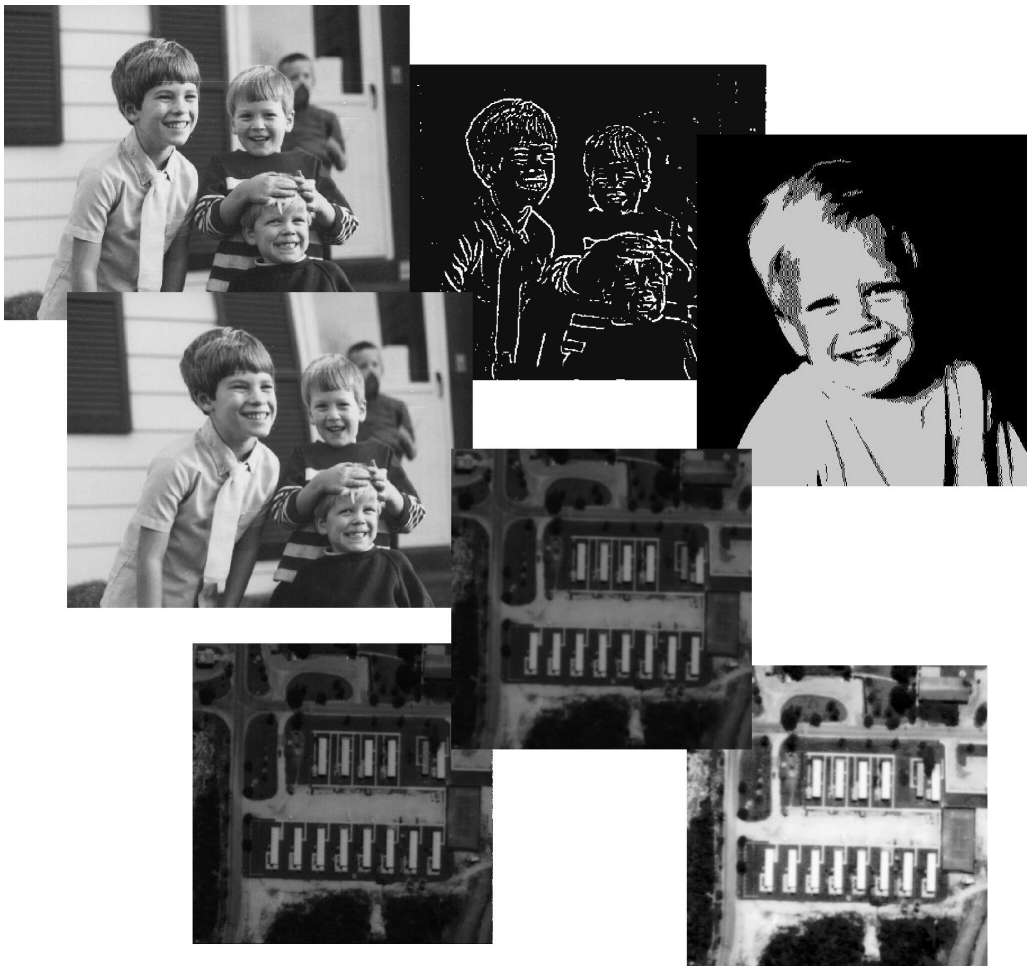
Image Processing in C

Second Edition



Dwayne Phillips

This first edition of "Image Processing in C" (Copyright 1994, ISBN 0-13-104548-2) was published by

R & D Publications
1601 West 23rd Street, Suite 200
Lawrence, Kansas 66046-0127

R & D Publications has since been purchased by Miller-Freeman, Inc. which has been purchased by CMP Media, Inc. I recommend reading "The C/C++ Users Journal" now published by CMP Media, Inc. See http://www.cuj.com.

The Electronic Second Edition of this text is Copyright ©2000 by Dwayne Phillips. Dwayne Phillips owns the electronic rights of this text. No part of this text may be copied without the written permission of Dwayne Phillips. If you have purchased the electronic edition of this text, you may print a copy.
Electronic Edition 1.0, 26 April 2000

The Source Code listed in this text is available at
http://members.aol.com/dwaynephil/cips2edsrc.zip

Preface

This book is a tutorial on image processing. Each chapter explains basic concepts with words and figures, shows image processing results with photographs, and implements the operations in C. Information herein comes from articles published in *The C/C++ Users Journal* from 1990 through 1998 and from the first edition of this book published in 1994. This second (electronic) edition contains new material in every chapter.

The goals of the first edition of this book were to (1) teach image processing, (2) provide image processing tools, (3) provide an image processing software system as a foundation for growth, and (4) make all of the above available to anyone with a plain, garden variety PC.

These goals remain the same today, but much else has changed. The update to this text reflects many of these changes. The Internet exploded, and this brought a limitless supply of free images to those of us who like to process them. With these images have come inexpensive software packages that display and print images as well as convert file formats.

The operating systems on home desktop and laptop computers have come of age. These have brought flat, virtual memory models so that it is easy to pull entire image files into memory for processing. This permitted the software revisions that are the basis of this second edition.

The software presented in this book will run on any computer using a 32-bit operating system (Windows 95, 98, NT and all flavors of UNIX). I compiled it using D.J. Delorie's port of the (free) GNU C compiler (DJGPP, see www.delorie.com). It should compile fine using commercially available C/C++ compilers. The software works on 8-bit, gray scale images in TIFF and BMP file formats. Inexpensive programs are available to convert almost any image into one of these formats.

Chapter 0 introduces the C Image Processing System. This chapter ties together many of the concepts of the software and how to use it.

Chapter 1 presents the image file input and output (I/O) routines used by the image operators in the remainder of the text. These I/O routines underwent major changes from the first edition of this text. The changes in the I/O code means chapter 1 is much longer in this edition and the remaining chapters and their source code are shorter.

Chapter 2 describes showing image numbers on a screen and dumping them to a text file for printing. I now leave image viewing and printing in today's windows systems to other, inexpensive programs.

Chapter 3 describes the halftoning technique that transform a gray scale image to a black and white image that looks like it has shades of gray. This chapter also shows how to use this to print wall posters of images.

Chapter 4 delves into histograms and histogram equalization. Histogram equalization allows you to correct for poor contrast in images. It presents a program that creates a picture of an image's histogram. It also gives a program that pastes images together.

Chapter 5 introduces edge detection — a basic operation in image processing.

Chapter 6 explains advanced edge detection techniques. We will use these techniques later in the book for segmentation.

Chapter 7 addresses spatial frequency filtering. It shows how to use various high-pass and low-pass filters to enhance images by removing noise and sharpening edges.

Chapter 8 considers sundry image operations. It demonstrates how to add and subtract images and cut and paste parts of images.

Chapter 9 introduces image segmentation. Segmentation is an attempt to divide the image into parts representing real objects in the image. Chapter 9 shows how to use simple histogram based segmentation.

Chapter 10 continues image segmentation with several advanced techniques. It discusses using edges, gray shades, and complex region growing algorithms.

Chapter 11 demonstrates morphological filtering or manipulating shapes. It describes erosion, dilation, outlining, opening, closing, thinning, and medial axis transforms.

Chapter 12 discusses Boolean operations and image overlaying. It shows how to use Boolean algebra to place a label on an image and how to overlay images for a double exposure effect.

Chapter 13 describes how to alter the geometry of images by displacement, scaling, rotation, and cross products. It provides a utility I often use

that stretches and compresses images.

Chapter 14 presents image warping and morphing. Warping is a 1960s technique that Hollywood embraced in the early 1990s. It leads to morphing.

Chapter 15 looks at textures and texture operators. Texture is hard to explain and harder to classify with computers. Nevertheless, there are a few ways to work this problem.

Chapter 16 explains stereograms. These dot-filled images contain 3-D objects if viewed correctly. Stereograms flooded the market in the early 1990s. The theory and techniques are simple and easy to use.

Chapter 17 examines steganography — the ability to hide information in images. Steganography exploits the unnecessary resolution of gray in images.

Chapter 18 shows how to write DOS .bat programs to use the programs of the C Image Processing System.

Chapter 19 shows the Windows interface I created for the C Image Processing System. I used the tcl/tk language and the Visual Tcl tool to create this. The tcl/tk scripting language is perfect for gluing together a set of programs like the image processing ones in this book.

The appendices provide information on the programming aspects of this book. They discuss the makefile for the programs (appendix A) and the stand alone application programs in CIPS (appendix B). Appendix C lists the individual functions and the source code files containing them. Appendix D gives all the image processing algorithms and the chapters in which they appear. Appendix E is a bibliography enumerating the books that have been of great help to me.

Appendix F contains all the source code listings. I struggled with putting the listings in each chapter or all together at the end of thebook. I chose the end as that makes it easier to print the text without lots of source code listings. You may download a copy of the source code from
http://members.aol.com/dwaynephil/cips2edsrc.zip

Have fun with this. I had fun updating the software and the descriptions. Thanks to the Internet (lots of free images) and newer operating systems (32-bit), image processing is more fun than ever before. Everyone is doing image processing today. Use the tools and techniques described here to join in. Every technique brings with it ideas for other things to do. So much fun and so little time.

