {
                printf("%6.2f,%6.2f ",dget(real,j,i),dget(imaginary,j,i));
            }
            printf("\n");
        }
        getchar();
        goto start;
    }
    if (chl==3)
    {
        printf("\nEnter start x: ");
        scanf("%i",&x1);
        getchar();
        printf("Enter end x: ");
        scanf("%i",&x2);
        getchar();
        printf("Enter start y: ");
        scanf("%i",&y1);
        getchar();
        printf("Enter end y: ");
        scanf("%i",&y2);
        getchar();
        for (j=y1;j<=y2;j++)
        {
            for (i=x1;i<=x2;i++)
            {
                dput(real,j,i,0);
                dput(imaginary,j,i,0);
            }
        }
        goto start;
    }
}

stop: printf("\nREADY TO DISPLAY"); getchar();
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
}

```

```

    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float data;
    float pix_no;
    int i,j,x,y,ch1,ch,x3,y3;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15];
    unsigned char dummy;

    /*Read in image info*/
    infile1=fopen("input","r");

```

```

fscanf(infile1, "%d %d %d", &mode, &width, &height);
fscanf(infile1, "%s", string);
fclose(infile1);

/*Read in real and imaginary matrices and initialize extended memory*/
infile1=fopen("real.out", "r");
infile2=fopen("imag.out", "r");
xmallocate(&image, 2048);
xmallocate(&real, 2048);
xmallocate(&imaginary, 2048);
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fscanf (infile1, "%f ", &data);
        xputrow(&data, real, (j-1)*4, (i-1), 4, 2880);
        if (data>bias1 && (i!=1 && j!=1)) bias1=data;
        fscanf (infile2, "%f ", &data);
        xputrow(&data, imaginary, (j-1)*4, (i-1), 4, 2880);
        if (data>bias1) bias1=data;
    }
}
fclose(infile1);
fclose(infile2);

x1=1;
y1=1;
x2=width;
y2=height;

/*Edit the DFT*/
loop1: editdft();

/*Display real and imaginary sections*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [oursreg], 0A000h;

//Display real section
y3=y2-y1+1;
x3=x2-x1+1;
if (y3 > HEIGHT1)
    y3=HEIGHT1;
for (j=1; j<=y3; j++)
{
    for (i=1; i<=x3; i++)
    {
        pixel = (unsigned int) (dget(real, j+y1-1, i+x1-1)*128.0/bias+127.0);
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}

```



```

    }
}

getchar();

//Display imaginary section
for (j=1; j<=y3; j++)
{
    for (i=1; i<=x3; i++)
    {
        pixel = (unsigned int) (dget(imaginary, j+y1-1, i+x1-1)*128.0/biasi+127.0);
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}

getchar();

//Reset text mode
_asm mov ax, 3;
_asm int 10h;

printf("\n(0)-Continue (1)-Quit");
printf("\n?: ");
scanf("%i", &ch);
getchar();
if (!ch) goto loop1;

/*Save results to hard disk*/
printf("\n(0)-Save & Exit (1)-Exit");
printf("\n?: ");
scanf("%i", &chl);
getchar();
if (chl) goto endl;
outfile1=fopen("real.out", "w");
outfile2=fopen("imag.out", "w");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fprintf (outfile1, "%f ", dget(real, j, i));
        fprintf (outfile2, "%f ", dget(imaginary, j, i));
    }
}

/*Close files and free extended memory*/
endl:
fcloseall();
xfree(image);
xfree(real);
xfree(imaginary);
}
/*Main program ends*/

```

Appendix I

The Dilation Processor

The Dilation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,result,mode,width,height,mwidth,mheight;
int ourseg,WIDTH1,HEIGHT1,_height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

/*Dilation*/
void dilate()
{
    int i,j,ii,jj,index_r,index_c,max,test;

    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            max=255;
            for (jj=(-mheight/2);jj<=(mheight/2);jj++)
            {
                for (ii=(-mwidth/2);ii<=(mwidth/2);ii++)
                {
                    index_c=i+ii;
                    index_r=j+jj;
                    if (index_r>1&&index_r<=height&&index_c>1&&index_c<=width)
                    {
                        test=dget(image,index_r,index_c);
```

```

        if (test < max) max=test;
    }
}
    dput(result, j, i, max);
}
printf("%.~");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax, 13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax, 4f02h;
        _asm mov bx, 101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax, 4f02h;
        _asm mov bx, 103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax, 4f02h;
        _asm mov bx, 105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i, j, k;

    for (i=0; i<=255; i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=spalbuf;
    i=(int)p;
    _asm mov dx, i;
    _asm mov bx, k;
    _asm mov cx, j;
    _asm mov ax, 1012h;

```

```

    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15],string1[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile,"%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fscanf(infile,"%s", string1);
    fscanf(infile,"%d %d", &ewidth, &eheight);
    fclose(infile);

    /*Read in image and initialize extended memory*/
    infile=fopen(string,"r+b");
    xmallocate(&image,2048);
    xmallocate(&result,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile,"%c",&dummy);
            data = (int) dummy;
            dput(image,j,i,data);
        }
    }

    /*Process image*/
    dilate();

    /*Save dilation results to hard disk*/
    outfile=fopen(string1,"w+b");
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            data = dget(result,j,i);

```

```

        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Display
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1;j<=_height;j++)
{
    for (i=1;i<=width;i++)
    {
        pixel = (unsigned int) (dget(result,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(result);
}
/*Main program ends*/

```

Appendix J

The Erosion Processor

The Erosion Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,result,mode,width,height,mwidth,mheight;
int ourseg,WIDTH1,HEIGHT1,_height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
int handle,index_x,index_y;
int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

/*Erosion*/
void erode()
{
    int i,j,ii,jj,index_r,index_c,max,test;

    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            max=0;
            for (jj=-mheight/2;jj<=(mheight/2);jj++)
            {
                for (ii=-mwidth/2;ii<=(mwidth/2);ii++)
                {
                    index_c=i+ii;
                    index_r=j+jj;
                    if (index_r>=1&&index_r<=height&&index_c>=1&&index_c<=width)
                    {
                        test=dget(image,index_r,index_c);

```



```

        if (test > max) max=test;
    }
    }
    dput(result,j,i,max);
}
printf("%.");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;

```

```

    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15],string1[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile, "%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fscanf(infile,"%s", string1);
    fscanf(infile, "%d %d", &width, &height);
    fclose(infile);

    /*Read in image and initialize extended memory*/
    infile=fopen(string,"r+b");
    xmallocate(&image,2048);
    xmallocate(&result,2048);
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            fscanf (infile,"%c",&dummy);
            data = (int) dummy;
            dput(image,j,i,data);
        }
    }

    /*Process image*/
    erode();

    /*Save erosion results to hard disk*/
    outfile=fopen(string1,"w+b");
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            data = dget(result,j,i);

```

```

        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Display
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1;j<=_height;j++)
{
    for (i=1;i<=width;i++)
    {
        pixel = (unsigned int) (dget(result,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(result);
}
/*Main program ends*/

```

Appendix K

The Closure Processor

The Closure Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,result,mode,width,height,mwidth,mheight;
int ourseg.WIDTH1,HEIGHT1,_height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

/*Closure*/
void close()
{
    int i,j,ii,jj,index_r,index_c,max,test;

    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            max=255;
            for (jj=-mheight/2;jj<=(mheight/2);jj++)
            {
                for (ii=-mwidth/2;ii<=(mwidth/2);ii++)
                {
                    index_c=i+ii;
                    index_r=j+jj;
                    if (index_r>=1&&index_r<=height&&index_c>=1&&index_c<=width)
                    {
                        test=dget(image,index_r,index_c);

```

```

        if (test < max) max=test;
    }
}
    dput(result,j,i,max);
}
printf("%.");
}

for (j=1;j<=height;j++)
{
    for (i=1;i<=width;i++)
    {
        max=0;
        for (jj=-mheight/2;jj<=(mheight/2);jj++)
        {
            for (ii=-mwidth/2;ii<=(mwidth/2);ii++)
            {
                index_c=i+ii;
                index_r=j+jj;
                if (index_r>=1&&index_r<=height&&index_c>=1&&index_c<=width)
                {
                    test=dget(result,index_r,index_c);
                    if (test > max) max=test;
                }
            }
        }
        dput(image,j,i,max);
    }
    printf("%.");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
    }
}

```

```

    HEIGHT1=768;
    )
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
{
    int bank_no;
    {
        _asm mov ax,4f05h;
        _asm mov bx,00h;
        _asm mov dx,bank_no;
        _asm int 10h;
    }
}

/*Pixel display*/
void Display_Pixel(pixel,position)
{
    int pixel,position;
    {
        _asm mov es,[ourseg];
        _asm mov al,byte ptr pixel;
        _asm mov bx,position;
        _asm mov es:[bx],al;
    }
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15], string1[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile, "%d %d %d", &mode, &width, &height);
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%d %d", &width, &height);
    fclose(infile);

    /*Read in image and initialize extended memory*/

```

```

infile=fopen(string,"r+b");
xmallocate(&image,2048);
xmallocate(&result,2048);
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fscanf (infile,"%c",&dummy);
        data = (int) dummy;
        dput (image,j,i,data);
    }
}

/*Process image*/
dilate();

/*Save closure results to hard disk*/
outfile=fopen(string1,"w+b");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data = dget (image,j,i);
        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set_up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Display
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1;j<=_height;j++)
{
    for (i=1;i<=width;i++)
    {
        pixel = (unsigned int) (dget (image,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

```



```
//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
xmfree(image);
xmfree(result);
}
/*Main program ends*/
```

Appendix L

The Opening Processor

The Opening Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,result,mode,width,height,mwidth,mheight;
int ourseg,WIDTH1,HEIGHT1,height;
unsigned char palbuf[256][3];
void __far *p;

FILE *infile, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
    getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2048);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
    getchar();}
}

/*Opening*/
void open()
{
    int i,j,ii,jj,index_r,index_c,max,test;

    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            max=0;
            for (jj=-mheight/2;jj<=(mheight/2);jj++)
            {
                for (ii=-mwidth/2;ii<=(mwidth/2);ii++)
                {
                    index_c=i+ii;
                    index_r=j+jj;
                    if (index_r>=1&&index_r<=height&&index_c>=1&&index_c<=width)
                    {
                        test=dget(image,index_r,index_c);
```

```

        if (test > max) max=test;
    }
}
    }
    dput(result, j, i, max);
}
printf("\n");
}

for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        max=255;
        for (jj=-mheight/2; jj<=(mheight/2); jj++)
        {
            for (ii=-mwidth/2; ii<=(mwidth/2); ii++)
            {
                index_c=i+ii;
                index_r=j+jj;
                if (index_r>1&&index_r<=height&&index_c>=1&&index_c<=width)
                {
                    test=dget(result, index_r, index_c);
                    if (test < max) max=test;
                }
            }
        }
        dput(image, j, i, max);
    }
    printf("\n");
}
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax, 13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax, 4f02h;
        _asm mov bx, 101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax, 4f02h;
        _asm mov bx, 103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax, 4f02h;
        _asm mov bx, 105h;
        _asm int 10h;
        WIDTH1=1024;
    }
}

```

```

        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    char string[15],string1[15];
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile, "%d %d %d", &mode, &width, &height);
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%d %d", &width, &height);
    fclose(infile);

    /*Read in image and initialize extended memory*/

```

```

infile=fopen(string,"r+b");
xmallocate(&image,2048);
xmallocate(&result,2048);
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        fscanf (infile,"%c",&dummy);
        data = (int) dummy;
        dput(image,j,i,data);
    }
}

/*Process image*/
open();

/*Save opening results to hard disk*/
outfile=fopen(string1,"w+b");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data = dget(image,j,i);
        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set_up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Display
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1;j<=_height;j++)
{
    for (i=1;i<=width;i++)
    {
        pixel = (unsigned int) (dget(image,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

```

```
//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
_xmfree(image);
_xmfree(result);
}
```

Appendix M

The Windowed Convolution Processor

The Windowed Convolution Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\graph.h>
#include <c:\c700\include\conio.h>
#include <c:\c700\include\string.h>
#include <c:\c700\include\stdlib.h>
#include <c:\c700\include\math.h>
#define RGB(r,g,b) (0x3F3F3F | ((long)(b) << 16 | (g) << 8 | (r)))

/*Globals begin*/
char string[80], string1[80], savefile[80], confile[80];
unsigned char red[256], blue[256], green[256], cp,d[3][1024];
int num,i,j,k,ix,jx,xdim,ydim,mode,header,updown,xbound,ybound,k1lx,bias_flag;
float k1[20][20], k2[20][20], templ, positive, negative;

FILE *input_file, *inpalette, *infile, *out, *incon;
/*Globals begin*/

/*Main program begins*/
main()
{
    /*Read procedure information*/
    infile=fopen("tfile","r");
    fscanf(infile,"%i",&mode);
    fscanf(infile,"%s",string);
    fscanf(infile,"%s",string1);
    fscanf(infile,"%i",&xdim);
    fscanf(infile,"%i",&ydim);
    fscanf(infile,"%i",&header);
    fscanf(infile,"%i",&updown);
    fscanf(infile,"%s",savefile);
    fscanf(infile,"%s",confile);
    fscanf(infile,"%i",&bias_flag);

    /*Open files*/
    input_file=fopen(string1,"r+b");
    inpalette=fopen(string,"r+b");
    incon=fopen(confile,"r");
    out=fopen(savefile,"w+b");

    /*Read in convolution matrix*/
    fscanf(incon,"%i",&num);
    negative=positive=0.0;
    for (j=0;j<num;j++)
    {
        for (i=0;i<num;i++)
        {
            fscanf(incon,"%f",&k2[j][i]);
            if (k2[j][i]<0) negative-=k2[j][i];
            if (k2[j][i]>0) positive+=k2[j][i];
        }
    }

    /*Read in palette*/
    for (i=0;i<=255;i++)
    {
        fscanf(inpalette,"%c",&cp);
        red[i]=cp;
    }
    for (i=0;i<=255;i++)
    {
        fscanf(inpalette,"%c",&cp);
    }
}
```

```

        green[i]=cp;
    }
    for (i=0;i<=255;i++)
    {
        fscanf(inpalette,"%c",&cp);
        blue[i]=cp;
    }
    if (header !=0)
    {
        for (i = 1; i <= header; i++)
        {
            fscanf(input_file,"%c",&cp);
        }
    }

    /*Set video mode*/
    if (mode==1) _setvideomode(_MRES256COLOR);
    if (mode==2) _setvideomode(_VRES256COLOR);
    if (mode==3) _setvideomode(_SRES256COLOR);
    if (mode==4) _setvideomode(_XRES256COLOR);

    /*Enable palette*/
    for (i=0;i<=255;i++)
    {
        remappalette(i,RGB(red[i]/4,green[i]/4,blue[i]/4));
    }

    /*Display original image*/
    for (i=0;i<ydim;i++)
    {
        for (j=0;j<xdim;j++)
        {
            fscanf(input_file,"%c",&cp);
            k = cp;
            _setcolor(k);
            if (updown == 0) _setpixel(j,i);
            if (updown == 1) _setpixel(j,ydim-1-i);
        }
    }

    /*Perform convolution for arbitrary borders and display results*/
    templ=0;
    for (i=0;i<ydim;i++)
    {
        for (j=0;j<xdim;j++)
        {
            for (ix=0;ix<num;ix++)
            {
                for (jx=0;jx<num;jx++)
                {
                    xbound = j-1+jx;
                    ybound = i-1+ix;
                    if (xbound < 0) xbound = -xbound;
                    if (ybound < 0) ybound = -ybound;
                    if (xbound > (xdim-1)) xbound = xdim-1-(xbound-(xdim-1));
                    if (ybound > (ydim-1)) ybound = ydim-1-(ybound-(ydim-1));
                    k1[ix][jx]=(float) _getpixel(xbound,ybound);
                    templ+= k2[ix][jx]*k1[ix][jx];
                }
            }
            k = (int) templ;
            if (!bias_flag)
            {
                if (k>255) k=255;
                if (k<0) k=0;
            }
        }
    }

```

```

    }
    temp1=0;
    if (i>=((int) (num/2.0)+1))
    {
        kllx=d[(i-((int) (num/2.0)+1))*((int) (num/2.0)+1)][j];
        _setcolor(kllx);
        _setpixel(j,i-((int) (num/2.0)+1));
    }
    if (bias_flag)
    {
        if (k<0) d[i*((int) (num/2.0)+1)][j]=(unsigned char) (k/(2.0*nega-
tive)+128);
        if (k>=0) d[i*((int) (num/2.0)+1)][j]= (unsigned char) (k/(2.0*posi-
tive)+128);
    }
    else
    {
        d[i*((int) (num/2.0)+1)][j]=(unsigned char) k;
    }
}
}
for (i=ydim-1-((int) (num/2.0)+1);i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        k = d[i*((int) (num/2.0)+1)][j];
        _setcolor(k);
        _setpixel(j,i);
    }
}

/*Write results to hard disk*/
for (i=0;i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        k = _getpixel(j,i);
        fprintf(out,"%c",k);
    }
}

/*Close files*/
_fcloseall();

/*Pause*/
getch();

/*Reset text mode*/
_setvideomode(_DEFAULTMODE);
}
/*Main program ends*/

```

Appendix N

The Value Thresholding Processor

The Value Thresholding Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,mode,width,height,lower_threshold;
int ourseg,WIDTH1,HEIGHT1,upper_threshold;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
/*Globals end*/

FILE *infile, *outfile;

/*Video mode controller*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
}
```

```

    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel Display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile,"%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fscanf(infile,"%s", string1);
    fscanf(infile,"%i", &super_threshold);
    fscanf(infile,"%i", &slower_threshold);
    fclose(infile);

    /*Read in image, manipulate image, and display results*/
    //Set display mode
    Set_Mode();

    //Set up palette
    Set_Palette();

    //Set-up VGA memory
    Set_Bank(bank);
    _asm mov [ourseg],0A000h;

    //Read in data, threshold, write to hard disk, and display image
    infile=fopen(string,"r+b");
    outfile=fopen(string1,"w+b");
    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)

```

```

    {
        fscanf(infile, "%c", &dummy);
        data=dummy;
        if (data>upper_threshold) data=upper_threshold;
        if (data<lower_threshold) data=lower_threshold;
        dummy=(unsigned char) data;
        fprintf(outfile, "%c", dummy);
        pixel = (unsigned int) data;
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
fcloseall();
}
/*Main program ends*/

```

Appendix O

The Occurrence Thresholding Processor

The Occurrence Thresholding Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\videfs.h>
#include <c:\c700\include\vicfcts.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,mode,width,height,most_freq;
int ourseg,WIDTH1,HEIGHT1,_height;
unsigned char palbuf[256][3];
char string[15],string1[15];
float weight[256],_threshold;
void __far *p;
/*Globals end*/

FILE *infile, *outfile;

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
getchar();}
}

/*Obtain histogram*/
void histo()
{
    int i,j,index;
    unsigned char cp;

    infile=fopen(string,"r+b");
    for (i=0;i<256;i++)
    {
        weight[i]=0.0;
    }
    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            fscanf(infile,"%c",&cp);
            index=cp;

```

```

        weight[index]+=1.0;
        dput (image, j, i, index);
    }
    if (j%4==0) printf(".");
}
most_freq=0;
for (i=0; i<256; i++)
{
    if (weight[i]>weight[most_freq]) most_freq=i;
}
fclose(i-file);
}

/*Occurrence thresholding*/
void threshold_by_occurence()
{
    int i, j, index;
    float test;

    histo();
    for (j=1; j<=height; j++)
    {
        for (i=1; i<=width; i++)
        {
            index=dget (image, j, i);
            test=weight[index];
            while ((test<_threshold) && (index<most_freq))
            {
                index++;
                test=weight[index];
            }
            test=weight[index];
            while ((test<_threshold) && (index>most_freq))
            {
                index--;
                test=weight[index];
            }
            if (index>255) index=255;
            dput (image, j, i, index);
        }
        if (j%4==0) printf(".");
    }
}

/*Video mode controller*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax, 13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax, 4f02h;
        _asm mov bx, 101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax, 4f02h;

```

```

    _asm mov bx,103h;
    _asm int 10h;
    WIDTH1=800;
    HEIGHT1=600;
}
//1024X768
if (mode==4) {
    _asm mov ax,4f02h;
    _asm mov bx,105h;
    _asm int 10h;
    WIDTH1=1024;
    HEIGHT1=768;
}
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank controller*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    unsigned char dummy;

```

```

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile, "%d %d %d", &mode, &width, &height);
fscanf(infile, "%s", string);
fscanf(infile, "%s", string1);
fscanf(infile, "%f", &_threshold);
fclose(infile);

/*Initialize extended memory*/
xmallocate(&image, 2048);

/*Process image*/
threshold_by_occurence();

/*Save results to hard disk*/
outfile=fopen(string1, "w+b");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data=dget(image, j, i);
        dummy=(unsigned char) data;
        fprintf(outfile, "%c", dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg], 0A000h;

//Display image
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(image, j, i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel, position);
    }
}

getchar();

//Reset text mode

```

```
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
_fcloseall();
_xmfree(image);
}
/*Main program ends*/
```

Appendix P

The Square Contrast Manipulation Processor

The Square Contrast Manipulation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseg,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
```

```
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}
```

```
/*Palette load*/
```

```
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
```

```

k=0;
p=$palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
_asm mov ax,4f05h;
_asm mov bx,00h;
_asm mov dx,bank_no;
_asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
_asm mov es,[ourseg];
_asm mov al,byte ptr pixel;
_asm mov bx,position;
_asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
float pix_no,temp;
int i,j,x,y,data;
unsigned int bank=0,pixel,new_bank,position;
unsigned char dummy;

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile,"%d %d %d", &mode, &width, &height);
fscanf(infile,"%s", string);
fscanf(infile,"%s", string1);
fclose(infile);

/*Read in data, manipulate, display results, and store to hard disk*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Read in data, manipulate, display results, and store to hard disk*/
infile=fopen(string,"r+b");
outfile=fopen(string1,"w+b");
for (j=1;j<=height;j++)
{
for (i=1;i<=width;i++)
{
fscanf(infile,"%c",&dummy);
data=dummy;

```



```

temp=(float)data;
temp/=256.0;
temp=temp*temp*256.0;
data=(int)temp;
dummy=(unsigned char) data;
fprintf(outfile,"%c",dummy);
pixel = (unsigned int) data;
x=i-1;
y=j-1;
pix_no= (float) y*WIDTH1+x+1;
new_bank=pix_no/65536;
if (new_bank!=bank)
{
    bank=new_bank;
    Set_Bank(bank);
}
position=pix_no-new_bank*65536;
Display_Pixel(pixel,position);
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
_fcloseall();
}
/*Main program ends*/

```

Appendix Q

The Cube Contrast Manipulation Processor

The Cube Contrast Manipulation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseq,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
```

```
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}
```

```
/*Palette load*/
```

```
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
```

```

k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video Memory Bank Control*/
void Set_Bank(bank_no)
int bank_no;
{
_asm mov ax,4f05h;
_asm mov bx,00h;
_asm mov dx,bank_no;
_asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
_asm mov es,[ourseg];
_asm mov al,byte ptr pixel;
_asm mov bx,position;
_asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
float pix_no,temp;
int i,j,x,y,data;
unsigned int bank=0,pixel,new_bank,position;
unsigned char dummy;

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile,"%d %d %d", &mode, &width, &height);
fscanf(infile,"%s", string);
fscanf(infile,"%s", string1);
fclose(infile);

/*Read in data, manipulate, display results, and write to hard disk*/
//Set display mode
Set_Mode();

//Set_up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Read in data, manipulate, display results, and write to hard disk
infile=fopen(string,"r+b");
outfile=fopen(string1,"w+b");
for (j=1;j<=height;j++)
{
for (i=1;i<=width;i++)
{
fscanf(infile,"%c",&dummy);

```

```

data=dummy;
temp=(float)data;
temp/=256.0;
temp=temp*temp*temp*256.0;
data=(int)temp;
dummy=(unsigned char) data;
fprintf(outfile,"%c",dummy);
pixel = (unsigned int) data;
x=i-1;
y=j-1;
pix_no= (float) y*WIDTH1+x+1;
new_bank=pix_no/65536;
if (new_bank!=bank)
{
    bank=new_bank;
    Set_Bank(bank);
}
position=pix_no-new_bank*65536;
Display_Pixel(pixel,position);
}

}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
fcloseall();
}
/*Main program ends*/

```

Appendix R

The Square Root Contrast Manipulation Processor

The Square Root Contrast Manipulation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseg,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
void Set_Mode()
```

```
{
//320X200
if (mode==1) {
_asm mov ax,13h;
_asm int 10h;
WIDTH1=320;
HEIGHT1=200;
}
//640X480
if (mode==2) {
_asm mov ax,4f02h;
_asm mov bx,101h;
_asm int 10h;
WIDTH1=640;
HEIGHT1=480;
}
//800X600
if (mode==3) {
_asm mov ax,4f02h;
_asm mov bx,103h;
_asm int 10h;
WIDTH1=800;
HEIGHT1=600;
}
//1024X768
if (mode==4) {
_asm mov ax,4f02h;
_asm mov bx,105h;
_asm int 10h;
WIDTH1=1024;
HEIGHT1=768;
}
}
```

```
/*Palette load*/
```

```
void Set_Palette()
```

```
{
int i,j,k;

for (i=0;i<=255;i++)
{
palbuf[i][0]=i/4;
palbuf[i][1]=i/4;
palbuf[i][2]=i/4;
}
j=256;
```

```

k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
_asm mov ax,4f05h;
_asm mov bx,00h;
_asm mov dx,bank_no;
_asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
_asm mov es,[ourseg];
_asm mov al,byte ptr pixel;
_asm mov bx,position;
_asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
float pix_no,temp;
int i,j,x,y,data;
unsigned int bank=0,pixel,new_bank,position;
unsigned char dummy;

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile, "%d %d %d", &mode, &width, &height);
fscanf(infile, "%s", string);
fscanf(infile, "%s", string1);
fclose(infile);

/*Read in data, manipulate, display image, and write to hard disk*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Read in data, manipulate, display image, and write to hard disk
infile=fopen(string,"r+b");
outfile=fopen(string1,"w+b");
for (j=1;j<=height;j++)
{
for (i=1;i<=width;i++)
{
fscanf(infile, "%c", &dummy);

```



```

        data=dummy;
        temp=(float)data;
        temp/=256.0;
        temp=sqrt(temp)*256.0;
        data=(int)temp;
        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
        pixel = (unsigned int) data;
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }

}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
_fcloseall();
}
/*Main program ends*/

```

Appendix S

The Cube Root Contrast Manipulation Processor

The Cube Root Contrast Manipulation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseg,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
```

```
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}
```

```
/*Palette load*/
```

```
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
}
```

```

    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no,temp;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile,"%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fscanf(infile,"%s", string1);
    fclose(infile);

    /*Read in data, manipulate, display image, and write to hard disk*/
    //Set display mode
    Set_Mode();

    //Set_up palette
    Set_Palette();

    //Set-up VGA memory
    Set_Bank(bank);
    _asm mov [ourseg],0A000h;

    //Read in data, manipulate, display image, and write to hard disk
    infile=fopen(string,"r+b");
    outfile=fopen(string1,"w+b");
    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            fscanf(infile,"%c",&dummy);

```

```

data=dummy;
temp=(float)data;
temp/=256.0;
temp=pow(temp,.33333)*256.0;
data=(int)temp;
dummy=(unsigned char) data;
fprintf(outfile,"%c",dummy);
pixel = (unsigned int) data;
x=i-1;
y=j-1;
pix_no= (float) y*WIDTH1+x+1;
new_bank=pix_no/65536;
if (new_bank!=bank)
{
    bank=new_bank;
    Set_Bank(bank);
}
position=pix_no-new_bank*65536;
Display_Pixel(pixel,position);
}

}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
_fcloseall();
}
/*Main program ends*/

```

Appendix T

The 1/x (Inverse) Manipulation Processor

The 1/x (Inverse) Manipulation Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseg,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
void Set_Mode()
```

```
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}
```

```
/*Palette load*/
void Set_Palette()
```

```
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
```

```

k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
_asm mov ax,4f05h;
_asm mov bx,00h;
_asm mov dx,bank_no;
_asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
_asm mov es,[ourseg];
_asm mov al,byte ptr pixel;
_asm mov bx,position;
_asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
float pix_no,temp;
int i,j,x,y,data;
unsigned int bank=0,pixel,new_bank,position;
unsigned char dummy;

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile,"%d %d %d",&mode,&width,&height);
fscanf(infile,"%s",string);
fscanf(infile,"%s",string1);
fclose(infile);

/*Read in data, manipulate, display results, and write to hard disk*/
//Set display mode
Set_Mode();

//Set_up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Read in data, manipulate, display results, and write to hard disk
infile=fopen(string,"r+b");
outfile=fopen(string1,"w+b");
for (j=1;j<=height;j++)
{
for (i=1;i<=width;i++)
{
fscanf(infile,"%c",&dummy);

```



```

    data=dummy;
    temp=(float)data;
    if (temp<=1.0) temp=255.0;
    else
    {
        temp/=256.0;
        temp=1.0/temp;
    }
    data=(int)temp;
    dummy=(unsigned char) data;
    fprintf(outfile,"%c",dummy);
    pixel = (unsigned int) data;
    x=i-1;
    y=j-1;
    pix_no= (float) y*WIDTH1+x+1;
    new_bank=pix_no/65536;
    if (new_bank!=bank)
    {
        bank=new_bank;
        Set_Bank(bank);
    }
    position=pix_no-new_bank*65536;
    Display_Pixel(pixel,position);
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
_fcloseall();
}
/*Main program ends*/

```

Appendix U

The Inverting Processor

The Inverting Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\graph.h>
```

```
/*Globals begin*/
int image,mode,width,height;
int ourseg,WIDTH1,HEIGHT1;
unsigned char palbuf[256][3];
char string[15],string1[15];
void __far *p;
```

```
FILE *infile, *outfile;
/*Globals end*/
```

```
/*Video mode control*/
```

```
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
    }
}
```

```
/*Palette load*/
```

```
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
```

```

k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
int bank_no;
{
_asm mov ax,4f05h;
_asm mov bx,00h;
_asm mov dx,bank_no;
_asm int 10h;
}

/*Pixel display*/
void Display_Pixel(pixel,position)
int pixel,position;
{
_asm mov es,[ourseg];
_asm mov al,byte ptr pixel;
_asm mov bx,position;
_asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
float pix_no,temp;
int i,j,x,y,data;
unsigned int bank=0,pixel,new_bank,position;
unsigned char dummy;

/*Read in image info*/
infile=fopen("tfile","r");
fscanf(infile,"%d %d %d",&mode,&width,&height);
fscanf(infile,"%s",string);
fscanf(infile,"%s",string1);
fclose(infile);

/*Read in data, manipulate, display image, and write to hard disk*/
//Set display mode
Set_Mode();

//Set_up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Read in data, manipulate, display image, and write to hard disk
infile=fopen(string,"r+b");
outfile=fopen(string1,"w+b");
for (j=1;j<=height;j++)
{
for (i=1;i<=width;i++)
{
fscanf(infile,"%c",&dummy);

```

```

        data=dummy;
        temp=(float)data;
        if (temp<=1.0) temp=255.0;
        else
        {
            temp/=256.0;
            temp=1.0/temp;
        }
        data=(int)temp;
        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
        pixel = (unsigned int) data;
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files*/
fcloseall();
}
/*Main program ends*/

```

Appendix V

The Image Extraction Processor

The Image Extraction Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\math.h>
#include <c:\c700\include\vicdefs.h>
#include <c:\c700\include\vicfcta.h>
#include <c:\c700\include\vicerror.h>
#include <c:\c700\include\graph.h>

/*Globals begin*/
int image,mode,width,height,lowest,highest;
int ourseg,WIDTH1,HEIGHT1,height;
unsigned char palbuf[256][3];
char string[15],string1[15];
float factor,range,weight[256];
void __far *p;

FILE *infile, *outfile;
/*Globals end*/

/*Extended memory data retrieval*/
int dget(handle,index_x,index_y)
    int handle,index_x,index_y;
{
    int data;
    int test;

    test=xmgetrow(&data,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
    return(data);
}

/*Extended memory data placement*/
void dput(handle,index_x,index_y,data)
    int handle,index_x,index_y;
    int data;
{
    int dummy;
    int test;

    dummy = data;
    test=xmputrow(&dummy,handle,(index_x-1)*2,(index_y-1),2,2050);
    if (test!=NO_ERROR) {printf("dput %i,%i,%i",handle,index_x,index_y);
getchar();}
}

/*Obtain histogram*/
void histo()
{
    int i,j,index;
    unsigned char cp;

    infile=fopen(string,"r+b");
    for (i=0;i<256;i++)
    {
        weight[i]=0.0;
    }
    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            fscanf(infile,"%c",&cp);
            index=cp;

```

```

        weight[index]+=1.0;
        dput(image,j,i,index);
    }
    if (j%4==0) printf(".");
}
i=0;
while (weight[i]==0.0) i++;
lowest=i;
i=255;
while (weight[i]==0.0) i--;
highest=i;
range=(float) (highest-lowest);
fclose(infile);
}

/*Spread out histogram*/
void extract()
{
    int i,j,index;
    float base;
    histo();
    factor=(float) 256.0/range;
    for (j=1;j<=height;j++)
    {
        for (i=1;i<=width;i++)
        {
            index=dget(image,j,i);
            base=(float) (index-lowest);
            index=(int) (base*factor);
            if (index>255) index=255;
            dput(image,j,i,index);
        }
        if (j%4==0) printf(".");
    }
}

/*Video mode control*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
    }
    //1024X768
    if (mode==4) {
        _asm mov ax,4f02h;

```



```

    _asm mov bx,105h;
    _asm int 10h;
    WIDTH1=1024;
    HEIGHT1=768;
}
}

/*Palette load*/
void Set_Palette()
{
    int i,j,k;

    for (i=0;i<=255;i++)
    {
        palbuf[i][0]=i/4;
        palbuf[i][1]=i/4;
        palbuf[i][2]=i/4;
    }
    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;
}

/*Video memory bank control*/
void Set_Bank(bank_no)
    int bank_no;
{
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,bank_no;
    _asm int 10h;
}

/*Pixel display
void Display_Pixel(pixel,position)
    int pixel,position;
{
    _asm mov es,[ourseg];
    _asm mov al,byte ptr pixel;
    _asm mov bx,position;
    _asm mov es:[bx],al;
}

/*Main program begins*/
main ()
{
    float pix_no;
    int i,j,x,y,data;
    unsigned int bank=0,pixel,new_bank,position;
    unsigned char dummy;

    /*Read in image info*/
    infile=fopen("tfile","r");
    fscanf(infile,"%d %d %d", &mode, &width, &height);
    fscanf(infile,"%s", string);
    fscanf(infile,"%s", string1);
    fclose(infile);

```

```

/*Initialize extended memory*/
xmallocate(&image,2048);

/*Process image*/
extract();

/*Save extraction results to hard disk*/
outfile=fopen(string1,"w+b");
for (j=1; j<=height; j++)
{
    for (i=1; i<=width; i++)
    {
        data=dget(image,j,i);
        dummy=(unsigned char) data;
        fprintf(outfile,"%c",dummy);
    }
}

/*Display results*/
//Set display mode
Set_Mode();

//Set up palette
Set_Palette();

//Set-up VGA memory
Set_Bank(bank);
_asm mov [ourseg],0A000h;

//Display results
if (height> HEIGHT1)
    _height=HEIGHT1;
else
    _height=height;
for (j=1; j<=_height; j++)
{
    for (i=1; i<=width; i++)
    {
        pixel = (unsigned int) (dget(image,j,i));
        x=i-1;
        y=j-1;
        pix_no= (float) y*WIDTH1+x+1;
        new_bank=pix_no/65536;
        if (new_bank!=bank)
        {
            bank=new_bank;
            Set_Bank(bank);
        }
        position=pix_no-new_bank*65536;
        Display_Pixel(pixel,position);
    }
}

getchar();

//Reset text mode
_asm mov ax,3;
_asm int 10h;

/*Close files and free extended memory*/
fcloseall();
xmfree(image);
}
/*Main program ends*/

```

Appendix W

The Median Filter Processor

The Median Filter Processor

```
#include <c:\c700\include\stdio.h>
#include <c:\c700\include\graph.h>
#include <c:\c700\include\conio.h>
#include <c:\c700\include\string.h>
#include <c:\c700\include\stdlib.h>
#include <c:\c700\include\math.h>
#define RGB(r,g,b) (0x3F3F3F | ((long)(b) << 16 | (g) << 8 | (r)))

/*Globals begin*/
char string[80], string1[80], savefile[80];
unsigned char red[256], blue[256], green[256], cp, d[15][1024], kl[400];
int block_size, i, j, k, ix, jx, xdim, ydim, mode, header, updown, z;
int xbound, ybound, kllx;
float temp1;

FILE *input_file, *inpalette, *infile, *out;
/*Globals end*/

/*Performs a standard bubble sort*/
void bubble()
{
    int ii, jj;
    unsigned char temp;

    for (ii=block_size*block_size; ii>=1; ii--)
        for (jj=2; jj<=ii; jj++)
            if (kl[jj-1]>kl[jj])
            {
                temp=kl[jj-1];
                kl[jj-1]=kl[jj];
                kl[jj]=temp;
            }
}

/*Main program begins*/
main()
{
    /*Read procedure information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%i", &mode);
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%i", &xdim);
    fscanf(infile, "%i", &ydim);
    fscanf(infile, "%i", &header);
    fscanf(infile, "%i", &updown);
    fscanf(infile, "%s", savefile);
    fscanf(infile, "%i", &block_size);

    /*Open files*/
    input_file=fopen(string1, "r+b");
    inpalette=fopen(string, "r+b");
    out=fopen(savefile, "w+b");

    /*Read in palette*/
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        red[i]=cp;
    }
}
```

```

for (i=0;i<=255;i++)
{
    fscanf(inpalette,"%c",&cp);
    green[i]=cp;
}
for (i=0;i<=255;i++)
{
    fscanf(inpalette,"%c",&cp);
    blue[i]=cp;
}
if (header !=0)
{
    for (i = 1; i <= header; i++)
    {
        fscanf(input_file,"%c",&cp);
    }
}

/*Set video mode*/
if (mode==1) _setvideomode(_MRES256COLOR);
if (mode==2) _setvideomode(_VRES256COLOR);
if (mode==3) _setvideomode(_SRES256COLOR);
if (mode==4) _setvideomode(_XRES256COLOR);

/*Enable palette*/
for (i=0;i<=255;i++)
{
    remappalette(i,RGB(red[i]/4,green[i]/4,blue[i]/4));
}

/*Display original image*/
for (i=0;i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        fscanf(input_file,"%c",&cp);
        k = cp;
        _setcolor(k);
        if (updown == 0) _setpixel(j,i);
        if (updown == 1) _setpixel(j,ydim-1-i);
    }
}

/*Perform 3x3 windowed median for arbitrary borders*/
templ=0;
for (i=0;i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        for (ix=0;ix<block_size;ix++)
        {
            for (jx=0;jx<block_size;jx++)
            {
                xbound = j-1+jx;
                ybound = i-1+ix;
                if (xbound < 0) xbound = -xbound;
                if (ybound < 0) ybound = -ybound;
                if (xbound > (xdim-1)) xbound = xdim-1-(xbound-(xdim-1));
                if (ybound > (ydim-1)) ybound = ydim-1-(ybound-(ydim-1));
                kl[ix*block_size+jx+1]=(unsigned char)_getpixel(xbound,ybound);
            }
        }
        bubble();
        k=kl[(block_size*block_size+1)/2];
        templ=0;
    }
}

```

```

        if (i>=((int) (block_size/2.0)+1))
        {
            kllx=d[(i-((int) (block_size/2.0)+1))*((int) (block_size/2.0)+1)][j];
            _setcolor(kllx);
            _setpixel(j,i-((int) (block_size/2.0)+1) );
        }
        d[i*((int) (block_size/2.0)+1)][j]=(unsigned char) k;
    }
}
for (i=ydim-1-((int) (block_size/2.0)+1);i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        k = d[i * ((int) (block_size/2.0)+1)][j];
        _setcolor(k);
        _setpixel(j,i);
    }
}

/*Write results to hard disk*/
for (i=0;i<ydim;i++)
{
    for (j=0;j<xdim;j++)
    {
        k = _getpixel(j,i);
        fprintf(out,"%c",k);
    }
}

/*Close files*/
_fcloseall();

/*Pause*/
getch();

/*Reset text mode*/
_setvideomode(_DEFAULTMODE);
}

```

Appendix X

The Transmit Processor

The Transmit Processor

```
#include <io.h>
#include <dos.h>
#include <abc.h>
#include <bios.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include "avdos.h"
#include <fcntl.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#include "loaddrv.c"
#include <process.h>
#include <abcvoice.h>
#include <sys\types.h>
#include <sys\timex.h>
#include <pctcp\types1.h>
#include <pctcp\pctcp.h>
#include <pctcp\ipconfig.h>
#include <pctcp\options.h>
#include <\ftp\src\alarm\alarm.h>

#define SOURCE "DefaultSource"
#define FORMAT BITMAP_SVW_NATIVE

#ifdef TRUE
#define TRUE 1
#define FALSE 0
#endif

/*Globals begin*/
struct addr a;
int templ, alarm_con, rc;
float sn1[2000], sn2[2000];
unsigned char buffer[2000];
long tmo=ALARM_TIMEOUT, lBufSize=0x400L;
char string1[80], superbuf[10000], __huge *hbuf, far *lpVoiceBuf;

FILE *input_file;
/*Globals end*/

/*Raised Cosine Filter for sound packets*/
void raised_cosine_filtering()
{
    for (templ=16; templ<=(rc+16); templ++)
    {
        superbuf[templ]=superbuf[templ]-128;
        superbuf[templ]=(superbuf[templ]*sn1[templ]);
        superbuf[templ]=superbuf[templ]+128;
    }
    for (templ=(1016-rc); templ<=1016; templ++)
    {
        superbuf[templ]=superbuf[templ]-128;
        superbuf[templ]=(superbuf[templ]*sn2[templ]);
        superbuf[templ]=superbuf[templ]+128;
    }
    net_write(alarm_con, superbuf, 1100, 0);
    ctvm_input(lpVoiceBuf, lBufSize, 8000);
}
```



```

/*Main program begins*/
main()
{
    VIDEO vid;
    int key,i,j;
    long int kt;
    Winport wp,nwp;
    extern char far *near voice_drv;

    /*Network and video setup*/
    a.fhost=nm_res_name("134.164.40.121", (char *) 0,0);
    a.lsocket = 0;
    a.fsocket = SOCK_ALARM_RECV;
    hbuf=(char __huge *)_halloc(556020, sizeof(char));
    alarm_con=net_connect(-1,DGRAM,&a);
    set_option(alarm_con,DGRAM,NET_OPT_TIMEOUT, (char far *)tmo, 4);
    AVinit("video.ini");
    vid = AVcreate(SOURCE);
    wp.width = 162;
    wp.height = 100;
    wp.x = wp.y = 0;
    AVconfigure(vid, &wp, &nwp, 0);
    AVfitMode(vid, FIT_STRETCH);
    AVcolorKey(vid, KEY_COLOR, 0);
    AVenable(vid);

    /*Audio setup*/
    if (!GetEnvSetting())
    {
        if (sbc_check_card() != 4)
        {
            if (sbc_test_int())
            {
                if (sbc_test_dma() >= 0)
                {
                    if (voice_drv=LoadDriver("CT-VOICE.DRV"))
                    {
                        if (!ctvm_init())
                        {
                            lpVoiceBuf=superbuf;
                            ctvm_speaker(0);
                            ctvm_input(lpVoiceBuf, lBufSize, 8000);
                        }
                    }
                }
            }
            else
            {
                printf("Error on DMA channel.\n");
            }
            else
            {
                printf("Error on interrupt.\n");
            }
            else
            {
                printf("Sound Blaster Card not found or wrong I/O settings.\n") ;
            }
        }
        else
        {
            printf("BLASTER environment not set or incomplete or invalid.\n");
        }

        /*Filter setup*/
        rc=80;
        for (templ=16;templ<=(rc+16);templ++)
            sn1[templ]=sin(3.14159*(templ-16.0)/(2.0*rc));
        for (templ=(1016-rc);templ<=1016;templ++)
            sn2[templ]=sin(3.14159*(1016.0-templ)/(2.0*rc));
    }
}

```

```

/*Video statement*/
AVfreeze(vid,FALSE);

/*Packet transmission*/
while (1)
{
    if (ct_voice_status!=-1) raised_cosine_filtering();
    AVgrabToBitmap(vid,FORMAT,NULL,(AVbitmapPtr *)shbuf);
    if (ct_voice_status!=-1) raised_cosine_filtering();
    kt=0L;
    for (i=0;i<=97;i++)
    {
        i++;
        _asm mov ax,1
        _asm mov j,ax
        label2:
        buffer[j+1]= hbuf[kt+j+j+20L];
        _asm inc j
        _asm cmp j,320
        _asm jne label2
        buffer[1]=(char)i;
        kt=kt+640L;
        if (ct_voice_status!=-1) raised_cosine_filtering();
        net_write(alarm_con,buffer,322,0);
        if (ct_voice_status!=-1) raised_cosine_filtering();
        if (_kbhit()!=0) goto label1;
    }
}

/*Job termination*/
label1: ctvm_stop();
    getchar();
    ctvm_terminate();
    _dos_freemem(FP_SEG(lpVoiceBuf));
    AVdestroy(vid);
    AVend();
    net_release(alarm_con);
    _fcloseall();
    return 1;
}

/*Main program ends*/

```

Appendix Y

The Receive Processor

The Receive Processor

```
#include <io.h>
#include <abc.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <string.h>
#include "loaddrv.c"
#include <process.h>
#include <abcvoice.h>
#include <pctcp/types.h>
#include <pctcp/pctcp.h>
#include <pctcp/error.h>
#include <pctcp/asynch.h>
#include <pctcp/types1.h>
#include <pctcp/options.h>
#include <pctcp/ipconfig.h>
#include </ftp/src/alarm/alarm.h>

#define REPLY (struct addr *)&reply_to

/*Globals begin*/
char *lpVoiceBuf;
struct addr reply_to;
long tmo=ALARM_TIMEOUT;
void __far *p, __far *fptr;
unsigned int soundflag=1, offvalue, i, b;
int offd, segd, ourseg, data_length, width1=320;
unsigned char read_buf[1025], read_buf1[1025], read_buf2[1025];
unsigned char read_pal[770], palbuf[256][3], *read_buffer;

FILE *inpalette, *input_file1;
/*Globals end*/

/*Process video packet*/
void buffer_call()
{
    i=(read_buffer+1);
    b=i*width1;
    fptr=read_buffer+2;
    segd=_FP_SEG(fptr);
    offd=_FP_OFF(fptr);
    _asm mov ds, segd
    _asm mov si, offd
    _asm mov es, [ourseg]
    _asm mov di, b
    _asm mov cx, 80
    _asm rep movsw
    b=b+width1;
    fptr=read_buffer+162;
    segd=_FP_SEG(fptr);
    offd=_FP_OFF(fptr);
    _asm mov ds, segd
    _asm mov si, offd
    _asm mov es, [ourseg]
    _asm mov di, b
    _asm mov cx, 80
    _asm rep movsw
}

/*Fix sound header*/
```

```

void sounder()
{
    lpVoiceBuf[0]=9;
    lpVoiceBuf[1]=-12;
    lpVoiceBuf[2]=3;
    lpVoiceBuf[3]=0;
    lpVoiceBuf[4]=30;
    lpVoiceBuf[5]=31;
    lpVoiceBuf[1016]=0;
    if (soundflag==1) ctvm_output(lpVoiceBuf);
}

/*Main program begins*/
main()
{
    int j,k,alarm_con,terminate=TRUE;
    int key,index1,index2,index3,key1;
    extern char far *near voice_drv;
    char string1[80];
    unsigned char cp;
    struct SREGS seg;
    struct addr a;

    /*Network setup*/
    terminate=FALSE;
    segread(&seg);
    a.lsocket=SOCK_ALARM_RECV;
    a.fsocket=0;
    a.fhost=0L;

    /*Audio setup*/
    if (!GetEnvSetting())
    {
        if (sbc_check_card()<4)
        {
            if (sbc_test_int())
            {
                if (sbc_test_dma()>=0)
                {
                    if (voice_drv=LoadDriver("CT-VOICE.DRV"))
                    {
                        ctvm_init();
                    }
                }
            }
            else
            {
                printf("Error on DMA channel.\n");
            }
            else
            {
                printf("Error on interrupt.\n");
            }
            else
            {
                printf("Sound Blaster Card not found or wrong I/O settings.\n") ;
            }
        }
        else
        {
            printf("BLASTER environment not set or incomplete or invalid.\n");
        }

        /*Set 320x200 graphics mode*/
        _asm mov ax,13h
        _asm int 10h

        /*Get a global network descriptor and make network connection*/
        alarm_con=net_getglobdesc();
        set_option(alarm_con,DGRAM,NET_OPT_TIMEOUT,(char far *)tmo,4);
        net_connect(alarm_con,DGRAM,&a);
    }
}

```

```

/*Fill grayscale palette buffer*/
j=0;
for (i=0;i<=255;i++)
{
    palbuf[i][0]=i/4;
    palbuf[i][1]=i/4;
    palbuf[i][2]=i/4;
}

/*Set up palette*/
j=256;
k=0;
p=&palbuf;
offvalue=_FP_OFF(p);
_asm mov dx,offvalue;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;

/*Set a pointer to the beginning of video memory*/
_asm mov [ourseg],0A000h;

/*Determine if the packet is video or audio*/
labelx: data_length=net_read(alarm_con,read_buf,1020,REPLY,0);
        if (data_length>700) goto sound_buffer;
        else goto display_buffer;

/*Process audio packet*/
sound_buffer: lpVoiceBuf=read_buf;
              sounder();

/*Determine if the packet is video or audio*/
labelg: data_length=net_read(alarm_con,read_buf1,1020,REPLY,0);
        if (data_length>700) goto sound_buffer1;
        else goto display_buffer1;

/*Process audio packet*/
sound_buffer1: lpVoiceBuf=read_buf1;
               sounder();

/*Determine if the packet is video or audio*/
labelz: data_length=net_read(alarm_con,read_buf2,1020,REPLY,0);
        if (data_length>700) goto sound_buffer2;
        else goto display_buffer2;

/*Process audio packet*/
sound_buffer2: lpVoiceBuf=read_buf2;
               sounder();
               goto labelx;

/*Process video packet*/
display_buffer: read_buffer=read_buf;
               buffer_call();
               if (_kbhit()!=0) goto labely;
               goto labelx;

/*Process video packet*/
display_buffer1: read_buffer=read_buf1;
                buffer_call();
                goto labelg;

/*Process video packet*/
display_buffer2: read_buffer=read_buf2;
                buffer_call();

```

```

        goto labels;

/*Special functions*/
labely: key=getch();
        switch (key)
        {
            case 'z': /*Job termination*/
                net_release(alarm_con);
                /*Go to Text Mode*/
                _asm mov ax,3
                _asm int 10h;
                _fcloseall();
                ctvm_terminate();
                exit(1);
                break;

            case '1': /*Sound toggle*/
                if (soundflag==0) soundflag=1;
                else soundflag=0;
                break;

            case '2': /*Video capture-quadrant 1*/
                strcpy(string1,"pict.000");
                key1=4;
                index1=index2=index3=0;
                input_file1=fopen(string1,"r+b");
                while (input_file1!=(FILE *) NULL)
                {
                    fclose(input_file1);
                    index1++;
                    if (index1==10)
                    {
                        index1=0;
                        index2++;
                        if (index2==10)
                        {
                            index2=0;
                            index3++;
                        }
                    }
                    string1[key1+3]=index1+'0';
                    string1[key1+2]=index2+'0';
                    string1[key1+1]=index3+'0';
                    input_file1=fopen(string1,"r+b");
                }
                fclose(input_file1);
                input_file1=fopen(string1,"w+b");
                for (i=0;i<=99;i++)
                {
                    for (j=0;j<=159;j++)
                    {
                        b=i*width1;
                        b=b+j;
                        _asm mov es,[ourseq]
                        _asm mov bx,b
                        _asm mov al,es:[bx]
                        _asm mov cp,al
                        b=b+160;
                        _asm mov es,[ourseq]
                        _asm mov al,cp
                        _asm mov bx,b
                        _asm mov es:[bx],al
                        fprintf(input_file1,"%c",cp);
                    }
                }

```

```

        fclose(input_file1);
        break;

case '3': /*Video capture-quadrant 3*/
    strcpy(string1,"pict.000");
    key1=4;
    index1=index2=index3=0;
    input_file1=fopen(string1,"r+b");
    while (input_file1!=(FILE *) NULL)
    {
        fclose(input_file1);
        index1++;
        if (index1==10)
        {
            index1=0;
            index2++;
            if (index2==10)
            {
                index2=0;
                index3++;
            }
        }
        string1[key1+3]=index1+'0';
        string1[key1+2]=index2+'0';
        string1[key1+1]=index3+'0';
        input_file1=fopen(string1,"r+b");
    }
    fclose(input_file1);
    input_file1=fopen(string1,"w+b");
    for (i=0;i<=99;i++)
    {
        for (j=0;j<=159;j++)
        {
            b=i*width1;
            b=b+j;
            _asm mov es,[ourseg]
            _asm mov bx,b
            _asm mov al,es:[bx]
            _asm mov cp,al
            if (width1==640) b=b+320;
            else b=b+width1*100;
            _asm mov es,[ourseg]
            _asm mov al,cp
            _asm mov bx,b
            _asm mov es:[bx],al
            fprintf(input_file1,"%c",cp);
        }
    }
    fclose(input_file1);
    break;

case '4': /*Video capture-quadrant 4*/
    strcpy(string1,"pict.000");
    key1=4;
    index1=index2=index3=0;
    input_file1=fopen(string1,"r+b");
    while (input_file1!=(FILE *) NULL)
    {
        fclose(input_file1);
        index1++;
        if (index1==10)
        {
            index1=0;
            index2++;
            if (index2==10)

```



```

        {
            index2=0;
            index3++;
        }
        string1[key1+3]=index1+'0';
        string1[key1+2]=index2+'0';
        string1[key1+1]=index3+'0';
        input_file1=fopen(string1,"r+b");
    }
    fclose(input_file1);
    input_file1=fopen(string1,"w+b");
    for (i=0;i<=99;i++)
    {
        for (j=0;j<=159;j++)
        {
            b=i*width1;
            b=b+j;
            _asm mov es,[ourseq]
            _asm mov bx,b
            _asm mov al,es:[bx]
            _asm mov cp,al
            if (width1==640) b=b+480;
            else b=b+width1*100+160;
            _asm mov es,[ourseG]
            _asm mov al,cp
            _asm mov bx,b
            _asm mov es:[bx],al
            fprintf(input_file1,"%c",cp);
        }
        fclose(input_file1);
        break;
    }

case '5': /*640*480 graphics mode*/
    _asm mov ax,4f02h;
    _asm mov bx,101h;
    _asm int 10h;

    /*Set up palette*/
    j=256;
    k=0;
    p=&palbuf;
    offvalue=FP_OFF(p);
    _asm mov dx,offvalue
    _asm mov bx,k
    _asm mov cx,j
    _asm mov ax,1012h
    _asm int 10h
    width1 = 640;
    break;

case '6': /*320x200 graphics mode*/
    _asm mov ax,13h
    _asm int 10h

    /*Set up palette*/
    j=256;
    k=0;
    p=&palbuf;
    offvalue=FP_OFF(p);
    _asm mov dx,offvalue
    _asm mov bx,k
    _asm mov cx,j
    _asm mov ax,1012h

```

```
        _asm int 10h
        width1=320;
        break;
    }
    goto labelx;
}
/*Main program ends*/
```

Appendix Z

The Virtual Receive Processor

The Virtual Receive Processor

```
#define SOURCE "DefaultSource"
#include <io.h>
#include <dos.h>
#include <bios.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>

/*Globals begin*/
unsigned char inbuf[450], voice_buffer[2000];
unsigned char buffer[2000], palbuf[770], inpalbuf[770];
char palbuf1[256][3], string1[100], superbuf[10000], newaddr[100];
char store1[10], __huge *hbuf, cp1, cp2, cp3, src_test[6], __far *newadd;

unsigned int typ1, temp1, temp2, temp3, temp5, temp6, b, ourseg, temp4;
int count=0, xdim=320, _intseg, _intoff, flag=0;
union _REGS inregs, outregs;
void __far *fptr, __far *p;
struct _SREGS segregs;

typedef struct {
    unsigned char destaddr[6];
    unsigned char srcaddr[6];
    int pkttype;
    char data[5000];
} buffertype;

buffertype buffer_set;
/*Globals end*/

/*Access network status*/
ip_error()
{
    if (outregs.h.dh==1) printf("Bad Handle\n");
    if (outregs.h.dh==2) printf("No Class\n");
    if (outregs.h.dh==3) printf("No Type\n");
    if (outregs.h.dh==4) printf("No Number\n");
    if (outregs.h.dh==5) printf("Bad Type\n");
    if (outregs.h.dh==6) printf("No Multicast\n");
    if (outregs.h.dh==7) printf("Can't Terminate\n");
    if (outregs.h.dh==8) printf("Bad Mode\n");
    if (outregs.h.dh==9) printf("No Space\n");
    if (outregs.h.dh==10) printf("Type In Use\n");
    if (outregs.h.dh==11) printf("Bad Command\n");
    if (outregs.h.dh==12) printf("Can't Send\n");
    if (outregs.h.dh==13) printf("Can't Set\n");
    if (outregs.h.dh==14) printf("Bad Address\n");
    if (outregs.h.dh==15) printf("Can't Reset\n");
}

/*Interrupt service routine*/
void __interrupt _receiver()
{
    __asm
    {
        cli
        OR ax, ax
        jnz second_call
        mov ax, SEG buffer_set
    }
}
```

```

    mov es,ax
    mov ax,OFFSET buffer_set
    mov di,ax
    mov cx,0800h
    sti
    iret

second_call:
    cli
    mov ax,cx
    cmp ax, 364
    jne ret_label
    xor ax,ax
    mov al, byte ptr buffer_set.data[29]
    mov dx,320
    mul dx
    mov b,ax

    mov ax, SEG buffer_set
    mov ds,ax
    mov ax, OFFSET buffer_set
    add ax,44
    mov si,ax
    mov ax,0A000h
    mov es, ax
    mov di, b
    mov cx, 160
    rep movsb

    mov ax,b
    add ax, 320
    mov b, ax
    mov ax,SEG buffer_set
    mov ds,ax
    mov ax, OFFSET buffer_set
    add ax,204
    mov si,ax
    mov ax,0A000h
    mov es,ax
    mov di, b
    mov cx, 160
    rep movsb
    sti
ret_label: iret
    }
}

/*Main program begins*/
main()
{
    int ix,b,key,type,ouroff,oursegl,rc,j,l,k,int_no,typ,handle,tyipx=80;
    char jc,far *lpVoiceBuf,far *tempx,far *tempy,far *tempz,*ch_prt,tyip[2];
    unsigned char far *string1,cp;
    unsigned int segd,offd;
    long i;

    FILE *outfile,*infile;

    newadd=newaddr;
    tyip[0]='8';
    tyip[1]='0';

    /*Scan interrupts 0x60 to 0x80 to find packet driver*/
    p=4*96;
    int_no=-1;

```

```

for (temp1=0;temp1<32;temp1++)
{
    ourseg1=0;
    _asm mov es,[ourseg1]
    _asm mov bx,b
    _asm mov ax,es:[bx]
    _asm mov ouroff,ax
    _asm inc b
    _asm inc b

    _asm mov bx,b
    _asm mov ax,es:[bx]
    _asm mov ourseg1,ax
    _asm inc b
    _asm inc b

    for (temp2=0;temp2<10;temp2++)
    {
        _asm mov es,[ourseg1]
        _asm mov bx,[ouroff]
        _asm mov al,es:[bx]
        _asm mov cp,al
        _asm inc ouroff
        storel[temp2]=cp;
    }

    if ((storel[3]=='P') && (storel[4]=='K') && (storel[5]=='T'))
    {
        ouroff-=10;
        int_no=temp1+96;
        goto drvinfo;
    }
}

drvinfo:

/*Set address to beginning of video memory*/
ourseg = 0x0A000;

/*Get information*/
inregs.h.ah=0x01;
inregs.h.al=0xff;
_int86x(int_no,&inregs,&outregs,&segregs);
temp1=outregs.x.bx;
cp=outregs.h.ch;
typ=outregs.x.dx;
cp1=outregs.h.cl;
cp2=outregs.h.al;
temp2=segregs.ds;
temp3=outregs.x.si;
printf("Version:  %i\n",temp1);
printf("Class:   %i\n",cp);
printf("Type:    %i\n",typ);
printf("Number:  %i\n",cp1);
printf("Name:    ~");
_FP_SEG(tempx)=temp2;
_FP_OFF(tempx)=temp3;
printf("%s",tempx);
printf("\n");
printf("Functionality:  %i\n",cp2);

/*Get ethernet address of network card*/
getaddr:
inregs.h.ah=6;
segregs.es=_FP_SEG(string1);

```

```

inregs.x.di=_FP_OFF(string1);
inregs.x.cx=12;
_int86x(int_no,&inregs,&outregs,&segregs);
printf("Address: ");
for (i=0;i<5;i++) printf("%02x",string1[i]);
printf("\n");

/*Get ethernet address to a common address*/
setaddr:
newaddr[0]=0x00;
newaddr[1]=0x00;
newaddr[2]=0xC0;
newaddr[3]=0x4E;
newaddr[4]=0xA6;
newaddr[5]=0x2C;
inregs.h.ah=25;
inregs.x.bx=handle;
segregs.es=_FP_SEG(newadd);
inregs.x.di=_FP_OFF(newadd);
inregs.x.cx=6;
_int86x(int_no,&inregs,&outregs,&segregs);

/*Get new ethernet address to insure change*/
getaddr1:
inregs.h.ah=6;
segregs.es=_FP_SEG(string1);
inregs.x.di=_FP_OFF(string1);
inregs.x.cx=12;
_int86x(int_no,&inregs,&outregs,&segregs);
printf("Address: ");
for (i=0;i<5;i++) printf("%02x",string1[i]);
printf("\n");

/*Access type and set up interrupt service routine*/
acctype:
inregs.h.ah=2;
inregs.h.al=1;
inregs.x.bx=typ;
inregs.h.dl=cpl;
segregs.ds=_FP_SEG(typx);
inregs.x.si=_FP_OFF(typx);
inregs.x.cx=0;
outfile=fopen("data.out","w");
fprintf(outfile,"%p",_receiver);
fclose(outfile);
infile=fopen("data.out","r");
fscanf(infile,"%x:%x",&intseg,&intoff);
fclose(infile);
remove("data.out");
segregs.es=_intseg;
inregs.x.di=_intoff+23;
printf("%p \n",_receiver);
_int86x(int_no,&inregs,&outregs,&segregs);
handle=outregs.x.ax;
if (outregs.h.dh != 0) ip_error ();
else printf("Handle %i\n",handle);

/*Set receive mode*/
setrecemode:
inregs.h.ah=20;
inregs.x.bx=handle;
inregs.x.cx=3;
_int86x(int_no,&inregs,&outregs,&segregs);
if (outregs.h.dh != 0) ip_error ();

```

```

/*Get receive mode*/
getrcmode:
inregs.h.ah=21;
inregs.x.bx=handle;
_int86x(int_no,&inregs,&outregs,&segregs);
if (outregs.h.dh != 0) ip_error();
else printf("Receive mode %i\n ",outregs.x.ax);
getch();

/*Set text mode*/
_asm mov ax,13h
_asm int 10h

/*Set palette to grayscale*/
for (k=0;k<=255;k++)
{
    palbuf1[k][0]=k/4;
    palbuf1[k][1]=k/4;
    palbuf1[k][2]=k/4;
}

/*Enable palette*/
j=256;
k=0;
p=&palbuf1;
l=(int)p;
_asm mov dx,l
_asm mov bx,k
_asm mov cx,j
_asm mov ax,1012h
_asm int 10h

/*Polling loop*/
label2:
if (_kbhit() != 0) goto label1;
goto label2;

label1:
key=getch();
switch (key)
{
    case 'z': /*Quit*/
        _asm mov ax,3
        _asm int 10h;

        /*Reset address*/
        setaddrl:
        inregs.h.ah=25;
        segregs.es=_FP_SEG(string1);
        inregs.x.di=_FP_OFF(string1);
        inregs.x.cx=6;
        _int86x(int_no,&inregs,&outregs,&segregs);

        /*Release type*/
        inregs.h.ah=20;
        inregs.x.bx=handle;
        inregs.x.cx=3;
        _int86x(int_no,&inregs,&outregs,&segregs);
        if (outregs.h.dh!=0) ip_error ();

        /*Termination handle*/
        inregs.x.ax=0x0300;
        inregs.x.bx=handle;
        _int86x(int_no,&inregs,&outregs,&segregs);

```



```

        exit(0);
        break;

    case '2': /*Record video frame to quadrant-1*/
        for (i=0;i<=99;i++)
        {
            for (j=0;j<=159;j++)
            {
                b=i*320;
                b=b+j;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov bx,b;
                _asm mov al,es:[bx]
                _asm mov cp,al
                b=b+160;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov al,cp
                _asm mov bx,b;
                _asm mov es:[bx],al
            }
        }
        break;

    case '3': /*Record video frame to quadrant-3*/
        for (i=0;i<=99;i++)
        {
            for (j=0;j<=159;j++)
            {
                b=i*320;
                b=b+j;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov bx,b;
                _asm mov al,es:[bx]
                _asm mov cp,al
                b=b+320*100;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov al,cp
                _asm mov bx,b;
                _asm mov es:[bx],al
            }
        }
        break;

    case '4': /*Record video frame to quadrant-4*/
        for (i = 0;i<=99;i++)
        {
            for (j=0;j<=159;j++)
            {
                b=i*320;
                b=b+j;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov bx,b;
                _asm mov al,es:[bx]
                _asm mov cp,al
                b=b+320*100+160;
                _asm mov ax,0A000h
                _asm mov es,ax
                _asm mov al,cp
                _asm mov bx,b;
            }
        }

```

```
        _asm mov es:[bx],al
    }
    break;
}
/*Continue polling*/
goto label2;
}
/*Main program ends*/
```

Appendix AA

The IMAGE93 Host Processor

The IMAGE93 Host Processor

```
DECLARE SUB box (row!, column!, width!, depth!, fore!, back!)
'include
10 DIM red$(256), green$(256), blue$(256), red(256), green(256), blue(256),
ix(100)
15 DIM a(100)
20 CLS
22 updown = 0: headbits = 0
25 g$ = "gray.pal"
26 SHELL "vesal.exe"
27 OPEN "vesa.ini" FOR INPUT AS #1
28 INPUT #1, vdepth
29 CLOSE #1
30 vdepth = vdepth + 3
31 LOCATE 1, 1: COLOR 15, 1: PRINT "File Edit Video Window Compress De-
compress Enhancement Tools"
35 LOCATE 1, 1: COLOR 0, 7: PRINT "File ": keys = 1
39 GOSUB display_status_box
40 COLOR 7, 0
42 a$ = INKEY$
44 IF a$ = "" THEN GOTO 42
50 IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 77 THEN keys = keys + 1:
GOSUB Print_title_bar
52 IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 75 THEN keys = keys - 1:
GOSUB Print_title_bar
53 IF a$ <> CHR$(13) THEN GOTO 42
54 IF (keys = 1) THEN GOSUB 6000
56 IF (keys = 2) THEN GOSUB 6500
57 IF (keys = 3) THEN GOSUB 7000
58 IF (keys = 4) THEN GOSUB 7500
59 IF (keys = 5) THEN GOSUB 8000
60 IF (keys = 6) THEN GOSUB 8500
61 IF (keys = 7) THEN GOSUB 9000
62 IF (keys = 8) THEN GOSUB 9500
67 GOTO 42

Print_title_bar: IF keys <= 0 THEN keys = 8
4010 IF keys > 8 THEN keys = 1
4015 VIEW PRINT 1 TO 18
4020 LOCATE 1, 1: COLOR 15, 1: PRINT "File Edit Video Window Compress De-
compress Enhancement Tools"
4025 COLOR 0, 7
4030 IF keys = 1 THEN LOCATE 1, 1: PRINT "File "
4040 IF keys = 2 THEN LOCATE 1, 6: PRINT " Edit "
4050 IF keys = 3 THEN LOCATE 1, 12: PRINT " Video "
4060 IF keys = 4 THEN LOCATE 1, 20: PRINT " Window "
4070 IF keys = 5 THEN LOCATE 1, 29: PRINT " Compress "
4080 IF keys = 6 THEN LOCATE 1, 40: PRINT " Decompress "
4085 IF keys = 7 THEN LOCATE 1, 53: PRINT " Enhancement "
4087 IF keys = 8 THEN LOCATE 1, 65: PRINT " Tools "
4090 COLOR 7, 0
4100 RETURN

6000 row = 2: column = 1: width1 = 21: depth = 14: fore = 15: back = 1
6005 VIEW PRINT 2 TO 18
6010 CALL box(row, column, width1, depth, fore, back)
6020 COLOR 7, 0: LOCATE 3, 2: PRINT "Load BIN Image ": select1 = 3
6030 COLOR fore, back: LOCATE 4, 2: PRINT "Display BMP "
6035 LOCATE 5, 2: PRINT "Display PCX "
6040 LOCATE 6, 2: PRINT "Load Palette "
6045 LOCATE 7, 2: PRINT "Save Palette/Gray "
6050 LOCATE 8, 2: PRINT "Convert PCX to BIN"
6070 LOCATE 9, 2: PRINT "Copy File "
```

```

6080 LOCATE 10, 2: PRINT "Rename File      "
6090 LOCATE 11, 2: PRINT "Delete File     "
6097 LOCATE 12, 2: PRINT "Print Image    "
6098 LOCATE 13, 2: PRINT "Shell       "
6100 LOCATE 14, 2: PRINT "Quit       "
6120 a$ = INKEY$
6130 IF a$ = "" THEN GOTO 6120
6132 IF a$ <> CHR$(13) THEN GOTO 6165
6135 IF (select1 = 14) THEN VIEW PRINT: CLS : END
6136 IF (select1 = 3) THEN GOSUB Get_RGB: GOTO 6000
6137 IF (select1 = 4) THEN GOSUB Get_BMP: GOTO 6000
6138 IF (select1 = 6) THEN GOSUB Get_PAL: GOTO 6000
6139 IF (select1 = 13) THEN GOSUB SHELL1: GOTO 6000
6140 IF (select1 = 7) THEN GOSUB palgray: GOTO 6000
6150 IF (select1 = 8) THEN GOSUB conver: GOTO 6000
6152 IF (select1 = 5) THEN GOSUB pcx: GOTO 6000
6155 IF (select1 = 12) THEN GOSUB printf: GOTO 6000
6160 IF ((select1 >= 9) AND (select1 <= 11)) THEN GOSUB copyfile: GOTO 6000
6165 GOSUB std_box_stuff
6180 GOTO 6120
6190 RETURN

6500 row = 2: column = 4: width1 = 30: depth = 15: fore = 15: back = 1
6505 VIEW PRINT 2 TO 18
6510 CALL box(row, column, width1, depth, fore, back)
6520 COLOR 7, 0: LOCATE 3, 5: PRINT "Chop Image      ": se-
lect1 = 3
6530 COLOR fore, back: LOCATE 4, 5: PRINT "View Bit Plane(s) "
6540 LOCATE 5, 5: PRINT "Show RED Palette    "
6550 LOCATE 6, 5: PRINT "Show GREEN Palette  "
6560 LOCATE 7, 5: PRINT "Show BLUE Palette   "
6570 LOCATE 8, 5: PRINT "Show as Grayscale   "
6580 LOCATE 9, 5: PRINT "Subtract Files Byte-by-Byte"
6590 LOCATE 10, 5: PRINT "Add Files Byte-by-Byte "
6600 LOCATE 11, 5: PRINT "Examine an Image     "
6610 LOCATE 12, 5: PRINT "Examine Two Images   "
6612 LOCATE 13, 5: PRINT "Paste Top-to-Bottom  "
6614 LOCATE 14, 5: PRINT "Paste Side-by-Side   "
6616 LOCATE 15, 5: PRINT "Edit DFT File        "
6620 a$ = INKEY$
6630 IF a$ = "" THEN GOTO 6620
6635 IF a$ <> CHR$(13) THEN GOTO 6679
6640 IF (select1 = 3) THEN GOSUB chop_image: GOTO 6500
6650 IF (select1 >= 4) AND (select1 <= 8) THEN GOSUB bit_planes: GOTO 6505
6660 IF (select1 = 9) THEN GOSUB differ: GOTO 6500
6662 IF (select1 = 10) THEN GOSUB differ: GOTO 6500
6664 IF (select1 = 11) THEN GOSUB shell_lookone: GOTO 6500
6666 IF (select1 = 12) THEN GOSUB shell_looktwo: GOTO 6500
6667 IF (select1 = 13) THEN GOSUB paste: GOTO 6500
6668 IF (select1 = 14) THEN GOSUB paste: GOTO 6500
6669 IF (select1 = 15) THEN GOSUB edit_dft: GOTO 6500
6679 GOSUB std_box_stuff
6680 GOTO 6620
6690 RETURN 42

7000 IF keys = 3 THEN row = 2: column = 10: width1 = 21: depth = vdepth: fore =
15: back = 1
7005 VIEW PRINT 2 TO 18
7010 CALL box(row, column, width1, depth, fore, back)
7020 COLOR 7, 0: LOCATE 3, 11: PRINT "320X200  256 Colors": select1 = 3
7030 IF vdepth >= 4 THEN COLOR fore, back: LOCATE 4, 11: PRINT "640X480  256 Col-
ors"
7040 IF vdepth >= 5 THEN COLOR fore, back: LOCATE 5, 11: PRINT "800X600  256 Col-
ors"
7050 IF vdepth >= 6 THEN COLOR fore, back: LOCATE 6, 11: PRINT "1024X768 256 Col-
ors"

```

```

7120 a$ = INKEY$
7130 IF a$ = "" THEN GOTO 7120
7135 IF a$ = CHR$(13) THEN dmode = select1 - 3: GOSUB display_status_box: GOSUB
7000
GOSUB std_box_stuff
7180 GOTO 7120
7190 RETURN

7500 IF keys = 4 THEN row = 2: column = 16: width1 = 21: depth = 3: fore = 15:
back = 1
7505 VIEW PRINT 2 TO 18
7510 CALL box(row, column, width1, depth, fore, back)
7520 IF f$ <> "" THEN COLOR 7, 0: LOCATE 3, column + 1: PRINT "Display
Image": select1 = 3
7620 a$ = INKEY$
7630 IF a$ = "" THEN GOTO 7620
7635 IF (a$ = CHR$(13)) AND (f$ <> "") THEN GOSUB Load_TFILE: GOTO 7505
GOSUB std_box_stuff
7680 GOTO 7620
7690 RETURN

8000 row = 2: column = 24: width1 = 22: depth = 12: fore = 15: back = 1
8005 VIEW PRINT 2 TO 18
8010 CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE 3, column + 1: PRINT "Huffman": select1 =
3
COLOR fore, back: LOCATE 4, column + 1: PRINT "Adaptive Huffman"
LOCATE 5, column + 1: PRINT "Arithmetic Order 0"
LOCATE 6, column + 1: PRINT "Arithmetic Order 1"
LOCATE 7, column + 1: PRINT "Arithmetic Order 2"
LOCATE 8, column + 1: PRINT "LZW (ZIP)"
LOCATE 9, column + 1: PRINT "LZW (LHARC)"
LOCATE 10, column + 1: PRINT "Cosine"
LOCATE 11, column + 1: PRINT "Sine"
LOCATE 12, column + 1: PRINT "Hadamard"
8120 a$ = INKEY$
8130 IF a$ = "" THEN GOTO 8120
8135 IF a$ <> CHR$(13) THEN GOTO 8162
8140 IF (select1 = 8) THEN GOSUB SHELL_zip: GOTO 8000
8142 IF (select1 = 4) THEN GOSUB SHELL_ahuff: GOTO 8000
8144 IF (select1 = 3) THEN GOSUB SHELL_huff: GOTO 8000
8146 IF (select1 = 5) THEN GOSUB SHELL_ari0: GOTO 8000
8148 IF (select1 = 6) THEN GOSUB SHELL_ari1: GOTO 8000
8150 IF (select1 = 7) THEN GOSUB SHELL_ari2: GOTO 8000
8152 IF (select1 = 9) THEN GOSUB SHELL_lharc: GOTO 8000
8154 IF (select1 = 10) THEN GOSUB dct_box: GOTO 8000
8156 IF (select1 = 11) THEN GOSUB dct_box: GOTO 8000
8158 IF (select1 = 12) THEN GOSUB dct_box: GOTO 8000
8162 GOSUB std_box_stuff
8180 GOTO 8120
8190 RETURN

8500 IF keys = 6 THEN row = 2: column = 35: width1 = 22: depth = 12: fore = 15:
back = 1
8505 VIEW PRINT 2 TO 18
8510 CALL box(row, column, width1, depth, fore, back)
8520 COLOR 7, 0: LOCATE 3, column + 1: PRINT "Huffman": se-
lect1 = 3
8530 COLOR fore, back: LOCATE 4, column + 1: PRINT "Adaptive Huffman"
8540 LOCATE 5, column + 1: PRINT "Arithmetic Order 0"
8542 LOCATE 6, column + 1: PRINT "Arithmetic Order 1"
8544 LOCATE 7, column + 1: PRINT "Arithmetic Order 2"
8550 LOCATE 8, column + 1: PRINT "LZW (ZIP)"
8552 LOCATE 9, column + 1: PRINT "LZW (LHARC)"
8560 LOCATE 10, column + 1: PRINT "Cosine"

```

```

8570 LOCATE 11, column + 1: PRINT "Sine"
8580 LOCATE 12, column + 1: PRINT "Hadamard"
8620 a$ = INKEY$
8630 IF a$ = "" THEN GOTO 8620
8635 IF a$ <> CHR$(13) THEN GOTO 8675
8640 IF (select1 = 8) THEN GOSUB SHELL_unzip: GOTO 8500
8642 IF (select1 = 4) THEN GOSUB SHELL_aunhuff: GOTO 8500
8644 IF (select1 = 3) THEN GOSUB SHELL_unhuff: GOTO 8500
8646 IF (select1 = 5) THEN GOSUB SHELL_unari0: GOTO 8500
8648 IF (select1 = 6) THEN GOSUB SHELL_unari1: GOTO 8500
8650 IF (select1 = 7) THEN GOSUB SHELL_unari2: GOTO 8500
8652 IF (select1 = 9) THEN GOSUB SHELL_unlharc: GOTO 8500
8654 IF (select1 = 10) THEN GOSUB invdct_box: GOTO 8500
8656 IF (select1 = 11) THEN GOSUB invdct_box: GOTO 8500
8658 IF (select1 = 12) THEN GOSUB invdct_box: GOTO 8500
8675 GOSUB std_box_stuff
8680 GOTO 8620
8690 RETURN

9000 row = 2: column = 50: width1 = 27: depth = 11: fore = 15: back = 1
9005 VIEW PRINT 2 TO 18
9010 CALL box(row, column, width1, depth, fore, back)
9020 COLOR 7, 0: LOCATE 3, column + 1: PRINT "2-D Fourier.....":
select1 = 3
9030 COLOR fore, back: LOCATE 4, column + 1: PRINT "Singular Valued Decomp..":
""
9031 LOCATE 5, column + 1: PRINT "Detect Edges....."
9032 LOCATE 6, column + 1: PRINT "Noise Cleaning....."
9033 LOCATE 7, column + 1: PRINT "Crispen Image....."
9035 LOCATE 8, column + 1: PRINT "Contrast Manipulation..."
9040 LOCATE 9, column + 1: PRINT "Morphological....."
9051 LOCATE 10, column + 1: PRINT "Median Filter....."
9060 LOCATE 11, column + 1: PRINT "Image Thresholding....."