Many thanks to the staff of *The C/C++ Users Journal* and Miller-Freeman past and present. In particular I want to thank Howard Hyten, Diane Thomas, Martha Masinton, Bernie Williams, P.J. Plauger, and Robert

and Donna Ward. They allowed me to keep writing installments to this series and put this book together.

Thanks also to my wife Karen. Marrying her was the smartest thing I ever did.

Dwayne Phillips Reston, Virginia May 2000

Contents

0 Introduction to CIPS **1**
0.1 Introduction 1
0.2 System Considerations 2
0.3 The Three Methods of Using CIPS 3
0.4 Implementation 3
0.5 Conclusions 4
0.6 References 4

1 Image File Input and Output **7**
1.1 Introduction 7
1.2 Image Data Basics 7
1.3 Image File I/O Requirements 8
1.4 TIFF 10
1.4.1 The IFD 11
1.4.2 The TIFF Code 15
1.5 BMP 17
1.5.1 The BMP Code 20
1.6 A Simple Program 21
1.7 Converting Between TIFF and BMP 21
1.8 Conclusions 22
1.9 References 22

2 Viewing and Printing Image Numbers **23**
2.1 Introduction 23
2.2 Displaying Image Numbers 23
2.3 Printing Image Numbers 24
2.4 Viewing and Printing Images 24
2.5 Conclusions 25

3 Halftoning **27**
3.1 Introduction 27
3.2 The Halftoning Algorithm 27
3.3 Sample Output 29
3.4 Printing an Image 31
3.5 Conclusions 31
3.6 Reference 31

4 Histograms and Equalization **33**
4.1 Introduction 33
4.2 Histograms 33
4.3 Histogram Equalization 35
4.4 Equalization Results 39
4.5 Implementation 44
4.6 The *side* Program 44
4.7 Conclusions 45
4.8 Reference 45

5 Basic Edge Detection **47**
5.1 Introduction 47
5.2 Edge Detection 47
5.3 Implementing Edge Detectors 51
5.4 Results 52
5.5 Conclusion 52
5.6 References 56

6 Advanced Edge Detection **57**
6.1 Introduction 57
6.2 Homogeneity Operator 58
6.3 Difference Operator 58
6.4 Difference of Gaussians 60
6.5 More Differences 65
6.6 Contrast-based Edge Detector 66
6.7 Edge Enhancement 69
6.8 Variance and Range 70
6.9 Applications 70
6.10 Conclusions 73
6.11 References 73

7 Spatial Frequency Filtering 75
7.1 Spatial Frequencies 75
7.2 Filtering 75
7.3 Application of Spatial Image Filtering 77
7.4 Frequency vs. Spatial Filtering 77
7.5 Low-Pass Filtering 78
7.6 Median Filters 80
7.7 Effects of Low-Pass Filtering 81
7.8 Implementing Low-Pass Filtering 87
7.9 High-Pass Filtering 88
7.10 Effects of High-Pass Filtering 88
7.11 Implementing High-Pass Filtering 92
7.12 Conclusion 92
7.13 References 93

8 Image Operations 95
8.1 Introduction 95
8.2 Addition and Subtraction 95
8.3 Rotation and Flipping 98
8.4 Cut and Paste 98
8.5 Image Scaling 99
8.6 Blank Images 99
8.7 Inverting Images 100
8.8 Conclusion 101

9 Histogram-Based Segmentation 103
9.1 Histogram-Based Segmentation 103
9.2 Histogram Preprocessing 106
9.3 Thresholding and Region Growing 110
9.4 Histogram-Based Techniques 113
9.4.1 Manual Technique 113
9.4.2 Histogram Peak Technique 117
9.4.3 Histogram Valley Technique 119
9.4.4 Adaptive Histogram Technique 121
9.5 An Application Program 122
9.6 Conclusions 122
9.7 Reference 123

10 Segmentation via Edges & Gray Shades 125
10.1 Introduction 125
10.2 Segmentation Using Edges & Gray Shades 125
10.3 Problems 129
10.4 Solutions 132
10.4.1 Preprocessing 132
10.4.2 Improved Edge Detection 136
10.4.3 Improved Region Growing 138
10.5 The Three New Techniques 145
10.5.1 Edges Only 145
10.5.2 Gray Shades Only 146
10.5.3 Edges and Gray Shade Combined 146
10.6 Integrating the New Techniques 149
10.7 Conclusions 149
10.8 Reference 151

11 Manipulating Shapes 153
11.1 Introduction 153
11.2 Working with Shapes 153
11.3 Erosion and Dilation 156
11.4 Opening and Closing 160
11.5 Special Opening and Closing 163
11.6 Outlining 171
11.7 Thinning and Skeletonization 176
11.8 A Shape Operations Application Program 179
11.9 Conclusions 179
11.10References 181

12 Boolean and Overlay Operations 183
12.1 Introduction 183
12.2 Boolean Operations 183
12.3 Applications of Boolean Operations 184
12.4 Overlay Operations 188
12.5 Applications of Overlay Operations 188
12.6 Conclusions 196

13 Geometric Operations **197**
13.1 Introduction 197
13.2 Geometric Operations 197
13.3 Rotation About Any Point 202
13.4 Bi-Linear Interpolation 203
13.5 An Application Program 206
13.6 A Stretching Program 207
13.7 Conclusions 208
13.8 References 208

14 Warping and Morphing **209**
14.1 Introduction 209
14.2 Image Warping 209
14.3 The Warping Technique 210
14.4 Two Ways to Warp 212
14.5 Shearing Images 216
14.6 Morphing 218
14.7 A Warping Application Program 221
14.8 Conclusions 222
14.9 References 222

15 Basic Textures Operations **223**
15.1 Introduction 223
15.2 Textures 223
15.3 Edge Detectors as Texture Operators 225
15.4 The Difference Operator 231
15.5 The Hurst Operator 234
15.6 The Compare Operator 239
15.7 An Application Program 241
15.8 Conclusions 241
15.9 References 241

16 Random Dot Stereograms **243**
16.1 Introduction 243
16.2 Stereogram Basics 243
16.3 Stereogram Algorithms 249
16.4 Source Code and Examples 252
16.5 Colorfield Stereograms 256

16.6 Conclusions 264
16.7 Reference 264

17 Steganography: Hiding Information 265
17.1 Introduction 265
17.2 Hidden Writing 265
17.3 Watermarking 266
17.4 Hiding Images in Images 269
17.5 Extensions 274
17.6 Conclusions 275
17.7 Reference 275

18 Command-Line Programming 277
18.1 Introduction 277
18.2 Batch Programming with .bat Files 277
18.3 Basics of .bat Programming 278
18.4 Uses and Examples 280
18.5 Conclusions 282

19 A Tcl/Tk Windows Interface 283
19.1 Introduction 283
19.2 The Need for a Windows Interface 283
19.3 Options 284
19.4 The Tcl/Tk Graphical User Interface 285
19.5 Conclusions 288
19.6 Reference 288

A The makefile 289
A.1 The Listings 290
A.2 Commands to Build The C Image Processing System 291
A.3 Reference 291
A.4 Code Listings 291

B The Stand-Alone Application Programs 301

C Source Code Tables of Contents 307
C.1 Listings 307

D Index of Image Processing Algorithms **319**
D.1 Algorithms Listed in Order of Appearance 319
D.2 Algorithms Listed Alphabetical Order 322

E Bibliography **327**
E.1 Image Processing Books 327
E.2 Programming Books 329

F Source Code Listings **331**
F.1 Code Listings for Chapter 1 331
F.2 Code Listings for Chapter 2 395
F.3 Code Listings for Chapter 3 401
F.4 Code Listings for Chapter 4 409
F.5 Code Listings for Chapter 5 425
F.6 Code Listings for Chapter 6 440
F.7 Code Listings for Chapter 7 459
F.8 Code Listings for Chapter 8 471
F.9 Code Listings for Chapter 9 487
F.10 Code Listings for Chapter 10 512
F.11 Code Listings for Chapter 11 538
F.12 Code Listings for Chapter 12 591
F.13 Code Listings for Chapter 13 623
F.14 Code Listings for Chapter 14 642
F.15 Code Listings for Chapter 15 661
F.16 Code Listings for Chapter 16 683
F.17 Code Listings for Chapter 17 729
F.18 Code Listings for Chapter 18 743
F.19 Code Listings for Chapter 19 753

List of Figures

1.1 A Sample Program 9
1.2 Existing Standard TIFF Tags 12
1.3 The Structure of a TIFF File 13
1.4 The Beginning of a TIFF File 14
1.5 Possible Data Types and Lengths 15
1.6 The BMP File Header 18
1.7 The Bit Map Header 19

3.1 The Basic Halftoning Algorithm 28
3.2 Input Boy Image 30
3.3 Output Halftoned Boy Image 30
3.4 Poster Created with the dumpb Program 32