9120 a$ = INKEY$
9130 IF a$ = "" THEN GOTO 9120
9132 IF a$ <> CHR$(13) THEN GOTO 9111
9135 IF (select1 = 6) THEN GOSUB noiseclean: GOTO 9000
9140 IF (select1 = 7) THEN GOSUB convolution: GOTO 9000
9042 IF (select1 = 5) THEN GOSUB edge: GOTO 9000
9044 IF (select1 = 9) THEN GOSUB dilating: GOTO 9000
9050 IF (select1 = 4) THEN GOSUB SVD: GOTO 9000
9052 IF (select1 = 3) THEN GOSUB Fourier: GOTO 9000
9054 IF (select1 = 8) THEN GOSUB contrast: GOTO 9000
9057 IF (select1 = 11) THEN GOSUB thresher: GOTO 9000
9058 IF (select1 = 10) THEN GOSUB median: GOTO 9000
9111 GOSUB std_box_stuff
9180 GOTO 9120
9190 RETURN

9500 IF keys = 8 THEN row = 2: column = 55: width1 = 22: depth = 6: fore = 15:
back = 1
9505 VIEW PRINT 2 TO 18
9510 CALL box(row, column, width1, depth, fore, back)
9520 COLOR 7, 0: LOCATE 3, column + 1: PRINT "Calculate Entropy ": se-
lect1 = 3
9530 COLOR fore, back: LOCATE 4, column + 1: PRINT "Mean Square Error"
9540 LOCATE 5, column + 1: PRINT "Histogram"
9609 LOCATE 6, column + 1: PRINT "Load Macros"
9620 a$ = INKEY$
9630 IF a$ = "" THEN GOTO 9620
9640 IF (select1 = 3) AND (a$ = CHR$(13)) THEN GOSUB entropy: GOTO 9500
9645 IF (select1 = 4) AND (a$ = CHR$(13)) THEN GOSUB mse: GOTO 9500
9650 IF (select1 = 5) AND (a$ = CHR$(13)) THEN GOSUB histogram: GOTO 9500
9669 IF (select1 = 6) AND (a$ = CHR$(13)) THEN GOSUB macro: GOTO 9500

```

```

GOSUB std_box_stuff
9680 GOTO 9620
9690 RETURN

11000 REM
11010 FOR i = column + 1 TO column + width1 - 2
11020 ix(i) = SCREEN(select1, i)
11030 NEXT i
11040 FOR i = column + 1 TO column + width1 - 2
11050 LOCATE select1, i: COLOR fore, back: PRINT CHR$(ix(i))
11060 NEXT
11070 IF select1 >= (row + depth - 2) THEN select1 = (row + 1) ELSE select1 = se-
lect1 + 1
11080 FOR i = column + 1 TO column + width1 - 2
11090 ix(i) = SCREEN(select1, i): NEXT
11100 FOR i = column + 1 TO column + width1 - 2
11110 LOCATE select1, i: COLOR 7, 0: PRINT CHR$(ix(i))
11120 NEXT
11130 RETURN

Get_RGB:
11200 REM subroutine for collecting RGB data and displaying image
11205 headbits = 0: updown = 0
11210 rowx = 3: columnx = 22: widthlx = 58: depthx = 6: forex = 15: backx = 1
11220 VIEW PRINT 2 TO 24
11230 CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11240 LOCATE rowx + 1, columnx + 1: INPUT "Type Image Filename "; f$
11242 IF f$ = "" THEN forex = 0: backx = 0
11244 IF f$ = "" THEN CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11246 IF f$ = "" THEN GOSUB display_status_box: GOTO 11285
11250 LOCATE rowx + 2, columnx + 1: INPUT "Type X Dimension "; xdim
11260 LOCATE rowx + 3, columnx + 1: INPUT "Type Y Dimension "; ydim
11262 IF f$ = "" THEN forex = 0: backx = 0
11264 IF f$ = "" THEN CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11270 GOSUB display_status_box 'Print these in status box
11280 GOSUB Load_TFILE 'load tfile and display image
11285 VIEW PRINT 2 TO 18
11290 RETURN

macro:
20200 OPEN "macro.lst" FOR INPUT AS #1
20205 ic = 0
20210 ic = ic + 1
IF EOF(1) THEN GOTO 20260
20220 LINE INPUT #1, a$(ic)
20230 LINE INPUT #1, b$(ic)
20250 GOTO 20210
20260 CLOSE #1
ic = ic - 1
row = 2: column = 19: width1 = 36: depth = ic + 2: fore = 15: back = 1
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0:
FOR i = 1 TO ic
LOCATE row + 1, column + 1: PRINT a$(i); : select1 = row + 1: COLOR fore, back
NEXT i
rep64: a$ = INKEY$
IF a$ = "" THEN GOTO rep64:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF a$ = CHR$(13) THEN GOSUB macrol: GOSUB ent64: GOSUB clear_box: RETURN
GOSUB std_box_stuff
GOTO rep64

ent64: row = 2: column = 19: width1 = 36: depth = ic + 1: fore = 15: back = 1
RETURN

```



```

macrol:
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, f$
PRINT #1, g$
PRINT #1, xdim
PRINT #1, ydim
CLOSE #1
SHELL b$(ic)
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

pcx:
REM subroutine for collecting PCX data and displaying image
'headbits = 0: updown = 0
rowx = 3: columnx = 22: widthlx = 58: depthx = 6: forex = 15: backx = 1
VIEW PRINT 2 TO 24
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE rowx + 1, columnx + 1: INPUT "Type Image Filename "; f$
IF f$ = "" THEN forex = 0: backx = 0
IF f$ = "" THEN CALL box(rowx, columnx, widthlx, depthx, forex, backx)
IF f$ = "" THEN GOSUB display_status_box: RETURN
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, f$
CLOSE #1
SHELL "PCXLOAD" "loadgif"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

printf:
REM subroutine for printing image to laserprinter
headbits = 0: updown = 0
IF f$ = "" THEN forex = 0: backx = 0
IF f$ = "" THEN CALL box(rowx, columnx, widthlx, depthx, forex, backx)
IF f$ = "" THEN GOSUB display_status_box: RETURN
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, f$
PRINT #1, g$
PRINT #1, xdim
PRINT #1, ydim
CLOSE #1
SHELL "printb"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

Get_BMP:
11300 REM subroutine for collecting BMP data and displaying image
11310 rowx = 3: columnx = 22: widthlx = 58: depthx = 4: forex = 15: backx = 1
11320 VIEW PRINT 2 TO 24
11330 CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11340 LOCATE rowx + 1, columnx + 1: INPUT "Type Image Filename "; f$
11342 forex = 0: backx = 0
11344 CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11346 IF f$ = "" THEN GOSUB display_status_box: GOTO 11390
11350 GOSUB Get_BMP_file 'Get BMP file
11370 GOSUB display_status_box 'Print these in status box
11380 GOSUB Load_TFILE 'load tfile and display image
11390 VIEW PRINT 2 TO 18
11395 RETURN

```

```

display_status_box:
11400 VIEW PRINT: rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15:
backz = 4
11402 CALL box(rowz, columnz, widthlz, depthz, forez, backz)
11404 LOCATE 20, 30: COLOR 15, 4: PRINT "Status Window"
11405 LOCATE 21, 12: COLOR 15, 4: IF dmode = 0 THEN PRINT "320 X 200 Video Mode"
11410 IF dmode = 1 THEN PRINT "640 X 480 Video Mode"
11420 IF dmode = 2 THEN PRINT "800 X 600 Video Mode"
11430 IF dmode = 3 THEN PRINT "1024 X 768 Video Mode"
11440 LOCATE 21, 45: PRINT "Image is "; f$
11450 LOCATE 22, 12: PRINT "Image Dimensions are "; xdim; ydim
11460 LOCATE 22, 45: PRINT "Palette is "; g$
11590 RETURN

Get_PAL:
11600 REM subroutine for collecting Palette
11610 rowx = 3: columnx = 22: widthlx = 58: depthx = 4: forex = 15: backx = 1
11620 VIEW PRINT 2 TO 24
11630 CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11640 LOCATE rowx + 1, columnx + 1: INPUT "Type Palette "; g$
11642 forex = 0: backx = 0
11644 CALL box(rowx, columnx, widthlx, depthx, forex, backx)
11670 GOSUB display_status_box 'Print these in status box
11675 VIEW PRINT 2 TO 18
11690 RETURN

12000 REM
12010 FOR i = column + 1 TO column + widthl - 2
12020 ix(i) = SCREEN(selectl, i)
12030 NEXT i
12040 FOR i = column + 1 TO column + widthl - 2
12050 LOCATE selectl, i: COLOR fore, back: PRINT CHR$(ix(i))
12060 NEXT
12070 IF selectl <= (row + 1) THEN selectl = (row + depth - 2) ELSE selectl = se-
lectl - 1
12080 FOR i = column + 1 TO column + widthl - 2
12090 ix(i) = SCREEN(selectl, i): NEXT
12100 FOR i = column + 1 TO column + widthl - 2
12110 LOCATE selectl, i: COLOR 7, 0: PRINT CHR$(ix(i))
12120 NEXT
12130 RETURN

Load_TFILE:
12200 REM "Load up Tfile and Shell to Ellis1.exe
12280 OPEN "tfile" FOR OUTPUT AS #1
12290 PRINT #1, dmode + 1
12300 PRINT #1, g$
12310 PRINT #1, f$
12320 PRINT #1, xdim
12330 PRINT #1, ydim
12340 PRINT #1, headbits 'headerbits to skip is zero
12350 PRINT #1, updown 'start on first line
12355 CLOSE #1
12360 SHELL "display.exe"
12370 VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
12375 GOSUB Print_title_bar
12380 RETURN

Get_BMP_file:
21000 OPEN f$ FOR BINARY AS #1
21005 a$ = INPUT$(1, 1): b$ = INPUT$(1, 1)
21010 a$ = INPUT$(1, 1)
21020 b$ = INPUT$(1, 1)
21030 c$ = INPUT$(1, 1)

```

```

21040 d$ = INPUT$(1, 1)
21050 faise = (ASC(a$) + ASC(b$) * 256 + ASC(C$) * 256 ^ 2 + ASC(d$) * 256 ^ 3)
21060 a$ = INPUT$(1, 1)
21070 b$ = INPUT$(1, 1)
21080 a$ = INPUT$(1, 1): b$ = INPUT$(1, 1): C$ = INPUT$(1, 1): d$ = INPUT$(1, 1)
21090 headbits = (ASC(C$) + ASC(d$) * 256)
21100 a$ = INPUT$(1, 1): b$ = INPUT$(1, 1): C$ = INPUT$(1, 1): d$ = INPUT$(1, 1)
21110 a$ = INPUT$(1, 1): b$ = INPUT$(1, 1): C$ = INPUT$(1, 1): d$ = INPUT$(1, 1)
21120 xdim = ASC(C$) + 256 * ASC(d$)
21130 a$ = INPUT$(1, 1): b$ = INPUT$(1, 1): C$ = INPUT$(1, 1): d$ = INPUT$(1, 1)
21140 ydim = ASC(C$) + 256 * ASC(d$)
21150 updown = 1
21160 g$ = "bmp.pal"
21170 FOR i = 1 TO 5
21180 a$ = INPUT$(1, 1)
21190 NEXT i
21200 FOR i = 1 TO 25
21210 a$ = INPUT$(1, 1)
21220 IF i = 18 THEN numcolors = ASC(a$)
21230 NEXT i
21240 OPEN g$ FOR OUTPUT AS #2
21250 FOR j = 1 TO numcolors
21260 blue$(j) = INPUT$(1, 1): green$(j) = INPUT$(1, 1): red$(j) = INPUT$(1, 1): a$ = INPUT$(1, 1)
21270 NEXT j
21280 IF numcolors = 256 THEN GOTO 21400
21290 FOR j = numcolors + 1 TO 256
21300 blue$(j) = CHR$(0): green$(j) = CHR$(0): red$(j) = CHR$(0)
21310 NEXT j
21400 FOR j = 1 TO 256
21410 PRINT #2, red$(j);
21420 NEXT j
21430 FOR j = 1 TO 256
21440 PRINT #2, green$(j);
21450 NEXT j
21460 FOR j = 1 TO 256
21470 PRINT #2, blue$(j);
21480 NEXT j
21490 CLOSE #2
21495 CLOSE #1
21498 GOSUB display_status_box
21500 RETURN

```

```

median:
blocksize = 3
ent29: row = 7: column = 12: width1 = 38: depth = 8: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As....."; : select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Type Palette = "; g$
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Type Input Filename = "; f$
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Type X-Dimension = "; xdim
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Type Y-Dimension = "; ydim
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Type Blocksize (odd) = "; blocksize
rep29: a$ = INKEY$
IF a$ = "" THEN GOTO rep29:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RETURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB shell_median: GOSUB ent49: GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q29: GOTO ent29
GOSUB std_box_stuff
GOTO rep29

```

```

ent49: row = 7: column = 4: width1 = 46: depth = 8: fore = 15: back = 1
      RETURN

q29:
REM "Load up Tfile and get ready to shell to median.exe
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Palette "; g$
IF select1 = (row + 3) THEN INPUT "Type Input Filename "; f$
IF select1 = (row + 4) THEN INPUT "Type X-Dimension of Filename "; xdim
IF (select1 = (row + 5)) THEN INPUT "Type Y-Dimension of Filename "; ydim
IF (select1 = (row + 6)) THEN INPUT "Type Blocksize (must be odd) "; blocksize
IF blocksize MOD 2 = 0 THEN blocksize = blocksize + 1
GOSUB display_status_box
RETURN

shell_median:
REM "Load up Tfile and Shell to MEDIAN.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, f$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, header
PRINT #1, updown
PRINT #1, h2$
PRINT #1, blocksize
CLOSE #1
SHELL "median.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

RETURN

edgel:
ent53: row = 7: column = 28: width1 = 22: depth = 5: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Laplace 4 Neighbor"; : select1 =
row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Prewitt 8 Neighbor"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Separable 8 Neighbor"
GOSUB second3_box
fin = 0
repax: a$ = INKEY$
IF a$ = "" THEN GOTO repax
oldsel = select1
fin = 0
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent54: GOSUB
clear_box: RETURN

```

```

IF (a$ = CHR$(13)) THEN GOSUB convoll: select1 = oldsel: IF fin = 1 THEN RETURN
GOSUB std_box_stuff
GOSUB second3_box
GOTO repax

```

```

ent54: row = 6: column = 1: width1 = 50: depth = 11: fore = 15: back = 1
RETURN

```

```

second3_box:
IF select1 = 8 THEN h1$ = "laplace4"
IF select1 = 9 THEN h1$ = "previtt8"
IF select1 = 10 THEN h1$ = "sep8"
biasflag = 1
CALL box(6, 1, 24, 11, 0, 0)
OPEN h1$ FOR INPUT AS #1
INPUT #1, n
rown = 6: columnn = 24 - 5 * n: widthn = 28 - columnn: depthn = n + 2
CALL box(rown, columnn, widthn, depthn, 15, 1)
k = n
FOR i = 1 TO n
FOR j = 1 TO k
INPUT #1, a: LOCATE 6 + i, columnn - 4 + 5 * j: PRINT a
NEXT j
NEXT i
CLOSE #1
RETURN

```

```

std_box_stuff: IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN
COLOR 7, 0: CLS
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN RETURN 42
IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 80 THEN GOSUB 11000
IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 72 THEN GOSUB 12000
IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 77 THEN keys = keys + 1:
COLOR 7, 0: CLS : GOSUB Print_title_bar: RETURN 54
IF ASC(LEFT$(a$, 1)) = 0 AND ASC(RIGHT$(a$, 1)) = 75 THEN keys = keys - 1:
COLOR 7, 0: CLS : GOSUB Print_title_bar: RETURN 54
RETURN

```

```

thresher:
row = 5: column = 33: width1 = 17: depth = 4: fore = 15: back = 1
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "By Occurrence": select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "By Value"
repmx: a$ = INKEY$
IF a$ = "" THEN GOTO repmx
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (a$ = CHR$(13)) THEN GOTO labelmx
GOSUB std_box_stuff
GOTO repmx
labelmx:
savesel = select1

```

```

h1$ = f$
entmx: row = 9: column = 5: width1 = 45: depth = 7: fore = 15: back = 1
IF savesel = 7 THEN depth = 8
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As...": : select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Width="; xdim
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Height="; ydim
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Input Filename = "; h1$
IF savesel = 6 THEN COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Occur-
rence Threshold = "; occth

```

```

IF savesel = 7 THEN COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Upper
Threshold = "; upperth
IF savesel = 7 THEN COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Lower
Threshold = "; lowerth

```

```

repm: a$ = INKEY$
IF a$ = "" THEN GOTO repm:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent46: GOSUB
clear_box: RETURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB thres: GOSUB ent46:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q26: GOTO entmx
GOSUB std_box_stuff
GOTO repm

```

```

ent46: row = 2: column = 5: width1 = 45: depth = 15: fore = 15: back = 1
RETURN

```

```

q26:
REM "Load up Tfile and Shell dilate
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Width"; xdim
IF select1 = (row + 3) THEN INPUT "Type Height"; ydim
IF select1 = (row + 4) THEN INPUT "Type Input Filename "; h1$
IF (savesel = 6) AND select1 = (row + 5) THEN INPUT "Type Occurrence Threshold
"; occth
IF (savesel = 7) AND select1 = (row + 5) THEN INPUT "Type Upper Threshold "; up-
perth
IF (savesel = 7) AND select1 = (row + 6) THEN INPUT "Type Lower Threshold ";
lowerth
GOSUB display_status_box
RETURN

```

```

thres:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, h1$
PRINT #1, h2$
IF savesel = 6 THEN PRINT #1, occth
IF savesel = 7 THEN PRINT #1, upperth
IF savesel = 7 THEN PRINT #1, lowerth
CLOSE #1
IF savesel = 6 THEN SHELL "thresbyo.exe"
IF savesel = 7 THEN SHELL "thresbyv.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

```

```

contrast:

```

```

row = 2: column = 21: width1 = 29: depth = 9: fore = 15: back = 1
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Extract Image": select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Square Normalization"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Cube Normalization"
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Square Root Normalization"
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Cube Root Normalization"
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Inverse Normalization"
COLOR fore, back: LOCATE row + 7, column + 1: PRINT "Invert"
repmn: a$ = INKEY$
IF a$ = "" THEN GOTO repmn
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (a$ = CHR$(13)) THEN GOTO labelmn
GOSUB std_box_stuff
GOTO repmn
labelmn:
saveel = select1

hl$ = f$
entmn: row = 11: column = 5: width1 = 45: depth = 6: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As...": : select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Width="; xdim
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Height="; ydim
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Input Filename = "; hl$
repmo: a$ = INKEY$
IF a$ = "" THEN GOTO repmo:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent45: GOSUB
clear_box: RETURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB morp: GOSUB ent45:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q25: GOTO entmn
GOSUB std_box_stuff
GOTO repmo

ent45: row = 2: column = 5: width1 = 45: depth = 16: fore = 15: back = 1
RETURN

q25:
REM "Load up Tfile and Shell dilate
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Width"; xdim
IF select1 = (row + 3) THEN INPUT "Type Height"; ydim
IF select1 = (row + 4) THEN INPUT "Type Input Filename "; hl$
GOSUB display_status_box
RETURN

morp:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS

```

```

OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, h1$
PRINT #1, h2$
CLOSE #1
IF savesel = 3 THEN SHELL "extract.exe"
IF savesel = 4 THEN SHELL "squareh.exe"
IF savesel = 5 THEN SHELL "cubeh.exe"
IF savesel = 6 THEN SHELL "sqrth.exe"
IF savesel = 7 THEN SHELL "cuberth.exe"
IF savesel = 8 THEN SHELL "invh.exe"
IF savesel = 9 THEN SHELL "revh.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

Fourier:
SVD$ = "DFT"
h1$ = f$
ent24: row = 7: column = 19: width1 = 31: depth = 6: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Input and Output Files ...": : se-
lect1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Fourier =": SVD$: " <RET>"
REM COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Input Filename = ": h1$
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Type X-Dimension = ": xdim
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Type Y-Dimension = ": ydim
rep24: a$ = INKEY$
IF a$ = "" THEN GOTO rep24:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB fourier1: GOSUB ent44:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q24: GOTO ent24
GOSUB std_box_stuff
GOTO rep24

ent44: row = 7: column = 19: width1 = 31: depth = 6: fore = 15: back = 1
RETURN

q24:
REM "Load up Tfile and Shell dilate
IF (select1 = (row + 2)) AND (SVD$ = "DFT") THEN SVD$ = "Inverse DFT": RETURN
IF (select1 = (row + 2)) AND (SVD$ = "Inverse DFT") THEN SVD$ = "FFT": RETURN
IF (select1 = (row + 2)) AND (SVD$ = "FFT") THEN SVD$ = "Inverse FFT": RETURN
IF (select1 = (row + 2)) AND (SVD$ = "Inverse FFT") THEN SVD$ = "DFT": RETURN
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
tr: LOCATE 20, 2
IF select1 = (row + 3) THEN INPUT "Type X-Dimension of Filename that contains
the transform coefficients ": xdim
IF (SVD$ = "DFT") OR (SVD$ = "Inverse DFT") THEN GOTO stk
IF (select1 = (row + 3)) AND INT(LOG(xdim) / LOG(2)) <> (LOG(xdim) / LOG(2))
THEN BEEP: LOCATE 21, 2: PRINT "Must be power of 2": GOTO tr
stk: IF (select1 = (row + 4)) THEN INPUT "Type Y-Dimension of Filename that con-
tains the transform coefficients ": ydim
IF (SVD$ = "DFT") OR (SVD$ = "Inverse DFT") THEN GOTO stl
IF (select1 = (row + 4)) AND INT(LOG(ydim) / LOG(2)) <> (LOG(ydim) / LOG(2))
THEN BEEP: PRINT "Must be power of 2": GOTO stk
stl: GOSUB display_status_box

```


RETURN

fourier1:

REM "Load up Tfile and Shell to DCT.EXE

VIEW PRINT 18 TO 24

COLOR 7, 0

rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4

CALL box(rowz, columnz, widthlz, depthz, forez, backz)

IF (SVD\$ = "DFT") THEN LOCATE 20, 2: INPUT "Type Input Filename "; h1\$

IF (SVD\$ = "FFT") THEN LOCATE 20, 2: INPUT "Type Input Filename "; h1\$

IF (SVD\$ = "Inverse DFT") THEN LOCATE 20, 2: INPUT "Type Real Input Filename "; h1\$

IF (SVD\$ = "Inverse FFT") THEN LOCATE 20, 2: INPUT "Type Real Input Filename "; h1\$

IF h1\$ = "" THEN row = 7: column = 46: widthl = 22: depth = 7: fore = 15: back = 1: GOSUB clear_box

IF h1\$ = "" THEN GOSUB display_status_box: RETURN

IF (SVD\$ = "DFT") THEN LOCATE 21, 2: INPUT "Type Real Output Filename "; h2\$

IF (SVD\$ = "FFT") THEN LOCATE 21, 2: INPUT "Type Real Output Filename "; h2\$

IF (SVD\$ = "Inverse DFT") THEN LOCATE 21, 2: INPUT "Type Imaginary Input Filename "; h2\$

IF (SVD\$ = "Inverse FFT") THEN LOCATE 21, 2: INPUT "Type Imaginary Input Filename "; h2\$

IF h2\$ = "" THEN row = 7: column = 46: widthl = 22: depth = 7: fore = 15: back = 1: GOSUB clear_box

IF h2\$ = "" THEN GOSUB display_status_box: RETURN

IF (SVD\$ = "DFT") THEN LOCATE 22, 2: INPUT "Type Imaginary Output Filename "; h3\$

IF (SVD\$ = "FFT") THEN LOCATE 22, 2: INPUT "Type Imaginary Output Filename "; h3\$

IF (SVD\$ = "Inverse DFT") THEN LOCATE 22, 2: INPUT "Type Output Filename "; h3\$

IF (SVD\$ = "Inverse FFT") THEN LOCATE 22, 2: INPUT "Type Output Filename "; h3\$

IF h3\$ = "" THEN row = 7: column = 46: widthl = 22: depth = 7: fore = 15: back = 1: GOSUB clear_box

IF h3\$ = "" THEN GOSUB display_status_box: RETURN

COLOR 7, 0

CLS

OPEN "tfile" FOR OUTPUT AS #1

PRINT #1, dmode + 1

PRINT #1, xdim

PRINT #1, ydim

PRINT #1, h1\$

PRINT #1, h2\$

PRINT #1, h3\$

CLOSE #1

IF SVD\$ = "DFT" THEN SHELL "dft.exe"

IF SVD\$ = "Inverse DFT" THEN SHELL "invdft.exe"

IF SVD\$ = "FFT" THEN SHELL "fft.exe"

IF SVD\$ = "Inverse FFT" THEN SHELL "invfft.exe"

VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box

GOSUB Print_title_bar

RETURN

SVD:

SVD\$ = "SVD"

h1\$ = f\$

ent23: row = 7: column = 19: widthl = 31: depth = 6: fore = 15: back = 1

VIEW PRINT 2 TO 18

CALL box(row, column, widthl, depth, fore, back)

COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Input and Output Files..."; : select1 = row + 1

COLOR fore, back: LOCATE row + 2, column + 1: PRINT "SVD ="; SVD\$; " <RET>"

COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Type X-Dimension = "; xdim

COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Type Y-Dimension = "; ydim

```

rep23: a$ = INKEY$
IF a$ = "" THEN GOTO rep23:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB svd1: GOSUB ent43:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q23: GOTO ent23
GOSUB std_box_stuff
GOTO rep23

ent43: row = 7: column = 19: width1 = 31: depth = 8: fore = 15: back = 1
RETURN

q23:
REM "Load up Tfile and Shell dilate
IF (select1 = (row + 2)) AND (SVD$ = "SVD") THEN SVD$ = "Inverse SVD": RETURN
IF (select1 = (row + 2)) AND (SVD$ = "Inverse SVD") THEN SVD$ = "SVD": RETURN
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 3) THEN INPUT "Type X-Dimension of Filename that contains
the transform coefficients "; xdim
IF (select1 = (row + 4)) THEN INPUT "Type Y-Dimension of Filename that contains
the transform coefficients "; ydim
GOSUB display_status_box
RETURN

svd1:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
IF (SVD$ <> "SVD") THEN GOTO xi
LOCATE 20, 2: INPUT "Type Input Filename "; h1$
IF h1$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h1$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type U Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type V Filename to Save "; h3$
IF h3$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h3$ = "" THEN GOSUB display_status_box: RETURN
GOTO xj

xi:
LOCATE 20, 2: INPUT "Type U Filename "; h1$
IF h1$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h1$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type V Filename"; h2$
IF h2$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN

xj:
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1

```

```

PRINT #1, xdim
PRINT #1, ydim
PRINT #1, h1$
PRINT #1, h2$
PRINT #1, h3$
CLOSE #1
IF SVD$ = "SVD" THEN SHELL "SVD.exe"
IF SVD$ = "Inverse SVD" THEN SHELL "invsvd.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

edit_dft:
ent22: row = 7: column = 35: width1 = 31: depth = 7: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Real Input File = "; h1$:
select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Imag Input File = "; h2$
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Type X-Dimension = "; xdim
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Type Y-Dimension = "; ydim
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "DO IT"

rep22: a$ = INKEY$
IF a$ = "" THEN GOTO rep22:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 <> (row + 5)) AND (a$ = CHR$(13)) THEN GOSUB q22: GOTO ent22
IF (select1 = (row + 5)) AND (a$ = CHR$(13)) THEN GOSUB editdft: GOSUB ent42:
GOSUB clear_box: RETURN
GOSUB std_box_stuff
GOTO rep22

ent42: row = 7: column = 4: width1 = 31: depth = 8: fore = 15: back = 1
RETURN

q22:
REM "Load up Tfile and Shell dilate
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 1) THEN INPUT "Real Input Filename = "; h1$
IF select1 = (row + 2) THEN INPUT "Imag Input Filename "; h2$
IF select1 = (row + 3) THEN INPUT "Type X-Dimension of Filename that contains
the transform coefficients "; xdim
IF (select1 = (row + 4)) THEN INPUT "Type Y-Dimension of Filename that contains
the transform coefficients "; ydim
GOSUB display_status_box
RETURN

editdft:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, h1$
PRINT #1, h2$

```

```

CLOSE #1
SHELL "editdft.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

dilating:
row = 7: column = 39: width1 = 11: depth = 6: fore = 15: back = 1
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Dilating": select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Eroding"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Opening"
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Closing"

repqt: a$ = INKEY$
IF a$ = "" THEN GOTO repqt
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (a$ = CHR$(13)) THEN GOTO labelpx
GOSUB std_box_stuff
GOTO repqt
labelpx:
savesel = select1

blocksize = 8: thresh = 0: zonal = 8: k$ = "DON'T_STRIP_ZEROES"
maskwidth = 3: maskheight = 3
hl$ = f$
ent21: row = 7: column = 19: width1 = 31: depth = 8: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As...": select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Masksize width=": maskwidth
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Masksize height=":
maskheight
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Input Filename = ": hl$
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Type X-Dimension = ": xdim
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Type Y-Dimension = ": ydim
rep21: a$ = INKEY$
IF a$ = "" THEN GOTO rep21:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB dilate: GOSUB ent41:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q21: GOTO ent21
GOSUB std_box_stuff
GOTO rep21

ent41: row = 7: column = 19: width1 = 31: depth = 8: fore = 15: back = 1
RETURN

q21:
REM "Load up Tfile and Shell dilate
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Masksize Width ": maskwidth
IF select1 = (row + 3) THEN INPUT "Type Masksize Height ": maskheight
IF select1 = (row + 4) THEN INPUT "Type Input Filename ": hl$
IF select1 = (row + 5) THEN INPUT "Type X-Dimension of Filename that contains
the transform coefficients ": xdim
IF (select1 = (row + 6)) THEN INPUT "Type Y-Dimension of Filename that contains
the transform coefficients ": ydim

```

```

IF ((maskwidth MOD 2) = 0) THEN maskwidth = maskwidth + 1
IF ((maskheight MOD 2) = 0) THEN maskheight = maskheight + 1
GOSUB display_status_box
RETURN

dilate:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: widthl = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, hl$
PRINT #1, h2$
PRINT #1, maskwidth
PRINT #1, maskheight
CLOSE #1
IF savesel = 8 THEN SHELL "dil.exe"
IF savesel = 9 THEN SHELL "erode.exe"
IF savesel = 10 THEN SHELL "open.exe"
IF savesel = 11 THEN SHELL "close.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

rowx = 7: columnx = 46: widthlx = 22: depthx = 7: forex = 15: backx = 1
GOSUB clear_box:
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

edge:
row = 7: column = 36: widthl = 14: depth = 4: fore = 15: back = 1
CALL box(row, column, widthl, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "First Order": select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Second Order"
repat: a$ = INKEY$
IF a$ = "" THEN GOTO repat
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (a$ = CHR$(13)) THEN GOTO labelpu
GOSUB std_box_stuff
GOTO repat
labelpu:
IF select1 = 8 THEN GOSUB ent13: RETURN
IF select1 = 9 THEN GOSUB edgel: RETURN

ent13: row = 2: column = 31: widthl = 19: depth = 16: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, widthl, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Pixel Dif Row "; : select1 = row
+ 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Pixel Dif Col"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Separated Dif Row"
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Separated Dif Col"

```

```

COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Sobel Row  "
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Sobel Column"
COLOR fore, back: LOCATE row + 7, column + 1: PRINT "Roberts Row  "
COLOR fore, back: LOCATE row + 8, column + 1: PRINT "Roberts Column"
COLOR fore, back: LOCATE row + 9, column + 1: PRINT "Prewitt Row  "
COLOR fore, back: LOCATE row + 10, column + 1: PRINT "Prewitt Column"
COLOR fore, back: LOCATE row + 11, column + 1: PRINT "Frei-Chen Row"
COLOR fore, back: LOCATE row + 12, column + 1: PRINT "Frei-Chen Column"
COLOR fore, back: LOCATE row + 13, column + 1: PRINT "Nevatia-Babu 0"
COLOR fore, back: LOCATE row + 14, column + 1: PRINT "Nevatia-Babu 90"
GOSUB second2_box
fin = 0
reps: a$ = INKEY$
IF a$ = "" THEN GOTO reps
oldsel = select1
fin = 0
biasflag = 1
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent14: GOSUB
clear_box: RETURN
IF (a$ = CHR$(13)) THEN GOSUB convoll: select1 = oldsel: IF fin = 1 THEN RETURN
GOSUB std_box_stuff
GOSUB second2_box
GOTO reps

ent14: row = 2: column = 1: width1 = 50: depth = 16: fore = 15: back = 1
RETURN

second2_box:
IF select1 = 3 THEN hl$ = "pdr"
IF select1 = 4 THEN hl$ = "pdc"
IF select1 = 5 THEN hl$ = "sdr"
IF select1 = 6 THEN hl$ = "sdc"
IF select1 = 7 THEN hl$ = "sobelr"
IF select1 = 8 THEN hl$ = "sobelc"
IF select1 = 9 THEN hl$ = "robertsr"
IF select1 = 10 THEN hl$ = "robertsc"
IF select1 = 11 THEN hl$ = "prewitr"
IF select1 = 12 THEN hl$ = "prewittc"
IF select1 = 13 THEN hl$ = "fcr"
IF select1 = 14 THEN hl$ = "fcc"
IF select1 = 15 THEN hl$ = "neva0"
IF select1 = 16 THEN hl$ = "neva90"
CALL box(6, 1, 30, 11, 0, 0)
OPEN hl$ FOR INPUT AS #1
INPUT #1, n
rown = 6: columnn = 27 - 5 * n: widthn = 31 - columnn: depthn = n + 2
CALL box(rown, columnn, widthn, depthn, 15, 1)
k = n
FOR i = 1 TO n
FOR j = 1 TO k
INPUT #1, a: LOCATE 6 + i, columnn - 4 + 5 * j: PRINT a
NEXT j
NEXT i
CLOSE #1
RETURN

noiseclean:
ent11: row = 7: column = 31: width1 = 19: depth = 5: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Noise Mask 1"; : select1 = row +
1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Noise Mask 2"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Noise Mask 3"
GOSUB second1_box

```

```

fin = 0
rept: a$ = INKEY$
IF a$ = "" THEN GOTO rept
oldsel = select1
fin = 0
biasflag = 0
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent12: GOSUB
clear_box: RETURN
IF (a$ = CHR$(13)) THEN GOSUB convoll: select1 = oldsel: IF fin = 1 THEN RETURN
GOSUB std_box_stuff
GOSUB second1_box
GOTO rept

ent12: row = 6: column = 1: width1 = 50: depth = 11: fore = 15: back = 1
RETURN

convolution:
ent9: row = 7: column = 31: width1 = 19: depth = 7: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Crispening Mask 1"; : select1 =
row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Crispening Mask 2"
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Crispening Mask 3"
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Crispening Mask 4"
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Crispening Mask 5"
GOSUB second_box
fin = 0
repv: a$ = INKEY$
IF a$ = "" THEN GOTO repv
oldsel = select1
fin = 0
biasflag = 0
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent10: GOSUB
clear_box: RETURN
IF (a$ = CHR$(13)) THEN GOSUB convoll: select1 = oldsel: IF fin = 1 THEN RETURN
GOSUB std_box_stuff
GOSUB second_box
GOTO repv

second_box:
IF select1 = 8 THEN h1$ = "crisp1"
IF select1 = 9 THEN h1$ = "crisp2"
IF select1 = 10 THEN h1$ = "crisp3"
IF select1 = 11 THEN h1$ = "crisp4"
IF select1 = 12 THEN h1$ = "crisp5"
OPEN h1$ FOR INPUT AS #1
CALL box(6, 1, 30, 11, 0, 0)
INPUT #1, n
rown = 6: columnn = 27 - 5 * n: widthn = 31 - columnn: depthn = n + 2
CALL box(rown, columnn, widthn, depthn, 15, 1)
k = n
FOR i = 1 TO n
FOR j = 1 TO k
INPUT #1, a: LOCATE 6 + i, columnn - 4 + 5 * j: PRINT a
NEXT j
NEXT i
CLOSE #1
RETURN

second1_box:
IF select1 = 8 THEN h1$ = "noise1"
IF select1 = 9 THEN h1$ = "noise2"
IF select1 = 10 THEN h1$ = "noise3"
CALL box(6, 1, 30, 11, 0, 0)

```

```

OPEN h1$ FOR INPUT AS #1
INPUT #1, n
rown = 6: columnn = 27 - 5 * n: widthn = 31 - columnn: depthn = n + 2
CALL box(rown, columnn, widthn, depthn, 15, 1)
k = n
FOR i = 1 TO n
FOR j = 1 TO k
INPUT #1, a: LOCATE 6 + i, columnn - 4 + 5 * j: PRINT a
NEXT j
NEXT i
CLOSE #1
RETURN

ent10: row = 6: column = 1: width1 = 50: depth = 11: forex = 15: backx = 1
RETURN

convol1:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: width1x = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, width1x, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type Filename to Save "; h$
IF (h$ = "") THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, f$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, updown
PRINT #1, headbits
PRINT #1, h$
PRINT #1, h1$
PRINT #1, biasflag      'for Roy's modification to Mask
CLOSE #1
SHELL "mask.exe"
PRINT "Type any key to continue "
CLS
GOSUB Print_title_bar
GOSUB display_status_box
fin = 1
RETURN
convol2:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: width1x = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, width1x, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type Filename to Save "; h$
IF (h$ = "") THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, f$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, updown
PRINT #1, headbits
PRINT #1, h$
PRINT #1, h1$
PRINT #1, biasflag
CLOSE #1

```



```

SHELL "mask.exe"
PRINT "Type any key to continue "
CLS
GOSUB Print_title_bar
GOSUB display_status_box
fin = 1
RETURN

'conver:
'ent5: row = 7: column = 20: width1 = 12: depth = 6: fore = 15: back = 1
'VIEW PRINT 2 TO 18
'CALL box(row, column, width1, depth, fore, back)
'COLOR fore, back: LOCATE row, column + 4: PRINT "FROM"
'COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "8-Bit BIN": select1 = row + 1
'COLOR fore, back: LOCATE row + 2, column + 1: PRINT "8-Bit BMP"
'COLOR fore, back: LOCATE row + 3, column + 1: PRINT "24-Bit BIN"
'COLOR fore, back: LOCATE row + 4, column + 1: PRINT "24-Bit YIQ"
'fin = 0
'repa: a$ = INKEY$
'IF a$ = "" THEN GOTO repa
'oldsel = select1
'IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB ent6: GOSUB
clear_box: RETURN
'IF (a$ = CHR$(13)) THEN GOSUB convert0: select1 = oldsel: IF fin = 1 THEN RE-
TURN
'GOSUB std_box_stuff
'GOTO repa

'ent6: row = 7: column = 20: width1 = 12: depth = 6: fore = 15: back = 1
'
RETURN

'convert0:
'ent7: row = 7: column = 32: width1 = 12: depth = 6: fore = 15: back = 1
'VIEW PRINT 2 TO 18
'CALL box(row, column, width1, depth, fore, back)
'COLOR fore, back: LOCATE row, column + 5: PRINT "TO"
'COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "8-Bit BIN": select1 = row + 1
'COLOR fore, back: LOCATE row + 2, column + 1: PRINT "8-Bit BMP"
'COLOR fore, back: LOCATE row + 3, column + 1: PRINT "24-Bit BIN"
'COLOR fore, back: LOCATE row + 4, column + 1: PRINT "24-Bit YIQ"
'repb: a$ = INKEY$
'IF a$ = "" THEN GOTO repb
'IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN a$ = " ": GOSUB
clear_box: GOSUB ent6: RETURN
'IF (a$ = CHR$(13)) THEN GOSUB convert1: IF fin = 1 THEN RETURN
'GOSUB std_box_stuff
'GOTO repb

'ent8: row = 7: column = 32: width1 = 12: depth = 6: fore = 15: back = 1
'
RETURN

'convert1:
'VIEW PRINT 18 TO 24
'COLOR 7, 0
'rowx = 19: columnx = 1: width1x = 79: depthx = 5: forex = 15: backx = 4
'CALL box(rowx, columnx, width1x, depthx, forex, backx)
'IF (oldsel > 9) OR (select1 <> 11) THEN GOSUB display_status_box: RETURN
'IF oldsel = 9 THEN LOCATE 20, 2: INPUT "Type Filename for BMP file "; f$
'IF oldsel = 8 THEN LOCATE 20, 2: INPUT "Type Filename for BIN File "; f$
'IF oldsel = 8 THEN LOCATE 20, 2: PRINT SPC(60);
'IF oldsel = 8 THEN LOCATE 20, 2: INPUT "Type Palette for BIN file "; g$
'IF oldsel = 8 THEN updown = 0: headbits = 0
'IF f$ = "" THEN GOSUB display_status_box: RETURN
'i = INSTR(f$, ".")
'IF i <> 0 THEN h1$ = LEFT$(f$, i - 1) ELSE h1$ = f$

```

```

'h2$ = h1$ + ".Y"
'h3$ = h1$ + ".I"
'h4$ = h1$ + ".Q"
'IF select1 <> 11 THEN GOSUB display_status_box: RETURN
'LOCATE 21, 2: PRINT "Output Files will be ", h2$, h3$, h4$
'LOCATE 22, 2: INPUT "Type any key to continue "; a$
'COLOR 7, 0
'CLS
'IF oldsel = 9 THEN GOSUB 21000
'OPEN "tfile" FOR OUTPUT AS #1
'PRINT #1, dmode + 1
'PRINT #1, g$
'PRINT #1, f$
'PRINT #1, xdim
'PRINT #1, ydim
'PRINT #1, headbits      'headerbits to skip is zero
'PRINT #1, updown       'start on first line
'PRINT #1, h2$
'PRINT #1, h3$
'PRINT #1, h4$
'CLOSE #1
'SHELL "BMP2YIQ.exe"
'COLOR 7, 0
'CLS
'GOSUB Print_title_bar
'GOSUB display_status_box
'fin = 1
'RETURN

conver:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type Input Filename "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type Output Palette "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type Output Image Name "; h2$
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, h1$
PRINT #1, h2$
CLOSE #1
VIEW PRINT
CLS
RUN "pcxtobin.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

RETURN

SHELL1:
VIEW PRINT
CLS
SHELL
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

```

```

copyfile:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
IF select1 = 9 THEN LOCATE 20, 2: INPUT "Type Filename to Copy From "; h$
IF (select1 = 9) AND (h$ = "") THEN GOSUB display_status_box: RETURN
IF select1 = 9 THEN LOCATE 21, 2: INPUT "Type Filename to Copy To "; h1$
IF (select1 = 9) AND (h1$ = "") THEN GOSUB display_status_box: RETURN
IF select1 = 10 THEN LOCATE 20, 2: INPUT "Type Filename to Rename "; h$
IF (select1 = 10) AND (h$ = "") THEN GOSUB display_status_box: RETURN
IF select1 = 10 THEN LOCATE 21, 2: INPUT "Type New Filename "; h1$
IF (select1 = 10) AND (h1$ = "") THEN GOSUB display_status_box: RETURN
IF select1 = 11 THEN LOCATE 20, 2: INPUT "Type Filename to Delete "; h$
IF (select1 = 11) AND (h$ = "") THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
IF select1 = 10 THEN NAME h$ AS h1$
IF select1 = 11 THEN KILL h$
IF select1 = 9 THEN SHELL "copy " + h$ + " " + h1$
PRINT "Type any key to continue "
qL: a$ = INKEY$
IF a$ = "" THEN GOTO qL
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

paste:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type First Filename "; h$
IF (h$ = "") THEN GOSUB display_status_box: RETURN
LOCATE 20, 42: INPUT "Type Second Filename "; h1$
IF (h1$ = "") THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type Filename to save composite file "; h2$
IF (h2$ = "") THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type X-Dimension "; xxdim
IF xxdim = 0 THEN GOSUB display_status_box: RETURN
LOCATE 22, 42: INPUT "Type Y-Dimension "; yydim
IF yydim = 0 THEN GOSUB display_status_box: RETURN
IF select1 = 13 THEN top$ = "TOP"
IF select1 = 14 THEN top$ = "SIDE"
COLOR 7, 0
CLS
OPEN "TFILE" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, h1$
PRINT #1, h2$
PRINT #1, xxdim
PRINT #1, yydim
PRINT #1, top$
CLOSE #1
SHELL "paste.exe"
PRINT "Type any key to continue "
qm: a$ = INKEY$
IF a$ = "" THEN GOTO qm
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

entropy:

```

```

VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Analyze "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
CLOSE #1
SHELL "entropy.exe"
PRINT "Type any key to continue "
qw: a$ = INKEY$
IF a$ = "" THEN GOTO qw
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

```

```

shell_lookone:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type Filename to Examine "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type X-Dimension "; xxdim
IF xxdim = 0 THEN GOSUB display_status_box: RETURN
LOCATE 21, 40: INPUT "Type Y-Dimension "; yydim
IF yydim = 0 THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type Blocksize "; bsize
IF bsize = 0 THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, xxdim
PRINT #1, yydim
PRINT #1, bsize
CLOSE #1
VIEW PRINT
CLS
SHELL "lookone.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

```

```

shell_looktwo:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type First Filename "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 20, 40: INPUT "Type Second Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type X-Dimension "; xxdim
IF xxdim = 0 THEN GOSUB display_status_box: RETURN
LOCATE 21, 40: INPUT "Type Y-Dimension "; yydim
IF yydim = 0 THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type Blocksize "; bsize
IF bsize = 0 THEN GOSUB display_status_box: RETURN
COLOR 7, 0
VIEW PRINT

```

```

CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, h1$
PRINT #1, xxdim
PRINT #1, yydim
PRINT #1, bsize
CLOSE #1
SHELL "looktwo.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

mse:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type First Filename "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type Second Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, h1$
CLOSE #1
SHELL "mse.exe"
PRINT
PRINT "Type any key to continue "
qx: a$ = INKEY$
IF a$ = "" THEN GOTO qx
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

histogram:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Display Histogram "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Filename to Save Histogram "; h1$
IF h1$ = "" THEN h1$ = "none"
IF h1$ = "none" THEN temp = 0 ELSE temp = 1
COLOR 7, 0
CLS
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, h$
PRINT #1, temp
PRINT #1, h1$
CLOSE #1
SHELL "histo.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL _zip:
VIEW PRINT 18 TO 24
COLOR 7, 0

```

```

rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
i = INSTR(h$, ".")
IF i <> 0 THEN h1$ = LEFT$(h$, i - 1) ELSE h1$ = h$
h1$ = h1$ + ".zip"
COLOR 7, 0
CLS
SHELL "pkzip -a ~ + h1$ + " ~ + h$
PRINT "Type any key to continue ";
L1: a$ = INKEY$
IF a$ = "" THEN GOTO L1:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN
SHELL_unzip:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "pkunzip ~ + h$
PRINT "Type any key to continue ";
L2: a$ = INKEY$
IF a$ = "" THEN GOTO L2:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN
SHELL_ahuff:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "ahuff ~ + h$ + " ~ + h1$
PRINT "Type any key to continue ";
L3: a$ = INKEY$
IF a$ = "" THEN GOTO L3:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN
SHELL_aunhuff:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0

```

```

CLS
SHELL "aunhuff " + h$ + " " + h1$
PRINT "Type any key to continue ";
L4: a$ = INKEY$
IF a$ = "" THEN GOTO L4:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_huff:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "huff " + h$ + " " + h1$
PRINT "Type any key to continue ";
L5: a$ = INKEY$
IF a$ = "" THEN GOTO L5:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_unhuff:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "unhuff " + h$ + " " + h1$
PRINT "Type any key to continue ";
L6: a$ = INKEY$
IF a$ = "" THEN GOTO L6:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_ari0:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "arith " + h$ + " " + h1$
PRINT "Type any key to continue ";
L7: a$ = INKEY$
IF a$ = "" THEN GOTO L7:

```

```

CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_ar11:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "arith1 ~ + h$ + " ~ + h1$
PRINT "Type any key to continue ";
L8: a$ = INKEY$
IF a$ = "" THEN GOTO L8:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_ar12:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "arithn ~ + h$ + " ~ + h1$ + " -o 2"
PRINT "Type any key to continue ";
L9: a$ = INKEY$
IF a$ = "" THEN GOTO L9:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_unar10:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "unarith ~ + h$ + " ~ + h1$
PRINT "Type any key to continue ";
L10: a$ = INKEY$
IF a$ = "" THEN GOTO L10:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

```



```

SHELL_unar11:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "unar11 " + h$ + " " + h1$
PRINT "Type any key to continue ";
L11: a$ = INKEY$
IF a$ = "" THEN GOTO L11:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_unar12:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
SHELL "unar12 " + h$ + " " + h1$ + " -o 2"
PRINT "Type any key to continue ";
L12: a$ = INKEY$
IF a$ = "" THEN GOTO L12:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_lharc:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Compress "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 11: INPUT "Type Output Filename (no extension) "; h1$
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
LOCATE 18, 1
SHELL "lharc a " + h1$ + " " + h$
PRINT "Type any key to continue ";
up1: a$ = INKEY$
IF a$ = "" THEN GOTO up1:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

SHELL_unlharc:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4

```

```

CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 21, 11: INPUT "Type Filename to Decompress (no extension)"; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
LOCATE 18, 1
SHELL "lharc x ~ + h$"
PRINT "Type any key to continue ";
up: a$ = INKEY$
IF a$ = "" THEN GOTO up:
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

chop_image:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: PRINT "Upper Left Coordinate for X "; : INPUT xup
LOCATE 20, 45: PRINT "For Y "; : INPUT yup
LOCATE 21, 2: PRINT "Bottom Right Coordinate for X "; : INPUT xdown
LOCATE 21, 45: PRINT "For Y "; : INPUT ydown
LOCATE 22, 2: INPUT "Type Filename to Save "; h1$
IF (xup + xdown + yup + ydown) = 0 THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, xup
PRINT #1, yup
PRINT #1, xdown
PRINT #1, ydown
PRINT #1, h1$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, f$
CLOSE #1
CLS
SHELL "chop.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

differ:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type First Filename "; j$
IF j$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 21, 2: INPUT "Type Second Filename "; j1$
IF j1$ = "" THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type Filename to Store Difference "; j2$
IF j2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, j$
PRINT #1, j1$
PRINT #1, j2$
CLOSE #1
CLS
IF select1 = 9 THEN SHELL "differ.exe"
IF select1 = 10 THEN SHELL "adder.exe"
CLS

```

```

GOSUB Print_title_bar
GOSUB display_status_box
RETURN

bit_planes:
s = select1
IF s = 8 THEN gray$ = "GRAYSCALE" ELSE gray$ = "COLOR"
IF (s >= 5) THEN j$ = "NOSAVE": h = 0: h1 = 255: GOTO tf:
VIEW PRINT 18 TO 24

COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type Bit-Plane to View (1 is LSB, 8 is MSB), or <RET> "; h
h = INT(h): IF ((h < 0) OR (h > 8)) THEN GOSUB display_status_box: RETURN
IF h = 0 THEN LOCATE 21, 2: INPUT "Type Binary Number between 1 and 255 to AND
with Pixels "; h1
h1 = INT(h1): IF (h = 0) AND ((h1 < 0) OR (h1 > 255)) THEN GOSUB display_sta-
tus_box: RETURN
IF (h = 0) AND (h1 = 0) THEN GOSUB display_status_box: RETURN
LOCATE 22, 2: INPUT "Type Filename to Save New File or <RET> "; j$
IF j$ = "" THEN j$ = "NOSAVE"
COLOR 7, 0
CLS
tf: OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, f$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, headbits          'headerbits to skip is zero
PRINT #1, updown            'start on first line
IF h <> 0 THEN hh = 2 ^ (h - 1) ELSE hh = h1
PRINT #1, hh                'bitplane number from 1 to 8
PRINT #1, j$
IF h <> 0 THEN PRINT #1, 255 ELSE PRINT #1, 0 'full intensity if h <> 0
IF select1 = 4 THEN PRINT #1, "ALL"
IF select1 = 5 THEN PRINT #1, "RED"
IF select1 = 6 THEN PRINT #1, "GREEN"
IF select1 = 7 THEN PRINT #1, "BLUE"
IF select1 = 8 THEN PRINT #1, gray$
CLOSE #1
SHELL "bitplane.exe"
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

palgray:
VIEW PRINT 18 TO 24
COLOR 7, 0
rowx = 19: columnx = 1: widthlx = 79: depthx = 5: forex = 15: backx = 4
CALL box(rowx, columnx, widthlx, depthx, forex, backx)
LOCATE 20, 2: INPUT "Type New Palette Name "; h$
IF h$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
CLS
OPEN g$ FOR BINARY AS #1
FOR i = 1 TO 256
a$ = INPUT$(1, 1)
red(i) = ASC(a$)
NEXT i
FOR i = 1 TO 256
a$ = INPUT$(1, 1)
green(i) = ASC(a$)

```

```

NEXT i
FOR i = 1 TO 256
a$ = INPUT$(1, 1)
blue(i) = ASC(a$)
NEXT
CLOSE #1
FOR i = 1 TO 256
red(i) = .299 * red(i) + .587 * green(i) + .114 * blue(i)
IF red(i) > 255 THEN red(i) = 255
red(i) = INT(red(i))
blue(i) = red(i)
green(i) = blue(i)
NEXT i
OPEN h$ FOR OUTPUT AS #1
FOR i = 1 TO 256
PRINT #1, CHR$(red(i));
NEXT i
FOR i = 1 TO 256
PRINT #1, CHR$(green(i));
NEXT i
FOR i = 1 TO 256
PRINT #1, CHR$(blue(i));
NEXT i
CLOSE #1
CLS
GOSUB Print_title_bar
GOSUB display_status_box
RETURN

shell_dct:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h1$
IF h1$ = "" THEN row = 7: column = 46: width1 = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h1$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, f$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, headbits           'headerbits to skip is zero
PRINT #1, updown            'start on first line
PRINT #1, TRANS$
PRINT #1, blocksize
PRINT #1, "TRANSFORM"
PRINT #1, h1$
PRINT #1, thresh
PRINT #1, zonal
PRINT #1, k$
PRINT #1, blocksize          'expandsize entry
CLOSE #1
SHELL "dct.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

dct_box:
IF select1 = 10 THEN TRANS$ = "COSINE"
IF select1 = 11 THEN TRANS$ = "SINE"

```

```

IF select1 = 12 THEN TRANS$ = "HADAMARD"
blocksize = 8: thresh = 0: zonal = 8: k$ = "DON'T_STRIP_ZEROES"
expandsize = blocksize
ent1: row = 7: column = 46: width1 = 27: depth = 10: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As.....": : se-
lect1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Blocksize =": blocksize
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Threshold =": thresh
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Zonal Mask =": zonal
COLOR fore, back: LOCATE row + 5, column + 1: PRINT k$
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "X-Dimension = ": xdim
COLOR fore, back: LOCATE row + 7, column + 1: PRINT "Y-Dimension = ": ydim
COLOR fore, back: LOCATE row + 8, column + 1: PRINT "Filename is ": f$
repl: a$ = INKEY$
IF a$ = "" THEN GOTO repl:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB shell_dct: GOSUB ent3:
GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q1: GOTO ent1
GOSUB std_box_stuff
GOTO repl

ent3: row = 7: column = 46: width1 = 27: depth = 10: fore = 15: back = 1
RETURN

invdct_box:
IF select1 = 10 THEN TRANS$ = "COSINE"
IF select1 = 11 THEN TRANS$ = "SINE"
IF select1 = 12 THEN TRANS$ = "HADAMARD"
blocksize = 8: thresh = 0: zonal = 8: k$ = "DON'T_STRIP_ZEROES"
expandsize = blocksize
ent2: row = 7: column = 4: width1 = 31: depth = 8: fore = 15: back = 1
VIEW PRINT 2 TO 18
CALL box(row, column, width1, depth, fore, back)
COLOR 7, 0: LOCATE row + 1, column + 1: PRINT "Save As...": : select1 = row + 1
COLOR fore, back: LOCATE row + 2, column + 1: PRINT "Blocksize =": blocksize
COLOR fore, back: LOCATE row + 3, column + 1: PRINT "Type Filename = ": h1$
COLOR fore, back: LOCATE row + 4, column + 1: PRINT "Type X-Dimension = ": xdim
COLOR fore, back: LOCATE row + 5, column + 1: PRINT "Type Y-Dimension = ": ydim
COLOR fore, back: LOCATE row + 6, column + 1: PRINT "Expand Block to ": expand-
ize
rep2: a$ = INKEY$
IF a$ = "" THEN GOTO rep2:
IF ASC(LEFT$(a$, 1)) = 27 AND ASC(RIGHT$(a$, 1)) = 27 THEN GOSUB clear_box: RE-
TURN
IF (select1 = (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB shell_invdct: GOSUB
ent4: GOSUB clear_box: RETURN
IF (select1 <> (row + 1)) AND (a$ = CHR$(13)) THEN GOSUB q2: GOTO ent2
GOSUB std_box_stuff
GOTO rep2

ent4: row = 7: column = 4: width1 = 31: depth = 8: fore = 15: back = 1
RETURN

q2:
REM "Load up Tfile and Shell to dct.exe
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthy = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthy, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Blocksize ": blocksize

```

```

IF select1 = (row + 3) THEN INPUT "Type Filename that contains the transform co-
efficients "; h1$
IF select1 = (row + 4) THEN INPUT "Type X-Dimension of Filename that contains
the transform coefficients "; xdim
IF (select1 = (row + 5)) THEN INPUT "Type Y-Dimension of Filename that contains
the transform coefficients "; ydim
IF (select1 = (row + 6)) THEN INPUT "Expand Blocksize to NxN by padding with ze-
roes, N = "; expandsize
GOSUB display_status_box
RETURN

```

```

rowx = 7: columnx = 46: widthlx = 22: depthx = 7: forex = 15: backx = 1
GOSUB clear_box:
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

```

```

q1:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowy = 19: columny = 1: widthly = 79: depthy = 5: forey = 15: backy = 4
CALL box(rowy, columny, widthly, depthy, forey, backy)
LOCATE 20, 2
IF select1 = (row + 2) THEN INPUT "Type Blocksize "; blocksize
IF select1 = (row + 3) THEN INPUT "Type Threshold "; thresh
IF select1 = (row + 4) THEN INPUT "Type Zonal Mask "; zonal
IF (select1 = (row + 5)) AND (k$ = "DON'T_STRIP_ZEROS") THEN temp$ =
"STRIP_ZEROS_OUTSIDE_MASK"
IF (select1 = (row + 5)) AND (k$ = "STRIP_ZEROS_OUTSIDE_MASK") THEN temp$ =
"DON'T_STRIP_ZEROS"
IF (select1 = (row + 5)) THEN k$ = temp$
IF (select1 = (row + 6)) THEN INPUT "Type X-Dimension = "; xdim
IF (select1 = (row + 7)) THEN INPUT "Type Y-Dimension = "; ydim
IF (select1 = (row + 8)) THEN INPUT "Type Filename = "; f$
GOSUB display_status_box
RETURN

```

```

clear_box:
VIEW PRINT 2 TO 18
COLOR 0, 0
FOR i = 1 TO depth + 1
LOCATE row + i - 1, column: PRINT SPC(widthlx);
NEXT i
RETURN

```

```

shell_invdct:
REM "Load up Tfile and Shell to DCT.EXE
VIEW PRINT 18 TO 24
COLOR 7, 0
rowz = 19: columnz = 1: widthlz = 79: depthz = 5: forez = 15: backz = 4
CALL box(rowz, columnz, widthlz, depthz, forez, backz)
LOCATE 20, 2: INPUT "Type Filename to Save "; h2$
IF h2$ = "" THEN row = 7: column = 46: widthl = 22: depth = 7: fore = 15: back
= 1: GOSUB clear_box
IF h2$ = "" THEN GOSUB display_status_box: RETURN
COLOR 7, 0
OPEN "tfile" FOR OUTPUT AS #1
PRINT #1, dmode + 1
PRINT #1, g$
PRINT #1, h1$
PRINT #1, xdim
PRINT #1, ydim
PRINT #1, headerbits      'headerbits to skip is zero
PRINT #1, updown          'start on first line

```

```

PRINT #1, TRANS$
PRINT #1, blocksize
PRINT #1, "INVERSE_TRANSFORM"
PRINT #1, h2$
PRINT #1, thresh
PRINT #1, sonal
PRINT #1, k$
PRINT #1, expandsize
CLOSE #1
SHELL "dot.exe"
VIEW PRINT: COLOR 7, 0: CLS : GOSUB display_status_box
GOSUB Print_title_bar
RETURN

10000 STATIC SUB box (row, column, width1, depth, fore, back)
10010 DEFINT A, I-L, P
10020 K1 = row: L = column: j = width1 - 1: kend = depth - 1: idepth = depth -
1: ifore = fore: iback = back
10030 COLOR ifore, iback
10040 k = 0: j1 = j + 1
10050 FOR i = K1 + 1 TO K1 + idepth
10060 LOCATE i, L + 1: PRINT SPC(j - 1);
10070 NEXT
10080 FOR i = L + 1 TO L + j - 1
10090 LOCATE K1, i: PRINT CHR$(205)
10100 NEXT
10110 LOCATE K1, L: PRINT CHR$(201): LOCATE K1, L + j: PRINT CHR$(187)
10120 LOCATE K1 + kend, L: PRINT CHR$(200): LOCATE K1 + kend, L + j: PRINT
CHR$(188)
10230 FOR k = K1 + 1 TO K1 + kend - 1
10240 LOCATE k, L: PRINT CHR$(186)
10250 LOCATE k, L + j: PRINT CHR$(186)
10270 NEXT
10280 FOR i = L + 1 TO L + j - 1
10290 LOCATE K1 + kend, i: PRINT CHR$(205)
10300 NEXT
10310 END SUB

```

Appendix BB

The VESA_INITIALIZER

The VESA Initializer

```
#include <stdio.h>
#include <dos.h>

/*Globals begin*/
struct VGABLOCK
{
    char VESAS[4];
    int VESAV;
    char far *OEMSP;
    long CAPAB;
    int far *MODES;
    int TOTMEM;
    char data[236];
} VGADATA;

void __far *p;

FILE *out;
/*Globals end*/

/*Main program begins*/
main()
{
    int i,k;
    union REGS r;
    struct SREGS sr;

    p=&VGADATA;
    r.h.ah=0x4f;
    r.h.al=0x00;
    sr.es= _FP_SEG(p);
    r.x.di= _FP_OFF(p);
    int86x(0x10,&r,&r,&sr);

    i=0;
    k=0;

    while (VGADATA.MODES[i] != -1)
    {
        if (VGADATA.MODES[i]==257) k++;
        if (VGADATA.MODES[i]==259) k++;
        if (VGADATA.MODES[i]==261) k++;
        i++;
    }

    out=fopen("vesa.ini","w");
    fprintf(out,"%i",k);
    _fcloseall();
}
/*Main program ends*/
```

Appendix CC

The Display Processor

The Display Processor

```
#include <stdio.h>
#include <font1.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#define RGB(r,g,b) (0x3F3F3FL & ((long)(b) << 16 | (g) << 8 | (r)))

/*Globals begin*/
int width1,height1,a,b,olddb,c,d=0,i,j,k,xdim,ydim;
int ourseg,mode,header,updown,il,kl,input_file;
char string[80],string1[80],buffer[2048];
unsigned char red[256],blue[256],green[256];
unsigned char palbuf[256][3],cp;
float bl,cl;
void __far *p;

FILE *inpalette,*infile,*out;
/*Globals end*/

/*Main program begins*/
main()
{

/*Read in display information*/
infile=fopen("tfile","r");
fscanf(infile,"%i",&mode);
fscanf(infile,"%s",string);
fscanf(infile,"%s",string1);
fscanf(infile,"%i",&xdim);
fscanf(infile,"%i",&ydim);
fscanf(infile,"%i",&header);
fscanf(infile,"%i",&updown);
input_file=_open(string1,_O_BINARY);

/*Read in palette*/
inpalette=fopen(string,"r+b");
for (i=0;i<=255;i++)
{
    fscanf(inpalette,"%c",&cp);
    red[i]=cp;
}
for (i=0;i<=255;i++)
{
    fscanf(inpalette,"%c",&cp);
    green[i]=cp;
}
for (i=0;i<=255;i++)
{
    fscanf(inpalette,"%c",&cp);
    blue[i]=cp;
}
if (header!=0) _read(input_file,buffer,header);
for (i=0;i<=255;i++)
{
    palbuf[i][0]=red[i]/4;
    palbuf[i][1]=green[i]/4;
    palbuf[i][2]=blue[i]/4;
}

/*Set video mode*/
if (mode==1)
```

```

    {
        _asm mov ax,13h
        _asm int 10h
        width1=320;
        height1=200;
    }
    if (mode==2)
    {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        width1=640;
        height1=480;
    }
    if (mode==3)
    {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        width1=800;
        height1=600;
    }
    if (mode==4)
    {
        _asm mov ax,4f02h;
        _asm mov bx,105h;
        _asm int 10h;
        width1=1024;
        height1=768;
    }

    /*Enable palette*/
    j=256;
    k=0;
    p=4palbuf;
    i=(int)p;
    _asm mov dx,i;
    _asm mov bx,k;
    _asm mov cx,j;
    _asm mov ax,1012h;
    _asm int 10h;

    /*Start at Bank 0*/
    _asm mov ax,4f05h;
    _asm mov bx,00h;
    _asm mov dx,0;
    _asm int 10h;

    d=0;
    a=0;
    oldb=0;

    /*Define base video memory segment*/
    _asm mov [ourseg],0A000h;

    /*Display*/
    for (i=0;i<=ydim-1;i++)
    {
        if (updown == 1) il = ydim-1-i;
        else il = i;
        read((int)input_file,buffer,xdim);
        if (il < height1)
        {
            for (j=0;j<=(xdim-1):j++)
            {

```

```

        k1 = buffer[j+1];
        _asm mov ax, j
        _asm cmp ax, width1
        _asm jg label3

        _asm mov ax, j
        _asm add ax, 0
        _asm je label6
        {
            _asm mov ax, b
            _asm add ax, xdim
            _asm jnc label5
            _asm jmp label6
label5:    _asm inc b
            _asm jmp label2
        }

        /*The next group of code computes the bank.*/
label6:    {
            _asm mov ax, 11
            _asm mul width1
            _asm add ax, j
            _asm jnc label1
            _asm inc dx
label1:    _asm mov d, dx
            _asm mov b, ax

            /* The next two statements compare oldb to d*/
            _asm cmp dx, oldb
            _asm je label2
            {
                _asm mov ax, 4f05h;
                _asm mov bx, 00h;
                _asm mov dx, d;
                _asm int 10h;
            }

            /*Display pixel*/
label2:    oldb=d;
            _asm mov es, [ourseg];
            _asm mov al, byte ptr k1;
            _asm mov bx, b;
            _asm mov es:[bx], al
        }
label3:    _asm nop
    }
}
getch();

/*Go to text mode*/
_asm mov ax, 3
_asm int 10h;

/*Close files*/
_close (input_file);
_fcloseall();
}
/*Main program ends*/

```

Appendix DD

The PCX Loader

The PCX Loader

```
#include <vicdefs.h>
#include <vicfcts.h>
#include <vicerror.h>
#include <graph.h>
#include <stdio.h>

/*Main program begins*/
main()
{
    PcxData pdat;
    char *fname;
    imgdes image;
    unsigned char Paltab[768];
    int rcode, vmode, width, length, mode, no_modes;

    FILE *infile;

    /*Read in information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%i", &mode);
    fscanf(infile, "%s", &fname);
    fclose(infile);
    infile=fopen("vesa.ini", "r");
    fscanf(infile, "%i", &no_modes);
    if (no_modes==0)
    {
        _setvideomode(3);
        printf("Error-No Super VGA Modes!\n");
        printf("Type <RETURN> to continue.");
        getchar();
        goto out;
    }
    if (pcxinfo(fname, &pdat) != NO_ERROR)
    {
        printf("\nError in reading %s", fname);
        exit(1);
    }

    /*Get dimensions*/
    width=pdat.BytesPerLine;
    length=pdat.Ymax-pdat.Ymin+1;

    /*Initialize image space*/
    zeroimgdes(&image);

    /*Allocate image space*/
    if (xmallocimage(&image, width, length) != NO_ERROR)
    {
        printf("\nNot enough memory to load %s", fname);
        exit(1);
    }

    /*Set palette pointer*/
    image.palette=Paltab;

    /*Load PCX image to extended memory, set video mode, enable palette, and display image*/
    if (loadpcx(fname, &image) == NO_ERROR)
    {
        if (mode==1 || mode==2)
        {

```

```

        _asm mov ax,4f02h
        _asm mov bx, 101h
        _asm int 10h
        vmode=0x101;
    }
    if (mode==3)
    {
        _asm mov ax,4f02h
        _asm mov bx,103h
        _asm int 10h
        vmode=0x103;
    }
    if (mode==4)
    {
        _asm mov ax,4f02h
        _asm mov bx,105h
        _asm int 10h
        vmode=0x105;
    }
    setvgapalette(image.palette);
    if ((rcode=viewvesaevga(vmode,0,0,&image))==NO_ERROR) pause();
}

/*Deallocate image*/
freeimage(&image);
out:

/*Go to text mode*/
_setvideomode(3);

/*Close files*/
_fcloseall();
}
/*Main program ends*/

```


Appendix EE

The PCX to Binary Converter

The PCX to Binary Converter

```
#include <vicdefs.h>
#include <vicfcts.h>
#include <vicerror.h>
#include <graph.h>
#include <stdio.h>

/*Globals begin*/
imgdes image;
/*Globals end*/

/*Retrieve data from extended memory*/
char dget(index_x,index_y,width)
int index_x,index_y;
{
    char data;
    int test;
    int handle;

    handle = (int) image.xhandle;

    test=xmgetrow(&data,handle,index_x,index_y,1,width);
    if (test!=NO_ERROR) {printf("dget %i,%i,%i",handle,index_x,index_y);
getchar();}
    return(data);
}

/*Main program begins*/
main()
{
    PcxData pdat;
    char fname[20],palname[20],imagename[20],image_data;
    unsigned char Paltab[768];
    int rcode,width,length,i,j;

    FILE *infile,*outpal,*outimage;

    /*Clear screen*/
    _clearscreen(_GCLEARSCREEN);

    /*Read in information*/
    infile=fopen("tfile","r");
    fscanf(infile,"%s",fname);
    fscanf(infile,"%s",palname);
    fscanf(infile,"%s",imagename);

    /*Open files*/
    outpal=fopen(palname,"w+b");
    outimage=fopen(imagename,"w+b");

    /*Get PCX information*/
    if (pcxinfo(fname,&pdat)!=NO_ERROR)
    {
        printf("\nError in reading %s",fname);
        exit(1);
    }

    /*Get dimensions*/
    width=pdat.BytesPerLine;
    length=pdat.Ymax-pdat.Ymin+1;
    printf("\nWIDTH = %i",width);
    printf("\nHEIGHT = %i",length);
}
```

```

/*Initialize image space*/
zeromgdes(&image);

/*Allocate image space*/
if (xmallocimage(&image,width,length)!=NO_ERROR)
{
    printf("\nNot enough memory to load %s",fname);
    exit(1);
}

/*Set palette pointer*/
image.palette=Paltab;

/*Load PCX image and convert to binary image and palette*/
if (loadpcx(fname,&image)==NO_ERROR)
{
    for (j=0;j<3;j++)
    {
        for (i=0;i<256;i++)
        {
            fprintf(outpal,"%c",image.palette[3*i+j]);
        }
    }
    for (j=0;j<length;j++)
    {
        for (i=0;i<width;i++)
        {
            image_data=dget(i,j,width);
            fprintf(outimage,"%c",image_data);
        }
    }
}
else
{
    printf("\nError in loading %s",fname);
}

/*Close files*/
_fcloseall();

/*Deallocation image space*/
freeimage(&image);

/*Recall host program*/
execl("image.exe",NULL,NULL,NULL);
}

/*Main program ends*/

```

Appendix FF

The Image Printer

The Image Printer

```
#include <vicdefs.h>
#include <vicfcts.h>
#include <vicerror.h>
#include <graph.h>
#include <stdio.h>
#include <bios.h>

/*Globals begin*/
imgdes image;
int vmode,width,length,i,mode,prtmode=1,test,WIDTH1,HEIGHT1;
int ptr_ex=0,ptr_ey=0,ptr_ex=5*300,ptr_ey=5*300,dpi=300,frame=2;
unsigned char palbuf[256][3],cp;
float factorx, factory, factor;
char fname[80],palname[80];
void __far *p;

FILE *infile;
FILE *inpalette;

typedef struct {
    unsigned timeout :1;
    unsigned :2;
    unsigned ioerr :1;
    unsigned select :1;
    unsigned nopaper :1;
    unsigned :1;
    unsigned notbusy :1;
} prtbits;

typedef union {unsigned i;prtbits b;} prt_status;
/*Globals end*/

/*Set video mode*/
void Set_Mode()
{
    //320X200
    if (mode==1) {
        _asm mov ax,13h;
        _asm int 10h;
        WIDTH1=320;
        HEIGHT1=200;
        vmode=0x13;
    }
    //640X480
    if (mode==2) {
        _asm mov ax,4f02h;
        _asm mov bx,101h;
        _asm int 10h;
        WIDTH1=640;
        HEIGHT1=480;
        vmode=0x101;
    }
    //800X600
    if (mode==3) {
        _asm mov ax,4f02h;
        _asm mov bx,103h;
        _asm int 10h;
        WIDTH1=800;
        HEIGHT1=600;
        vmode=0x103;
    }
    //1024X768
```

```

    if (mode==4) {
        _asm mov ax, 4f02h;
        _asm mov bx, 105h;
        _asm int 10h;
        WIDTH1=1024;
        HEIGHT1=768;
        vmode=0x105;
    }
}

/*Enable palette*/
void Set_Palette()
{
    int i, j, k;

    j=256;
    k=0;
    p=&palbuf;
    i=(int)p;
    _asm mov dx, i;
    _asm mov bx, k;
    _asm mov cx, j;
    _asm mov ax, 1012h;
    _asm int 10h;
}

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%i", &mode);
    fscanf(infile, "%s", &fname);
    fscanf(infile, "%s", &palname);
    fscanf(infile, "%i", &width);
    fscanf(infile, "%i", &length);
    fclose(infile);

    ptr_ex=5*width;
    ptr_ey=5*length;
    factorx = 1;
    factory = 1;
    factor = 1.5;
    if (ptr_ex > 2399) {factorx = 2399.0/(float)ptr_ex;}
    if (ptr_ey > 3149) {factory = 3149.0/(float)ptr_ey;}
    if (factorx < factory) {factor = factorx;}
    if (factory < factorx) {factor = factory;}

    ptr_ex = ptr_ex*factor;
    ptr_ey=ptr_ey*factor;

    /*Read in palette*/
    inpalette=fopen(palname, "r+b");
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        palbuf[i][0]=cp/4;
    }
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        palbuf[i][1]=cp/4;
    }
    for (i=0; i<=255; i++)

```

```

        {
            fscanf(inpalette, "%c", &cp);
            palbuf[i][2]=cp/4;
        }
fclose(inpalette);

/*Initialize image space*/
zeroimgdes(&image);

/*Allocate image space*/
if (!xmallocimage(&image,width,length)!=NO_ERROR)
{
    printf("\nNot enough memory to load %s",fname);
    exit(1);
}

/*Load binary image and set video mode*/
test=loadbif(fname,&image);
if (test==NO_ERROR) Set_Mode();
else
{
    printf ("Error in LOADBIF!");
    exit(1);
}

/*Enable palette*/
Set_Palette();

/*View image*/
viewvesaevga(vmode,0,0,&image);

/*Print image*/
printimage(prtmode,dpi,&image,ptr_sx,ptr_sy,ptr_ex,ptr_ey,frame);

/*Form feed*/
_asm mov ah,0
_asm mov al,12
_asm mov dx,0
_asm int 17h

/*Reset text mode*/
_asm mov ax,3;
_asm int 10h;

/*Deallocate image space*/
freeimage(&image);
}
/*Main program ends*/

```

Appendix GG

The Image Chopper

The Image Chopper

```
#include <stdio.h>

/*Globals begin*/
int i, j, xdim, ydim, xup, yup, xdown, ydown;
char filein[80], fileout[80], cp;

FILE *input_file, *output_file, *infile, *outfile;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    input_file=fopen("tfile", "r");
    fscanf(input_file, "%i", &xup);
    fscanf(input_file, "%i", &yup);
    fscanf(input_file, "%i", &xdown);
    fscanf(input_file, "%i", &ydown);
    fscanf(input_file, "%s", fileout);
    fscanf(input_file, "%i", &xdim);
    fscanf(input_file, "%i", &ydim);
    fscanf(input_file, "%s", filein);

    /*Open files*/
    infile=fopen(filein, "r+b");
    outfile=fopen(fileout, "w+b");

    /*Chop image*/
    for (i=0; i<=ydim-1; i++)
    {
        for (j=0; j<=xdim-1; j++)
        {
            fscanf(infile, "%c", &cp);
            if (j>=xup && j<=xdown)
                if (i>=yup && i<=ydown)
                {
                    fprintf(outfile, "%c", cp);
                }
        }
    }

    /*Close files*/
    _fcloseall();
}
/*Main program ends*/
```

Appendix HH

The Bitplane Processor

The Bitplane Processor

```
#include <stdio.h>
#include <fcntl.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#define RGB(x,g,b) (0x3F3F3F | ((long)(b) << 16 | (g) << 8 | (x)))

/*Globals begin*/
char buffer[2048], string[80], string1[80], string2[80], rgbstring[80];
unsigned char red[256], blue[256], green[256], cp, palbuf[256][3];
int fullintensity, a, d, kl, i, j, k, xdim, ydim, ch3, header, updown;
int bitplane, temp, oldb, ourseg, b, width1, height1, input_file;
void __far *p;

FILE *inpalette, *infile, *out;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%i", &ch3);
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%i", &xdim);
    fscanf(infile, "%i", &ydim);
    fscanf(infile, "%i", &header);
    fscanf(infile, "%i", &updown);
    fscanf(infile, "%i", &bitplane);
    fscanf(infile, "%s", string2);
    fscanf(infile, "%i", &fullintensity);
    fscanf(infile, "%s", rgbstring);

    /*Open files*/
    input_file=open(string1, O_BINARY);
    inpalette=fopen(string, "r+b");
    if (strcmp(string2, "NOSAVE", 6) != 0) out = fopen(string2, "w+b");

    /*Read in palette*/
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        red[i]=cp;
    }
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        green[i]=cp;
    }
    for (i=0; i<=255; i++)
    {
        fscanf(inpalette, "%c", &cp);
        blue[i]=cp;
    }

    if (fullintensity == 255)
    {
        red[0]=0;
        blue[0]=0;
    }
}
```

```

    green[0]=0;
    red[255]=255;
    green[255]=255;
    blue[255]=255;
}

if (header !=0)
{
    for (i=1;i<=header;i++)
    {
        _read(input_file,&cp,1);
    }
}

/*Set video mode*/
if (ch3==1)
{
    _asm mov ax,13h
    _asm int 10h
    width1=320;
    height1=200;
}
if (ch3==2)
{
    _asm mov ax,4f02h
    _asm mov bx,101h
    _asm int 10h
    width1=640;
    height1=480;
}
if (ch3==3)
{
    _asm mov ax,4f02h
    _asm mov bx, 103h
    _asm int 10h
    width1=800;
    height1=600;
}
if (ch3==4)
{
    _asm mov ax,4f02h
    _asm mov bx,105h
    _asm int 10h
    width1=1024;
    height1=768;
}

/*Manipulate viewing parameters*/
for (i=0;i<=255;i++)
{
    if (strcmp(rgbstring,"RED",3)==0)
    {
        green[i] = 0;
        blue[i]=0;
    }
    if (strcmp(rgbstring,"GREEN",5)==0)
    {
        red[i]=0;
        blue[i]=0;
    }
    if (strcmp(rgbstring,"BLUE",4)==0)
    {
        red[i]=0;
        green[i]=0;
    }
}

```

```

    if (strcmp(rgbstring, "GRAYSCALE", 9) == 0)
    {
        temp = .299*red[i]+.587*green[i]+.114*blue[i];
        if (temp > 255) temp = 255;
        red[i]=(int)temp;
        green[i]=(int)temp;
        blue[i]=(int)temp;
    }
    palbuf[i][0]=red[i]/4;
    palbuf[i][1]=green[i]/4;
    palbuf[i][2]=blue[i]/4;
}

/*Enable palette*/
j=256;
k=0;
p=&palbuf;
i=(int)p;
_asm mov dx,i;
_asm mov bx,k;
_asm mov cx,j;
_asm mov ax,1012h;
_asm int 10h;

/*Set starting bank number to 0*/
_asm mov ax,4f05h
_asm mov bx,00h
_asm mov dx,0
_asm int 10h

d=0;
a=0;
olddb=0;

/*Set starting address to beginning of video memory*/
_asm mov [ourseg],0A000h;

/*Display with manipulated environment*/
for (i=0;i<ydlim;i++)
{
    _read(input_file,buffer,xdim);
    for (j=0;j<xdim;j++)
    {
        cp=buffer[j+1];
        if (fullintensity == 255)
        {
            if ((cp & bitplane) != 0) kl=255;
            if ((cp & bitplane) == 0) kl=0;
        }
        if (fullintensity != 255)
        {
            kl=(cp & bitplane);
        }
        if (strcmp(string2, "NOSAVE", 6) != 0)
        {
            fprintf(out, "%c", (char)kl);
        }
        if (i < height1)
        {
            _asm mov ax,j
            _asm cmp ax,width1
            _asm jg label3
            _asm mov ax,j
            _asm add ax,0
            _asm je label6

```

```

        {
            _asm mov ax,b
            _asm add ax,xdim
            _asm jnc label5
            _asm jmp label6

label5:
            _asm inc b
            _asm jmp label2
        }

label6:
        {
            _asm mov ax,i
            _asm mul width1
            _asm add ax, j
            _asm jnc label1
            _asm inc dx

label1:
            _asm mov d,dx
            _asm mov b,ax
            _asm cmp dx,oldb
            _asm je label2
            {
                _asm mov ax,4f05h;
                _asm mov bx,00h;
                _asm mov dx,d;
                _asm int 10h;
            }

label2:
            oldb=d;
            _asm mov es,[ourseg];
            _asm mov al,byte ptr k1;
            _asm mov bx,b;
            _asm mov es:[bx],al
        }

label3:
            _asm nop
        }
    }

/*Pause*/
getch();

/*Reset text mode*/
_asm mov ax,3
_asm int 10h

/*Close files*/
_close(input_file);
_fcloseall();
}

/*Main program ends*/

```

Appendix II

The File Subtractor

The File Subtractor

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
char string[80],string1[80],string2[80];
unsigned char c,c1,c2;
int k,k1,k2;

FILE *input_file1,*input_file2,*out,*infile;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile","r");
    fscanf(infile,"%s",string);
    fscanf(infile,"%s",string1);
    fscanf(infile,"%s",string2);

    /*Open files*/
    input_file1=fopen(string,"r+b");
    input_file2=fopen(string1,"r+b");
    out=fopen(string2,"w+b");

    /*Subtract files*/
    label:
    fscanf(input_file1,"%c",&c1);
    fscanf(input_file2,"%c",&c2);
    k1 = c1-128;
    k2 = c2-128;
    k = k1 - k2;
    if (k > 127) {k = k - 256;}
    if (k < -128) {k = 256+k;}
    c = (char) (k+128);
    if (feof(input_file1)==0)
    {
        fprintf(out,"%c",c);
    }
    if (feof(input_file1)!=0)
    {
        goto label1;
    }
    if (feof(input_file2) !=0)
    {
        goto label1;
    }
    goto label;

    /*Close files*/
    label1:
    _fcloseall();
}
/*Main program ends*/
```


Appendix JJ

The File Adder

The File Adder

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
char string[80],string1[80],string2[80];
unsigned char c,c1,c2;
int k,k1,k2;

FILE *input_file1,*input_file2,*out,*infile;
/*Globals end*/

/*Main program begins*/
main()
{

/*Read in information*/
infile=fopen("tfile","r");
fscanf(infile,"%s",string);
fscanf(infile,"%s",string1);
fscanf(infile,"%s",string2);

/*Open files*/
input_file1=fopen(string,"r+b");
input_file2=fopen(string1,"r+b");
out=fopen(string2,"w+b");

/*Add files*/
label:
fscanf(input_file1,"%c",&c1);
fscanf(input_file2,"%c",&c2);
k1 = c1-128;
k2 = c2-128;
k = k1 + k2;
if (k > 127) k=k-256;
if (k < -128) k=256+k;
c=(char) (k+128);
if (feof(input_file1)==0)
{
    fprintf(out,"%c",c);
}
if (feof(input_file1) !=0)
{
    goto label1;
}
if (feof(input_file2) !=0)
{
    goto label1;
}
goto label;