4.1 Simple Histogram 34
4.2 Histogram of a Poorly Scanned Image 35
4.3 Boy Image with Histogram 36
4.4 House Image with Histogram 37
4.5 Image with Poor Contrast 38
4.6 Histogram Equalization Algorithm 40
4.7 Equalized Version of Figure 4.5 41
4.8 Comparing Figures 4.6 and 4.7 42
4.9 Equalizing a Properly Scanned Image 43

5.1 Graphs of Gray Scale Values at Edges 48
5.2 Masks Used by Faler for Edge Detection 49
5.3 Masks for Edge Detection 50
5.4 The House Image 53
5.5 The Result of the Kirsch Masks 53
5.6 The Result of the Prewitt Masks 54

5.7 The Result of the Sobel Masks 54
5.8 The Result of the Sobel Masks Without Thresholding 55
5.9 The Result of the Quick Mask 55

6.1 Original House Image 57
6.2 An Example of the Homogeneity Operator 59
6.3 Result of Homogeneity Edge Detector 60
6.4 An Example of the Difference Operator 61
6.5 Result of Difference Edge Detector 62
6.6 Gaussian "Mexican Hat" Masks 63
6.7 Detecting Small Edges 64
6.8 Result of Gaussian Edge Detector with 7x7 Mask 65
6.9 Result of Gaussian Edge Detector with 9x9 Mask 66
6.10 Contrast-Based Edge Detector 67
6.11 Result of Quick Edge Detector 68
6.12 Result of Contrast-Based Edge Detector 69
6.13 Result of Edge Enhancement 70
6.14 The Results of Applying the Variance and Range Operators to an Array of Numbers 71
6.15 Result of Variance Edge Detector 72
6.16 Result of Range Edge Detector 72

7.1 Side View of an Image with Low Spatial Frequencies 76
7.2 Side View of an Image with High Spatial Frequencies 76
7.3 Low-Pass Filter Convolution Masks 78
7.4 An Image Segment with Low Spatial Frequencies 79
7.5 An Image Segment with High Spatial Frequencies 79
7.6 Low-Pass Filtering of Figure 7.4 80
7.7 Low-Pass Filtering of Figure 7.5 81
7.8 Noisy Aerial Image 82
7.9 Result of Low-Pass Filter Mask #6 82
7.10 Result of Low-Pass Filter Mask #9 83
7.11 Result of Low-Pass Filter Mask #10 83
7.12 Result of Low-Pass Filter Mask #16 84
7.13 Result of 3x3 Median Filter 84
7.14 House Image 85
7.15 Result of 3x3 Median Filter 86
7.16 Result of 5x5 Median Filter 86

7.17 Result of 7x7 Median Filter 87
7.18 High-Pass Filter Convolution Masks 89
7.19 Result of High-Pass Filter on Figure 7.4 89
7.20 Result of High-Pass Filter on Figure 7.5 90
7.21 Result of High-Pass Filter Mask #1 91
7.22 Result of High-Pass Filter Mask #2 91
7.23 Result of High-Pass Filter Mask #3 92

8.1 Addition and Subtraction of Images 96
8.2 A House Image 96
8.3 Edge Detector Output of Figure 8.2 97
8.4 Figure 8.2 Minus Figure 8.3 (Edges Subtracted) 97
8.5 Cutting and Pasting 98
8.6 Section of Figure 8.3 Cut and Pasted Into Figure 8.2 99
8.7 Two Images Pasted Onto a Blank Image 100

9.1 An Image Example 104
9.2 A Histogram of the Image of Figure 9.1 105
9.3 The Image in Figure 9.1 with All the Pixels Except the 8s Blanked Out 105
9.4 Figure 9.1 with a Threshold Point of 5 106
9.5 Aerial Image with Poor Contrast 107
9.6 Result of Histogram Equalization on Figure 9.5 108
9.7 Result of High-Pass Filtering on Figure 9.6 109
9.8 The Result of Smoothing the Histogram Given in Figure 9.2 . 110
9.9 The Result of Correctly Thresholding Figure 9.1 111
9.10 The Result of Region Growing Performed on Figure 9.9 112
9.11 Pseudocode for Region Growing 114
9.12 Input Image for Segmentation Examples 115
9.13 Threshold of Figure 9.12 with High=255 and Low=125 115
9.14 Threshold of Figure 9.12 with High=255 and Low=175 116
9.15 Threshold of Figure 9.12 with High=255 and Low=225 116
9.16 Result of Incorrect Peak Separation 118
9.17 A Histogram in which the Highest Peak Does Not Correspond to the Background 119
9.18 Threshold of Figure 9.12 Using Peak Technique (High=255 and Low=166) 120

9.19 Threshold of Figure 9.12 Using Valley Technique (High=255 and Low=241) 121
9.20 Threshold of Figure 9.12 Using Adaptive Technique (High=255 and Low=149) 122

10.1 Using Edges to Segment an Image 126
10.2 Growing Objects Using Gray Shades 127
10.3 Growing Objects Using Gray Shades and Edges 128
10.4 Aerial Image of House Trailers 129
10.5 House Image 130
10.6 Edge Detector Output from Figure 10.5 130
10.7 A Small Edge Detector Error Leads to a Big Segmentation Error 131
10.8 Edge Detector Output from Figure 10.4 132
10.9 Triple-Thick Edges Distort Objects 133
10.10 Result of Mistaking Edges for Objects 134
10.11 Output of Median, High-Pixel, and Low-Pixel Filters 135
10.12 Low-Pixel Filtering Performed on Figure 10.5 136
10.13 Edge Detector Output from Figure 10.12 137
10.14 Edge Detector Output from Figure 10.4 — Thresholded at 70% 138
10.15 Result of Eroding Stray Edges 139
10.16 Eroding Away Thick Edges 140
10.17 Result of Eroding the Edges in Figure 10.13 141
10.18 The Region Growing Algorithm from Chapter 9 142
10.19 The Improved Region Growing Algorithm (Part 1) 143
10.19 The Improved Region Growing Algorithm (Part 2) 144
10.20 Sobel Edge Detector Output from Figure 10.4 (after Erosion) . 147
10.21 Result of Edge-Only Segmentation of Figure 10.4 148
10.22 Result of Gray-Shade-Only Segmentation of Figure 10.4 148
10.23 Result of Edge and Gray Shade Segmentation of Figure 10.4 . 149
10.24 Result of Edge-Only Segmentation of Figure 10.5 150
10.25 Result of Gray-Shade-Only Segmentation of Figure 10.5 150
10.26 Result of Edge and Gray Shade Segmentation of Figure 10.5 . 151

11.1 Aerial Image 154
11.2 Segmentation of Aerial Image 154
11.3 House Image 155
11.4 Segmentation of House Image 155
11.5 A Binary Image 156

11.6 The Result of Eroding Figure 11.5 156
11.7 The Result of Dilating Figure 11.5 157
11.8 The Result of Eroding Figure 11.5 Using a Threshold of 2 . . 158
11.9 The Result of Dilating Figure 11.5 Using a Threshold of 2 . . 158
11.10 Four 3x3 Masks 158
11.11 The Result of Dilating Figure 11.5 with the Four Masks of Figure 11.9 159
11.12 Examples of Masked Vertical and Horizontal Dilations 161
11.13 Two Objects Joined by a Thread, Separated by opening and a Hole Enlarged by opening 162
11.14 A Segmentation and the Result of Opening 163
11.15 Two Objects that Should be Joined, How closing Removes the Break and Fills Unwanted Holes 164
11.16 An Unwanted Merging of Two Objects 165
11.17 Closing of Segmentation in Figure 11.2 166
11.18 An Unwanted Splitting of an Object 167
11.19 Result of Special Routines that Open and Close Objects but do not Join or Break Them 168
11.20 Result of Opening of a 2-Wide Object 169
11.21 Cases Where Objects Can and Cannot be Eroded 169
11.22 Cases that do and do not Require a Special Closing Routine . 170
11.23 Special Closing of Segmentation of Figure 11.2 171
11.24 Erosion of Segmentation in Figure 11.4 172
11.25 Special Closing of Figure 11.24 172
11.26 Outline of Segmentation in Figure 11.4 173
11.27 The Interior Outline of an Object 174
11.28 The Exterior Outline of an Object 175
11.29 Thinning a Rectangle until it is One Pixel Wide 176
11.30 A Square, its Euclidean Distance Measure, and its Medial Axis Transform (Part 1) 178
11.30 A Square, its Euclidean Distance Measure, and its Medial Axis Transform (Part 2) 179
11.31 A Rectangle and its Medial Axis Transform 180
11.32 (Clockwise from Upper Left) A, Its Outline, Medial Axis Transform, and Thinning 181