/*Close files*/
label1:
fcloseall();
}
/*Main program ends*/
```

Appendix KK

The Single File Examiner

The Single File Examiner

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
float sum,n;
unsigned char s,c,c1,c2;
char string[80],string1[80],string2[80];
int nl,yblock,xblock,i,j,jj,xdim,ydim;
int blocksize,k,k2,k1,a[32][1024],z;

FILE *input_file1,*input_file2,*infile;
/*Globals end*/

/*Main program begins*/
main()
{

/*Read in information*/
infile=fopen("tfile","r");
fscanf(infile,"%s",string);
fscanf(infile,"%i",&xdim);
fscanf(infile,"%i",&ydim);
fscanf(infile,"%i",&blocksize);

/*Open file*/
input_file1=fopen(string,"r+b");

/*Examine file*/
n = 0;
sum = 0;
if (xdim/blocksize != (float)xdim/(float)blocksize)
{
    xblock = xdim/blocksize + 1;
}
else
{
    xblock = xdim/blocksize;
}
if (ydim/blocksize != (float)ydim/(float)blocksize)
{
    yblock = ydim/blocksize + 1;
}
else
{
    yblock = ydim/blocksize;
}
for (nl=1;nl<=yblock;nl++)
{
    for (i=0;i<1024;i++)
    {
        for (j=0;j<32;j++)
        {
            a[j][i]=0;
        }
    }
    for (i=1;i<=blocksize;i++)
    {
        for (j=1;j<=xdim;j++)
        {
            fscanf(input_file1,"%c",&c1);
            a[i][j]=c1;
        }
    }
}
```

```

        if (feof(input_file1) != 0) break;
    }
}
for (k=0; k<=xblock-1; k++)
{
    for (j=1; j<=blocksize; j++)
    {
        for (i=1; i<=blocksize; i++)
        {
            jj=i+blocksize*k;
            printf("%4i", a[j][jj]-128);
        }
        if (j!=blocksize)
        {
            printf("\n");
        }
        else
        {
            printf("    Block %i,%i \n", n1, k+1);
        }
    }
    printf("\n");
    s=getch();
    z = (int)s;
    if (z==27) goto labell;
}

/*Close file*/
labell:
fcloseall();
}
/*Main program ends*/

```

Appendix LL

The Two File Examiner

The Two File Examiner

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
float sum,n;
unsigned char s,c,c1,c2;
char string[80],string1[80];
int k,k2,k1,z,a[32][1024],b[1024][32];
int nl,yblock,xblock,i,j,jj,xdim,ydim,blocksize;

FILE *input_file1,*input_file2,*infile;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile","r");
    fscanf(infile,"%s",string);
    fscanf(infile,"%s",string1);
    fscanf(infile,"%i",&xdim);
    fscanf(infile,"%i",&ydim);
    fscanf(infile,"%i",&blocksize);

    /*Open files*/
    input_file1=fopen(string,"r+b");
    input_file2=fopen(string1,"r+b");

    /*Examine two files*/
    n = 0;
    sum = 0;
    if (xdim/blocksize != (float)xdim/(float)blocksize)
    {
        xblock = xdim/blocksize + 1;
    }
    else
    {
        xblock = xdim/blocksize;
    }
    if (ydim/blocksize != (float)ydim/(float)blocksize)
    {
        yblock = ydim/blocksize + 1;
    }
    else
    {
        yblock = ydim/blocksize;
    }

    for (nl=1;nl<=yblock;nl++)
    {
        for (i=0;i<=1024;i++)
        {
            for (j=0;j<=32;j++)
            {
                a[j][i]=0;
                b[j][i]=0;
            }
        }
        for (i=1;i<=blocksize;i++)
        {
```

```

        for (j=1;j<=xdim;j++)
        {
            fscanf(input_file1,"%c",&c1);
            fscanf(input_file2,"%c",&c2);
            a[i][j]=c1;
            b[i][j]=c2;
            if (feof(input_file1)!=0) break;
            if (feof(input_file2)!=0) break;
        }
    for (k=0;k<=xblock-1;k++)
    {
        for (j=1;j<=blocksize;j++)
        {
            for (i=1;i<=blocksize;i++)
            {
                jj=i+blocksize*k;
                printf("%4i/%4i",a[j][jj]-128,b[j][jj]-128);
            }
            if (j!=blocksize) printf("\n");
            else printf("  Block %i,%i \n",n1,k+1);
        }
        printf("\n");
        s=getch();
        z = (int)s;
        if (z == 27) goto labell;
    }
}

/*Close files*/
labell:
fcloseall();
}

/*Main program ends*/

```


Appendix MM

The Image Paster

The Image Paster

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
int i, j, k, xdim, ydim;
char string[80], string1[80], string2[80], top[80];
unsigned char cp;

FILE *input_file1, *input_file2, *infile, *out;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%s", string2);
    fscanf(infile, "%i", &xdim);
    fscanf(infile, "%i", &ydim);
    fscanf(infile, "%s", top);

    /*Open files*/
    input_file1=fopen(string, "r+b");
    input_file2=fopen(string1, "r+b");
    out=fopen(string2, "w+b");

    /*Paste images*/
    if (strcmp(top, "TOP", 3)==0)
    {
        for (k=1; k<=ydim; k++)
        {
            for (j=1; j<=xdim; j++)
            {
                fscanf(input_file1, "%c", &cp);
                fprintf(out, "%c", cp);
            }
        }
        for (k=1; k<=ydim; k++)
        {
            for (j=1; j<=xdim; j++)
            {
                fscanf(input_file2, "%c", &cp);
                fprintf(out, "%c", cp);
            }
        }
    }
    if (strcmp(top, "SID", 3)==0)
    {
        for (k=1; k<=ydim; k++)
        {
            for (j=1; j<=xdim; j++)
            {
                fscanf(input_file1, "%c", &cp);
                fprintf(out, "%c", cp);
            }
        }
        for (j=1; j<=xdim; j++)
        {

```

```
        fscanf(input_file2,"%c",&cp);
        fprintf(out,"%c",cp);
    }
}

/*Close files*/
fcloseall();
}
/*Main program ends*/
```

Appendix NN

The Huffman Encoder

The Huffman Encoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-C.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "bitio.h"
#include "errhand.h"
#include "main.h"

typedef struct tree_node {
    unsigned int count;
    unsigned int saved_count;
    int child_0;
    int child_1;
} NODE;

typedef struct code {
    unsigned int code;
    int code_bits;
} CODE;

#define END_OF_STREAM 256

#ifdef __STDC__

void count_bytes( FILE *input, unsigned long *long_counts );
void scale_counts( unsigned long *long_counts, NODE *nodes );
int build_tree( NODE *nodes );
void convert_tree_to_code( NODE *nodes,
                           CODE *codes,
                           unsigned int code_so_far,
                           int bits,
                           int node );
void output_counts( BIT_FILE *output, NODE *nodes );
void input_counts( BIT_FILE *input, NODE *nodes );
void print_model( NODE *nodes, CODE *codes );
void compress_data( FILE *input, BIT_FILE *output, CODE *codes );
void expand_data( BIT_FILE *input, FILE *output, NODE *nodes,
                  int root_node );
void print_char( int c );

#else /* __STDC__ */

void count_bytes();
void scale_counts();
int build_tree();
void convert_tree_to_code();
void output_counts();
void input_counts();
void print_model();
void compress_data();
void expand_data();
void print_char();

#endif /* __STDC__ */

char *CompressionName = "static order 0 model with Huffman coding";
char *Usage =
    "infile outfile [-d]\n\nSpecifying -d will dump the modeling data\n";
```

```

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    unsigned long *counts;
    NODE *nodes;
    CODE *codes;
    int root_node;

    counts = (unsigned long *) calloc( 256, sizeof( unsigned long ) );
    if ( counts == NULL )
        fatal_error( "Error allocating counts array\n" );
    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    if ( ( codes = (CODE *) calloc( 257, sizeof( CODE ) ) ) == NULL )
        fatal_error( "Error allocating codes array\n" );
    count_bytes( input, counts );
    scale_counts( counts, nodes );
    output_counts( output, nodes );
    root_node = build_tree( nodes );
    convert_tree_to_code( nodes, codes, 0, 0, root_node );
    if ( argc > 0 && strcmp( argv[ 0 ], "-d" ) == 0 )
        print_model( nodes, codes );
    compress_data( input, output, codes );
    free( (char *) counts );
    free( (char *) nodes );
    free( (char *) codes );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    NODE *nodes;
    int root_node;

    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    input_counts( input, nodes );
    root_node = build_tree( nodes );
    if ( argc > 0 && strcmp( argv[ 0 ], "-d" ) == 0 )
        print_model( nodes, 0 );
    expand_data( input, output, nodes, root_node );
    free( (char *) nodes );
}

void output_counts( output, nodes )
BIT_FILE *output;
NODE *nodes;
{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && nodes[ first ].count == 0 )
        first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;

```

```

    for ( ; ; ) {
        for ( ; last < 256 ; last++ )
            if ( nodes[ last ].count == 0 )
                break;
        last--;
        for ( next = last + 1; next < 256 ; next++ )
            if ( nodes[ next ].count != 0 )
                break;
        if ( next > 255 )
            break;
        if ( ( next - last ) > 3 )
            break;
        last = next;
    };
    if ( putc( first, output->file ) != first )
        fatal_error( "Error writing byte counts\n" );
    if ( putc( last, output->file ) != last )
        fatal_error( "Error writing byte counts\n" );
    for ( i = first ; i <= last ; i++ ) {
        if ( putc( nodes[ i ].count, output->file ) !=
            (int) nodes[ i ].count )
            fatal_error( "Error writing byte counts\n" );
    }
    if ( putc( 0, output->file ) != 0 )
        fatal_error( "Error writing byte counts\n" );
}

void input_counts( input, nodes )
BIT_FILE *input;
NODE *nodes;
{
    int first;
    int last;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        nodes[ i ].count = 0;
    if ( ( first = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input->file ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                nodes[ i ].count = (unsigned int) c;
        if ( ( first = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
        if ( first == 0 )
            break;
        if ( ( last = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
    }
    nodes[ END_OF_STREAM ].count = 1;
}

#ifdef SEEK_SET
#define SEEK_SET 0
#endif

void count_bytes( input, counts )
FILE *input;

```

```

unsigned long *counts;
{
    long input_marker;
    int c;

    input_marker = ftell( input );
    while ( ( c = getc( input ) ) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, nodes )
unsigned long *counts;
NODE *nodes;
{
    unsigned long max_count;
    int i;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
        if ( counts[ i ] > max_count )
            max_count = counts[ i ];
    if ( max_count == 0 ) {
        counts[ 0 ] = 1;
        max_count = 1;
    }
    max_count = max_count / 255;
    max_count = max_count + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        nodes[ i ].count = (unsigned int) ( counts[ i ] / max_count );
        if ( nodes[ i ].count == 0 && counts[ i ] != 0 )
            nodes[ i ].count = 1;
    }
    nodes[ END_OF_STREAM ].count = 1;
}

int build_tree( nodes )
NODE *nodes;
{
    int next_free;
    int i;
    int min_1;
    int min_2;

    nodes[ 513 ].count = 0xffff;
    for ( next_free = END_OF_STREAM + 1 ; ; next_free++ ) {
        min_1 = 513;
        min_2 = 513;
        for ( i = 0 ; i < next_free ; i++ )
            if ( nodes[ i ].count != 0 ) {
                if ( nodes[ i ].count < nodes[ min_1 ].count ) {
                    min_2 = min_1;
                    min_1 = i;
                } else if ( nodes[ i ].count < nodes[ min_2 ].count )
                    min_2 = i;
            }
        if ( min_2 == 513 )
            break;
        nodes[ next_free ].count = nodes[ min_1 ].count
                                + nodes[ min_2 ].count;
        nodes[ min_1 ].saved_count = nodes[ min_1 ].count;
        nodes[ min_1 ].count = 0;
        nodes[ min_2 ].saved_count = nodes[ min_2 ].count;
        nodes[ min_2 ].count = 0;
        nodes[ next_free ].child_0 = min_1;
        nodes[ next_free ].child_1 = min_2;
    }
}

```



```

    }
    next_free--;
    nodes[ next_free ].saved_count = nodes[ next_free ].count;
    return( next_free );
}

void convert_tree_to_code( nodes, codes, code_so_far, bits, node )
NODE *nodes;
CODE *codes;
unsigned int code_so_far;
int bits;
int node;
{
    if ( node <= END_OF_STREAM ) {
        codes[ node ].code = code_so_far;
        codes[ node ].code_bits = bits;
        return;
    }
    code_so_far <<= 1;
    bits++;
    convert_tree_to_code( nodes, codes, code_so_far, bits,
                        nodes[ node ].child_0 );
    convert_tree_to_code( nodes, codes, code_so_far | 1,
                        bits, nodes[ node ].child_1 );
}

void print_model( nodes, codes )
NODE *nodes;
CODE *codes;
{
    int i;

    for ( i = 0 ; i < 513 ; i++ ) {
        if ( nodes[ i ].saved_count != 0 ) {
            printf( "node=" );
            print_char( i );
            printf( " count=%3d", nodes[ i ].saved_count );
            printf( " child_0=" );
            print_char( nodes[ i ].child_0 );
            printf( " child_1=" );
            print_char( nodes[ i ].child_1 );
            if ( codes && i <= END_OF_STREAM ) {
                printf( " Huffman code=" );
                FilePrintBinary( stdout, codes[ i ].code, codes[ i ].code_bits
            );
            }
            printf( "\n" );
        }
    }
}

void print_char( c )
int c;
{
    if ( c >= 0x20 && c < 127 )
        printf( "'%c'", c );
    else
        printf( "%3d", c );
}

void compress_data( input, output, codes )
FILE *input;
BIT_FILE *output;
CODE *codes;

```

```

(
    int c;

    while ( ( c =getc( input ) ) != EOF )
        OutputBits( output, (unsigned long) codes[ c ].code,
                    codes[ c ].code_bits );
    OutputBits( output, (unsigned long) codes[ END_OF_STREAM ].code,
                codes[ END_OF_STREAM ].code_bits );
)

void expand_data( input, output, nodes, root_node )
BIT_FILE *input;
FILE *output;
NODE *nodes;
int root_node;
{
    int node;

    for ( ; ; ) {
        node = root_node;
        do {
            if ( InputBit( input ) )
                node = nodes[ node ].child_1;
            else
                node = nodes[ node ].child_0;
        } while ( node > END_OF_STREAM );
        if ( node == END_OF_STREAM )
            break;
        if ( ( putc( node, output ) ) != node )
            fatal_error( "Error trying to write expanded byte to output" );
    }
}

```

Appendix OO

The Adaptive Huffman Encoder

The Adaptive Huffman Encoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-C.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "bitio.h"
#include "errhand.h"

char *CompressionName = "Adaptive Huffman coding, with escape codes";
char *Usage           = "infile outfile [ -d ]";

#define END_OF_STREAM      256
#define ESCAPE             257
#define SYMBOL_COUNT       258
#define NODE_TABLE_COUNT  ( ( SYMBOL_COUNT * 2 ) - 1 )
#define ROOT_NODE         0
#define MAX_WEIGHT         0x8000
#define TRUE               1
#define FALSE              0

typedef struct tree {
    int leaf[ SYMBOL_COUNT ];
    int next_free_node;
    struct node {
        unsigned int weight;
        int parent;
        int child_is_leaf;
        int child;
    } nodes[ NODE_TABLE_COUNT ];
} TREE;

TREE Tree;

#ifdef __STDC__

void CompressFile( FILE *input, BIT_FILE *output, int argc, char *argv[] );
void ExpandFile( BIT_FILE *input, FILE *output, int argc, char *argv[] );
void InitializeTree( TREE *tree );
void EncodeSymbol( TREE *tree, unsigned int c, BIT_FILE *output );
int DecodeSymbol( TREE *tree, BIT_FILE *input );
void UpdateModel( TREE *tree, int c );
void RebuildTree( TREE *tree );
void swap_nodes( TREE *tree, int i, int j );
void add_new_node( TREE *tree, int c );
void PrintTree( TREE *tree );
void print_codes( TREE *tree );
void print_code( TREE *tree, int c );
void calculate_rows( TREE *tree, int node, int level );
int calculate_columns( TREE *tree, int node, int starting_guess );
int find_minimum_column( TREE *tree, int node, int max_row );
void rescale_columns( int factor );
void print_tree( TREE *tree, int first_row, int last_row );
void print_connecting_lines( TREE *tree, int row );
void print_node_numbers( int row );
void print_weights( TREE *tree, int row );
void print_symbol( TREE *tree, int row );

#else

void CompressFile();
```

```

void ExpandFile();
void InitializeTree();
void EncodeSymbol();
int DecodeSymbol();
void UpdateModel();
void RebuildTree();
void swap_nodes();
void add_new_node();
void PrintTree();
void print_codes();
void print_code();
void calculate_rows();
int calculate_columns();
void rescale_columns();
void print_tree();
void print_connecting_lines();
void print_node_numbers();
void print_weights();
void print_symbol();

```

```

#endif

```

```

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = getc( input ) ) != EOF ) {
        EncodeSymbol( &Tree, c, output );
        UpdateModel( &Tree, c );
    }
    EncodeSymbol( &Tree, END_OF_STREAM, output );
    while ( argc > 0 ) {
        if ( strcmp( *argv, "-d" ) == 0 )
            PrintTree( &Tree );
        else
            printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

```

```

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = DecodeSymbol( &Tree, input ) ) != END_OF_STREAM ) {
        if ( putc( c, output ) == EOF )
            fatal_error( "Error writing character" );
        UpdateModel( &Tree, c );
    }
    while ( argc > 0 ) {
        if ( strcmp( *argv, "-d" ) == 0 )
            PrintTree( &Tree );
        else
            printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

```

```

    }
}

void InitializeTree( tree )
TREE *tree;
{
    int i;

    tree->nodes[ ROOT_NODE ].child      = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE ].child_is_leaf = FALSE;
    tree->nodes[ ROOT_NODE ].weight     = 2;
    tree->nodes[ ROOT_NODE ].parent     = -1;

    tree->nodes[ ROOT_NODE + 1 ].child   = END_OF_STREAM;
    tree->nodes[ ROOT_NODE + 1 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 1 ].weight  = 1;
    tree->nodes[ ROOT_NODE + 1 ].parent  = ROOT_NODE;
    tree->leaf[ END_OF_STREAM ]          = ROOT_NODE + 1;

    tree->nodes[ ROOT_NODE + 2 ].child   = ESCAPE;
    tree->nodes[ ROOT_NODE + 2 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 2 ].weight  = 1;
    tree->nodes[ ROOT_NODE + 2 ].parent  = ROOT_NODE;
    tree->leaf[ ESCAPE ]                = ROOT_NODE + 2;

    tree->next_free_node                 = ROOT_NODE + 3;

    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        tree->leaf[ i ] = -1;
}

```

```

void EncodeSymbol( tree, c, output )
TREE *tree;
unsigned int c;
BIT_FILE *output;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    if ( current_node == -1 )
        current_node = tree->leaf[ ESCAPE ];
    while ( current_node != ROOT_NODE ) {
        if ( ( current_node & 1 ) == 0 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    OutputBits( output, code, code_size );
    if ( tree->leaf[ c ] == -1 ) {
        OutputBits( output, (unsigned long) c, 8 );
        add_new_node( tree, c );
    }
}

```

```

int DecodeSymbol( tree, input )
TREE *tree;
BIT_FILE *input;
{

```

```

int current_node;
int c;

current_node = ROOT_NODE;
while ( !tree->nodes[ current_node ].child_is_leaf ) {
    current_node = tree->nodes[ current_node ].child;
    current_node += InputBit( input );
}
c = tree->nodes[ current_node ].child;
if ( c == ESCAPE ) {
    c = (int) InputBits( input, 8 );
    add_new_node( tree, c );
}
return( c );
}

void UpdateModel( tree, c )
TREE *tree;
int c;
{
    int current_node;
    int new_node;

    if ( tree->nodes[ ROOT_NODE ].weight == MAX_WEIGHT )
        RebuildTree( tree );
    current_node = tree->leaf[ c ];
    while ( current_node != -1 ) {
        tree->nodes[ current_node ].weight++;
        for ( new_node = current_node ; new_node > ROOT_NODE ; new_node-- )
            if ( tree->nodes[ new_node - 1 ].weight >=
                tree->nodes[ current_node ].weight )
                break;
        if ( current_node != new_node ) {
            swap_nodes( tree, current_node, new_node );
            current_node = new_node;
        }
        current_node = tree->nodes[ current_node ].parent;
    }
}

void RebuildTree( tree )
TREE *tree;
{
    int i;
    int j;
    int k;
    unsigned int weight;

    printf( "R" );
    j = tree->next_free_node - 1;
    for ( i = j ; i >= ROOT_NODE ; i-- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            tree->nodes[ j ] = tree->nodes[ i ];
            tree->nodes[ j ].weight = ( tree->nodes[ j ].weight + 1 ) / 2;
            j--;
        }
    }

    for ( i = tree->next_free_node - 2 ; j >= ROOT_NODE ; i -= 2, j-- ) {
        k = i + 1;
        tree->nodes[ j ].weight = tree->nodes[ i ].weight +
            tree->nodes[ k ].weight;
        weight = tree->nodes[ j ].weight;
        tree->nodes[ j ].child_is_leaf = FALSE;
        for ( k = j + 1 ; weight < tree->nodes[ k ].weight ; k++ )

```

```

        k--;
        memmove( &tree->nodes[ j ], &tree->nodes[ j + 1 ],
            ( k - j ) * sizeof( struct node ) );
        tree->nodes[ k ].weight = weight;
        tree->nodes[ k ].child = i;
        tree->nodes[ k ].child_is_leaf = FALSE;
    }
    for ( i = tree->next_free_node - 1 ; i >= ROOT_NODE ; i- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            k = tree->nodes[ i ].child;
            tree->leaf[ k ] = i;
        } else {
            k = tree->nodes[ i ].child;
            tree->nodes[ k ].parent = tree->nodes[ k + 1 ].parent = i;
        }
    }
}

void swap_nodes( tree, i, j )
TREE *tree;
int i;
int j;
{
    struct node temp;

    if ( tree->nodes[ i ].child_is_leaf )
        tree->leaf[ tree->nodes[ i ].child ] = j;
    else {
        tree->nodes[ tree->nodes[ i ].child ].parent = j;
        tree->nodes[ tree->nodes[ i ].child + 1 ].parent = j;
    }
    if ( tree->nodes[ j ].child_is_leaf )
        tree->leaf[ tree->nodes[ j ].child ] = i;
    else {
        tree->nodes[ tree->nodes[ j ].child ].parent = i;
        tree->nodes[ tree->nodes[ j ].child + 1 ].parent = i;
    }
    temp = tree->nodes[ i ];
    tree->nodes[ i ] = tree->nodes[ j ];
    tree->nodes[ j ].parent = temp.parent;
    temp.parent = tree->nodes[ j ].parent;
    tree->nodes[ j ] = temp;
}

void add_new_node( tree, c )
TREE *tree;
int c;
{
    int lightest_node;
    int new_node;
    int zero_weight_node;

    lightest_node = tree->next_free_node - 1;
    new_node = tree->next_free_node;
    zero_weight_node = tree->next_free_node + 1;
    tree->next_free_node += 2;

    tree->nodes[ new_node ] = tree->nodes[ lightest_node ];
    tree->nodes[ new_node ].parent = lightest_node;
    tree->leaf[ tree->nodes[ new_node ].child ] = new_node;

    tree->nodes[ lightest_node ].child = new_node;
    tree->nodes[ lightest_node ].child_is_leaf = FALSE;
}

```



```

        tree->nodes[ zero_weight_node ].child      = c;
        tree->nodes[ zero_weight_node ].child_is_leaf = TRUE;
        tree->nodes[ zero_weight_node ].weight     = 0;
        tree->nodes[ zero_weight_node ].parent     = lightest_node;
        tree->leaf[ c ] = zero_weight_node;
    }

    struct row {
        int first_member;
        int count;
    } rows[ 32 ];

    struct location {
        int row;
        int next_member;
        int column;
    } positions[ NODE_TABLE_COUNT ];

    void PrintTree( tree )
    TREE *tree;
    {
        int i;
        int min;

        print_codes( tree );
        for ( i = 0 ; i < 32 ; i++ ) {
            rows[ i ].count = 0;
            rows[ i ].first_member = -1;
        }
        calculate_rows( tree, ROOT_NODE, 0 );
        calculate_columns( tree, ROOT_NODE, 0 );

        min = find_minimum_column( tree, ROOT_NODE, 31 );
        rescale_columns( min );
        print_tree( tree, 0, 31 );
    }

    void print_codes( tree )
    TREE *tree;
    {
        int i;

        printf( "\n" );
        for ( i = 0 ; i < SYMBOL_COUNT ; i++ )
            if ( tree->leaf[ i ] != -1 ) {
                if ( isprint( i ) )
                    printf( "%5c: ", i );
                else
                    printf( "<3d>: ", i );
                printf( "%5u", tree->nodes[ tree->leaf[ i ] ].weight );
                printf( " " );
                print_code( tree, i );
                printf( "\n" );
            }
    }

    void print_code( tree, c )
    TREE *tree;
    int c;
    {
        unsigned long code;
        unsigned long current_bit;
        int code_size;
        int current_node;
        int i;
    }

```

```

code = 0;
current_bit = 1;
code_size = 0;
current_node = tree->leaf[ c ];
while ( current_node != ROOT_NODE ) {
    if ( current_node & 1 )
        code |= current_bit;
    current_bit <<= 1;
    code_size++;
    current_node = tree->nodes[ current_node ].parent;
};
for ( i = 0 ; i < code_size ; i++ ) {
    current_bit >>= 1;
    if ( code & current_bit )
        putc( '1', stdout );
    else
        putc( '0', stdout );
}
}

void calculate_rows( tree, node, level )
TREE *tree;
int node;
int level;
{
    if ( rows[ level ].first_member == -1 ) {
        rows[ level ].first_member = node;
        rows[ level ].count = 0;
        positions[ node ].row = level;
        positions[ node ].next_member = -1;
    } else {
        positions[ node ].row = level;
        positions[ node ].next_member = rows[ level ].first_member;
        rows[ level ].first_member = node;
        rows[ level ].count++;
    }
    if ( !tree->nodes[ node ].child_is_leaf ) {
        calculate_rows( tree, tree->nodes[ node ].child, level + 1 );
        calculate_rows( tree, tree->nodes[ node ].child + 1, level + 1 );
    }
}

int calculate_columns( tree, node, starting_guess )
TREE *tree;
int node;
int starting_guess;
{
    int next_node;
    int right_side;
    int left_side;

    next_node = positions[ node ].next_member;
    if ( next_node != -1 ) {
        if ( positions[ next_node ].column < ( starting_guess + 4 ) )
            starting_guess = positions[ next_node ].column - 4;
    }
    if ( tree->nodes[ node ].child_is_leaf ) {
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }
    right_side = calculate_columns( tree, tree->nodes[ node ].child, starting_guess + 2 );
    left_side = calculate_columns( tree, tree->nodes[ node ].child + 1, right_side - 4 );
}

```

```

        starting_guess = ( right_side + left_side ) / 2;
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }

    int find_minimum_column( tree, node, max_row )
    TREE *tree;
    int node;
    int max_row;
    {
        int min_right;
        int min_left;

        if ( tree->nodes[ node ].child_is_leaf || max_row == 0 )
            return( positions[ node ].column );
        max_row--;
        min_right = find_minimum_column( tree, tree->nodes[ node ].child + 1,
max_row );
        min_left = find_minimum_column( tree, tree->nodes[ node ].child, max_row );
        if ( min_right < min_left )
            return( min_right );
        else
            return( min_left );
    }

    void rescale_columns( factor )
    int factor;
    {
        int i;
        int node;

        for ( i = 0 ; i < 30 ; i++ ) {
            if ( rows[ i ].first_member == -1 )
                break;
            node = rows[ i ].first_member;
            do {
                positions[ node ].column -= factor;
                node = positions[ node ].next_member;
            } while ( node != -1 );
        }
    }

    void print_tree( tree, first_row, last_row )
    TREE *tree;
    int first_row;
    int last_row;
    {
        int row;

        for ( row = first_row ; row <= last_row ; row++ ) {
            if ( rows[ row ].first_member == -1 )
                break;
            if ( row > first_row )
                print_connecting_lines( tree, row );
            print_node_numbers( row );
            print_weights( tree, row );
            print_symbol( tree, row );
        }
    }

#ifdef ALPHANUMERIC

#define LEFT_END 218
#define RIGHT_END 191
#define CENTER 193

```

```

#define LINE      196
#define VERTICAL  179

#else

#define LEFT_END  '+'
#define RIGHT_END '+'
#define CENTER    '+'
#define LINE      '-'
#define VERTICAL  '|'

#endif

void print_connecting_lines( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int start_col;
    int end_col;
    int center_col;
    int node;
    int parent;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        start_col = positions[ node ].column + 2;
        node = positions[ node ].next_member;
        end_col = positions[ node ].column + 2;
        parent = tree->nodes[ node ].parent;
        center_col = positions[ parent ].column;
        center_col += 2;
        for ( ; current_col < start_col ; current_col++ )
            putc( ' ', stdout );
        putc( LEFT_END, stdout );
        for ( current_col++; current_col < center_col ; current_col++ )
            putc( LINE, stdout );
        putc( CENTER, stdout );
        for ( current_col++; current_col < end_col ; current_col++ )
            putc( LINE, stdout );
        putc( RIGHT_END, stdout );
        current_col++;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_node_numbers( row )
int row;
{
    int current_col;
    int node;
    int print_col;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( "%03d", node );
        current_col += 3;
        node = positions[ node ].next_member;
    }
}

```

```

    }
    printf( "\n" );
}

void print_weights( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;
    int print_size;
    int next_col;
    char buffer[ 10 ];

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        sprintf( buffer, "%u", tree->nodes[ node ].weight );
        if ( strlen( buffer ) < 3 )
            sprintf( buffer, "%03u", tree->nodes[ node ].weight );
        print_size = 3;
        if ( strlen( buffer ) > 3 ) {
            if ( positions[ node ].next_member == -1 )
                print_size = strlen( buffer );
            else {
                next_col = positions[ positions[ node ].next_member ].column;
                if ( ( next_col - print_col ) > 6 )
                    print_size = strlen( buffer );
                else {
                    strcpy( buffer, "--" );
                    print_size = 3;
                }
            }
        }
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( buffer );
        current_col += print_size;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_symbol( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        if ( tree->nodes[ node ].child_is_leaf )
            break;
        node = positions[ node ].next_member;
    }
    if ( node == -1 )
        return;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;

```

```

for ( ; current_col < print_col ; current_col++ )
    puts( ' ', stdout );
if ( tree->nodes[ node ].child_is_leaf ) {
    if ( isprint( tree->nodes[ node ].child ) )
        printf( "'%c'", tree->nodes[ node ].child );
    else if ( tree->nodes[ node ].child == END_OF_STREAM )
        printf( "EOF" );
    else if ( tree->nodes[ node ].child == ESCAPE )
        printf( "ESC" );
    else
        printf( "%02XH", tree->nodes[ node ].child );
} else
    printf( " %c ", VERTICAL );
current_col += 3;
node = positions[ node ].next_member;
}
printf( "\n" );
}

```

Appendix PP

The Arithmetic Order 0 Encoder

The Arithmetic Order 0 Encoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-C.C*/

#include <stdio.h>
#include <stdlib.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#ifdef __STDC__

void build_model( FILE *input, FILE *output );
void scale_counts( unsigned long counts[], unsigned char scaled_counts[] );
void build_totals( unsigned char scaled_counts[] );
void count_bytes( FILE *input, unsigned long counts[] );
void output_counts( FILE *output, unsigned char scaled_counts[] );
void input_counts( FILE *stream );
void convert_int_to_symbol( int symbol, SYMBOL *s );
void get_symbol_scale( SYMBOL *s );
int convert_symbol_to_int( int count, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );

#else

void build_model();
void scale_counts();
void build_totals();
void count_bytes();
void output_counts();
void input_counts();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();

#endif

#define END_OF_STREAM 256
short int totals[ 258 ];

char *CompressionName = "Fixed order 0 model with arithmetic coding";
char *Usage           = "in-file out-file\n\n";

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
```



```

char *argv[];
{
    int c;
    SYMBOL s;

    build_model( input, output->file );
    initialize_arithmetic_encoder();

    while ( ( c = getc( input ) ) != EOF ) {
        convert_int_to_symbol( c, &s );
        encode_symbol( output, &s );
    }
    convert_int_to_symbol( END_OF_STREAM, &s );
    encode_symbol( output, &s );
    flush_arithmetic_encoder( output );
    OutputBits( output, 0L, 16 );
    while ( argc- > 0 ) {
        printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int count;

    input_counts( input->file );
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale( &s );
        count = get_current_count( &s );
        c = convert_symbol_to_int( count, &s );
        if ( c == END_OF_STREAM )
            break;
        remove_symbol_from_stream( input, &s );
        putc( (char) c, output );
    }
    while ( argc- > 0 ) {
        printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

void build_model( input, output )
FILE *input;
FILE *output;
{
    unsigned long counts[ 256 ];
    unsigned char scaled_counts[ 256 ];

    count_bytes( input, counts );
    scale_counts( counts, scaled_counts );
    output_counts( output, scaled_counts );
    build_totals( scaled_counts );
}

#ifdef SEEK_SET
#define SEEK_SET 0

```

```

#endif

void count_bytes( input, counts )
FILE *input;
unsigned long counts[];
{
    long input_marker;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        counts[ i ] = 0;
    input_marker = ftell( input );
    while ( ( c = getc( input ) ) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, scaled_counts )
unsigned long counts[];
unsigned char scaled_counts[];
{
    int i;
    unsigned long max_count;
    unsigned int total;
    unsigned long scale;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
        if ( counts[ i ] > max_count )
            max_count = counts[ i ];
    scale = max_count / 256;
    scale = scale + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        scaled_counts[ i ] = (unsigned char) ( counts[ i ] / scale );
        if ( scaled_counts[ i ] == 0 && counts[ i ] != 0 )
            scaled_counts[ i ] = 1;
    }
    total = 1;
    for ( i = 0 ; i < 256 ; i++ )
        total += scaled_counts[ i ];
    if ( total > ( 32767 - 256 ) )
        scale = 4;
    else if ( total > 16383 )
        scale = 2;
    else
        return;
    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] /= scale;
}

void build_totals( scaled_counts )
unsigned char scaled_counts[];
{
    int i;

    totals[ 0 ] = 0;
    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        totals[ i + 1 ] = totals[ i ] + scaled_counts[ i ];
    totals[ END_OF_STREAM + 1 ] = totals[ END_OF_STREAM ] + 1;
}

void output_counts( output, scaled_counts )
FILE *output;
unsigned char scaled_counts[];

```

```

{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && scaled_counts[ first ] == 0 )
        first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;
        for ( ; ; ) {
            for ( ; last < 256 ; last++ )
                if ( scaled_counts[ last ] == 0 )
                    break;
            last--;
            for ( next = last + 1; next < 256 ; next++ )
                if ( scaled_counts[ next ] != 0 )
                    break;
            if ( next > 255 )
                break;
            if ( ( next - last ) > 3 )
                break;
            last = next;
        };
        if ( putc( first, output ) != first )
            fatal_error( "Error writing byte counts\n" );
        if ( putc( last, output ) != last )
            fatal_error( "Error writing byte counts\n" );
        for ( i = first ; i <= last ; i++ ) {
            if ( putc( scaled_counts[ i ], output ) !=
                (int) scaled_counts[ i ] )
                fatal_error( "Error writing byte counts\n" );
        }
    }
    if ( putc( 0, output ) != 0 )
        fatal_error( "Error writing byte counts\n" );
}

void input_counts( input )
FILE *input;
{
    int first;
    int last;
    int i;
    int c;
    unsigned char scaled_counts[ 256 ];

    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] = 0;
    if ( ( first = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                scaled_counts[ i ] = (unsigned char) c;
        if ( ( first = getc( input ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
        if ( first == 0 )
            break;
        if ( ( last = getc( input ) ) == EOF )

```

```

        fatal_error( "Error reading byte counts\n" );
    }
    build_totals( scaled_counts );
}

static unsigned short int code; /* The present input code value */
static unsigned short int low;  /* Start of the current code range */
static unsigned short int high; /* End of the current code range */
long underflow_bits;           /* Number of underflow bits pending */
void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )
        OutputBit( stream, ~low & 0x4000 );
}

void convert_int_to_symbol( c, s )
int c;
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
    s->low_count = totals[ c ];
    s->high_count = totals[ c + 1 ];
}

void get_symbol_scale( s )
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, s )
int count;
SYMBOL *s;
{
    int c;

    for ( c = END_OF_STREAM ; count < totals[ c ] ; c- )
        ;
    s->high_count = totals[ c + 1 ];
    s->low_count = totals[ c ];
    return( c );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high - low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {

```

```

        OutputBit( stream, high & 0x8000 );
        while ( underflow_bits > 0 ) {
            OutputBit( stream, ~high & 0x8000 );
            underflow_bits--;
        }
    }
    else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
        underflow_bits += 1;
        low &= 0x3fff;
        high |= 0x4000;
    } else
        return ;
    low <<= 1;
    high <<= 1;
    high |= 1;
}

}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;
    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long)( high - low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
        }
        else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
            code ^= 0x4000;
            low &= 0x3fff;
            high |= 0x4000;
        } else
            return;
        low <<= 1;
        high <<= 1;
    }
}

```

```
    high |= 1;  
    code <<= 1;  
    code += InputBit( stream );  
}
```

Appendix QQ

The Arithmetic Order 1 Encoder

The Arithmetic Order 1 Encoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-C.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#define MAXIMUM_SCALE 16383 /* Maximum allowed frequency count */
#define END_OF_STREAM 256 /* The EOF symbol */

extern long underflow_bits; /* The present underflow count in */
/* the arithmetic coder. */
int *totals[ 257 ]; /* Pointers to the 257 context tables */

#ifdef __STDC__
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );
void initialize_model( void );
void update_model( int symbol, int context );
void convert_int_to_symbol( int symbol, int context, SYMBOL *s );
void get_symbol_scale( int context, SYMBOL *s );
int convert_symbol_to_int( int count, int context, SYMBOL *s );
#else
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();
void initialize_model();
void update_model();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
#endif

char *CompressionName = "Adaptive order 1 model with arithmetic coding";
char *Usage = "in-file out-file\n\n";

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int context;

    context = 0;
```



```

initialize_model();
initialize_arithmetic_encoder();
for ( ; ; ) {
    c = getc( input );
    if ( c == EOF )
        c = END_OF_STREAM;
    convert_int_to_symbol( c, context, &s );
    encode_symbol( output, &s );
    if ( c == END_OF_STREAM )
        break;
    update_model( c, context );
    context = c;
}
flush_arithmetic_encoder( output );
putchar( '\n' );
while ( argc- > 0 )
    printf( "Unknown argument: %s\n", *argv++ );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int count;
    int c;
    int context;

    context = 0;
    initialize_model();
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale( context, &s );
        count = get_current_count( &s );
        c = convert_symbol_to_int( count, context, &s );
        remove_symbol_from_stream( input, &s );
        if ( c == END_OF_STREAM )
            break;
        putc( (char) c, output );
        update_model( c, context );
        context = c;
    }
    putchar( '\n' );
    while ( argc- > 0 )
        printf( "Unknown argument: %s\n", *argv++ );
}

void initialize_model()
{
    int context;
    int i;

    for ( context = 0 ; context < END_OF_STREAM ; context++ ) {
        totals[ context ] = (int *) calloc( END_OF_STREAM + 2, sizeof(int) );
        if ( totals[ context ] == NULL )
            fatal_error( "Failure allocating context %d", context );
        for ( i = 0 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
            totals[ context ][ i ] = 1;
    }
}

void update_model( symbol, context )

```

```

int symbol;
int context;
{
    int i;

    for ( i = symbol + 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
        totals[ context ][ i ]++;
    if ( totals[ context ][ END_OF_STREAM + 1 ] < MAXIMUM_SCALE )
        return;
    for ( i = 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ ) {
        totals[ context ][ i ] /= 2;
        if ( totals[ context ][ i ] <= totals[ context ][ i - 1 ] )
            totals[ context ][ i ] = totals[ context ][ i - 1 ] + 1;
    }
}

void convert_int_to_symbol( c, context, s )
int c;
int context;
SYMBOL *s;
{
    s->scale = totals[ context ][ END_OF_STREAM + 1 ];
    s->low_count = totals[ context ][ c ];
    s->high_count = totals[ context ][ c + 1 ];
}

void get_symbol_scale( context, s )
int context;
SYMBOL *s;
{
    s->scale = totals[ context ][ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, context, s )
int count;
int context;
SYMBOL *s;
{
    int c;

    for ( c = 0; count >= totals[ context ][ c + 1 ] ; c++ )
        ;
    s->high_count = totals[ context ][ c + 1 ];
    s->low_count = totals[ context ][ c ];
    return( c );
}

static unsigned short int code;
static unsigned short int low;
static unsigned short int high;
long underflow_bits;

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )

```

```

        OutputBit( stream, ~low & 0x4000 );
        OutputBits( stream, 0L, 16 );
    }

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits--;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        }
        else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;

    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

```

```

range = (long)( high - low ) + 1;
high = low + (unsigned short int)
            (( range * s->high_count ) / s->scale - 1 );
low = low + (unsigned short int)
          (( range * s->low_count ) / s->scale );
for ( ; ; ) {
    if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
    }
    else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
        code ^= 0x4000;
        low  &= 0x3fff;
        high |= 0x4000;
    } else
        return;
    low <<= 1;
    high <<= 1;
    high |= 1;
    code <<= 1;
    code += InputBit( stream );
}
}

```

Appendix RR

The Arithmetic Order 2 Encoder

The Arithmetic Order 2 Encoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-C.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#define MAXIMUM_SCALE    16383 /* Maximum allowed frequency count */
#define ESCAPE           256  /* The escape symbol */
#define DONE             (-1) /* The output stream empty symbol */
#define FLUSH            (-2) /* The symbol to flush the model */

#ifdef __STDC__

void initialize_options( int argc, char **argv );
int check_compression( FILE *input, BIT_FILE *output );
void initialize_model( void );
void update_model( int symbol );
int convert_int_to_symbol( int symbol, SYMBOL *s );
void get_symbol_scale( SYMBOL *s );
int convert_symbol_to_int( int count, SYMBOL *s );
void add_character_to_model( int c );
void flush_model( void );
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );

#else

void initialize_options();
int check_compression();
void initialize_model();
void update_model();
int convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
void add_character_to_model();
void flush_model();
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();