12.1 Existing Standard TIFF Tags 184
12.2 Original Aerial Image 185

12.3 Segmentation of Aerial Image (from Chapter 10) 185
12.4 Segmented Aerial Image Masked with Original 186
12.5 *ilabel* Output on Left, Dilation in Center, XOR of Both on Right 187
12.6 Labeled Boy Image 187
12.7 Images A and B 189
12.8 Result of Overlay Non-Zero A 190
12.9 Result of Overlay Zero A 190
12.10 Result of Overlay Greater A 191
12.11 Result of Overlay Less A 191
12.12 Result of Average Overlay 192
12.13 Two Images Side by Side 192
12.14 Two Images Averaged 193
12.15 Leafy Texture Image 193
12.16 House Image 194
12.17 Averaging Leafy Texture and House Image 194
12.18 White Frame in Blank Image 195
12.19 Frame Overlaid on Boy Image 196

13.1 The Three Basic Geometric Operations: Displacement, Stretching, and Rotation 198
13.2 Examples of Displacement 199
13.3 Examples of Stretching 200
13.4 Examples of Rotation about the Origin 200
13.5 Examples of Cross Products 201
13.6 Combining All Four Geometric Operations 202
13.7 Rotation About any Point m,n 203
13.8 Examples of Rotation About Any Point 204
13.9 A Comparison of Not Using Bi-Linear Interpolation and Using Bi-Linear Interpolation 205
13.10 Bi-Linear Interpolation 206
13.11 The Boy Image Enlarged Horizontally and Shrunk Vertically . 207

14.1 Bi-Linear Interpolation of a Quadrilateral 211
14.2 The Control Point Warping Process 213
14.3 Examples of Control Point Warping 215
14.4 The Object Warping Process 215
14.5 Examples of Object Warping 217
14.6 Warped House Image 217

14.7 Another Warped House Image 218
14.8 Examples of Image Shearing 219
14.9 Morphing a Black Circle to a White Pentagon 220
14.10 A Morphing Sequence 220

15.1 Examples of Textures 224
15.2 Four Textures 224
15.3 An Example of How the Sobel Edge Detector Does Not Work Well with a Texture 226
15.4 The Result of Applying the Range Edge Detector to a Texture 227
15.5 The Result of Applying the Variance Edge Detector to a Texture 228
15.6 The Result of Applying the Sigma Edge Detector to a Texture 230
15.7 The Result of Applying the Skewness Operator to a Texture . 231
15.8 The Result of Applying the Difference Operator to a Texture . 232
15.9 The Result of Applying the Mean Operator to the Same Texture as in Figure 15.8 233
15.10 Three Size Areas for the Hurst Operator 235
15.11 Two Example Image Sections 236
15.12 Values Calculated by the Hurst Operator 237
15.13 The Result of Applying the Hurst Operator to a Texture . . . 238
15.14 The Failed Result of Applying the Hurst Operator to the House Image 239
15.15 The Result of Applying the Compare Operator to a Texture . 240
15.16 The Result of Applying the Compare Operator to the House Image 241

16.1 Divergent Viewing 244
16.2 The Repeating Pattern 244
16.3 Deleting an Element from the Pattern 245
16.4 Inserting an Element into the Pattern 246
16.5 Deleting and Inserting to Create an Object 247
16.6 A Character Stereogram 247
16.7 A Random Character Stereogram 248
16.8 Substitution Values for the First Line of Figures 16.6 and 16.7 249
16.9 A Depth Image and Random Character Stereogram Image . . 250
16.10 The Stereogram Processing Loop 251
16.11 The Shorten Pattern Algorithm 252
16.12 The Lengthen Pattern Algorithm 253

16.13A Simple Depth File Image 254
16.14A Random Dot Stereogram from Figure 16.13 255
16.15A Random Dot Stereogram 256
16.16A "Colorfield" Image of Boys 257
16.17A Colorfield Stereogram from Figure 16.16 258
16.18A Colorfield Image of Houses 259
16.19A Depth Image 260
16.20The Stereogram from Figures 16.18 and 16.19 261
16.21A Character Depth File 262
16.22A Character Colorfied Stereogram 263

17.1 The Original Boy Image 267
17.2 The Watermark Image 267
17.3 Overlaying the Watermark on the Boy Image 268
17.4 Hiding the Watermark on the Boy Image 268
17.5 Hiding Message Image Pixels in a Cover Image 270
17.6 The Cover Image 271
17.7 The Message Image 272
17.8 The Cover Image with the Message Image Hidden In It 272
17.9 The Unhidden Message Image 273

18.1 A .bat File 278
18.2 Another Simple .bat File 279
18.3 A .bat File with Replaceable Parameters 279
18.4 A .bat File that Checks for Parameters 280

19.1 The Main CIPS Window 286
19.2 The Window for the stretch Program 287
19.3 The Window for the Various Texture Operators 287

Chapter 0

Introduction to CIPS

0.1 Introduction

This chapter presents the underlying concepts of the remaining chapters in this electronic book. The first edition of this book [0.18] was released in 1994 from R&D Publications. That book was first written as separate articles for *The C Users Journal* from 1991 through 1993 [0.2- 0.12]. Since then, R&D Publications was purchased by Miller-Freeman, *The C Users Journal* became *The C/C++ Users Journal*, and much has changed in the world of image processing. *The C/C++ Users Journal* published five more articles from 1995 through 1998 [0.13-0.17]. Versions of these articles are included.

Every chapter in this edition of the book is different from the first edition. All the source code has been modified. The goals of the following chapters are to (1) teach image processing, (2) provide image processing tools, (3) provide an image processing software system as a foundation for growth, and (4) make all of the above available to anyone with a plain, garden variety PC. *The C/C++ Users Journal* is an excellent forum for teaching. The publisher and editors have allowed me to explain image processing from the basic to the advanced. Each chapter reviews image processing techniques with words, figures, and photographs. After examining the ideas, each chapter discusses C source code to implement the operations. The complete source code is listed in Appendix F. The source code can be downloaded from

http://members.aol.com/dwaynephil/cips2edsrc.zip

The techniques in this collection would fill a large part of a college or graduate level textbook. The textbooks, however, do not give useful source

code.

It is the source code that keeps this book from being another academic or reference work. The intent was to give people working edge detectors, filters, and histogram equalizers so they would not need to write them again. An equally important goal was to give people disk I/O, display, and print routines. These routines make a collection of operators into a software system. They handle the dull necessities and allow you to concentrate on exciting, new operations and techniques.

The overriding condition continues to do all this using a basic personal computer. The basic personal computer of 2000 is much different from 1994, but the routines do not require special hardware.

0.2 System Considerations

Image processing software performs image disk I/O, manipulates images, and outputs the results. This book will be easier to understand if you know how the C Image Processing System (CIPS) performs these three tasks.

The first task is image disk I/O, and the first item needed is an image file format. The file format specifies how to store the image and information about itself. The software in this book works with Tagged Image File Format (TIFF) files and Windows bit mapped (BMP) files. Aldus (of PageMaker fame) invented TIFF in the mid-1980s and worked with several scanner manufacturers and software developers to create an established and accepted standard. The source code in this text works with 8-bit gray scale TIFF files (no compression). The source code also works with 8-bit BMP files (again, no compression). These are basic pixel-based image files. Images are available from many sources today (the Internet is a limitless source). Also available are inexpensive programs ($50 down to free) that convert images to the formats I support. Chapter 1 discusses these formats and shows code that will read and write these files.

The second task is image manipulation. This is how the software holds the image data in memory and processes it. The CIPS described in this edition is much better than the first edition in this respect. The first edition used a 16-bit compiler and was limited by the 64K byte memory segments in the PC. This edition uses a 32-bit compiler that can use virtually limitless memory. Therefore, this software reads entire images into a single array. This allows the image processing programmer to concentrate on image processing.

The final task is outputting results. CIPS can write results to TIFF and BMP image files, display image numbers on the screen, and dump image numbers to a text file. This is less than the CIPS presented in the first edition. I now leave image display and printing to others. This is because Windows-based image display programs are available free or at low cost on the Internet and elsewhere. (If you double click on a .bmp file, Microsoft Paint will display it). I use and recommend VuePrint from Hamrick Software (http://www.hamrick.com).

0.3 The Three Methods of Using CIPS

There are three ways to use CIPS: (1) interactively, (2) by writing C programs, and (3) by writing .bat files. Now that we are all in the Windows world, I have included a Windows application that allows the user to click buttons and fill in blanks. I created this using The Visual tcl toolkit and the tcl scripting language. Chapter 18 describes this process.

All the image processing subroutines share a common format that allows you to call them from your own C programs. The common format has a parameter list containing image file names, and listing items specific to each operator. The subroutines call the disk I/O routines and perform their specific image processing function. Writing stand-alone application programs is not difficult with CIPS. This book contains more than a dozen such programs as examples (Appendix B gives a list and explanation of these).

The third method of using the CIPS software is by writing DOS .bat files. The stand-alone programs in this book are all command-line driven. The advantage to this is that you can call them from .bat files. Chapter 17 explains this technique in detail and gives several examples.

0.4 Implementation

I implemented this software using a DOS port of the GNU C compiler. This is the well known D.J. Delorie port (see http://delorie.com). It is free to download, is updated regularly, and works well. The source code ports to other C compilers and systems reasonably well. I created a large makefile to help manage the software (see Appendix A). It allows you to make code changes and rebuild the programs with simple commands.

0.5 Conclusions

Enjoy this book. Use the source code and experiment with images. One of the good things about image processing is you can see the result of your work. Investigate, explore different combinations of the techniques mentioned. There are no right or wrong answers to what you are doing. Each image has its own characteristics, challenges, and opportunities.

0.6 References

0.1 "TIFF, Revision 6.0, Final, June 3, 1993," Aldus Developers Desk, For a copy of the TIFF 6.0 specification, call (206) 628-6593. See also http:www.adobe.comsupportservicedevrelationsPDFSTNTIFF6.pdf.
0.2 "Image Processing, Part 11: Boolean and Overlay Operations," Dwayne Phillips, The C Users Journal, August 1993.
0.3 "Image Processing, Part 10: Segmentation Using Edges and Gray Shades," Dwayne Phillips, The C Users Journal, June 1993.
0.4 "Image Processing, Part 9: Histogram-Based Image Segmentation," Dwayne Phillips, The C Users Journal, February 1993.
0.5 "Image Processing, Part 8: Image Operations," Dwayne Phillips, The C Users Journal, November 1992.
0.6 "Image Processing, Part 7: Spatial Frequency Filtering," Dwayne Phillips, The C Users Journal, October 1992.
0.7 "Image Processing, Part 6: Advanced Edge Detection," Dwayne Phillips, The C Users Journal, January 1992.
0.8 "Image Processing, Part 5: Writing Images to Files and Basic Edge Detection," Dwayne Phillips, The C Users Journal, November 1991.
0.9 "Image Processing, Part 4: Histograms and Histogram Equalization," Dwayne Phillips, The C Users Journal, August 1991.
0.10 "Image Processing, Part 3: Displaying and Printing Images Using Halftoning," Dwayne Phillips, The C Users Journal, June 1991.
0.11 "Image Processing, Part 2: Displaying Images and Printing Numbers," Dwayne Phillips, The C Users Journal, May 1991.
0.12 "Image Processing, Part 1: Reading the Tag Image File Format," Dwayne Phillips, The C Users Journal, March 1991.
0.13 "Geometric Operations," Dwayne Phillips, The C/C++ Users Journal, August 1995.

0.14 "Warping and Morphing," Dwayne Phillips, The C/C++ Users Journal, October 1995.
0.15 "Texture Operations," Dwayne Phillips, The C/C++ Users Journal, November 1995.
0.16 "Stereograms," Dwayne Phillips, The C/C++ Users Journal, April 1996.
0.17 "Steganography," Dwayne Phillips, The C/C++ Users Journal, November 1998.
0.18 "Image Processing in C," Dwayne Phillips, R&D Publications Inc., 1994, ISBN 0-13-104548-2.

Chapter 1

Image File Input and Output

1.1 Introduction

Image processing involves processing or altering an existing image in a desired manner. The first step is obtaining an image in a readable format. This is much easier today than five years back. The Internet and other sources provide countless images in standard formats. This chapter describes the TIFF and BMP file formats and presents source code that reads and writes images in these formats.

Once the image is in a readable format, image processing software needs to read it so it can be processed and written back to a file. This chapter presents a set of routines that do the reading and writing in a manner that frees the image processing programming from the details.

1.2 Image Data Basics

An image consists of a two-dimensional array of numbers. The color or gray shade displayed for a given picture element (pixel) depends on the number stored in the array for that pixel. The simplest type of image data is black and white. It is a binary image since each pixel is either 0 or 1.

The next, more complex type of image data is gray scale, where each pixel takes on a value between zero and the number of gray scales or gray levels that the scanner can record. These images appear like common black-and-white photographs — they are black, white, and shades of gray. Most gray scale images today have 256 shades of gray. People can distinguish about 40

shades of gray, so a 256-shade image "looks like a photograph." This book concentrates on gray scale images.

The most complex type of image is color. Color images are similar to gray scale except that there are three bands, or channels, corresponding to the colors red, green, and blue. Thus, each pixel has three values associated with it. A color scanner uses red, green, and blue filters to produce those values.

Images are available via the Internet, scanners, and digital cameras. Any picture shown on the Internet can be downloaded by pressing the right mouse button when the pointer is on the image. This brings the image to your PC usually in a JPEG format. Your Internet access software and other software packages can convert that to a TIFF or BMP.

Image scanners permit putting common photographs into computer files. The prices of full-color, full-size scanners are lower than ever (some available for less than $100). Be prepared to experiment with scanning photographs. The biggest problem is file size. Most scanners can scan 300 dots per inch (dpi), so a 3"x5" photograph at 300 dpi provides 900x1500 pixels. At eight bits per pixel, the image file is over 1,350,000 bytes.

Digital cameras have come out of the research lab and into consumer electronics. These cameras store images directly to floppy disk. Most cameras use the JPEG file format to save space. Again, there are many inexpensive image viewing software packages available today that convert JPEG to TIFF and BMP.

1.3 Image File I/O Requirements

Image file I/O routines need to read and write image files in a manner that frees the programmer from worrying about details. The routines need to hide the underlying disk files.

Figure 1.1 shows what a programmer would like to write when creating a routine. The first three lines declare the basic variables needed. Line 3 creates the output image to be just like the input image (same type and size). The output image is needed because the routines cannot write to an image file that does not exist. Line 4 reads the size of the input image. The height and width are necessary to allocate the image array. The allocation takes place in line 5. The size (height and width) does not matter to the programmer. Line 6 reads the image array from the input file. The type of

```
Line
0     char  *in_name, *out_name;
1     short **the_image;
2     long  height, width;

3     create_image_file(in_name, out_name);
4     get_image_size(in_name, &height, &width);
5     the_image = allocate_image_array(height, width);
6     read_image_array(in_name, the_image);

7        call an image processing routine

8     write_image_array(out_name, the_image);
9     free_image_array(the_image, height);
```

Figure 1.1: A Sample Program

input file (TIFF or BMP) does not matter. Line 7 is where the programmer calls the desired processing routine. Line 8 writes the resulting image array to the output file, and line 9 frees the memory array allocated in line 5.

The routines in figure 1.1 are the top-level of a family of routines. These hide the image file details from the programmer. The underlying routines do the specific work. This structure removes all file I/O from the image processing routines. All routines receive an array of numbers and the size of the array. This improves the portability of the image processing routines. They do not depend on image formats or sources.

This structure also makes it easier to add more image file formats. The read_image_array function rides on top of a set of routines that determine the specific file format and read it. Adding new routines below read_image_array will not affect the vast majority of code in the image processing system.

Listing 1.1 shows the high-level I/O routines. It begins with the basic read_image_array and write_image_array. These routines call routines that check the file format and call the read and write routines for those specific formats. Adding new file formats means adding calls to those routines here. The next routine in listing 1.1 is create_image_file. It also calls routines to determine the specific file format and create files for those formats.

The get_image_size routine determines the size of the image to process. This information is needed to allocate image arrays and to pass to processing routines. The processing routines will receive a pointer to an array. They must also receive the size of the image or they cannot process through the numbers. The get_image_size routine determines the specific file format and calls routines to read the image headers.

The next two routines, allocate_image_array and free_image_array, create and free memory for arrays of numbers. This completes the routines shown in figure 1.1. The remaining routines in listing 1.1 are used in many of the programs presented in this book. Like the routines described earlier, they ride on top of other routines that work with specific image file formats. They create files, determine if files exist, manipulate headers, and pull important information from image headers.

1.4 TIFF

Several computer and scanner companies created an industry standard for digital image data communication [1.1]. Their collaboration resulted in the TIFF specification. Since most scanner manufacturers support the standard in their PC and Macintosh products, TIFF is a natural for PC-based image processing.

The goals of the TIFF specification are extensibility, portability, and revisability. TIFF must be extensible in the future. TIFF must be able to adapt to new types of images and data and must be portable between different computers, processors, and operating systems. TIFF must be revisable — it is not a read-only format. Software systems should be able to edit, process, and change TIFF files.

The tag in Tag Image File Format refers to the file's basic structure. A TIFF tag provides information about the image, such as its width, length, and number of pixels. Tags are organized in tag directories. Tag directories have no set length or number, since pointers lead from one directory to another. The result is a flexible file format that can grow and survive in the future. Figure 1.2 contains the existing standard tags. Figure 1.3 shows the structure of TIFF. The first eight bytes of the file are the header. These eight bytes have the same format on all TIFF files. They are the only items set in concrete for TIFF files. The remainder of the file differs from image to image. The IFD, or Image File Directory, contains the number of direc-

tory entries and the directory entries themselves. The right-hand column in Figure 1.2 shows the structure of each directory entry. Each entry contains a tag indicating what type of information the file holds, the data type of the information, the length of the information, and a pointer to the information or the information itself.

Figure 1.4 shows the beginning of a TIFF file. The addresses are located on the left side in decimal, and the bytes and their values are in the table in hex.