#endif

char *CompressionName = "Adaptive order n model with arithmetic coding";
char *Usage           = "in-file out-file [ -o order ]\n\n";
int max_order         = 3;
```

```

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int escaped;
    int flush = 0;
    long int text_count = 0;

    initialize_options( argc, argv );
    initialize_model();
    initialize_arithmetic_encoder();
    for ( ; ; ) {
        if ( ( ++text_count & 0x0ff ) == 0 )
            flush = check_compression( input, output );
        if ( !flush )
            c = getc( input );
        else
            c = FLUSH;
        if ( c == EOF )
            c = DONE;
        do {
            escaped = convert_int_to_symbol( c, &s );
            encode_symbol( output, &s );
        } while ( escaped );
        if ( c == DONE )
            break;
        if ( c == FLUSH ) {
            flush_model();
            flush = 0;
        }
        update_model( c );
        add_character_to_model( c );
    }
    flush_arithmetic_encoder( output );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int count;

    initialize_options( argc, argv );
    initialize_model();
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        do {
            get_symbol_scale( &s );
            count = get_current_count( &s );
            c = convert_symbol_to_int( count, &s );
            remove_symbol_from_stream( input, &s );
        } while ( c == ESCAPE );
        if ( c == DONE )
            break;
        if ( c != FLUSH )
            putc( (char) c, output );
    }
}

```

```

        else
            flush_model();
            update_model( c );
            add_character_to_model( c );
    }
}

void initialize_options( argc, argv )
int argc;
char *argv[];
{
    while ( argc > 0 ) {
        if ( strcmp( *argv, "-o" ) == 0 ) {
            argc--;
            max_order = atoi( *++argv );
        } else
            printf( "Unknown argument on command line: %s\n", *argv );
        argc--;
        argv++;
    }
}

int check_compression( input, output )
FILE *input;
BIT_FILE *output;
{
    static long local_input_marker = 0L;
    static long local_output_marker = 0L;
    long total_input_bytes;
    long total_output_bytes;
    int local_ratio;

    total_input_bytes = ftell( input ) - local_input_marker;
    total_output_bytes = ftell( output->file );
    total_output_bytes -= local_output_marker;
    if ( total_output_bytes == 0 )
        total_output_bytes = 1;
    local_ratio = (int)( ( total_output_bytes * 100 ) / total_input_bytes );

    local_input_marker = ftell( input );
    local_output_marker = ftell( output->file );

    return( local_ratio > 90 );
}

typedef struct {
    unsigned char symbol;
    unsigned char counts;
} STATS;
typedef struct {
    struct context *next;
} LINKS;

typedef struct context {
    int max_index;
    LINKS *links;
    STATS *stats;
    struct context *lesser_context;
} CONTEXT;

CONTEXT **contexts;

int current_order;

short int totals[ 258 ];

```



```

char scoreboard[ 256 ];

#ifdef __STDC__
void update_table( CONTEXT *table, int symbol );
void rescale_table( CONTEXT *table );
void totalize_table( CONTEXT *table );
CONTEXT *shift_to_next_context( CONTEXT *table, int c, int order);
CONTEXT *allocate_next_order_table( CONTEXT *table,
                                   int symbol,
                                   CONTEXT *lesser_context );

void recursive_flush( CONTEXT *table );
#else
void update_table();
void rescale_table();
void totalize_table();
CONTEXT *shift_to_next_context();
CONTEXT *allocate_next_order_table();
void recursive_flush();
#endif

void initialize_model()
{
    int i;
    CONTEXT *null_table;
    CONTEXT *control_table;

    current_order = max_order;
    contexts = (CONTEXT **) calloc( sizeof( CONTEXT * ), 10 );
    if ( contexts == NULL )
        fatal_error( "Failure #1: allocating context table!" );
    contexts += 2;
    null_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    if ( null_table == NULL )
        fatal_error( "Failure #2: allocating null table!" );
    null_table->max_index = -1;
    contexts[ -1 ] = null_table;
    for ( i = 0 ; i <= max_order ; i++ )
        contexts[ i ] = allocate_next_order_table( contexts[ i-1 ],
                                                  0,
                                                  contexts[ i-1 ] );

    free( (char *) null_table->stats );
    null_table->stats =
        (STATS *) calloc( sizeof( STATS ), 256 );
    if ( null_table->stats == NULL )
        fatal_error( "Failure #3: allocating null table!" );
    null_table->max_index = 255;
    for ( i=0 ; i < 256 ; i++ ) {
        null_table->stats[ i ].symbol = (unsigned char) i;
        null_table->stats[ i ].counts = 1;
    }

    control_table = (CONTEXT *) calloc( sizeof(CONTEXT), 1 );
    if ( control_table == NULL )
        fatal_error( "Failure #4: allocating null table!" );
    control_table->stats =
        (STATS *) calloc( sizeof( STATS ), 2 );
    if ( control_table->stats == NULL )
        fatal_error( "Failure #5: allocating null table!" );
    contexts[ -2 ] = control_table;
    control_table->max_index = 1;
    control_table->stats[ 0 ].symbol = -FLUSH;
    control_table->stats[ 0 ].counts = 1;
    control_table->stats[ 1 ].symbol = -DONE;
    control_table->stats[ 1 ].counts = 1;
}

```

```

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

CONTEXT *allocate_next_order_table( table, symbol, lesser_context )
CONTEXT *table;
int symbol;
CONTEXT *lesser_context;
{
    CONTEXT *new_table;
    int i;
    unsigned int new_size;

    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == (unsigned char) symbol )
            break;
    if ( i > table->max_index ) {
        table->max_index++;
        new_size = sizeof( LINKS );
        new_size *= table->max_index + 1;
        if ( table->links == NULL )
            table->links = (LINKS *) calloc( new_size, 1 );
        else
            table->links = (LINKS *)
                realloc( (char *) table->links, new_size );
        new_size = sizeof( STATS );
        new_size *= table->max_index + 1;
        if ( table->stats == NULL )
            table->stats = (STATS *) calloc( new_size, 1 );
        else
            table->stats = (STATS *)
                realloc( (char *) table->stats, new_size );
        if ( table->links == NULL )
            fatal_error( "Failure #6: allocating new table" );
        if ( table->stats == NULL )
            fatal_error( "Failure #7: allocating new table" );
        table->stats[ i ].symbol = (unsigned char) symbol;
        table->stats[ i ].counts = 0;
    }
    new_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    if ( new_table == NULL )
        fatal_error( "Failure #8: allocating new table" );
    new_table->max_index = -1;
    table->links[ i ].next = new_table;
    new_table->lesser_context = lesser_context;
    return( new_table );
}

void update_model( symbol )
int symbol;
{
    int i;
    int local_order;

    if ( current_order < 0 )
        local_order = 0;
    else
        local_order = current_order;
    if ( symbol >= 0 ) {
        while ( local_order <= max_order ) {
            if ( symbol >= 0 )
                update_table( contexts[ local_order ], symbol );
            local_order++;
        }
    }
}

```

```

        current_order = max_order;
        for ( i = 0 ; i < 256 ; i++ )
            scoreboard[ i ] = 0;
    }

void update_table( table, symbol )
CONTEXT *table;
int symbol;
{
    int i;
    int index;
    unsigned char temp;
    CONTEXT *temp_ptr;
    unsigned int new_size;
    index = 0;
    while ( index <= table->max_index &&
            table->stats[index].symbol != (unsigned char) symbol )
        index++;
    if ( index > table->max_index ) {
        table->max_index++;
        new_size = sizeof( LINKS );
        new_size *= table->max_index + 1;
        if ( current_order < max_order ) {
            if ( table->max_index == 0 )
                table->links = (LINKS *) calloc( new_size, 1 );
            else
                table->links = (LINKS *)
                    realloc( (char *) table->links, new_size );
            if ( table->links == NULL )
                fatal_error( "Error #9: reallocating table space!" );
            table->links[ index ].next = NULL;
        }
        new_size = sizeof( STATS );
        new_size *= table->max_index + 1;
        if (table->max_index==0)
            table->stats = (STATS *) calloc( new_size, 1 );
        else
            table->stats = (STATS *)
                realloc( (char *) table->stats, new_size );
        if ( table->stats == NULL )
            fatal_error( "Error #10: reallocating table space!" );
        table->stats[ index ].symbol = (unsigned char) symbol;
        table->stats[ index ].counts = 0;
    }
    i = index;
    while ( i > 0 &&
            table->stats[ index ].counts == table->stats[ i-1 ].counts )
        i--;
    if ( i != index ) {
        temp = table->stats[ index ].symbol;
        table->stats[ index ].symbol = table->stats[ i ].symbol;
        table->stats[ i ].symbol = temp;
        if ( table->links != NULL ) {
            temp_ptr = table->links[ index ].next;
            table->links[ index ].next = table->links[ i ].next;
            table->links[ i ].next = temp_ptr;
        }
        index = i;
    }
    table->stats[ index ].counts++;
    if ( table->stats[ index ].counts == 255 )
        rescale_table( table );
}

int convert_int_to_symbol( c, s )

```

```

int c;
SYMBOL *s;
{
    int i;
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
    if ( current_order == -2 )
        c = -c;
    for ( i = 0 ; i <= table->max_index ; i++ ) {
        if ( c == (int) table->stats[ i ].symbol ) {
            if ( table->stats[ i ].counts == 0 )
                break;
            s->low_count = totals[ i+2 ];
            s->high_count = totals[ i+1 ];
            return( 0 );
        }
    }
    s->low_count = totals[ 1 ];
    s->high_count = totals[ 0 ];
    current_order--;
    return( 1 );
}

void get_symbol_scale( s )
SYMBOL *s;
{
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
}

int convert_symbol_to_int( count, s )
int count;
SYMBOL *s;
{
    int c;
    CONTEXT *table;

    table = contexts[ current_order ];
    for ( c = 0; count < totals[ c ] ; c++ )
        ;
    s->high_count = totals[ c - 1 ];
    s->low_count = totals[ c ];
    if ( c == 1 ) {
        current_order--;
        return( ESCAPE );
    }
    if ( current_order < -1 )
        return( (int) -table->stats[ c-2 ].symbol );
    else
        return( table->stats[ c-2 ].symbol );
}

void add_character_to_model( c )
int c;
{
    int i;
    if ( max_order < 0 || c < 0 )
        return;
    contexts[ max_order ] =

```

```

        shift_to_next_context( contexts[ max_order ], c, max_order );
    for ( i = max_order-1 ; i > 0 ; i-- )
        contexts[ i ] = contexts[ i+1 ]->lesser_context;
}

CONTEXT *shift_to_next_context( table, c, order )
CONTEXT *table;
int c;
int order;
{
    int i;
    CONTEXT *new_lesser;
    table = table->lesser_context;
    if ( order == 0 )
        return( table->links[ 0 ].next );
    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == (unsigned char) c )
            if ( table->links[ i ].next != NULL )
                return( table->links[ i ].next );
            else
                break;
    new_lesser = shift_to_next_context( table, c, order-1 );
    table = allocate_next_order_table( table, c, new_lesser );
    return( table );
}

void rescale_table( table )
CONTEXT *table;
{
    int i;

    if ( table->max_index == -1 )
        return;
    for ( i = 0 ; i <= table->max_index ; i++ )
        table->stats[ i ].counts /= 2;
    if ( table->stats[ table->max_index ].counts == 0 &&
        table->links == NULL ) {
        while ( table->stats[ table->max_index ].counts == 0 &&
            table->max_index >= 0 )
            table->max_index--;
        if ( table->max_index == -1 ) {
            free( (char *) table->stats );
            table->stats = NULL;
        } else {
            table->stats = (STATS *)
                realloc( (char *) table->stats,
                    sizeof( STATS ) * ( table->max_index + 1 ) );
            if ( table->stats == NULL )
                fatal_error( "Error #11: reallocating stats space!" );
        }
    }
}

void totalize_table( table )
CONTEXT *table;
{
    int i;
    unsigned char max;

    for ( ; ; ) {
        max = 0;
        i = table->max_index + 2;
        totals[ i ] = 0;
        for ( ; i > 1 ; i-- ) {
            totals[ i-1 ] = totals[ i ];

```

```

        if ( table->stats[ i-2 ].counts )
            if ( ( current_order == -2 ) ||
                scoreboard[ table->stats[ i-2 ].symbol ] == 0 )
                totals[ i-1 ] += table->stats[ i-2 ].counts;
        if ( table->stats[ i-2 ].counts > max )
            max = table->stats[ i-2 ].counts;
    }
    if ( max == 0 )
        totals[ 0 ] = 1;
    else {
        totals[ 0 ] = (short int) ( 256 - table->max_index );
        totals[ 0 ] *= table->max_index;
        totals[ 0 ] /= 256;
        totals[ 0 ] /= max;
        totals[ 0 ]++;
        totals[ 0 ] += totals[ 1 ];
    }
    if ( totals[ 0 ] < MAXIMUM_SCALE )
        break;
    rescale_table( table );
}
for ( i = 0 ; i < table->max_index ; i++ )
    if (table->stats[i].counts != 0)
        scoreboard[ table->stats[ i ].symbol ] = 1;
}

void recursive_flush( table )
CONTEXT *table;
{
    int i;

    if ( table->links != NULL )
        for ( i = 0 ; i <= table->max_index ; i++ )
            if ( table->links[ i ].next != NULL )
                recursive_flush( table->links[ i ].next );
    rescale_table( table );
}

void flush_model()
{
    putc( 'F', stdout );
    recursive_flush( contexts[ 0 ] );
}

static unsigned short int code; /* The present input code value */
static unsigned short int low; /* Start of the current code range */
static unsigned short int high; /* End of the current code range */
long underflow_bits; /* Number of underflow bits pending */

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )
        OutputBit( stream, ~low & 0x4000 );
}

```

```

    OutputBits( stream, 0L, 16 );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits--;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        }
        else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;

    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long) ( high - low ) + 1;

```

```

high = low + (unsigned short int)
            (( range * s->high_count ) / s->scale - 1 );
low = low + (unsigned short int)
          (( range * s->low_count ) / s->scale );
for ( ; ; ) {
    if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
    }
    else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
        code ^= 0x4000;
        low  &= 0x3fff;
        high |= 0x4000;
    } else
        return;
    low <<= 1;
    high <<= 1;
    high |= 1;
    code <<= 1;
    code += InputBit( stream );
}
}

```


Appendix SS

The Huffman Decoder

The Huffman Decoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-E.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "bitio.h"
#include "errhand.h"
#include "main.h"

typedef struct tree_node {
    unsigned int count;
    unsigned int saved_count;
    int child_0;
    int child_1;
} NODE;

typedef struct code {
    unsigned int code;
    int code_bits;
} CODE;

#define END_OF_STREAM 256

#ifdef __STDC__

void count_bytes( FILE *input, unsigned long *long_counts );
void scale_counts( unsigned long *long_counts, NODE *nodes );
int build_tree( NODE *nodes );
void convert_tree_to_code( NODE *nodes,
                           CODE *codes,
                           unsigned int code_so_far,
                           int bits,
                           int node );
void output_counts( BIT_FILE *output, NODE *nodes );
void input_counts( BIT_FILE *input, NODE *nodes );
void print_model( NODE *nodes, CODE *codes );
void compress_data( FILE *input, BIT_FILE *output, CODE *codes );
void expand_data( BIT_FILE *input, FILE *output, NODE *nodes,
                  int root_node );
void print_char( int c );

#else /* __STDC__ */

void count_bytes();
void scale_counts();
int build_tree();
void convert_tree_to_code();
void output_counts();
void input_counts();
void print_model();
void compress_data();
void expand_data();
void print_char();

#endif /* __STDC__ */

char *CompressionName = "static order 0 model with Huffman coding";
char *Usage =
    "infile outfile [-d]\n\nSpecifying -d will dump the modeling data\n";
```

```

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    unsigned long *counts;
    NODE *nodes;
    CODE *codes;
    int root_node;

    counts = (unsigned long *) calloc( 256, sizeof( unsigned long ) );
    if ( counts == NULL )
        fatal_error( "Error allocating counts array\n" );
    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    if ( ( codes = (CODE *) calloc( 257, sizeof( CODE ) ) ) == NULL )
        fatal_error( "Error allocating codes array\n" );
    count_bytes( input, counts );
    scale_counts( counts, nodes );
    output_counts( output, nodes );
    root_node = build_tree( nodes );
    convert_tree_to_code( nodes, codes, 0, 0, root_node );
    if ( argc > 0 && strcmp( argv[ 0 ], "-d" ) == 0 )
        print_model( nodes, codes );
    compress_data( input, output, codes );
    free( (char *) counts );
    free( (char *) nodes );
    free( (char *) codes );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    NODE *nodes;
    int root_node;

    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    input_counts( input, nodes );
    root_node = build_tree( nodes );
    if ( argc > 0 && strcmp( argv[ 0 ], "-d" ) == 0 )
        print_model( nodes, 0 );
    expand_data( input, output, nodes, root_node );
    free( (char *) nodes );
}

void output_counts( output, nodes )
BIT_FILE *output;
NODE *nodes;
{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && nodes[ first ].count == 0 )
        first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;

```

```

    for ( ; ; ) {
        for ( ; last < 256 ; last++ )
            if ( nodes[ last ].count == 0 )
                break;
        last--;
        for ( next = last + 1; next < 256 ; next++ )
            if ( nodes[ next ].count != 0 )
                break;
        if ( next > 255 )
            break;
        if ( ( next - last ) > 3 )
            break;
        last = next;
    };
    if ( putc( first, output->file ) != first )
        fatal_error( "Error writing byte counts\n" );
    if ( putc( last, output->file ) != last )
        fatal_error( "Error writing byte counts\n" );
    for ( i = first ; i <= last ; i++ ) {
        if ( putc( nodes[ i ].count, output->file ) !=
            (int) nodes[ i ].count )
            fatal_error( "Error writing byte counts\n" );
    }
    if ( putc( 0, output->file ) != 0 )
        fatal_error( "Error writing byte counts\n" );
}

void input_counts( input, nodes )
BIT_FILE *input;
NODE *nodes;
{
    int first;
    int last;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        nodes[ i ].count = 0;
    if ( ( first = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input->file ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                nodes[ i ].count = (unsigned int) c;
        if ( ( first = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
        if ( first == 0 )
            break;
        if ( ( last = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
    }
    nodes[ END_OF_STREAM ].count = 1;
}

#ifdef SEEK_SET
#define SEEK_SET 0
#endif

void count_bytes( input, counts )
FILE *input;

```

```

unsigned long *counts;
{
    long input_marker;
    int c;

    input_marker = ftell( input );
    while ( ( c =getc( input )) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, nodes )
unsigned long *counts;
NODE *nodes;
{
    unsigned long max_count;
    int i;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
        if ( counts[ i ] > max_count )
            max_count = counts[ i ];
    if ( max_count == 0 ) {
        counts[ 0 ] = 1;
        max_count = 1;
    }
    max_count = max_count / 255;
    max_count = max_count + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        nodes[ i ].count = (unsigned int) ( counts[ i ] / max_count );
        if ( nodes[ i ].count == 0 && counts[ i ] != 0 )
            nodes[ i ].count = 1;
    }
    nodes[ END_OF_STREAM ].count = 1;
}

int build_tree( nodes )
NODE *nodes;
{
    int next_free;
    int i;
    int min_1;
    int min_2;

    nodes[ 513 ].count = 0xffff;
    for ( next_free = END_OF_STREAM + 1 ; ; next_free++ ) {
        min_1 = 513;
        min_2 = 513;
        for ( i = 0 ; i < next_free ; i++ )
            if ( nodes[ i ].count != 0 ) {
                if ( nodes[ i ].count < nodes[ min_1 ].count ) {
                    min_2 = min_1;
                    min_1 = i;
                } else if ( nodes[ i ].count < nodes[ min_2 ].count )
                    min_2 = i;
            }
        if ( min_2 == 513 )
            break;
        nodes[ next_free ].count = nodes[ min_1 ].count
                                + nodes[ min_2 ].count;
        nodes[ min_1 ].saved_count = nodes[ min_1 ].count;
        nodes[ min_1 ].count = 0;
        nodes[ min_2 ].saved_count = nodes[ min_2 ].count;
        nodes[ min_2 ].count = 0;
        nodes[ next_free ].child_0 = min_1;
        nodes[ next_free ].child_1 = min_2;
    }
}

```

```

    }
    next_free--;
    nodes[ next_free ].saved_count = nodes[ next_free ].count;
    return( next_free );
}

void convert_tree_to_code( nodes, codes, code_so_far, bits, node )
NODE *nodes;
CODE *codes;
unsigned int code_so_far;
int bits;
int node;
{
    if ( node <= END_OF_STREAM ) {
        codes[ node ].code = code_so_far;
        codes[ node ].code_bits = bits;
        return;
    }
    code_so_far <<= 1;
    bits++;
    convert_tree_to_code( nodes, codes, code_so_far, bits,
                          nodes[ node ].child_0 );
    convert_tree_to_code( nodes, codes, code_so_far | 1,
                          bits, nodes[ node ].child_1 );
}

void print_model( nodes, codes )
NODE *nodes;
CODE *codes;
{
    int i;

    for ( i = 0 ; i < 513 ; i++ ) {
        if ( nodes[ i ].saved_count != 0 ) {
            printf( "node=" );
            print_char( i );
            printf( " count=%3d", nodes[ i ].saved_count );
            printf( " child_0=" );
            print_char( nodes[ i ].child_0 );
            printf( " child_1=" );
            print_char( nodes[ i ].child_1 );
            if ( codes && i <= END_OF_STREAM ) {
                printf( " Huffman code=" );
                FilePrintBinary( stdout, codes[ i ].code, codes[ i ].code_bits
);
            }
            printf( "\n" );
        }
    }
}

void print_char( c )
int c;
{
    if ( c >= 0x20 && c < 127 )
        printf( "'%c'", c );
    else
        printf( "%3d", c );
}

void compress_data( input, output, codes )
FILE *input;
BIT_FILE *output;
CODE *codes;

```

```

{
    int c;

    while ( ( c = getc( input ) ) != EOF )
        OutputBits( output, (unsigned long) codes[ c ].code,
                    codes[ c ].code_bits );
    OutputBits( output, (unsigned long) codes[ END_OF_STREAM ].code,
                codes[ END_OF_STREAM ].code_bits );
}

void expand_data( input, output, nodes, root_node )
BIT_FILE *input;
FILE *output;
NODE *nodes;
int root_node;
{
    int node;

    for ( ; ; ) {
        node = root_node;
        do {
            if ( InputBit( input ) )
                node = nodes[ node ].child_1;
            else
                node = nodes[ node ].child_0;
        } while ( node > END_OF_STREAM );
        if ( node == END_OF_STREAM )
            break;
        if ( ( putc( node, output ) ) != node )
            fatal_error( "Error trying to write expanded byte to output" );
    }
}

```

Appendix TT

The Adaptive Huffman Decoder

The Adaptive Huffman Decoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-E.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "bitio.h"
#include "errhand.h"

char *CompressionName = "Adaptive Huffman coding, with escape codes";
char *Usage           = "infile outfile [ -d ]";

#define END_OF_STREAM      256
#define ESCAPE            257
#define SYMBOL_COUNT      258
#define NODE_TABLE_COUNT  ( ( SYMBOL_COUNT * 2 ) - 1 )
#define ROOT_NODE         0
#define MAX_WEIGHT        0x8000
#define TRUE              1
#define FALSE             0

typedef struct tree {
    int leaf[ SYMBOL_COUNT ];
    int next_free_node;
    struct node {
        unsigned int weight;
        int parent;
        int child_is_leaf;
        int child;
    } nodes[ NODE_TABLE_COUNT ];
} TREE;

TREE Tree;

#ifdef __STDC__

void CompressFile( FILE *input, BIT_FILE *output, int argc, char *argv[] );
void ExpandFile( BIT_FILE *input, FILE *output, int argc, char *argv[] );
void InitializeTree( TREE *tree );
void EncodeSymbol( TREE *tree, unsigned int c, BIT_FILE *output );
int DecodeSymbol( TREE *tree, BIT_FILE *input );
void UpdateModel( TREE *tree, int c );
void RebuildTree( TREE *tree );
void swap_nodes( TREE *tree, int i, int j );
void add_new_node( TREE *tree, int c );
void PrintTree( TREE *tree );
void print_codes( TREE *tree );
void print_code( TREE *tree, int c );
void calculate_rows( TREE *tree, int node, int level );
int calculate_columns( TREE *tree, int node, int starting_guess );
int find_minimum_column( TREE *tree, int node, int max_row );
void rescale_columns( int factor );
void print_tree( TREE *tree, int first_row, int last_row );
void print_connecting_lines( TREE *tree, int row );
void print_node_numbers( int row );
void print_weights( TREE *tree, int row );
void print_symbol( TREE *tree, int row );

#else

void CompressFile();
```

```

void ExpandFile();
void InitializeTree();
void EncodeSymbol();
int DecodeSymbol();
void UpdateModel();
void RebuildTree();
void swap_nodes();
void add_new_node();
void PrintTree();
void print_codes();
void print_code();
void calculate_rows();
int calculate_columns();
void rescale_columns();
void print_tree();
void print_connecting_lines();
void print_node_numbers();
void print_weights();
void print_symbol();

#endif

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = getc( input ) ) != EOF ) {
        EncodeSymbol( &Tree, c, output );
        UpdateModel( &Tree, c );
    }
    EncodeSymbol( &Tree, END_OF_STREAM, output );
    while ( argc- > 0 ) {
        if ( strcmp( *argv, "-d" ) == 0 )
            PrintTree( &Tree );
        else
            printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = DecodeSymbol( &Tree, input ) ) != END_OF_STREAM ) {
        if ( putc( c, output ) == EOF )
            fatal_error( "Error writing character" );
        UpdateModel( &Tree, c );
    }
    while ( argc- > 0 ) {
        if ( strcmp( *argv, "-d" ) == 0 )
            PrintTree( &Tree );
        else
            printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

```

```

    }
}

void InitializeTree( tree )
TREE *tree;
{
    int i;

    tree->nodes[ ROOT_NODE ].child      = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE ].child_is_leaf = FALSE;
    tree->nodes[ ROOT_NODE ].weight     = 2;
    tree->nodes[ ROOT_NODE ].parent     = -1;

    tree->nodes[ ROOT_NODE + 1 ].child  = END_OF_STREAM;
    tree->nodes[ ROOT_NODE + 1 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 1 ].weight = 1;
    tree->nodes[ ROOT_NODE + 1 ].parent = ROOT_NODE;
    tree->leaf[ END_OF_STREAM ]        = ROOT_NODE + 1;

    tree->nodes[ ROOT_NODE + 2 ].child  = ESCAPE;
    tree->nodes[ ROOT_NODE + 2 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 2 ].weight = 1;
    tree->nodes[ ROOT_NODE + 2 ].parent = ROOT_NODE;
    tree->leaf[ ESCAPE ]                = ROOT_NODE + 2;

    tree->next_free_node                = ROOT_NODE + 3;

    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        tree->leaf[ i ] = -1;
}

void EncodeSymbol( tree, c, output )
TREE *tree;
unsigned int c;
BIT_FILE *output;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    if ( current_node == -1 )
        current_node = tree->leaf[ ESCAPE ];
    while ( current_node != ROOT_NODE ) {
        if ( ( current_node & 1 ) == 0 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    OutputBits( output, code, code_size );
    if ( tree->leaf[ c ] == -1 ) {
        OutputBits( output, (unsigned long) c, 8 );
        add_new_node( tree, c );
    }
}

int DecodeSymbol( tree, input )
TREE *tree;
BIT_FILE *input;
{

```

```

int current_node;
int c;

current_node = ROOT_NODE;
while ( !tree->nodes[ current_node ].child_is_leaf ) {
    current_node = tree->nodes[ current_node ].child;
    current_node += InputBit( input );
}
c = tree->nodes[ current_node ].child;
if ( c == ESCAPE ) {
    c = (int) InputBits( input, 8 );
    add_new_node( tree, c );
}
return( c );
}

void UpdateModel( tree, c )
TREE *tree;
int c;
{
    int current_node;
    int new_node;

    if ( tree->nodes[ ROOT_NODE ].weight == MAX_WEIGHT )
        RebuildTree( tree );
    current_node = tree->leaf[ c ];
    while ( current_node != -1 ) {
        tree->nodes[ current_node ].weight++;
        for ( new_node = current_node ; new_node > ROOT_NODE ; new_node-- )
            if ( tree->nodes[ new_node - 1 ].weight >=
                tree->nodes[ current_node ].weight )
                break;
        if ( current_node != new_node ) {
            swap_nodes( tree, current_node, new_node );
            current_node = new_node;
        }
        current_node = tree->nodes[ current_node ].parent;
    }
}

void RebuildTree( tree )
TREE *tree;
{
    int i;
    int j;
    int k;
    unsigned int weight;

    printf( "R" );
    j = tree->next_free_node - 1;
    for ( i = j ; i >= ROOT_NODE ; i-- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            tree->nodes[ j ] = tree->nodes[ i ];
            tree->nodes[ j ].weight = ( tree->nodes[ j ].weight + 1 ) / 2;
            j--;
        }
    }

    for ( i = tree->next_free_node - 2 ; j >= ROOT_NODE ; i -= 2, j-- ) {
        k = i + 1;
        tree->nodes[ j ].weight = tree->nodes[ i ].weight +
            tree->nodes[ k ].weight;
        weight = tree->nodes[ j ].weight;
        tree->nodes[ j ].child_is_leaf = FALSE;
        for ( k = j + 1 ; weight < tree->nodes[ k ].weight ; k++ )

```

```

        k--;
        memmove( &tree->nodes[ j ], &tree->nodes[ j + 1 ],
            ( k - j ) * sizeof( struct node ) );
        tree->nodes[ k ].weight = weight;
        tree->nodes[ k ].child = i;
        tree->nodes[ k ].child_is_leaf = FALSE;
    }
    for ( i = tree->next_free_node - 1 ; i >= ROOT_NODE ; i- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            k = tree->nodes[ i ].child;
            tree->leaf[ k ] = i;
        } else {
            k = tree->nodes[ i ].child;
            tree->nodes[ k ].parent = tree->nodes[ k + 1 ].parent = i;
        }
    }
}

void swap_nodes( tree, i, j )
TREE *tree;
int i;
int j;
{
    struct node temp;

    if ( tree->nodes[ i ].child_is_leaf )
        tree->leaf[ tree->nodes[ i ].child ] = j;
    else {
        tree->nodes[ tree->nodes[ i ].child ].parent = j;
        tree->nodes[ tree->nodes[ i ].child + 1 ].parent = j;
    }
    if ( tree->nodes[ j ].child_is_leaf )
        tree->leaf[ tree->nodes[ j ].child ] = i;
    else {
        tree->nodes[ tree->nodes[ j ].child ].parent = i;
        tree->nodes[ tree->nodes[ j ].child + 1 ].parent = i;
    }
    temp = tree->nodes[ i ];
    tree->nodes[ i ] = tree->nodes[ j ];
    tree->nodes[ j ].parent = temp.parent;
    temp.parent = tree->nodes[ j ].parent;
    tree->nodes[ j ] = temp;
}

void add_new_node( tree, c )
TREE *tree;
int c;
{
    int lightest_node;
    int new_node;
    int zero_weight_node;

    lightest_node = tree->next_free_node - 1;
    new_node = tree->next_free_node;
    zero_weight_node = tree->next_free_node + 1;
    tree->next_free_node += 2;

    tree->nodes[ new_node ] = tree->nodes[ lightest_node ];
    tree->nodes[ new_node ].parent = lightest_node;
    tree->leaf[ tree->nodes[ new_node ].child ] = new_node;

    tree->nodes[ lightest_node ].child = new_node;
    tree->nodes[ lightest_node ].child_is_leaf = FALSE;
}

```

```

        tree->nodes[ zero_weight_node ].child      = c;
        tree->nodes[ zero_weight_node ].child_is_leaf = TRUE;
        tree->nodes[ zero_weight_node ].weight     = 0;
        tree->nodes[ zero_weight_node ].parent     = lightest_node;
        tree->leaf[ c ] = zero_weight_node;
    }

    struct row {
        int first_member;
        int count;
    } rows[ 32 ];

    struct location {
        int row;
        int next_member;
        int column;
    } positions[ NODE_TABLE_COUNT ];

    void PrintTree( tree )
    TREE *tree;
    {
        int i;
        int min;

        print_codes( tree );
        for ( i = 0 ; i < 32 ; i++ ) {
            rows[ i ].count = 0;
            rows[ i ].first_member = -1;
        }
        calculate_rows( tree, ROOT_NODE, 0 );
        calculate_columns( tree, ROOT_NODE, 0 );

        min = find_minimum_column( tree, ROOT_NODE, 31 );
        rescale_columns( min );
        print_tree( tree, 0, 31 );
    }

    void print_codes( tree )
    TREE *tree;
    {
        int i;

        printf( "\n" );
        for ( i = 0 ; i < SYMBOL_COUNT ; i++ )
            if ( tree->leaf[ i ] != -1 ) {
                if ( isprint( i ) )
                    printf( "%5c: ", i );
                else
                    printf( "<3d>: ", i );
                printf( "%5u", tree->nodes[ tree->leaf[ i ] ].weight );
                printf( " " );
                print_code( tree, i );
                printf( "\n" );
            }
    }

    void print_code( tree, c )
    TREE *tree;
    int c;
    {
        unsigned long code;
        unsigned long current_bit;
        int code_size;
        int current_node;
        int i;
    }

```

```

code = 0;
current_bit = 1;
code_size = 0;
current_node = tree->leaf[ c ];
while ( current_node != ROOT_NODE ) {
    if ( current_node & 1 )
        code |= current_bit;
    current_bit <<= 1;
    code_size++;
    current_node = tree->nodes[ current_node ].parent;
};
for ( i = 0 ; i < code_size ; i++ ) {
    current_bit >>= 1;
    if ( code & current_bit )
        putc( '1', stdout );
    else
        putc( '0', stdout );
}
}

void calculate_rows( tree, node, level )
TREE *tree;
int node;
int level;
{
    if ( rows[ level ].first_member == -1 ) {
        rows[ level ].first_member = node;
        rows[ level ].count = 0;
        positions[ node ].row = level;
        positions[ node ].next_member = -1;
    } else {
        positions[ node ].row = level;
        positions[ node ].next_member = rows[ level ].first_member;
        rows[ level ].first_member = node;
        rows[ level ].count++;
    }
    if ( !tree->nodes[ node ].child_is_leaf ) {
        calculate_rows( tree, tree->nodes[ node ].child, level + 1 );
        calculate_rows( tree, tree->nodes[ node ].child + 1, level + 1 );
    }
}

int calculate_columns( tree, node, starting_guess )
TREE *tree;
int node;
int starting_guess;
{
    int next_node;
    int right_side;
    int left_side;

    next_node = positions[ node ].next_member;
    if ( next_node != -1 ) {
        if ( positions[ next_node ].column < ( starting_guess + 4 ) )
            starting_guess = positions[ next_node ].column - 4;
    }
    if ( tree->nodes[ node ].child_is_leaf ) {
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }
    right_side = calculate_columns( tree, tree->nodes[ node ].child, starting_guess + 2 );
    left_side = calculate_columns( tree, tree->nodes[ node ].child + 1, right_side - 4 );
}

```

```

        starting_guess = ( right_side + left_side ) / 2;
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }

int find_minimum_column( tree, node, max_row )
TREE *tree;
int node;
int max_row;
{
    int min_right;
    int min_left;

    if ( tree->nodes[ node ].child_is_leaf || max_row == 0 )
        return( positions[ node ].column );
    max_row--;
    min_right = find_minimum_column( tree, tree->nodes[ node ].child + 1,
max_row );
    min_left = find_minimum_column( tree, tree->nodes[ node ].child, max_row );
    if ( min_right < min_left )
        return( min_right );
    else
        return( min_left );
}

void rescale_columns( factor )
int factor;
{
    int i;
    int node;

    for ( i = 0 ; i < 30 ; i++ ) {
        if ( rows[ i ].first_member == -1 )
            break;
        node = rows[ i ].first_member;
        do {
            positions[ node ].column -= factor;
            node = positions[ node ].next_member;
        } while ( node != -1 );
    }
}

void print_tree( tree, first_row, last_row )
TREE *tree;
int first_row;
int last_row;
{
    int row;

    for ( row = first_row ; row <= last_row ; row++ ) {
        if ( rows[ row ].first_member == -1 )
            break;
        if ( row > first_row )
            print_connecting_lines( tree, row );
        print_node_numbers( row );
        print_weights( tree, row );
        print_symbol( tree, row );
    }
}

#ifndef ALPHANUMERIC

#define LEFT_END 218
#define RIGHT_END 191
#define CENTER 193

```



```

#define LINE      196
#define VERTICAL  179

#else

#define LEFT_END  '+'
#define RIGHT_END '+'
#define CENTER    '+'
#define LINE      '-'
#define VERTICAL  '|'

#endif

void print_connecting_lines( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int start_col;
    int end_col;
    int center_col;
    int node;
    int parent;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        start_col = positions[ node ].column + 2;
        node = positions[ node ].next_member;
        end_col = positions[ node ].column + 2;
        parent = tree->nodes[ node ].parent;
        center_col = positions[ parent ].column;
        center_col += 2;
        for ( ; current_col < start_col ; current_col++ )
            putc( ' ', stdout );
        putc( LEFT_END, stdout );
        for ( current_col++ ; current_col < center_col ; current_col++ )
            putc( LINE, stdout );
        putc( CENTER, stdout );
        for ( current_col++; current_col < end_col ; current_col++ )
            putc( LINE, stdout );
        putc( RIGHT_END, stdout );
        current_col++;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_node_numbers( row )
int row;
{
    int current_col;
    int node;
    int print_col;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( "%03d", node );
        current_col += 3;
        node = positions[ node ].next_member;
    }
}

```

```

    }
    printf( "\n" );
}

void print_weights( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;
    int print_size;
    int next_col;
    char buffer[ 10 ];

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        sprintf( buffer, "%u", tree->nodes[ node ].weight );
        if ( strlen( buffer ) < 3 )
            sprintf( buffer, "%03u", tree->nodes[ node ].weight );
        print_size = 3;
        if ( strlen( buffer ) > 3 ) {
            if ( positions[ node ].next_member == -1 )
                print_size = strlen( buffer );
            else {
                next_col = positions[ positions[ node ].next_member ].column;
                if ( ( next_col - print_col ) > 6 )
                    print_size = strlen( buffer );
                else {
                    strcpy( buffer, "--" );
                    print_size = 3;
                }
            }
        }
        for ( ; current_col < print_col ; current_col++ )
            puts( ' ', stdout );
        printf( buffer );
        current_col += print_size;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_symbol( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        if ( tree->nodes[ node ].child_is_leaf )
            break;
        node = positions[ node ].next_member;
    }
    if ( node == -1 )
        return;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;

```

```

for ( ; current_col < print_col ; current_col++ )
    putc( ' ', stdout );
if ( tree->nodes[ node ].child_is_leaf ) {
    if ( isprint( tree->nodes[ node ].child ) )
        printf( "'%c'", tree->nodes[ node ].child );
    else if ( tree->nodes[ node ].child == END_OF_STREAM )
        printf( "EOF" );
    else if ( tree->nodes[ node ].child == ESCAPE )
        printf( "ESC" );
    else
        printf( "%02XH", tree->nodes[ node ].child );
} else
    printf( " %c ", VERTICAL );
current_col += 3;
node = positions[ node ].next_member;
}
printf( "\n" );
}

```

Appendix UU

The Arithmetic Order 0 Decoder

The Arithmetic Order 0 Decoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-E.C*/

#include <stdio.h>
#include <stdlib.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#ifdef __STDC__

void build_model( FILE *input, FILE *output );
void scale_counts( unsigned long counts[], unsigned char scaled_counts[] );
void build_totals( unsigned char scaled_counts[] );
void count_bytes( FILE *input, unsigned long counts[] );
void output_counts( FILE *output, unsigned char scaled_counts[] );
void input_counts( FILE *stream );
void convert_int_to_symbol( int symbol, SYMBOL *s );
void get_symbol_scale( SYMBOL *s );
int convert_symbol_to_int( int count, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );

#else

void build_model();
void scale_counts();
void build_totals();
void count_bytes();
void output_counts();
void input_counts();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();

#endif

#define END_OF_STREAM 256
short int totals[ 256 ];

char *CompressionName = "Fixed order 0 model with arithmetic coding";
char *Usage           = "in-file out-file\n\n";

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
```

```

char *argv[];
{
    int c;
    SYMBOL s;

    build_model( input, output->file );
    initialize_arithmetic_encoder();

    while ( ( c = getc( input ) ) != EOF ) {
        convert_int_to_symbol( c, &s );
        encode_symbol( output, &s );
    }
    convert_int_to_symbol( END_OF_STREAM, &s );
    encode_symbol( output, &s );
    flush_arithmetic_encoder( output );
    OutputBits( output, 0L, 16 );
    while ( argc- > 0 ) {
        printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int count;

    input_counts( input->file );
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale( &s );
        count = get_current_count( &s );
        c = convert_symbol_to_int( count, &s );
        if ( c == END_OF_STREAM )
            break;
        remove_symbol_from_stream( input, &s );
       putc( (char) c, output );
    }
    while ( argc- > 0 ) {
        printf( "Unused argument: %s\n", *argv );
        argv++;
    }
}

void build_model( input, output )
FILE *input;
FILE *output;
{
    unsigned long counts[ 256 ];
    unsigned char scaled_counts[ 256 ];

    count_bytes( input, counts );
    scale_counts( counts, scaled_counts );
    output_counts( output, scaled_counts );
    build_totals( scaled_counts );
}

#ifdef SEEK_SET
#define SEEK_SET 0