Glancing between Figures 1.3 and 1.4 should clarify the structure. The first eight bytes are the header. Bytes zero and one tell whether the file stores numbers with the most significant byte (MSB) first, or least significant byte (LSB) first. If bytes zero and one are II (0x4949), then the least significant byte is first (predominant in the PC world). If the value is MM (0x4D4D), the most significant byte is first (predominant in the Macintosh world). Your software needs to read both formats.

The example in Figure 1.4 shows LSB first. Bytes two and three give the TIFF version number, which should be 42 (0x2A) in all TIFF images. Bytes four to seven give the offset to the first Image File Directory (IFD). Note that all offsets in TIFF indicate locations with respect to the beginning of the file. The first byte in the file has the offset 0. The offset in Figure 1.4 is 8, so the IFD begins in byte nine of the file.

1.4.1 The IFD

The content of address eight is 27, indicating that this file has 27 12-byte directory entries. The first two bytes of the entry contain the tag, which tells the type of information the entry contains. The directory entry at location 0 (Figure 1.4) contains tag=255. This tag tells the file type. (Refer to Figure 1.2 for possible tags.) The next two bytes of the entry give the data type of the information (Figure 1.5 lists the possible data types and their lengths). Directory entry 0 in Figure 1.4 is type=3, a short (two-byte unsigned integer). The next four bytes of the entry give the length of the information. This length is not in bytes, but rather in multiples of the data type. If the data type is a short and the length is one, the length is one short, or two bytes. An entry's final four bytes give either the value of the information or a pointer to the value. If the size of the information is four bytes or less, the information is stored here. If it is longer than four bytes, a pointer to it is stored. The information in directory entry zero is two bytes

```
SubfileType
     Tag = 255 (FF)     Type = short       N = 1
     Indicates the kind of data in the subfile.

ImageWidth
     Tag = 256 (100)    Type = short       N = 1
     The width (x or horizontal) of the image in pixels.

ImageLength
     Tag = 257 (101)    Type = short       N = 1
     The length (y or height or vertical) of the image in pixels.

RowsPerStrip
     Tag = 278 (116)    Type = long        N = 1
     The number of rows per strip.
     The default is the entire image in one strip.

StripOffsets
     Tag = 273 (111)    Type = short or long  N = strips per image
     The byte offset for each strip.

StripByteCounts
     Tag = 279 (117)    Type = long        N = 1
     The number of bytes in each strip.

SamplesPerPixel
     Tag = 277 (115)    Type = short       N = 1
     The number of samples per pixel
     (1 for monochrome data, 3 for color).

BitsPerSample
     Tag = 258 (102)    Type = short       N = SamplesPerPixel
     The number of bits per pixel.  2**BitsPerSample = # of gray levels.
```

Figure 1.2: Existing Standard TIFF Tags

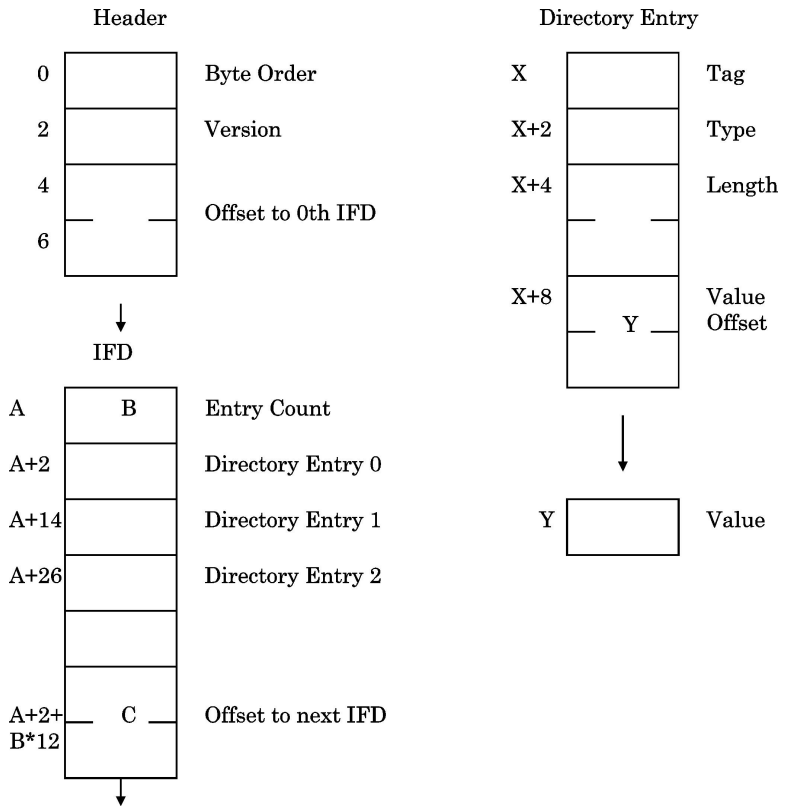


Figure 1.3: The Structure of a TIFF File

```
address          contents
(decimal)          (hex)
     header
0               49 49
2               2A 00
4               08 00 00 00

     IFD
8               1B 00
     0th directory entry
10               FF 00           tag=255
12               03 00           type=3 (short)
14               01 00 00 00     length=1
18               01 00 00 00     value=1
     1rst directory entry
22               00 01           tag=256
24               03 00           type=3 (short)
26               01 00 00 00     length=1
30               58 02 00 00     value=600
     2nd directory entry
34               01 01           tag=257
36               03 00           type=3 (short)
38               01 00 00 00     length=1
42               5A 02 00 00     value=602
     .
     .
     .
     offset to next IFD
334               00 00 00 00
     offset=0 so there are no more IFD's
```

Figure 1.4: The Beginning of a TIFF File

```
Type                Length of the Type
1 = byte            8 bit unsigned integer
2 = ASCII           8 bit bytes that store ASCII codes
                    (the last byte must be null)
3 = short           16 bit (2 byte) unsigned integer
4 = long            32 bit (4 byte) unsigned integer
5 = rational        Two long's: The first is the numerator,
                    the second is the denominator
```

Figure 1.5: Possible Data Types and Lengths

long and is stored here with a value of 1. (This value has no meaning for this tag.)

As for the next two entries, the first entry has tag=256. This is the image width of the image in number of columns. The type is short and the length of the value is one short, or two bytes. The value 600 means that there are 600 columns in the image. The second entry has tag=257. This is the image length or height in number of rows. The type is short, the length is one, and the value is 602, meaning that the image has 602 rows.

You continue through the directory entries until you reach the offset to the next IFD. If this offset is 0, as in Figure 1.4, no more IFDs follow in the file.

1.4.2 The TIFF Code

The code in Listing 1.2 reads image arrays from and writes them to TIFF files. The code works with eight-bit gray scale TIFF images. It sits one level closer to the files than the general routines given in listing 1.1.

Listing 1.4 (cips.h shown later) contains the #include files, constants and the data structures. The structure tiff_header_struct holds the essential tags we must extract from the TIFF header.

The function read_tiff_header in Listing 1.2 first determines whether the file uses LSB-first or MSB-first since the method used influences the manner in which the functions extract_long_from_buffer and extract_short_from_buffer read the remainder of the file header. Next, the offset to the Image File Directory is read. The next section seeks to the IFD and reads the entry

count, or number of entries in the IFD. Finally, the code loops over the number of entries. It reads each entry and picks out the necessary tags. The essential information is the width and length of the image, the bits per pixel (four-bit or eight-bit data), and the offset to the start of the data.

The function read_tiff_image in Listing 1.2 uses read_tiff_header and the header information to read data into an array of shorts. The code seeks to the beginning of the data and loops through the lines in the image to read all the data. The function read_line reads the image data into a buffer, and places the data into the array of shorts. read_line uses unions defined in cips.h and also depends on the number of bits per pixel.

The next functions in Listing 1.2 write TIFF files to disk. The function create_tiff_file_if_needed receives the name of an input file and an output file and looks for that output file on the disk. If the output file does not exist, it creates it to be the same basic size as the input file. create_tiff_file_if_needed uses the functions does_not_exist and create_allocate_tiff_file, both described below, to check for the existence of the output file and to create it.

The next function in Listing 1.2 is create_allocate_tiff_file. This function takes in a file name and information about the TIFF file header and creates a file on disk. It allocates disk space by writing enough zeros to the file to hold an image. The image width and length specified in the tiff_header_struct indicate how large an image the disk file must be able to hold. In writing the file header, create_allocate_tiff_file always specifies the least-significant-byte-first (LSB) order. It goes on to write all the tags required by the new TIFF [1.1] specification for gray scale image files. After writing the file header, it goes into a loop and writes out bytes of zeros to the file.

The next function in Listing 5.1 is write_tiff_image. Image processing functions will use this to write an array of pixels into existing TIFF files. It takes in the file name, looks at the file header, and uses the header information to write an array of pixels into the file. Its form is similar to that of the function read_tiff_image shown above. The function write_tiff_image seeks to where the image data begins and loops through the writing the lines.

The function write_line (shown next in Listing 1.2) actually writes the bytes into the file. It converts the short values (16 bits) to either eight- or four-bit values and writes them.

The other functions in the listing are often-used utilities. The function is_a_tiff looks at the file name and header information to determine a file is a TIFF file. The function equate_image_headers sets the primary information of two image headers to be equal. The following functions insert shorts and

longs into and extracts them from buffers. The TIFF I/O functions in this listing and the BMP file functions in listing 1.3 use these utilities.

1.5 BMP

The Microsoft Windows Bitmap (BMP) file format is a basic file format for digital images in the Microsoft Windows world. The BMP format is simpler and less capable than the TIFF format. It does what it is supposed to do — store digital images, but technically is not as good as TIFF. Simplicity, however, is a blessing in that the files are easier to read and write.

This is the native graphics format for the Windows world, so the vast majority of Windows-based software applications support this format. Since BMP was created for Microsoft Windows, it was created for the Intel processors only. Hence, it is all least significant byte first. This differs from the TIFF discussed earlier where it could be either least or most significant byte first. Microsoft's Paint program (free with all Windows) works with BMP files, so everyone using Windows can display and print BMP files. The down side of BMP is that most UNIX systems do not support BMP files.

The BMP file format has grown and changed as Microsoft Windows has grown and changed. There are five or six different versions of BMP files. The code presented herein works with version of BMP created for Windows 3.x, eight bits per pixel, gray shades, no compression.

An excellent source of information for BMP and all other image file formats is [1.2]. Further information for BMP is in [1.3] and [1.4] while source code to read and write all BMP formats is available at [1.5] and [1.6].

BMP files have (1) a file header, (2) a bit map header, (3) a color table, and (4) the image data. The file header, shown in figure 1.6, occupies the first 14 bytes of all BMP files. The first two bytes are the file type which always equals 4D42 hex or 'BM.' The next four bytes give the size of the BMP file. The next two bytes are reserved and are always zero. The last four bytes of the header give the offset to the image data. This value points to where in the file the image data begins. This value, and the other four byte values in the header, is an unsigned number (always positive).

The next 40 bytes, shown in figure 1.7, are the bit map header. These are unique to the 3.x version of BMP files. The bit map header begins with the size of the header (always 40). Next come the width and height of the image data (the numbers of columns and rows). If the height is a negative number,

File Type	0
File Size	2
Zero	6
Zero	8
Bit Map Offset	10

Figure 1.6: The BMP File Header

the image is stored bottom-up. That is the normal format for BMP files. The number of color planes is usually 1. The bits per pixel is important. My code works with eight bits per pixel only to provide 256 shades of gray.

The next two fields deal with image data compression. The compression field is 0 for no compression and 1 for run length encoding compression. My code does not work with compression. The size of bitmap field gives the size of the image data when the data is compressed. It is zero when not compressed, and the software calculates the size of the data.

The next two field deal with the resolution of the image data and the final two deal with the colors or gray shades in the image. The horizontal and vertical resolutions are expressed in pixels per meter. The color fields help the software decipher the color table discussed below. The colors field states how many colors or gray shades are in the image. The images do not always have 256 gray shades. If only 30 are present, this field equals 30 and the color table only has 30 entries. The important colors field states how many of the colors are important to the image.

After the headers come the color table. A color table is a lookup table that assigns a gray shade or color to a number given in the image data. In BMP files, just because the number 12 is in the image data does not mean that the pixel is the 12th darkest gray shade. It means that the pixel has the gray shade given in the 12th place in the color table. That gray shade might be 100, 200, or anything. Color tables offer the opportunity to save space in image files and to match the image colors to a display or printing device. They do not play an important role in the image processing routines in this text.

Header Size	0
Image Width	4
Image Height	8
Color Planes	12
Bits Per Pixel	14
Compression	16
Size of Bitmap	20
Horizontal Resolution	24
Vertical Resolution	28
Colors	32
Important Colors	36

Figure 1.7: The Bit Map Header

The BMP color table has four bytes in each color table entry. The bytes are for the blue, green, and red color values. The fourth byte is padding and is always zero. For a 256 gray shade image, the color table is 4x256 bytes long. The blue, green, and red values equal one another.

The final part of the BMP file is the image data. The data is stored row by row with padding on the end of each row. The padding ensures the image rows are multiples of four. The four, just like in the color table, makes it easier to read blocks and keep track of addresses.

1.5.1 The BMP Code

The code in Listing 1.3 reads image arrays from and writes them to BMP files. The code works with eight-bit gray scale BMP images. It sits one level closer to the files than the general routines given in listing 1.1.

Listing 1.4 (cips.h shown later) contains the #include files, constants and the data structures. The structures bmpfileheader and bitmapheader hold the information from the BMP file header and bit map header.

Listing 1.3 begins with the functions that read the essential header information from a BMP file. The function read_bmp_file_header reads the first information in the BMP file. The function read_bm_header reads the bit map header information. The function read_color_table reads the essential color table.

Next comes reading image data from file with read_bmp_image. This calls the above functions to learn the header information. It seeks to the start of the image data and reads the data one byte at a time. This function uses the color table information to convert the read byte to the proper short value. If necessary, flip_image_array is called to turn the image right-side up.

The next functions create a blank BMP file on disk. The function create_bmp_file_if_needed checks the file name given it does_not_exist. If it does not, create_bmp_file_if_needed calls create_allocate_bmp_file to create and fill the blank BMP file. It writes the necessary header information and color table before writing zeros to the file.

The function write_bmp_image writes an array of numbers into a BMP file. First, it reads the header information and seeks to the color table location. After writing the color table to file, it loops over the height and width of the array of numbers. It writes the image data to disk by filling a buffer with one row of image data at a time and writing this buffer to disk.

The remaining functions in listing 1.3 are utilities. The function is_a_bmp looks at a file name and then the information in the file header to determine if a file is a BMP image file. The function calculate_pad calculates the extra bytes padded to the end of a row of numbers to keep the four-byte boundaries in place. The function equate_bmpfileheaders sets the essential file header information equal for two headers, and flip_image_array flips the image numbers right-side up.

1.6 A Simple Program

Listing 1.5 shows how a simple program can use the I/O routines presented earlier. This listing shows the *round* program that rounds off a part of an input file and stores it to an output file. The basic use of this program is to create a new file that contains the desired part of an existing file. The user specifies the desired starting point in the existing file and the size of the output file.

The program first interprets the command line to obtain the file names and output image size. It calls the is_a routines to determine the type of file being used. The next calls create the desired output file. Calls to the previously discussed routines allocated two arrays of shorts and read the input image. The loop over out_length and out_width copy the desired part of the input image to an output image array. The final calls write the output image to the output file and free the memory used by the two image arrays.

1.7 Converting Between TIFF and BMP

The final two listings in this chapter show short programs that convert between the TIFF and BMP image file formats. Listing 1.6 shows the tif2bmp program. It checks the command line parameters to ensure the user entered the proper file names. The program obtains the size of the input TIFF image file, allocates an image array of that size, and creates an output BMP image file also of that size. It then reads the image data from the input file and writes it to the output file.

Listing 1.7 shows the bmp2tiff program. This program is similar to the tif2bmp program described above. It reads data from a BMP image file, creates a TIFF image file, reads the data from the BMP image file, and

writes it to the TIFF file.

1.8 Conclusions

This chapter has discussed image file input and output. Image I/O is a fundamental part of image processing. Images are easier to find today than ever before. The second edition of this book is based on the new image I/O routines described in this chapter. These routines allow the image processing programmer to concentrate on image processing operators. The current I/O routines work with 8-bit gray scale TIFF images and 8-bit gray scale BMP images. Inexpensive software products are available to convert almost any image to one of these formats.

1.9 References

1.1. "TIFF Revision 6.0," Final, June 3, 1993, Aldus Developers Desk, Aldus Corporation, 411 First Avenue South, Seattle, WA 98104-2871, (206) 628-6593. See also
http:www.adobe.comsupportservicedevrelationsPDFSTNTIFF6.pdf.
1.2. "Encyclopedia of Graphics File Formats," James D. Murray, William vanRyper, O'Reilly and Associates, 1996.
1.3. "The BMP File Format: Part I," David Charlap, Dr. Dobb's Journal, March 1995.
1.4. "The BMP File Format: Part II," David Charlap, Dr. Dobb's Journal, April 1995.
1.5. ftp://ftp.mv.com/pub/ddj/1995/1995.03/bmp.zip
1.6. ftp://ftp.mv.com/pub/ddj/1995/1995.04/bmp.zip

Chapter 2

Viewing and Printing Image Numbers

2.1 Introduction

Image processing is a visual task. Therefore, displaying images in various forms is a necessity. This chapter presents techniques to display the numbers in an image on a screen, print the numbers in an image, and display and print the image like a photograph.

2.2 Displaying Image Numbers

There are times when the best technique an image processor can use is to look at the raw image numbers. The numbers tell exactly what is happening. They show what the image operators are doing to the image.