```

```

#endif

void count_bytes( input, counts )
FILE *input;
unsigned long counts[];
{
    long input_marker;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        counts[ i ] = 0;
    input_marker = ftell( input );
    while ( ( c = getc( input ) ) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, scaled_counts )
unsigned long counts[];
unsigned char scaled_counts[];
{
    int i;
    unsigned long max_count;
    unsigned int total;
    unsigned long scale;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
        if ( counts[ i ] > max_count )
            max_count = counts[ i ];
    scale = max_count / 256;
    scale = scale + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        scaled_counts[ i ] = (unsigned char) ( counts[ i ] / scale );
        if ( scaled_counts[ i ] == 0 && counts[ i ] != 0 )
            scaled_counts[ i ] = 1;
    }
    total = 1;
    for ( i = 0 ; i < 256 ; i++ )
        total += scaled_counts[ i ];
    if ( total > ( 32767 - 256 ) )
        scale = 4;
    else if ( total > 16383 )
        scale = 2;
    else
        return;
    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] /= scale;
}

void build_totals( scaled_counts )
unsigned char scaled_counts[];
{
    int i;

    totals[ 0 ] = 0;
    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        totals[ i + 1 ] = totals[ i ] + scaled_counts[ i ];
    totals[ END_OF_STREAM + 1 ] = totals[ END_OF_STREAM ] + 1;
}

void output_counts( output, scaled_counts )
FILE *output;
unsigned char scaled_counts[];

```

```

{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && scaled_counts[ first ] == 0 )
        first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;
        for ( ; ) {
            for ( ; last < 256 ; last++ )
                if ( scaled_counts[ last ] == 0 )
                    break;
            last--;
            for ( next = last + 1; next < 256 ; next++ )
                if ( scaled_counts[ next ] != 0 )
                    break;
            if ( next > 255 )
                break;
            if ( ( next - last ) > 3 )
                break;
            last = next;
        };
        if ( putc( first, output ) != first )
            fatal_error( "Error writing byte counts\n" );
        if ( putc( last, output ) != last )
            fatal_error( "Error writing byte counts\n" );
        for ( i = first ; i <= last ; i++ ) {
            if ( putc( scaled_counts[ i ], output ) !=
                (int) scaled_counts[ i ] )
                fatal_error( "Error writing byte counts\n" );
        }
    }
    if ( putc( 0, output ) != 0 )
        fatal_error( "Error writing byte counts\n" );
}

void input_counts( input )
FILE *input;
{
    int first;
    int last;
    int i;
    int c;
    unsigned char scaled_counts[ 256 ];

    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] = 0;
    if ( ( first = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                scaled_counts[ i ] = (unsigned char) c;
        if ( ( first = getc( input ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
        if ( first == 0 )
            break;
        if ( ( last = getc( input ) ) == EOF )

```



```

        fatal_error( "Error reading byte counts\n" );
    }
    build_totals( scaled_counts );
}

static unsigned short int code; /* The present input code value */
static unsigned short int low;  /* Start of the current code range */
static unsigned short int high; /* End of the current code range */
long underflow_bits;           /* Number of underflow bits pending */
void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )
        OutputBit( stream, ~low & 0x4000 );
}

void convert_int_to_symbol( c, s )
int c;
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
    s->low_count = totals[ c ];
    s->high_count = totals[ c + 1 ];
}

void get_symbol_scale( s )
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, s )
int count;
SYMBOL *s;
{
    int c;

    for ( c = END_OF_STREAM ; count < totals[ c ] ; c- )
        ;
    s->high_count = totals[ c + 1 ];
    s->low_count = totals[ c ];
    return( c );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {

```

```

        OutputBit( stream, high & 0x8000 );
        while ( underflow_bits > 0 ) {
            OutputBit( stream, ~high & 0x8000 );
            underflow_bits--;
        }
    }
    else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
        underflow_bits += 1;
        low &= 0x3fff;
        high |= 0x4000;
    } else
        return ;
    low <<= 1;
    high <<= 1;
    high |= 1;
}

}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;
    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long) ( high - low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
        }
        else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
            code ^= 0x4000;
            low &= 0x3fff;
            high |= 0x4000;
        } else
            return;
        low <<= 1;
        high <<= 1;
    }
}

```

```
    high |= 1;  
    code <<= 1;  
    code += InputBit( stream );  
}  
}
```

Appendix VV

The Arithmetic Order 1 Decoder

The Arithmetic Order 1 Decoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-E.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#define MAXIMUM_SCALE 16383 /* Maximum allowed frequency count */
#define END_OF_STREAM 256 /* The EOF symbol */

extern long underflow_bits; /* The present underflow count in */
/* the arithmetic coder. */
int *totals[ 257 ]; /* Pointers to the 257 context tables */

#ifdef __STDC__
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );
void initialize_model( void );
void update_model( int symbol, int context );
void convert_int_to_symbol( int symbol, int context, SYMBOL *s );
void get_symbol_scale( int context, SYMBOL *s );
int convert_symbol_to_int( int count, int context, SYMBOL *s );
#else
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();
void initialize_model();
void update_model();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
#endif

char *CompressionName = "Adaptive order 1 model with arithmetic coding";
char *Usage = "in-file out-file\n\n";

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int context;

    context = 0;
```

```

initialize_model();
initialize_arithmetic_encoder();
for ( ; ; ) {
    c = getc( input );
    if ( c == EOF )
        c = END_OF_STREAM;
    convert_int_to_symbol( c, context, &s );
    encode_symbol( output, &s );
    if ( c == END_OF_STREAM )
        break;
    update_model( c, context );
    context = c;
}
flush_arithmetic_encoder( output );
putchar( '\n' );
while ( argc- > 0 )
    printf( "Unknown argument: %s\n", *argv++ );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int count;
    int c;
    int context;

    context = 0;
    initialize_model();
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale( context, &s );
        count = get_current_count( &s );
        c = convert_symbol_to_int( count, context, &s );
        remove_symbol_from_stream( input, &s );
        if ( c == END_OF_STREAM )
            break;
        putc( (char) c, output );
        update_model( c, context );
        context = c;
    }
    putchar( '\n' );
    while ( argc- > 0 )
        printf( "Unknown argument: %s\n", *argv++ );
}

void initialize_model()
{
    int context;
    int i;

    for ( context = 0 ; context < END_OF_STREAM ; context++ ) {
        totals[ context ] = (int *) calloc( END_OF_STREAM + 2, sizeof(int) );
        if ( totals[ context ] == NULL )
            fatal_error( "Failure allocating context %d", context );
        for ( i = 0 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
            totals[ context ][ i ] = i;
    }
}

void update_model( symbol, context )

```

```

int symbol;
int context;
{
    int i;

    for ( i = symbol + 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ )
        totals[ context ][ i ]++;
    if ( totals[ context ][ END_OF_STREAM + 1 ] < MAXIMUM_SCALE )
        return;
    for ( i = 1 ; i <= ( END_OF_STREAM + 1 ) ; i++ ) {
        totals[ context ][ i ] /= 2;
        if ( totals[ context ][ i ] <= totals[ context ][ i - 1 ] )
            totals[ context ][ i ] = totals[ context ][ i - 1 ] + 1;
    }
}

void convert_int_to_symbol( c, context, s )
int c;
int context;
SYMBOL *s;
{
    s->scale = totals[ context ][ END_OF_STREAM + 1 ];
    s->low_count = totals[ context ][ c ];
    s->high_count = totals[ context ][ c + 1 ];
}

void get_symbol_scale( context, s )
int context;
SYMBOL *s;
{
    s->scale = totals[ context ][ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, context, s )
int count;
int context;
SYMBOL *s;
{
    int c;

    for ( c = 0; count >= totals[ context ][ c + 1 ] ; c++ )
        ;
    s->high_count = totals[ context ][ c + 1 ];
    s->low_count = totals[ context ][ c ];
    return( c );
}

static unsigned short int code;
static unsigned short int low;
static unsigned short int high;
long underflow_bits;

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )

```

```

        OutputBit( stream, ~low & 0x4000 );
        OutputBits( stream, 0L, 16 );
    }

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits--;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        }
        else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;

    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

```



```

range = (long)( high - low ) + 1;
high = low + (unsigned short int)
           (( range * s->high_count ) / s->scale - 1 );
low = low + (unsigned short int)
           (( range * s->low_count ) / s->scale );
for ( ; ; ) {
    if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
    }
    else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
        code ^= 0x4000;
        low  &= 0x3fff;
        high |= 0x4000;
    } else
        return;
    low <<= 1;
    high <<= 1;
    high |= 1;
    code <<= 1;
    code += InputBit( stream );
}
}

```

Appendix WW

The Arithmetic Order 2 Decoder

The Arithmetic Order 2 Decoder

```
/*Source code is from The Data Compression Book, by Mark Nelson*/
/*It is compiled with BITIO.C, ERRHAND.C, and MAIN-E.C*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errhand.h"
#include "bitio.h"

typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;

#define MAXIMUM_SCALE    16383 /* Maximum allowed frequency count */
#define ESCAPE           256   /* The escape symbol */
#define DONE             (-1)  /* The output stream empty symbol */
#define FLUSH            (-2)  /* The symbol to flush the model */

#ifdef __STDC__

void initialize_options( int argc, char **argv );
int check_compression( FILE *input, BIT_FILE *output );
void initialize_model( void );
void update_model( int symbol );
int convert_int_to_symbol( int symbol, SYMBOL *s );
void get_symbol_scale( SYMBOL *s );
int convert_symbol_to_int( int count, SYMBOL *s );
void add_character_to_model( int c );
void flush_model( void );
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );

#else

void initialize_options();
int check_compression();
void initialize_model();
void update_model();
int convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
void add_character_to_model();
void flush_model();
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();

#endif

char *CompressionName = "Adaptive order n model with arithmetic coding";
char *Usage           = "in-file out-file [ -o order ]\n\n";
int max_order         = 3;
```

```

void CompressFile( input, output, argc, argv )
FILE *input;
BIT_FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int escaped;
    int flush = 0;
    long int text_count = 0;

    initialize_options( argc, argv );
    initialize_model();
    initialize_arithmetic_encoder();
    for ( ; ; ) {
        if ( ( ++text_count & 0x0ff ) == 0 )
            flush = check_compression( input, output );
        if ( !flush )
            c = getc( input );
        else
            c = FLUSH;
        if ( c == EOF )
            c = DONE;
        do {
            escaped = convert_int_to_symbol( c, &s );
            encode_symbol( output, &s );
        } while ( escaped );
        if ( c == DONE )
            break;
        if ( c == FLUSH ) {
            flush_model();
            flush = 0;
        }
        update_model( c );
        add_character_to_model( c );
    }
    flush_arithmetic_encoder( output );
}

void ExpandFile( input, output, argc, argv )
BIT_FILE *input;
FILE *output;
int argc;
char *argv[];
{
    SYMBOL s;
    int c;
    int count;

    initialize_options( argc, argv );
    initialize_model();
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        do {
            get_symbol_scale( &s );
            count = get_current_count( &s );
            c = convert_symbol_to_int( count, &s );
            remove_symbol_from_stream( input, &s );
        } while ( c == ESCAPE );
        if ( c == DONE )
            break;
        if ( c != FLUSH )
            putc( (char) c, output );
    }
}

```

```

        else
            flush_model();
            update_model( c );
            add_character_to_model( c );
        }
    }

void initialize_options( argc, argv )
int argc;
char *argv[];
{
    while ( argc > 0 ) {
        if ( strcmp( *argv, "-o" ) == 0 ) {
            argc--;
            max_order = atoi( *++argv );
        } else
            printf( "Unknown argument on command line: %s\n", *argv );
        argc--;
        argv++;
    }
}

int check_compression( input, output )
FILE *input;
BIT_FILE *output;
{
    static long local_input_marker = 0L;
    static long local_output_marker = 0L;
    long total_input_bytes;
    long total_output_bytes;
    int local_ratio;

    total_input_bytes = ftell( input ) - local_input_marker;
    total_output_bytes = ftell( output->file );
    total_output_bytes -= local_output_marker;
    if ( total_output_bytes == 0 )
        total_output_bytes = 1;
    local_ratio = (int)( ( total_output_bytes * 100 ) / total_input_bytes );

    local_input_marker = ftell( input );
    local_output_marker = ftell( output->file );

    return( local_ratio > 90 );
}

typedef struct {
    unsigned char symbol;
    unsigned char counts;
} STATS;
typedef struct {
    struct context *next;
} LINKS;

typedef struct context {
    int max_index;
    LINKS *links;
    STATS *stats;
    struct context *lesser_context;
} CONTEXT;

CONTEXT **contexts;

int current_order;

short int totals[ 256 ];

```

```

char scoreboard[ 256 ];

#ifdef _STDC_
void update_table( CONTEXT *table, int symbol );
void rescale_table( CONTEXT *table );
void totalize_table( CONTEXT *table );
CONTEXT *shift_to_next_context( CONTEXT *table, int c, int order);
CONTEXT *allocate_next_order_table( CONTEXT *table,
                                   int symbol,
                                   CONTEXT *lesser_context );

void recursive_flush( CONTEXT *table );
#else
void update_table();
void rescale_table();
void totalize_table();
CONTEXT *shift_to_next_context();
CONTEXT *allocate_next_order_table();
void recursive_flush();
#endif

void initialize_model()
{
    int i;
    CONTEXT *null_table;
    CONTEXT *control_table;

    current_order = max_order;
    contexts = (CONTEXT **) calloc( sizeof( CONTEXT * ), 10 );
    if ( contexts == NULL )
        fatal_error( "Failure #1: allocating context table!" );
    contexts += 2;
    null_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    if ( null_table == NULL )
        fatal_error( "Failure #2: allocating null table!" );
    null_table->max_index = -1;
    contexts[ -1 ] = null_table;
    for ( i = 0 ; i <= max_order ; i++ )
        contexts[ i ] = allocate_next_order_table( contexts[ i-1 ],
                                                    0,
                                                    contexts[ i-1 ] );

    free( (char *) null_table->stats );
    null_table->stats =
        (STATS *) calloc( sizeof( STATS ), 256 );
    if ( null_table->stats == NULL )
        fatal_error( "Failure #3: allocating null table!" );
    null_table->max_index = 255;
    for ( i=0 ; i < 256 ; i++ ) {
        null_table->stats[ i ].symbol = (unsigned char) i;
        null_table->stats[ i ].counts = 1;
    }

    control_table = (CONTEXT *) calloc( sizeof(CONTEXT), 1 );
    if ( control_table == NULL )
        fatal_error( "Failure #4: allocating null table!" );
    control_table->stats =
        (STATS *) calloc( sizeof( STATS ), 2 );
    if ( control_table->stats == NULL )
        fatal_error( "Failure #5: allocating null table!" );
    contexts[ -2 ] = control_table;
    control_table->max_index = 1;
    control_table->stats[ 0 ].symbol = -FLUSH;
    control_table->stats[ 0 ].counts = 1;
    control_table->stats[ 1 ].symbol = -DONE;
    control_table->stats[ 1 ].counts = 1;
}

```

```

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

CONTEXT *allocate_next_order_table( table, symbol, lesser_context )
CONTEXT *table;
int symbol;
CONTEXT *lesser_context;
{
    CONTEXT *new_table;
    int i;
    unsigned int new_size;

    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == (unsigned char) symbol )
            break;
    if ( i > table->max_index ) {
        table->max_index++;
        new_size = sizeof( LINKS );
        new_size *= table->max_index + 1;
        if ( table->links == NULL )
            table->links = (LINKS *) calloc( new_size, 1 );
        else
            table->links = (LINKS *)
                realloc( (char *) table->links, new_size );
        new_size = sizeof( STATS );
        new_size *= table->max_index + 1;
        if ( table->stats == NULL )
            table->stats = (STATS *) calloc( new_size, 1 );
        else
            table->stats = (STATS *)
                realloc( (char *) table->stats, new_size );
        if ( table->links == NULL )
            fatal_error( "Failure #6: allocating new table" );
        if ( table->stats == NULL )
            fatal_error( "Failure #7: allocating new table" );
        table->stats[ i ].symbol = (unsigned char) symbol;
        table->stats[ i ].counts = 0;
    }
    new_table = (CONTEXT *) calloc( sizeof( CONTEXT ), 1 );
    if ( new_table == NULL )
        fatal_error( "Failure #8: allocating new table" );
    new_table->max_index = -1;
    table->links[ i ].next = new_table;
    new_table->lesser_context = lesser_context;
    return( new_table );
}

void update_model( symbol )
int symbol;
{
    int i;
    int local_order;

    if ( current_order < 0 )
        local_order = 0;
    else
        local_order = current_order;
    if ( symbol >= 0 ) {
        while ( local_order <= max_order ) {
            if ( symbol >= 0 )
                update_table( contexts[ local_order ], symbol );
            local_order++;
        }
    }
}

```

```

    current_order = max_order;
    for ( i = 0 ; i < 256 ; i++ )
        scoreboard[ i ] = 0;
}

void update_table( table, symbol )
CONTEXT *table;
int symbol;
{
    int i;
    int index;
    unsigned char temp;
    CONTEXT *temp_ptr;
    unsigned int new_size;
    index = 0;
    while ( index <= table->max_index &&
            table->stats[index].symbol != (unsigned char) symbol )
        index++;
    if ( index > table->max_index ) {
        table->max_index++;
        new_size = sizeof( LINKS );
        new_size *= table->max_index + 1;
        if ( current_order < max_order ) {
            if ( table->max_index == 0 )
                table->links = (LINKS *) calloc( new_size, 1 );
            else
                table->links = (LINKS *)
                    realloc( (char *) table->links, new_size );
            if ( table->links == NULL )
                fatal_error( "Error #9: reallocating table space!" );
            table->links[ index ].next = NULL;
        }
        new_size = sizeof( STATS );
        new_size *= table->max_index + 1;
        if (table->max_index==0)
            table->stats = (STATS *) calloc( new_size, 1 );
        else
            table->stats = (STATS *)
                realloc( (char *) table->stats, new_size );
        if ( table->stats == NULL )
            fatal_error( "Error #10: reallocating table space!" );
        table->stats[ index ].symbol = (unsigned char) symbol;
        table->stats[ index ].counts = 0;
    }
    i = index;
    while ( i > 0 &&
            table->stats[ index ].counts == table->stats[ i-1 ].counts )
        i--;
    if ( i != index ) {
        temp = table->stats[ index ].symbol;
        table->stats[ index ].symbol = table->stats[ i ].symbol;
        table->stats[ i ].symbol = temp;
        if ( table->links != NULL ) {
            temp_ptr = table->links[ index ].next;
            table->links[ index ].next = table->links[ i ].next;
            table->links[ i ].next = temp_ptr;
        }
        index = i;
    }
    table->stats[ index ].counts++;
    if ( table->stats[ index ].counts == 255 )
        rescale_table( table );
}

int convert_int_to_symbol( c, s )

```



```

int c;
SYMBOL *s;
{
    int i;
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
    if ( current_order == -2 )
        c = -c;
    for ( i = 0 ; i <= table->max_index ; i++ ) {
        if ( c == (int) table->stats[ i ].symbol ) {
            if ( table->stats[ i ].counts == 0 )
                break;
            s->low_count = totals[ i+2 ];
            s->high_count = totals[ i+1 ];
            return( 0 );
        }
    }
    s->low_count = totals[ 1 ];
    s->high_count = totals[ 0 ];
    current_order--;
    return( 1 );
}

void get_symbol_scale( s )
SYMBOL *s;
{
    CONTEXT *table;

    table = contexts[ current_order ];
    totalize_table( table );
    s->scale = totals[ 0 ];
}

int convert_symbol_to_int( count, s )
int count;
SYMBOL *s;
{
    int c;
    CONTEXT *table;

    table = contexts[ current_order ];
    for ( c = 0; count < totals[ c ] ; c++ )
        ;
    s->high_count = totals[ c - 1 ];
    s->low_count = totals[ c ];
    if ( c == 1 ) {
        current_order--;
        return( ESCAPE );
    }
    if ( current_order < -1 )
        return( (int) -table->stats[ c-2 ].symbol );
    else
        return( table->stats[ c-2 ].symbol );
}

void add_character_to_model( c )
int c;
{
    int i;
    if ( max_order < 0 || c < 0 )
        return;
    contexts[ max_order ] =

```

```

        shift_to_next_context( contexts[ max_order ], c, max_order );
    for ( i = max_order-1 ; i > 0 ; i-- )
        contexts[ i ] = contexts[ i+1 ]->lesser_context;
}

CONTEXT *shift_to_next_context( table, c, order )
CONTEXT *table;
int c;
int order;
{
    int i;
    CONTEXT *new_lesser;
    table = table->lesser_context;
    if ( order == 0 )
        return( table->links[ 0 ].next );
    for ( i = 0 ; i <= table->max_index ; i++ )
        if ( table->stats[ i ].symbol == (unsigned char) c )
            if ( table->links[ i ].next != NULL )
                return( table->links[ i ].next );
            else
                break;
    new_lesser = shift_to_next_context( table, c, order-1 );
    table = allocate_next_order_table( table, c, new_lesser );
    return( table );
}

void rescale_table( table )
CONTEXT *table;
{
    int i;

    if ( table->max_index == -1 )
        return;
    for ( i = 0 ; i <= table->max_index ; i++ )
        table->stats[ i ].counts /= 2;
    if ( table->stats[ table->max_index ].counts == 0 &&
        table->links == NULL ) {
        while ( table->stats[ table->max_index ].counts == 0 &&
            table->max_index >= 0 )
            table->max_index--;
        if ( table->max_index == -1 ) {
            free( (char *) table->stats );
            table->stats = NULL;
        } else {
            table->stats = (STATS *)
                realloc( (char *) table->stats,
                    sizeof( STATS ) * ( table->max_index + 1 ) );
            if ( table->stats == NULL )
                fatal_error( "Error #11: reallocating state space!" );
        }
    }
}

void totalize_table( table )
CONTEXT *table;
{
    int i;
    unsigned char max;

    for ( ; ; ) {
        max = 0;
        i = table->max_index + 2;
        totals[ i ] = 0;
        for ( ; i > 1 ; i-- ) {
            totals[ i-1 ] = totals[ i ];

```

```

        if ( table->stats[ i-2 ].counts )
            if ( ( current_order == -2 ) ||
                scoreboard[ table->stats[ i-2 ].symbol ] == 0 )
                totals[ i-1 ] += table->stats[ i-2 ].counts;
        if ( table->stats[ i-2 ].counts > max )
            max = table->stats[ i-2 ].counts;
    }
    if ( max == 0 )
        totals[ 0 ] = 1;
    else {
        totals[ 0 ] = (short int) ( 256 - table->max_index ;
        totals[ 0 ] *= table->max_index;
        totals[ 0 ] /= 256;
        totals[ 0 ] /= max;
        totals[ 0 ]++;
        totals[ 0 ] += totals[ 1 ];
    }
    if ( totals[ 0 ] < MAXIMUM_SCALE )
        break;
    rescale_table( table );
}
for ( i = 0 ; i < table->max_index ; i++ )
    if (table->stats[i].counts != 0)
        scoreboard[ table->stats[ i ].symbol ] = 1;
}

void recursive_flush( table )
CONTEXT *table;
{
    int i;

    if ( table->links != NULL )
        for ( i = 0 ; i <= table->max_index ; i++ )
            if ( table->links[ i ].next != NULL )
                recursive_flush( table->links[ i ].next );
    rescale_table( table );
}

void flush_model()
{
    puts( 'F', stdout );
    recursive_flush( contexts[ 0 ] );
}

static unsigned short int code; /* The present input code value */
static unsigned short int low; /* Start of the current code range */
static unsigned short int high; /* End of the current code range */
long underflow_bits; /* Number of underflow bits pending */

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits > 0 )
        OutputBit( stream, ~low & 0x4000 );
}

```

```

    OutputBits( stream, 0L, 16 );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
        (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
        (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits--;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 ) ) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        } else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
        (((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;

    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long) ( high - low ) + 1;

```

```

high = low + (unsigned short int)
            (( range * s->high_count ) / s->scale - 1 );
low = low + (unsigned short int)
          (( range * s->low_count ) / s->scale );
for ( ; ; ) {
    if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
    }
    else if ((low & 0x4000) == 0x4000 && (high & 0x4000) == 0 ) {
        code ^= 0x4000;
        low  &= 0x3fff;
        high |= 0x4000;
    } else
        return;
    low <<= 1;
    high <<= 1;
    high |= 1;
    code <<= 1;
    code += InputBit( stream );
}
}

```

Appendix XX

The Discrete Transform Processor

The Discrete Transform Processor

```
#include <stdio.h>
#include <graph.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define RGB(r,g,b) (0x3F3F3F | ((long)(b) << 16 | (g) << 8 | (r)))

/*Globals begin*/
float block,c[64][64],templ,temp[64][64];
unsigned char red[256],blue[256],green[256],cp;
char string[80],string1[80],strip[80],cosine[80],transform[80],savefile[80];
int jj,jjj,iii,ii,n,L,m,i,j,k,xdim,ydim,mode,header,updown,expandsize;
int data[64][64],result1,result2,thresh,zonal,blocksize;
int expandx, expandy;

struct _videoconfig vc;

FILE *input_file,*inpalette,*infile,*out;
/*Globals end*/

void hadamard()
{
    for (i=1;i<=n/2;i++)
    {
        for (j=(n/2)+1;j<=n;j++)
        {
            c[i-1][j-1]=c[i-1][j-1-n/2];
        }
    }
    for (i = n/2+1;i<=n;i++)
    {
        for (j = 1;j<=n/2;j++)
        {
            c[i-1][j-1]=c[i-n/2-1][j-1];
        }
    }
    for (i = n/2+1;i<=n;i++)
    {
        for (j = (n/2)+1;j<=n;j++)
        {
            c[i-1][j-1]=-c[i-n/2-1][j-n/2-1];
        }
    }
}

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile","r");
    fscanf(infile,"%i",&mode);
    fscanf(infile,"%s",string);
    fscanf(infile,"%s",string1);
    fscanf(infile,"%i",&xdim);
    fscanf(infile,"%i",&ydim);
    fscanf(infile,"%i",&header);
    fscanf(infile,"%i",&updown);
    fscanf(infile,"%s",cosine);
    fscanf(infile,"%i",&blocksize);
    fscanf(infile,"%s",transform);
```

```

fscanf(infile, "%s", savefile);
fscanf(infile, "%i", &thresh);
fscanf(infile, "%i", &szonal);
fscanf(infile, "%s", strip);
fscanf(infile, "%i", &expandsize);

/*Open files*/
input_file=fopen(string1, "r+b");
inpalette=fopen(string, "r+b");
out=fopen(savefile, "w+b");

/*Read in palette*/
for (i=0; i<=255; i++)
{
    fscanf(inpalette, "%c", &cp);
    red[i]=cp;
}
for (i=0; i<=255; i++)
{
    fscanf(inpalette, "%c", &cp);
    green[i]=cp;
}
for (i=0; i<=255; i++)
{
    fscanf(inpalette, "%c", &cp);
    blue[i]=cp;
}
if (header!=0)
{
    for (i=1; i<=header; i++)
    {
        fscanf(input_file, "%c", &cp);
    }
}

/*Setup coefficient matrix*/
if (strcmp(cosine, "COSINE", 6)==0)
{
    for (k = 0; k <= (expandsize-1); k++)
    {
        for (n = 0; n<=(expandsize-1); n++)
        {
            if (k == 0) {c[k][n]=1.0/sqrt((float)expandsize);}
            if (k != 0) {c[k][n]=sqrt(2.0/(float)expandsize)*cos(
                3.14159*(2.0*n+1.0)*k/(2.0*(float)expandsize));}
        }
    }
}
if (strcmp(cosine, "SINE", 4)==0)
{
    for (k = 0; k <= (expandsize-1); k++)
    {
        for (n = 0; n<=(expandsize-1); n++)
        {
            c[k][n]=sqrt(2.0/((float)expandsize+1.0))*sin(
                3.14159*(n+1.0)*(k+1.0)/(1.0+(float)expandsize));
        }
    }
}
if (strcmp(cosine, "HAD", 3)==0)
{
    c[0][0]=1.0;
    n=2;
    hadamard();
    n=4;
}

```



```

        hadamard();
        n=4;
        hadamard();
        n=8;
        hadamard();
        n=16;
        hadamard();
        n=32;
        hadamard();
        n=64;
        hadamard();
        for (i = 0; i <=63;i++)
        {
            for (j = 0;j<=63;j++)
            {
                c[i][j]=c[i][j]/sqrt((float)expandsize);
            }
        }
    }
    result1 = strcmp(transform,"INVERS",6);
    if (result1 == 0)
    {
        zonal = expandsize;
        for (k = 0; k <= (expandsize-1); k++)
        {
            for (n = 0; n<=(expandsize-1);n++)
            {
                temp[n][k] = c[k][n];
            }
        }
        for (k = 0; k <= (expandsize-1); k++)
        {
            for (n = 0; n<=(expandsize-1);n++)
            {
                c[n][k] = temp[n][k];
            }
        }
        block = (float)expandsize;
    }
    else
    {
        block = 1.0/(float)expandsize;
    }
    /*Set video mode*/
    if (mode==1) _setvideomode(_MRES256COLOR);
    if (mode==2) _setvideomode(_VRES256COLOR);
    if (mode==3) _setvideomode(_SRES256COLOR);
    if (mode==4) _setvideomode(_XRES256COLOR);
    if (mode==5) _setvideomode(_ZRES256COLOR);
    _getvideoconfig(&vc);
    /*Enable palette*/
    for (i=0;i<=255;i++)
    {
        _remappalette(i,RGB(red[i]/4,green[i]/4,blue[i]/4));
    }

    /*Get dimensions*/
    ydim = (int)((float)ydim/(float)blocksize) * expandsize + ydim % blocksize;
    xdim = (int)((float)xdim/(float)blocksize) * expandsize + xdim % blocksize;

    /*Perform transform*/
    for (i=0;i<=(ydim-1);i++)
    {
        for (j=0;j<=(xdim-1);j++)
        {

```

```

        if (((i % expandsize) < blocksize) && ((j % expandsize) < blocksize))
        {
            fscanf(input_file, "%c", &cp);
            k = cp;
        }
        else
        {
            k = 128;
        }
        _setcolor(k);
        if (updown == 0) _setpixel(j,i);
        if (updown == 1) _setpixel(j,ydim-1-i);
    }
    for (jj = 0; jj <= ((ydim/expandsize)); jj++)
    {
        jjj = jj*expandsize;
        if ((ydim-jjj) != 0)
        {
            expandy = ydim-jjj;
            if (expandy > expandsize) expandy = expandsize;
            for (ii = 0; ii <= ((xdim/expandsize)); ii++)
            {
                iii = ii*expandsize;
                if ((xdim-iii) != 0)
                {
                    expandx = xdim-iii;
                    if (expandx > expandsize) expandx = expandsize;
                    for (i = 0; i <= (expandsize-1); i++)
                    {
                        for (j = 0; j <= (expandsize-1); j++)
                        {
                            data[j][i] = _getpixel(j+iii,i+jjj)-128;
                            if (j > (expandx-1)) data[j][i] = 0;
                            if (i > (expandy - 1)) data[j][i] = 0;
                        }
                    }
                    for (L = 0; L < expandsize; L++)
                    {
                        templ = 0;
                        for (m = 0; m < expandsize; m++)
                        {
                            for (n = 0; n < expandsize; n++)
                            {
                                templ += (float) (data[m][n]*c[L][n]);
                            }
                            temp[m][L]=templ;
                            templ = 0;
                        }
                    }
                    for (L = 0; L < expandsize; L++)
                    {
                        templ = 0;
                        for (m = 0; m < expandsize; m++)
                        {
                            for (n = 0; n < expandsize; n++)
                            {
                                templ += c[m][n]*temp[n][L];
                            }
                            if (templ >= 0)
                            {
                                data[m][L] = (int) (templ*block+0.5);
                            }
                            else
                            {

```

```

        data[m][L] = (int) (templ*block-0.5);
    }
    templ = 0;
}
for (i = 0; i<=(expandy-1);i++)
{
    for (j = 0;j<=(expandx-1);j++)
    {
        if (data[j][i] > 127) data[j][i] = 127;
        if (data[j][i] < -128) data[j][i] = -128;
        if (result1 != 0)
        {
            if (abs(data[j][i]) <= thresh) data[j][i] = 0;
            if ((j>=zonal) || (i>=zonal))
            {
                data[j][i] = 0;
            }
        }
        _setcolor((char) (data[j][i]+128));
        _setpixel(j+111,i+jjj);
    }
}
}
}
for (i = 0 ; i <=(ydim-1);i++)
{
    for (j = 0; j <=(xdim-1);j++)
    {
        k = _getpixel(j,i);
        if (strcmp(strip,"DON'T",5)==0) fprintf(out,"%c",k);
        else
        {
            if (((j % blocksize)< zonal) && ((i % blocksize) < zonal))
            {
                fprintf(out,"%c",k);
            }
        }
    }
}

/*Close files*/
fcloseall();

/*Pause*/
getch();

/*Return to text mode*/
_setvideomode(_DEFAULTMODE);
}
/*Main program ends*/

```

Appendix YY

The Entropy Calculator

The Entropy Calculator

```
#include <math.h>
#include <stdio.h>

/*Main program begins*/
main()
{
    char string[80];
    unsigned char p;
    int i,k;
    float j[256],pixel,entropy;

    FILE *input_file,*infile;

    /*Initialization*/
    entropy = 0;
    for (i=0;i<=255;i++) j[i]=0;
    pixel=0;

    /*Read in information*/
    input_file = fopen("tfile","r");
    fscanf(input_file,"%s",string);

    /*Open file*/
    infile = fopen(string,"r+b");

    /*Calculate and display entropy*/
    while (!feof(infile))
    {
        fscanf(infile,"%c",&p);
        pixel = pixel + 1;
        k = p;
        j[k]=j[k]+1;
    }
    for (i=0;i<=255;i++)
    {
        if (j[i] !=0)
        {
            entropy = entropy+j[i]*log(j[i]/pixel);
        }
    }
    entropy = -entropy/(pixel*log(2));
    printf("\nEntropy = %f",entropy);
    printf("\nEntropy encoding can achieve");
    printf(" a %f to 1 compression.\n",8/entropy);

    /*Close files*/
    _fcloseall();
}
/*Main program ends*/
```

Appendix ZZ

The Mean Squared Error Calculator

The Mean Squared Error Calculator

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*Globals begin*/
char string[80], string1[80], string2[80];
unsigned char c, c1, c2;
int k, k2, k1;
float sum, n;

FILE *input_file1, *input_file2, *infile;
/*Globals end*/

/*Main program begins*/
main()
{
    /*Read in information*/
    infile=fopen("tfile", "r");
    fscanf(infile, "%s", string);
    fscanf(infile, "%s", string1);
    fscanf(infile, "%s", string2);
    input_file1=fopen(string, "r+b");
    input_file2=fopen(string1, "r+b");

    /*Initialization*/
    n = 0;
    sum = 0;

    /*Calculate MSE*/
label:
    fscanf(input_file1, "%c", &c1);
    fscanf(input_file2, "%c", &c2);
    k1 = c1 - c2;
    k1 = k1 * k1;
    sum = sum + k1;
    n = n + 1;
    if (feof(input_file1) != 0)
    {
        goto label1;
    }
    if (feof(input_file2) != 0)
    {
        goto label1;
    }
    goto label;

    /*Display MSE*/
label1:
    printf("The Mean Square Error is %f ", sum/n);

    /*Close files*/
    _fcloseall();
}
/*Main program ends*/
```

Appendix AAA

The Histogram Processor

The Histogram Processor

```
/*Must be compiled with the small memory model*/
#include <math.h>
#include <stdio.h>
#include <graph.h>
#include <conio.h>
#include <bios.h>
#define NFonts 6

/*Main program begins*/
main()
{
    char string[80], string1[80];
    unsigned char p, list[20];
    int i, j, k, vc1, vc2, pix_val, ch;
    float sum, bigj, kl, jj[256], pixel, entropy, factor;

    struct _videoconfig vc;

    FILE *input_file, *infile, *outfile;

    /*Initialization*/
    _registerfonts("*.FON");
    _setfont(list);
    entropy=0;
    for (i=0; i<=255; i++) jj[i]=0;
    pixel=0.0;

    /*Read in information*/
    input_file = fopen("tfile", "r");
    fscanf(input_file, "%s", string);
    fscanf(input_file, "%i", &ch);

    /*Build histogram*/
    infile = fopen(string, "r+b");
    while (!feof(infile))
    {
        fscanf(infile, "%c", &p);
        pixel = pixel + 1.0;
        k = p;
        jj[k]=jj[k]+1;
    }

    /*Normalize matrix jj[k] to percentages of total*/
    bigj=0.0;
    for (k=0; k<=255; k++)
    {
        jj[k]=jj[k]/pixel;
        if (jj[k]>bigj) bigj=jj[k];
    }
    factor = 1.0/bigj;

    /*Display histogram*/
    _setvideomode(_VRES16COLOR);
    _getvideoconfig(&vc);
    vc1 = vc.numxpixels;
    vc2 = vc.numypixels;
    _setcolor(4);
```

```

    _settextposition(0,0);
    _moveto(0,0);
    sprintf(string,"%1.2f %1",100.0/factor);
    _outtext(string);
    for (k = 0;k<=255;k++)
    {
        k1 = ((float)k/255.0)*((float)vc.numxpixels-20.0);
        _moveto((int)(10.0+k1),vc2-10);
        _lineto((int)(10.0+k1), (int)((vc2-10)*(1.0-(float)factor*jj[k])) );
    }
    if (!ch) goto label;
    fscanf(input_file,"%s",string1);
    outfile=fopen(string1,"w+b");
    for (j=0;j<480;j++)
    {
        for (i=0;i<640;i++)
        {
            pix_val=_getpixel(i,j);
            if (pix_val==0) ch=255;
            else ch=0,
            fprintf(outfile,"%c",ch);
        }
    }

    label:

    /*Pause*/
    getch();

    /*Return to text mode*/
    _setvideomode(_DEFAULTMODE);

    /*Close files*/
    _fcloseall();
}
/*Main program ends*/

```

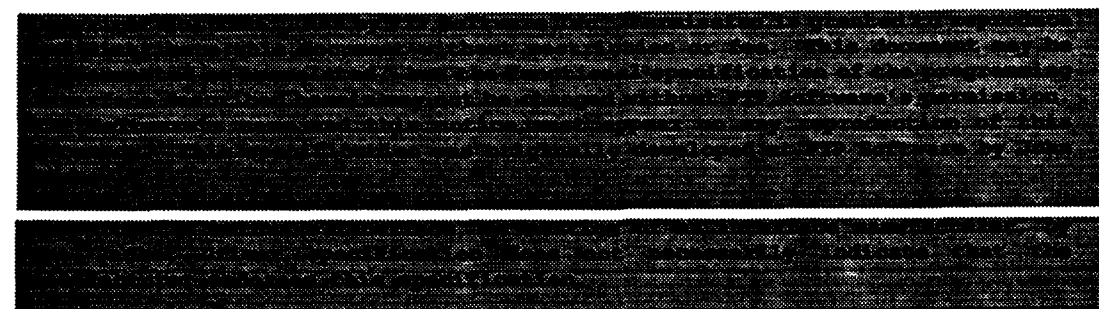
Appendix BBB

The Packet Driver Specifications

PC/TCP Packet Driver Specification

Revision 1.09
September-14-1989
Developed by:

FTP Software, Inc.
26 Princess St.
Wakefield, MA 01880-3004
(617) 246-0900



1 Document Conventions

All numbers in this document are given in C-style representation. Decimal is expressed as 11, hexadecimal is expressed as 0x0B, octal is expressed as 013. All reference to network hardware addresses (source, destination and multicast) and demultiplexing information for the packet headers assumes they are represented as they would be in a MAC-level packet header being passed to the `send_pkt()` function.

2 Introduction and Motivation

This document describes the programming interface to FTP Software Packet Drivers. Packet drivers provide a simple, common programming interface that allows multiple applications to share a network interface at the data link level. The packet drivers demultiplex incoming packets among the applications by using the network media's standard packet type or service access point field(s).

The intent of this specification is to allow protocol stack implementations to be independent of the actual brand or model of the network interface in use on a particular machine. Different versions of various protocol stacks still must exist for different network media (Ethernet, 802.5 ring, serial lines), because of differences in protocol-to-physical address mapping, header formats, maximum transmission units (MTUs) and so forth.

The packet driver provides calls to initiate access to a specific packet type, to end access to it, to send a packet, to get statistics on the network interface and to get information about the interface.

Protocol implementations that use the packet driver can completely coexist on a PC and can make use of one another's services, whereas multiple applications which do not use the driver do not coexist on one machine properly. Through use of the packet driver, a user could run TCP/IP, XNS, and a proprietary protocol

implementation such as DECnet, Banyan's, LifeNet's, Novell's or 3Com's without the difficulties associated with preempting the network interface.

Applications which use the packet driver can also run on new network hardware of the same class without being modified; only a new packet driver need be supplied.

Several levels of packet drivers are described in this specification. The first is the basic packet driver, which provides minimal functionality but should be simple to implement and which uses very few host resources. The basic driver provides operations to broadcast and receive packets. The second driver is the extended packet driver, which is a superset of the basic driver. The extended driver supports less commonly used functions of the network interface such as multicast, and also gathers statistics on use of the interface and makes these available to the application. The third level, the high-performance functions, supports performance improvements and tuning.

Functions which are available in only the extended packet driver are noted as such in their descriptions. All basic packet driver functions are available in the extended driver. The high-performance functions may be available with either basic or extended drivers.

3 Identifying Network Interfaces

Network interfaces are named by a triplet of integers, <class, type, number>. The first number is the class of interface. The class tells what kind of media the interface supports: DEC/Intel/Xerox (DIX or Bluebook) Ethernet, IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, ProNET-10, Appletalk, serial line, etc.

The second number is the type of interface: this specifies a particular instance of an interface supporting a class of network medium. Interface types for Ethernet might name these interfaces: 3Com 3C503 or 3C505, Interlan WIS210, Univation, BICC Data Networks ISOLAN, Ungermann-Bass NIC, etc. Interface types for IEEE 802.5 might name these interfaces: IBM Token Ring adapter, Proteon pl340.

The last number is the interface number. If a machine is equipped with more than one interface of a class and type, the interfaces must be numbered to distinguish between them.

An appendix details constants for classes and types. The class of an interface is an 8-bit integer, and its type is a 16 bit integer.

Class and type constants are managed by FTP Software. Contact FTP to register a new class or type number.

The type 0xFFFF is a wildcard type which matches any interface in the specified class. It is unnecessary to wildcard interface numbers, as 0 will always correspond to the first interface of the specified class and type.

This specification has no provision for the support of multiple network interfaces (with similar or different characteristics) via a single Packet Driver and associated interrupt. We feel that this issue is best addressed by loading several Packet Drivers, one per interface, with different interrupts (although all may be included in a single TSR software module). Applications software must check the class and type returned from a driver_info() call in any case, to make sure that the Packet Driver is for the correct media and packet format. This can easily be generalized by searching for another Packet Driver if the first is not of the right kind.

4 Initiating Driver Operations

The packet driver is invoked via a software interrupt in the range 0x60 through 0x80. This document does not specify a particular interrupt, but describes a mechanism for locating which interrupt the driver uses. The interrupt must be configurable to avoid conflicts with other pieces of software that also use software interrupts. The program which installs the packet driver should provide some mechanism for the user to specify the interrupt.

The handler for the interrupt is assumed to start with 3 bytes of executable code; this can either be a 3-byte jump instruction, or a 2-byte jump followed by a NOP (do not specify "jmp short" unless you also specify an explicit NOP). This must be followed by the null-terminated ASCII text string "PKT DRVR". To find the interrupt being used by the driver, an application should scan through the handlers for vectors 0x60 through 0x80 until it finds one with the text string "PKT DRVR" in the 12 bytes immediately following the entry point.

5 Link-layer Demultiplexing

If a network media standard is to support simultaneous use by different transport protocols (e.g. TCP/IP, XMS, OSI), it must define some link-level mechanism which allows a host to decide which protocol a packet is intended for. In DIX Ethernet, this is the 16-bit "ethertype" field immediately following the 6-byte destination and source addresses. In IEEE 802.3 where 802.2 headers are used, this information is in the variable-length 802.2

header. In Proteon's ProNET-10, this is done via the 32-bit "type" field. Other media standards may demultiplex via a method of their own, or 802.2 headers as in 802.3.

Our `access_type()` function provides access to this link-layer demultiplexing. Each call establishes a destination for a particular type of link-layer packet, which remains in effect until `release_type()` is called with the handle returned by that particular `access_type()`. The link-layer demultiplexing information is passed via the type and typelen fields, but values and interpretation depend on the class of packet driver (and thus on the media in use).

A class 1 driver (DIX Ethernet) should expect type to point at an "ethertype" value (in network byte order, and greater than 0x05EE), and might reasonably require typelen to equal 2. A class 2 driver could require 4 bytes. However, a class 3 (802.5) or 11 (Ethernet with 802.2 headers) driver should be prepared for typelen values between 2 (for the DSAP/SSAP fields only) and 8 (3 byte 802.2 header plus 3-byte Sub-Network Access Protocol extension header plus 2-byte "ethertype" as defined in RFC 1042).

6 Programming Interface

All functions are accessed via the software interrupt determined to be the driver's via the mechanism described earlier. On entry, register AH contains the code of the function desired.

The handle is an arbitrary integer value associated with each MAC-level demultiplexing type that has been established via the `access_type` call. Internally to the packet driver, it will probably be a pointer, or a table offset. The application calling the packet driver cannot depend on it assuming any particular range, or any other characteristics. In particular, if an application uses two or more packet drivers, handles returned by different drivers for the same or different types may have the same value.

Note that some of the functions defined below are labelled as extended driver functions and high-performance functions. Because these are not required for basic network operations, their implementation may be considered optional. Programs wishing to use these functions should use the `driver_info()` function to determine if they are available in a given packet driver.

6.1 Entry conditions

FTP Software applications which call the packet driver are coded in Microsoft C and assembly language. All necessary registers are saved by FTP's routines before the INT instruction to call the packet driver is executed. Our current `receiver()` functions behave as follows: DS, SS, SP and the flags are saved and restored. All other registers may be modified, and should be saved by the packet driver, if necessary. Processor interrupts may be enabled while in the upcall, but the upcall doesn't assume interrupts are disabled on entry. On entry, `receiver()` switches to a local stack, and switches back before returning.

Note that some older versions of PC/TCP may enable interrupts during the upcall, and leave them enabled on return to the Packet Driver.

When using a class 1 driver, PC/TCP will normally make 5 `access_type()` calls for IP, ARP and 3 kinds of Berkeley Trailer encapsulation packets. On other media, the number of handles we open will vary, but it is usually at least two (IP and ARP). Implementors should make their tables large enough to allow two protocol stacks to co-exist. We recommend support for at least 10 open handles simultaneously.

6.2 Byte and bit ordering

Developers should note that, on many networks and protocol families, the byte-ordering of 16-bit quantities on the network is opposite to the native byte-order of the PC. (802.5 Token Ring is an exception). This means that DEC-Intel-Xerox ethertype values passed to `access_type()` must be swapped (passed in network order). The IEEE 802.3 length field needs similar handling, and care should be taken with packets passed to `send_pkt()`, so all fields are in the proper order. Developers working with MSB LANs (802.5 and FDDI) should be aware that a MAC address changes bit order depending on whether it appears in the header or as data.

6.3 `driver_info()`

```

driver_info(handle) AH == 1, AL == 255
int handle;        BX /* Optional */

error return:
  carry flag set
  error code        DH
  possible errors:
  BAD_HANDLE        /* older drivers only */

non-error return:
  carry flag clear
  version            BX
  class              CH
  type               DX
  number             CL
  name               DS:SI
  functionality      AL

                        1 == basic functions present.
                        2 == basic and extended present.
                        5 == basic and high-performance.
                        6 == basic, high-performance, extended.
                        255 == not installed.
```

This function returns information about the interface. The version is assumed to be an internal hardware driver identifier. In earlier versions of this spec, the handle argument (which must have been obtained via access_type()) was required. It is now optional, but drivers developed according to versions of this spec previous to 1.07 may require it, so implementers should take care.

6.4 access_type()

```
int access_type(if_class, if_type, if_number, type, typelen, receiver) AH == 2
    int if_class;          AL
    int if_type;           BX
    int if_number;         DL
    char far *type;        DS:SI
    unsigned typelen;      CX
    int (far *receiver)(); ES:DI

error return:
    carry flag set
    error code            DH
    possible errors:
        NO_CLASS
        NO_TYPE
        NO_NUMBER
        BAD_TYPE
        NO_SPACE
        TYPE_INUSE

non-error return:
    carry flag clear
    handle                AX

receiver call:
    (*receiver)(handle, flag, len [, buffer])
        int handle;       BX
        int flag;          AX
        unsigned len;      CX
    if AX == 1,
        char far *buffer; DS:SI
```

Initiates access to packets of the specified type. The argument type is a pointer to a packet type specification. The argument typelen is the length in bytes of the type field. The argument receiver is a pointer to a subroutine which is called when a packet is received. If the typelen argument is 0, this indicates that the caller wants to match all packets (match all requests may be refused by packet drivers developed to conform to versions of this spec previous to 1.07).

When a packet is received, receiver is called twice by the packet driver. The first time it is called to request a buffer from the application to copy the packet into. AX == 0 on this call. The application should return a pointer to the buffer in ES:DI. If the application has no buffers, it may return 0:0 in ES:DI, and the driver should throw away the packet and not perform the second call.

It is important that the packet length (CX) be valid on the AX == 0 call, so that the receiver can allocate a buffer of the proper size. This length (as well as the copy performed prior to the AX == 1 call) must include the MAC header and all received data, but not the trailing Frame Check Sequence (if any).

On the second call, AX == 1. This call indicates that the copy has been completed, and the application may do as it wishes with the buffer. The buffer that the packet was copied into is pointed to by DS:SI.

6.5 release_type()

```
int release_type(handle)  AH == 3
    int handle;           BX
```

error return:

```
    carry flag set
    error code           DX
possible errors:
    BAD_HANDLE
```

non-error return:

```
    carry flag clear
```

This function ends access to packets associated with a handle returned by access_type(). The handle is no longer valid.

6.6 send_pkt()

```
int send_pkt(buffer, length)  AH == 4
    char far *buffer;         DS:SI
    unsigned length;          CX
```

error return:

```
    carry flag set
    error code           DX
possible errors:
    CANT_SEND
```

non-error return:

```
    carry flag clear
```

Transmits length bytes of data, starting at buffer. The application must supply the entire packet, including local network headers. Any MAC or LLC information in use for packet demulti-plexing (e.g. the DEC-Intel-Xerox Ethertype) must be filled in by the application as well. This cannot be performed by the driver, as no handle is specified in a call to the send_packet() function.

6.7 terminate()

```
terminate(handle)           AH == 5
    int handle;             BX
```

error return:

```
    carry flag set
    error code           DX
possible errors:
    BAD_HANDLE
    CANT_TERMINATE
```

non-error return:

```
    carry flag clear
```

Terminates the driver associated with handle. If possible, the driver will exit and allow MS-DOS to reclaim the memory it was using.

6.8 get_address()

```
get_address(handle, buf, len)  AH == 6
    int handle;               BX
    char far *buf;            ES:DI
    int len;                  CX
```

```

error return:
    carry flag set
    error code                DH
    possible errors:
        BAD_HANDLE
        NO_SPACE
non-error return:
    carry flag clear
    length                    CX

```

Copies the current local net address of the interface into buf. The buffer buf is len bytes long. The actual number of bytes copied is returned in CX. If the NO_SPACE error is returned, this indicates that len was insufficient to hold the local net address. If the address has been changed by set_address(), the new address should be returned.

6.9 reset_interface()

```

    reset_interface(handle)    AH == 7
    int handle;               BX

error return:
    carry flag set
    error code                DH
    possible errors:
        BAD_HANDLE
        CANT_RESET
non-error return:
    carry flag clear

```

Resets the interface associated with handle to a known state, aborting any transmits in process and reinitializing the receiver. The local net address is reset to the default (from ROM), the multicast list is cleared, and the receive mode is set to 3 (own address & broadcasts). If multiple handles are open, these actions might seriously disrupt other applications using the interface, so CANT_RESET should be returned.

6.10 get_parameters()

```

    get_parameters()          AH = 10

error return:
    carry flag set
    error code                DH
    possible errors:
        BAD_COMMAND
non error return:
    carry flag clear
    struct param far *;       ES:DI

struct param {
    unsigned char  major_rev;
    unsigned char  minor_rev;
    unsigned char  length;
    unsigned char  addr_len;
    unsigned short mtu;
    unsigned short multicast_aval;
    unsigned short rcv_bufs;
    unsigned short xmt_bufs;
    unsigned short int_num;

```

};

The performance of an application may benefit from using `get_parameters()` to obtain a number of driver parameters. This function was added to v1.09 of this specification, and may not be implemented by all drivers (see `driver_info()`).

The `major_rev` and `minor_rev` fields are the major and minor revision numbers of the version of this specification the driver conforms to. For this document, `major_rev` is 1 and `minor_rev` is 9. The `length` field may be used to determine which values are valid, should a later revision of this specification add more values at the end of this structure. For this document, `length` is 14. The `addr_len` field is the length of a MAC address, in bytes. Note the `param` structure is assumed to be packed, such that these fields occupy four consecutive bytes of storage.

In the `param` structure, the `mtu` is the maximum MAC-level packet the driver can handle (on Ethernet this number is fixed, but it may vary on other media, e.g. 802.5 or FDDI). The `multicast_aval` field is the number of bytes required to store all the multicast addresses supported by any "perfect filter" mechanism in the hardware. Calling `set_multicast_list()` with its `len` argument equal to this value should not fail with a `NO_SPACE` error. A value of zero implies no multicast support.

The `rcv_bufs` and `xmt_bufs` indicate the number of back-to-back receives or transmits the card/driver combination can accommodate, minus 1. The application can use this information to adjust flow-control or transmit strategies. A single-buffered card (for example, an Interlan NI5010) would normally return 0 in both fields. A value of 0 in `rcv_bufs` could also be used by a driver author to indicate that the hardware has limitations which prevent it from receiving as fast as other systems can send, and to recommend that the upper-layer protocols invoke lock-step flow control to avoid packet loss.

The `int_num` field should be set to a hardware interrupt that the application can hook in order to perform interrupt-time protocol processing after the EOI has been sent to the 8259 interrupt controller and the card is ready for more interrupts. A value of zero indicates that there is no such interrupt. Any application hooking this interrupt and finding a non-zero value in the vector must pass the interrupt down the chain and wait for its predecessors to return before performing any processing or stack switches. Any driver which implements this function via a separate `INT` instruction and vector, instead of just using the hardware interrupt, must prevent any saved context from being overwritten by a later interrupt. In other words, if the driver switches to its own stack, it must allow reentrancy.

6.11 `as_send_pkt()`

```
int as_send_pkt(buffer, length, upcall) AH == 11
    char far *buffer;                DS:SI
    unsigned length;                  CX
    int (far *upcall)();              ES:DI

error return:
    carry flag set
    error code                        DH
    possible errors:
        CANT_SEND
        BAD_COMMAND

non-error return:
    carry flag clear

buffer available upcall:
    (*upcall)(buffer, result)
    int result;                       AX
    char far *buffer;                 ES:DI
```

`as_send_pkt()` differs from `send_pkt()` in that the `upcall()` routine is called when the application's data has been copied out of the buffer, and the application can safely modify or re-use the buffer. The driver may pass a non-zero error code to `upcall()` if the copy failed, or some other error was detected, otherwise it should indicate success, even if the packet hasn't actually been transmitted yet. Note that the buffer passed to `send_pkt()` is assumed to be modifiable when that call returns, whereas with `as_send_pkt()`, the buffer may be queued by the driver and dealt with later. If an error is returned on the initial call, the `upcall` will not be executed. This function was added in v1.09 of this specification, and may not be implemented by all drivers (see `driver_info()`).

6.12 `set_rcv_mode()`

```

set_rcv_mode(handle, mode)    AE == 20
    int handle;               BX
    int mode;                  CX

```

error return:

carry flag set
error code DE

possible errors:
 BAD_HANDLE
 BAD_MODE

non-error return:

carry flag clear

Sets the receive mode on the interface associated with `handle`. The following values are accepted for `mode`:

mode	meaning
1	turn off receiver
2	receive only packets sent to this interface
3	mode 2 plus broadcast packets
4	mode 3 plus limited multicast packets
5	mode 3 plus all multicast packets
6	all packets

Note that not all interfaces support all modes. The receive mode affects all packets received by this interface, not just packets associated with the `handle` argument. See the extended driver functions `get_multicast_list()` and `set_multicast_list()` for programming "perfect filters" to receive specific multicast addresses.

Note that mode 3 is the default, and if the `set_rcv_mode()` function is not implemented, mode 3 is assumed to be in force as long as any handles are open (from the first `access_type()` to the last `release_type()`).

6.13 `get_rcv_mode()`

```

get_rcv_mode(handle, mode)    AE == 21
    int handle;               BX

```

error return:

carry flag set
error code DE

possible errors:
 BAD_HANDLE

non-error return:

```

    carry flag clear
mode                                     AX

```

Returns the current receive mode of the interface associated with handle.

6.14 set_multicast_list()

```

set_multicast_list(addrlist, len)  AH == 22
    char far *addrlist;  ES:DI
    int len;             CX

```

```

error return:
    carry flag set
    error code      DH
    possible errors:
        NO_MULTICAST
        NO_SPACE
        BAD_ADDRESS

```

```

non-error return:
    carry flag clear

```

The addrlist argument is assumed to point to an len-byte buffer containing a number of multicast addresses. BAD_ADDRESS is returned if len modulo the size of an address is not equal to 0, or the data is unacceptable for some reason. NO_SPACE is returned (and no addresses are set) if there are more addresses than the hardware supports directly.

The recommended procedure for setting multicast addresses is to issue a get_multicast_list(), copy the information to a local buffer, add any addresses desired, and issue a set_multicast_list(). This should be reversed when the application exits. If the set_multicast_list() fails due to NO_SPACE, use set_rcv_mode() to set mode 5 instead.

6.15 get_multicast_list()

```

get_multicast_list()              AH == 23

```

```

error return:
    carry flag set
    error code      DH
    possible errors:
        NO_MULTICAST
        NO_SPACE

```

```

non-error return:
    carry flag clear
    char far *addrlist;  ES:DI
    int len;             CX

```

On a successful return, addrlist points to len bytes of multicast addresses currently in use. The application program must not modify this information in-place. A NO_SPACE error indicates that there wasn't enough room for all active multicast addresses.

6.16 get_statistics()

```

[redacted]
get_statistics(handle)      AH == 24
    int handle;             BX

error return:
    carry flag set
    error code              DH
    possible errors:
        BAD_HANDLE

non-error return:
    carry flag clear
    char far *stats;        DS:SI

struct statistics {
    unsigned long packets_in;
    unsigned long packets_out;
    unsigned long bytes_in;
    unsigned long bytes_out;
    unsigned long errors_in;
    unsigned long errors_out;
    unsigned long packets_lost;
};
```

Returns a pointer to a statistics structure for the interface. The values are stored as to be normal 80xx 32-bit integers.

6.17 set_address()

```

[redacted]
set_address(addr, len)      AH == 25
    char far *addr;         ES:DI
    int len;                CX

error return:
    carry flag set
    error code              DH
    possible errors:
        CANT_SET
        BAD_ADDRESS

non-error return:
    carry flag clear
    length                  CX
```

This call is used when the application or protocol stack needs to use a specific LAN address. For instance, DECnet protocols on Ethernet encode the protocol address in the Ethernet address, requiring that it be set when the protocol stack is loaded. A BAD_ADDRESS error indicates that the Packet Driver doesn't like the len (too short or too long), or the data itself. Note that packet drivers should refuse to change the address (with a CANT_SET error) if more than one handle is open (lest it be changed out from under another protocol stack).

Appendix A

Interface classes and types

The following are defined as network interface classes, with their individual types listed immediately following the class.

DEC/Intel/Xerox "Bluebook" Ethernet

Class	1
3COM 3C500/3C501	1
3COM 3C505	2
Interlan Ni5010	3
BICC Data Networks 4110	4
BICC Data Networks 4117	5
MICOM-Interlan WF600	6
Ungermann-Bass PC-NIC	8
Univation NC-516	9
TRW PC-2000	10
Interlan Ni5210	11
3COM 3C503	12
3COM 3C523	13
Western Digital WD8003	14
Spider Systems S4	15
Torus Frame Level	16
10NET Communications	17
Gateway PC-bus	18
Gateway AT-bus	19
Gateway MCA-bus	20
IMC PCnic	21
IMC PCnic II	22
IMC PCnic 8bit	23
Tigan Communications	24
Micromatic Research	25
Clarkson "Multiplexor"	26
D-Link 8-bit	27
D-Link 16-bit	28
D-Link PS/2	29
Research Machines 8	30
Research Machines 16	31
Research Machines MCA	32
Radix Microsys. EXM1 16-bit	33
Interlan Ni9210	34
Interlan Ni6510	35
Vestra LANMASTER 16-bit	36
Vestra LANMASTER 8-bit	37
Allied Telesis PC/XT/AT	38
Allied Telesis NEC PC-98	39
Allied Telesis Fujitsu FMR	40
Ungermann-Bass NIC/PS2	41
Tiara LANCard/E AT	42
Tiara LANCard/E MC	43
Tiara LANCard/E TP	44
Spider Comm. SpiderComm 8	45
Spider Comm. SpiderComm 16	46
AT&T Starlan MAU	47
AT&T Starlan-10 MAU	48
AT&T Ethernet MAU	49
Intel smart card	50

ProNET-10

Class	2
Proteon p1300	1
Proteon p1800	2

IEEE 802.5/ProNET-4		
Class	3	
IBM Token ring adapter	1	
Proteon p1340	2	
Proteon p1344	3	
Gateway PC-bus	4	
Gateway AT-bus	5	
Gateway MCA-bus	6	
Omninet		
Class	4	
Appletalk		
Class	5	
Serial line		
Class	6	
Clarkson 8250-SLIP	1	
Clarkson "Multiplexor"	2	
Starlan		
Class	7	(NOTE: Class has been subsumed by Ethernet)
ArcNet		
Class	8	
Datapoint RIM	1	
AX.25Class	9	
KISS Class	10	
IEEE 802.3 w/802.2 hdrs		
Class	11	
Types as given under DIX Ethernet		
See Appendix D.		
FDDI w/802.2 hdrs		
Class	12	
Internet X.25		
Class	13	
Western Digital	1	
Frontier Technology	2	
N.T. LANSTAR (encapsulating DIX)		
Class	14	
NT LANSTAR/S	1	
NT LANSTAR/MC	2	

Appendix B

Function call numbers

The following decimal numbers are used to specify which operation the packet driver should perform. The number is stored in register AH on call to the packet driver.

driver_info	1
access_type	2
release_type	3
send_pkt	4
terminate	5
get_address	6
reset_interface	7
+get_parameters	10
+as_send_pkt	11
*set_rcv_mode	20
*get_rcv_mode	21
*set_multicast_list	22
*get_multicast_list	23
*get_statistics	24
*set_address	25

- + indicates a high-performance packet driver function
- * indicates an extended packet driver function

AH values from 128 through 255 (0x80 through 0xFF) are reserved for user-developed extensions to this specification. While FTP Software cannot support user extensions, we are willing to act as a clearing house for information about them. For more information, contact us.

Appendix C

Error codes

Packet driver calls indicate error by setting the carry flag on return. The error code is returned in register DH (a register not used to pass values to functions must be used to return the error code). The following error codes are defined:

1	BAD_HANDLE	Invalid handle number,
2	NO_CLASS	No interfaces of specified class found,
3	NO_TYPE	No interfaces of specified type found,
4	NO_NUMBER	No interfaces of specified number found,
5	BAD_TYPE	Bad packet type specified,
6	NO_MULTICAST	This interface does not support multicast,
7	CANT_TERMINATE	This packet driver cannot terminate,
8	BAD_MODE	An invalid receiver mode was specified,
9	NO_SPACE	Operation failed because of insufficient space,
10	TYPE_INUSE	The type had previously been accessed, and not released,
11	BAD_COMMAND	The command was out of range, or not implemented,
12	CANT_SEND	The packet couldn't be sent (usually hardware error),
13	CANT_SET	Hardware address couldn't be changed (more than 1 handle open),
14	BAD_ADDRESS	Hardware address has bad length or format,
15	CANT_RESET	Couldn't reset interface (more than 1 handle open).

Appendix D

802.3 vs. Blue Book Ethernet

One weakness of the present specification is that there is no provision for simultaneous support of 802.3 and Blue Book (the old DEC-Intel-Xerox standard) Ethernet headers via a single Packet Driver (as defined by its interrupt). The problem is that the "ethertype" of Blue Book packets is in bytes 12 and 13 of the header, and in 802.3 the corresponding bytes are interpreted as a length. In 802.3, the field which would appear to be most useful to begin the type check in is the 802.2 header, starting at byte 14. This is only a problem on Ethernet and variants (e.g. Starlan), where 802.3 headers and Blue Book headers are likely to need co-exist for many years to come.

One solution is to redefine class 1 as Blue Book Ethernet, and define a parallel class for 802.3 with 802.2 packet headers. This requires that a 2nd Packet Driver (as defined by its interrupt) be implemented where it is necessary to handle both kinds of packets, although they could both be part of the same TSR module.

As of v1.07 of this specification, class 11 was assigned to 802.3 using 802.2 headers, to implement the above.

Note: According to this scheme, an application wishing to receive IP encapsulated with an 802.2 SNAP header and "ethertype" of 0x800, per RFC 1042, would specify a typelen argument of 8, and type would point to:

```
char ieee_ip[] = {0xAA, 0xAA, 3, 0, 0, 0, 0x00, 0x08};
```

James B. VanBokkelen
jbvb@ftp.com

Table of Contents

1	Document Conventions	BBB2
2	Introduction and Motivation	BBB2
3	Identifying Network Interfaces	BBB3
4	Initiating Driver Operations	BBB4
5	Link-layer Demultiplexing	BBB4
6	Programming Interface	BBB4
6.1	Entry conditions	BBB5
6.2	Byte and bit ordering	BBB5
6.3	driver_info()	BBB5
6.4	access_type()	BBB6
6.5	release_type()	BBB7
6.6	send_pkt()	BBB7
6.7	terminate()	BBB7
6.8	get_address()	BBB7
6.9	reset_interface()	BBB8
6.10	get_parameters()	BBB8
6.11	as_send_pkt()	BBB9
6.12	set_rcv_mode()	BBB10
6.13	get_rcv_mode()	BBB10
6.14	set_multicast_list()	BBB11
6.15	get_multicast_list()	BBB11
6.16	get_statistics()	BBB12
6.17	set_address()	BBB12
Appendix A	Interface classes and types	BBB13
Appendix B	Function call numbers	BBB15
Appendix C	Error codes	BBB16
Appendix D	802.3 vs. Blue Book Ethernet	BBB17

Appendix CCC

VESA Graphics Interface

VESA Graphics Interface

Introduction

This appendix contains the specification for the Video Electronics Standards Association (VESA) graphics interface. Readers of this appendix should already be familiar with programming VGAs at the hardware level and real mode assembly language. Readers who are unfamiliar with programming the VGA should first read one of the many VGA programming tutorials before attempting to understand these extensions to the standard VGA.

The IBM VGA has become a *de facto* standard in the PC graphics world. A multitude of different VGA offerings exist in the marketplace, each one providing BIOS or register compatibility with the IBM VGA. More and more of these VGA compatible products implement various supersets of the VGA standard. These extensions range from higher resolutions and more colors to improved performance and even some graphics processing capabilities. Intense competition has dramatically improved the price/performance ratio, to the benefit of the end user.

However, several serious problems face a software developer who intends to take advantage of these Super VGA environments. Because there is no standard hardware implementation, the developer is faced with widely disparate Super VGA hardware architectures. Lacking a common software interface, designing applications for these environments is costly and technically difficult. Except for applications supported by OEM-specific display drivers, very few software packages can take advantage of the power and capabilities of Super VGA products.

The purpose of the VESA VGA BIOS extension is to remedy this situation. Being a common software interface to Super VGA graphics products, the primary objective is to enable application and system software to adapt to and exploit the wide range of features available in these VGA extensions.

Today, an application has no standard mechanism to determine what Super VGA hardware it is running on. Only by knowing OEM-specific features can an application determine the presence of a particular video board. This often involves reading and testing registers located at I/O addresses unique to each OEM. By not knowing what hardware an application is running on, few, if any, of the extended features of the underlying hardware can be used.

The VESA BIOS extension provides several functions to return information about the video environment. These functions return system level information as well as video mode specific details. Function 00h returns general system level information, including an OEM identification string. The function also returns a pointer to the supported video modes. Function 01h may be used by the application to obtain information about each supported video mode. Function 03h returns the current video mode.

Due to the fact that different Super VGA products have different hardware implementations, application software has great difficulty in adapting to each environment. However, since each is based on the VGA hardware architecture, differences are most common in video mode initialization and memory mapping. The rest of the architecture is usually kept intact, including I/O mapped registers, video buffer location in the CPU address space, DAC location and function, etc.

The VESA BIOS extension provides several functions to interface to the different Super VGA hardware implementations. The most important of these is Function 02h, set Super VGA video mode. This function isolates the application from the tedious and complicated task of setting up a video mode. Function 05h provides an interface to the underlying memory mapping hardware. Function 04h enables an application to save and restore a Super VGA state without knowing anything of the specific implementation.

A primary design objective of the VESA BIOS extension is to preserve maximum compatibility to the standard VGA environment. In no way should the BIOS extensions compromise compatibility or performance. Another but related concern is to minimize the changes necessary to an existing VGA BIOS. RAM as well as ROM-based implementations of the BIOS extension should be possible.

The purpose of the VESA BIOS extension is to provide support for extended VGA environments. Thus, the underlying hardware architecture is assumed to be a VGA. Graphics software that drives a Super VGA board will perform its graphics output in generally the same way it drives a standard VGA, i.e. writing directly to a VGA style frame buffer, manipulating graphics controller registers, directly programming the palette, etc. No significant graphics processing will be done in hardware. For this reason, the VESA BIOS extension does not provide any graphics output function, such as BitBit, line, or circle drawings, etc.

Outside the scope of this VESA BIOS extension is handling of different monitors and monitor timings. Such items are dealt with in other VESA fora. The purpose of the VESA BIOS extension is to provide a standardized software interface to Super VGA graphics modes, independent of monitor and monitor timing issues.

The following VESA mode numbers have been defined

<u>15-bit</u> <u>Mode Number</u>	<u>7-bit</u> <u>Mode Number</u>	<u>Resolution</u>	<u>Colors</u>
100h		640x400	256
101h		640x480	256
102h	6Ah	800x600	16
103h		800x600	256
104h		1024x768	16
105h		1024x768	256
106h		1280x1024	16
107h		1280x1024	256

The first consideration in implementing extended video memory is to give access to the memory to application software. The standard VGA CPU address space for 16 color graphics modes is typically at segment A000h for 64K. This gives access to the 256K bytes of a standard VGA, i.e. 64K per plane. Access to the extended video memory is accomplished by mapping portions of the video memory into the standard VGA CPU address space. Every super VGA hardware implementation provides a mechanism for software to specify the offset from the start of video memory which is to be mapped to the start of the CPU address space. Providing both read and write access to the mapped memory provides a necessary level of hardware support for an application to manipulate the extended video memory.

Several new BIOS calls have been defined to support Super VGA modes. For maximum compatibility with the standard VGA BIOS, these calls are grouped under one function number. This number is passed in the AH register to the int 10h handler.

The designated Super VGA extended function number is 4Fh. This function number is presently unused in most, if not all, VGA BIOS implementation. A standard VGA BIOS performs no action when function call 4F is made. Super VGA standard VS900602 defines subfunctions 00h through 07h. Subfunction numbers 08h through 0FFh are reserved for future use.

Every function returns status information in the AX register. The format of the status word is as follows:

```

AL == 4Fh:  Function is supported
AL != 4Fh:  Function is not supported
AH == 00h:  Function call successful
AH == 01h:  Function call failed

```

Software should treat a nonzero value in the AH register as a general failure condition. In later versions of the VESA BIOS extension new error codes might be defined.

Function 00h - Return Super VGA Information

The purpose of this function is to provide information to the calling program about the general capabilities of the Super VGA environment. The function fills an information block structure at the address specified by the caller. The information block size is 256 bytes.

Input: AH = 4Fh Super VGA support
AL = 00h Return Super VGA information
ES:DI = Pointer to Buffer

Output: AX = status
(All other registers preserved)

The information block has the following structure:

```
VgaInfoBlock struc
    VESASignature    db 'VESA'          ;4 signature bytes
    VESAVersion      dw ?               ;VESA version number
    OEMStringPtr     dd ?               ;Pointer to OEM string
    Capabilities     dd 4 dup (?)       ;capabilities of video
    VideoModePtr     dd ?               ;pointer to SVGA modes
    TotalMemory      dw ?               ;number of 64kb blocks
    Reserved         db 242 dup (?)     ;remainder of VgaInfoBlock
VgaInfoBlock ends
```

The VESASignature field contains the characters 'VESA' if this is a valid block.

The VESAVersion is a binary field which specifies what level of the VESA standard the Super VGA BIOS conforms to. The higher byte specifies the major version number. The lower byte specifies the minor version number. The current VESA version number is 1.1. Applications written to use the features of a specific version of the VESA BIOS extension are guaranteed to work in later versions. The VESA BIOS extension will be fully upwards compatible.

The OEMStringPtr is for a pointer to a null terminated OEM-defined string. The string may be used to identify the video chip, video board, memory configuration, etc., to hardware specific display drivers. There are no restrictions on the format of the string.

The Capabilities field describes what general features are supported in the video environment. The bits are defined as follows:

D0-31 = reserved.

The VideoModePtr points to a list of supported Super VGA (VESA-defined as well as OEM specific) mode numbers. Each mode number occupies one word (16 bits). The list of mode numbers is terminated by a -1 (0FFFFh). The pointer could point into either ROM or RAM, depending on the specific implementation. Either the list would be a static string stored in ROM, or the list would be generated at run-time in the information block in RAM. It is the applications responsibility to verify the current availability of any mode returned by this function through the Return Super VGA mode information (function 1) call. Some of the returned modes may not be available due to the video boards' current memory and monitor configuration.

The TotalMemory field indicates the amount of memory installed on the VGA board. Its value represents the number of 64kb blocks of memory currently installed.

Function 01h - Return Super VGA Mode Information

This function returns information about a specific Super VGA video mode that was returned by Function 0. The function fills a mode information block structure at the address specified by the caller. The mode information block size is maximum 256 bytes.

Some information provided by this function is implicitly defined by the VESA mode number. However, some Super VGA implementations might support video modes other than those defined by VESA. To provide access to these modes, this function also returns various other information about the mode.

Input: AL = 4Fh Super VGA support
AL = 01h Return Super VGA mode information
CX = Super VGA video mode number
ES:DI = Pointer to 256 byte buffer

Output: AX = status
(All other registers preserved)

The mode information block has the following structure:

ModeInfoBlock struc

```
;mandatory information
ModeAttributes      dw ? ;mode attributes
WinAAttributes      db ? ;window A attributes
WinBAttributes      db ? ;window B attributes
WinGranularity      dw ? ;window granularity
WinSize             dw ? ;window size
WinASegment          dw ? ;window A start segment
WinBSegment          dw ? ;window B start segment
WinFuncPtr          dd ? ;pointer to window function
BytesPerScanLine     dw ? ;bytes per scan line
                    ;extended information
;optional information
XResolution          dw ? ;horizontal resolution
YResolution          dw ? ;vertical resolution
XCharSize            db ? ;character cell width
YCharSize            db ? ;character cell height
NumberOfPlanes       db ? ;number of memory planes
BitsPerPixel         db ? ;bits per pixel
NumberOfBanks        db ? ;number of banks
MemoryMode           db ? ;memory model type
BankSize             db ? ;bank size in kb
NumberOfImagePages   db ? ;number of images
Reserved             db 1 ;reserved for page function
Reserved             db 255 dup (?) ;rest of ModeInfoBlock
```

Function 02h - Set Super VGA Video Mode

This function initializes a video mode. The BX register contains the video mode number. The format of VESA mode numbers is described previously. If the mode cannot be set, the BIOS should leave the video environment unchanged and return a failure error code.

Input: AH = 4Fh Super VGA support
AL = 02h Set Super VGA video mode
BX = Video mode
D0-D14 = video mode number
D15 = clear memory flag
0 = clear video memory
1 = Don't clear video memory

Output: AX = status
(All other registers are preserved.)

Function 03h - Return Current Video Mode

This function returns the current video mode in BX. The format of VESA video mode numbers is described previously.

Input: AH = 4Fh Super VGA support
AL = 03h Return current video mode

Output: AX = status
BX = current video mode number
(All other registers are preserved.)

Function 04h - Save/Restore Super VGA Video State

These functions provide a mechanism to save and restore the Super VGA video state. The functions are a superset of the three subfunctions under standard VGA BIOS function 1Ch (Save/Restore video state). The complete Super VGA video state (except video memory) should be savable/restorable by setting the requested states mask (in the CX register) to 000Fh.

Input: AH = 4Fh Super VGA support
AL = 04h Save/Restore Super VGA video states
DL = 00h Return save/restore state buffer size
CX = Requested states
D0 = Save/restore video hardware state
D1 = Save/restore video BIOS data state
D2 = Save/restore video DAC state
D3 = Save/restore Super VGA state

Output: AX = status
BX = Number of 64-byte blocks to hold the state buffer.

Input: AH = 4Fh Super VGA support
AL = 04h Save/Restore Super VGA video states
DL = 01h Save Super VGA video state
CX = Requested states
ES:BX = Pointer to buffer

Output: AX = status

Input: AH = 4Fh Super VGA support
AL = 04h Save/Restore Super VGA video states
DL = 02h Restore Super VGA video state
CX = Requested states
ES:BX = Pointer to buffer

Output: AX = status

Due to the goal of complete compatibility with the VGA environment, the standard VGA BIOS function 1Ch (Save/Restore VGA state) has not been extended to save the Super VGA video state. VGA BIOS compatibility requires that function 1Ch return a specific buffer size with specific contents, in which there is no room for the Super VGA state.

Function 05h - CPU Video Memory Window Control

This function sets or gets the position of the specified window in the video memory. The function allows direct access to the hardware paging registers. To use this function properly the software should use VESA BIOS function 01h (return Super VGA mode information) to determine the size, location, and granularity of the windows.

Input: AH = 4Fh Super VGA support
 AL = 05h Super VGA video memory window control
 BH = 00h Select super VGA video memory window
 BL = Window number
 0 = Window A
 1 = Window B
 DX = Window position in video memory (in window
 granularity units)

Output: AX = status

Input: AH = 4Fh Super VGA support
 AL = 05h Super VGA video memory window control
 BH = 01h Return super VGA video memory window
 BL = Window number
 0 = Window A
 1 = Window B

Output: AX = status
 DX = Window position in video memory

This function is also directly accessible through a far call from the application. The address of the BIOS function may be obtained by using VESA BIOS function 01h (return Super VGA mode information). A field in the ModeInfoBlock contains the address of this function. Note that this function may be different among video modes in a particular BIOS implementation so the function pointer should be obtained after each set mode.

In the far call version, no status information is returned to the application. Also, in the far call version, the AX and DX registers will be destroyed. Therefore if AX and/or DX must be preserved, the application must do so prior to making the far call.

The application must load the input arguments in BH, BL, and DX (for set window) but does not need to load either AH or AL in order to use the far call version of this function.

Function 06h - Set/Get Logical Scan Line Length

This function sets or gets the length of a logical scan line. This function allows an application to set up a logical video memory buffer that is wider than the displayed area. Function 07h then allows the application to set the starting position that is to be displayed.

Input: AH = 4Fh Super VGA support
 AL = 06h Logical Scan Line Length
 BL = 00h Select Scan Line Length

Output: AX = status
 BX = bytes per scan line
 CX = actual pixels per scan line
 DX = maximum number of scan lines

Input: AH = 4Fh Super VGA support
 AL = 06h Logical Scan Line Length
 BL = 01h Return Scan Line Length

Output: AX = status
 BX = bytes per scan line
 CX = actual pixels per scan line
 DX = maximum number of scan lines

The desired width in pixels may not be achievable because of VGA hardware considerations. The next larger value will be selected that will accommodate the desired number of pixels, and the actual number of pixels will be returned in CX. BX returns a value that when added to a pointer into video memory will point to the next scan line. For example, in a mode 13h this would be 320, but in mode 12h this would be 80. DX returns the number of logical scan lines based upon the new scan line length and the total memory installed and usable in this display mode. This function is also valid in text modes. In text modes, the application should find out the current character cell width through VESA function 01 (or VGA BIOS function 1Bh), multiply that times the desired number of characters per line, and pass that value in the CX register.

Function 07h - Set/Get Display Start

This function selects the pixel to be displayed in the upper left corner of the display from the logical page. This function can be used to pan and scroll around logical screens that are larger than the displayed screen. This function can also be used to rapidly switch between two different displayed screens for double buffered animation effects.

Input: AE = 4Fh Super VGA support
 AL = 07h Display Start Control
 BH = 00h Reserved and must be 0
 BL = 00h Select Display Start
 CX = First Displayed Pixel in Scan Line
 DX = First Displayed Scan Line

Output: AX = status

Input: AE = 4Fh Super VGA support
 AL = 07h Display Start Control
 BL = 01h Return Display Start

Output: AX = status
 BH = 00h Reserved and will be 0
 CX = First Displayed Pixel in Scan Line
 DX = First Displayed Scan Line

This function is also valid in text modes. In text modes the application should find out the current character cell width through VESA function 1 (or VGA BIOS function 1Bh), multiply that times the desired starting character column, and pass that value in the CX register. It should also multiply the current character cell height times the desired starting character row, and pass that value in the DX register.

REPORT DOCUMENTATION PAGEForm Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE February 1994	3. REPORT TYPE AND DATES COVERED Final report	
4. TITLE AND SUBTITLE Theory and Application of Image Enhancement			5. FUNDING NUMBERS	
6. AUTHOR(S) Michael G. Ellis, Sr. Roy L. Campbell, Jr.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Engineer Waterways Experiment Station Information Technology Laboratory 3909 Halls Ferry Road, Vicksburg, MS 39180-6199			8. PERFORMING ORGANIZATION REPORT NUMBER Technical Report ITL-94-1	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Corps of Engineers Washington, DC 20314-1000			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The conventional methods for image enhancement cannot be satisfactorily applied to the restoration of many types of images. This report explores standard image enhancement algorithms with emphasis on possible extensions to allow for real-time applications. The artifacts introduced by compression algorithms can often be compensated for by proper restoration techniques. A desktop conferencing system is prototyped in this report as a test-bed application to evaluate both real-time compression and real-time enhancement algorithms.				
14. SUBJECT TERMS Compression Enhancement Fourier			15. NUMBER OF PAGES 465	
Image restoration Palatte SVD			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	