The first method of looking at image numbers is to display them on the screen. The first program presented in this chapter shows the image numbers on a DOS text window on a PC. This would also work in a text window on a UNIX machine.

Listing 2.1 presents the *showi* program. It reads the image numbers from the input file and shows a portion of the numbers on the screen. The user can alter which part of the image is displayed via keystrokes. This is a short, simple program that is quite useful when trying to have a quick view of the image numbers.

2.3 Printing Image Numbers

Printing image numbers on paper gives the image processor something nothing else will — a hard copy of the image in your hand to study. The program in listing 2.2 takes the easy route by dumping all the image numbers to a text file. Common word processors can do the work of printing to paper. When the image exceeds page width (which happens almost all the time), the user can adjust font size and page orientation and resort to that tried and true technique of taping pages together.

Listing 2.2 shows the *dumpi* program. It reads the entire input image into an array and writes the numbers in each line to a buffer. The program writes each buffered line of numbers to a text file.

2.4 Viewing and Printing Images

Viewing and printing images like photographs is easier today than ever in the past. Discretion is the better part of valor, so I opted out of writing viewing and printing programs for the Windows environment. There are many excellent programs available at little or no cost. As in chapter 0, I recommend VuePrint from Hamrick Software (http://www.hamrick.com).

This approach also improves portability. Users of UNIX systems can also find free or inexpensive image viewers and printers.

Word processors are much better today than five years ago. Almost any word processor can import a file and print it alone and in a document. It seems a long time ago when that was not possible. That is why I struggled with photographing CRT displays and included those photographs in the first edition of this book. Those less than wonderful pictures are absent from this edition. The publisher, and almost anyone else, can now import the image files into a publishing package with much better results.

Use what is available in the marketplace to view and print images. This continues with the spirit of the C Image Processing System. Concentrate on image operators and leave the details of image I/O (including display) to someone else.

2.5 Conclusions

This chapter has discussed methods of viewing an image. Often, there is no substitute for looking at the raw image numbers. This chapter presented a program to see the numbers on the screen and another one that dumps the image numbers to a text file. Word processors can print those numbers to paper for examination. Viewing and printing images in visual format has been left to others. Inexpensive programs that display images on the screen are available. Today's word processors are much more powerful than five years ago. They can import and print images to paper.

Chapter 3

Halftoning

3.1 Introduction

Black and white printers put gray shade images on paper via the presence or absence of black ink on paper. Printers do not contain ink with varying shades of gray. These black and white output devices create the illusion of shades of gray via haltoning.

This chapter presents a halftoning algorithm that converts a gray scale image into an image containing only 1s and 0s. The image display sources of chapter 2 can output the resulting image.

This chapter also presents a program that dumps the 1/0 image to a text file as spaces and asterisks. A word processor can print that text file allowing the user to make a large wall display of an image.

3.2 The Halftoning Algorithm

Figure 3.1 shows the basic halftoning algorithm. Reference [3.1] is the source of the original algorithm. The basis of the algorithm is an error-diffusion technique. When the "error" reaches a certain value, turn a pixel on and reset the error. If the error is not great enough, leave the pixel turned off. Errors result from approximating a gray shade image with only ones and zeros.

Figure 3.1 shows the input image I with R rows and C columns. $E_p(m, n)$ is the sum of the errors propagated to position (m,n) due to earlier 1 or 0 assignments. $E_g(m, n)$ is the total error generated at location (m,n). $C(I, J)$

```
Define:

     I(R,C) - input image with R rows and C columns
     Ep(m,n) - sum of the errors propogated to position (m,n) due
         to prior assignments
     Eg(m,n) - the total error generated at position (m,n).
     C(i,j) - the error distribution function with I rows and J
         columns


1.  Set Ep(m,n) = Eg(m,n) = 0 for R rows and C columns

2.  loop m=1,R

     3.  loop n=1,C

         4.  Calculate the total propogated error at (m,n) due to
              prior assignments

         5.  Sum the current pixel value and the total propogated
              error:  T = I(m,n) + Ep(m,n)

         6. IF T > threshold
              THEN do steps 7. and 8.
              ELSE do steps 9. and 10.

              7.  Set pixel (m,n) on

              8.  Calculate error generated at current location
                   Eg(m,n) = T - 2*threshold

              9.  Set pixel (m,n) off

              10. Calculate error generated at current location
                   Eg(m,n) = threshold

     3.  end loop over n

2.  end loop over m
```

Figure 3.1: The Basic Halftoning Algorithm

is the error distribution function, whose size and values were set experimentally for the "best" results. The elements in C must add up to one. The authors of reference [3.1] set C to be a 2x3 matrix with the values shown in equation 3.1:

$$C_{ij} = \begin{matrix} 0.00.20.0 \\ 0.60.10.1 \end{matrix} \tag{3.1}$$

The results from using this error-diffusion method have been satisfactory. You may want to experiment with the equations. Equation 3.2 (in Figure 3.1) shows that $E_p(m,n)$, the error propagated to location (m,n), is the combination of the error distribution function C and the error generated $E_g(m,n)$.

After calculating the propagated error $E_p(m,n)$, add it to the input image $I(m,n)$ and give the sum to T. Now compare T to a threshold value, which is usually set to half the number of gray shades. For instance, for an image with 256 gray shades, set threshold = 128. If the results are unsatisfactory, experiment with different values for the threshold. If T is greater than the threshold, set the (m,n) pixel to 1 and reset the generated error $E_g(m,n)$. If T is less than the threshold, set the (m,n) pixel to 0 and set the generated error $E_g(m,n)$ to threshold.

Listing 3.1 shows the code that implements the halftoning algorithm via the function half_tone. First, the code initializes the error-distribution function in the array c. Next it sets the error arrays eg and ep to 0 (step 1 of the algorithm). The loops over m and n implement steps 2 and 3 of the algorithm. The code performs the total propagated error calculation of step 4 inside the loops over i and j. The code calculates t and then decides whether to set the pixel to 1 or 0.

3.3 Sample Output

Figure 3.2 shows a picture of a boy. Figure 3.3 shows the resulting of haltoning figure 3.2 with threshold set to 128 (half of 256). The result of halftoning is easy to recognize when compared with the input image. The true value of this 1/0 image comes in the next section because it can be printed as a poster.

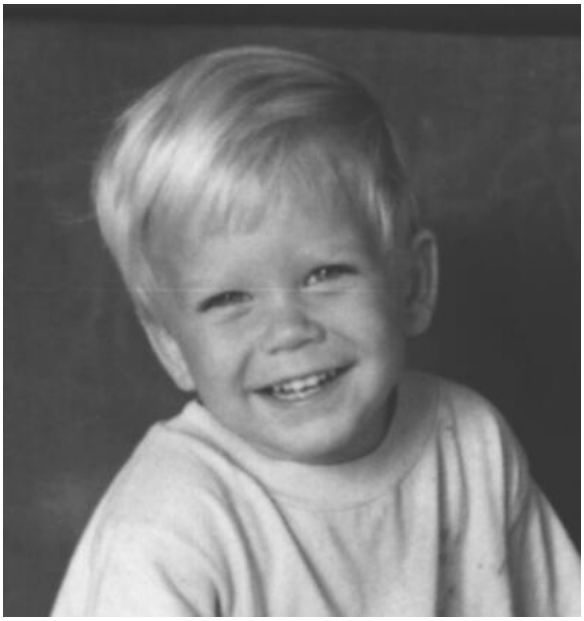


Figure 3.2: Input Boy Image

Figure 3.3: Output Halftoned Boy Image

3.4 Printing an Image

One use of halftoning and the *dumpb* program is creating a wall poster from a gray scale image. The first step is to halftone the original image as shown in figures 3.2 and 3.3. The next step is to run the *dumpb* program to create a large text file of spaces and asterisks. Finally, use a word processor to print the space-asterisk text file.

The final step takes some work and skill. The space-asterisk text file will probably be too wide to fit on a piece of printer paper. Use a word processor, set all the text to a fixed-space font (some type of courier), and use a small font (down to 2 or 3 points). That usually narrows the image to a page width. Maybe set the printer to print sideways instead of regular.

The "image" will still not be right because it will long stretched vertically. This is because the characters are taller than they are wide (the line feeds makes it look this way). Replace every two spaces with three spaces and do the same for the asterisks. Now the image "looks right." Print it. Cutting and pasting will be necessary to remove the white space from page breaks.

Figure 3.4 shows the result. This is a picture of a simple boy poster hanging on the wall. The more work and skill, the better the result. Still, this is a nice wall picture.

3.5 Conclusions

This chapter has discussed halftoning — a method for transforming a gray scale image to a black and white (1/0) image. The 1/0 image appears to have shades of gray. A program that performs this transformation was described. An interesting use of halftoning is to dump the 1/0 image to a text file and use a word processor to print a wall poster.

3.6 Reference

3.1. "Personal Computer Based Image Processing with Halftoning," John A Saghri, Hsieh S. Hou, Andrew F. Tescher, Optical Engineering, March 1986, Vol. 25, No. 3, pp. 499-504.

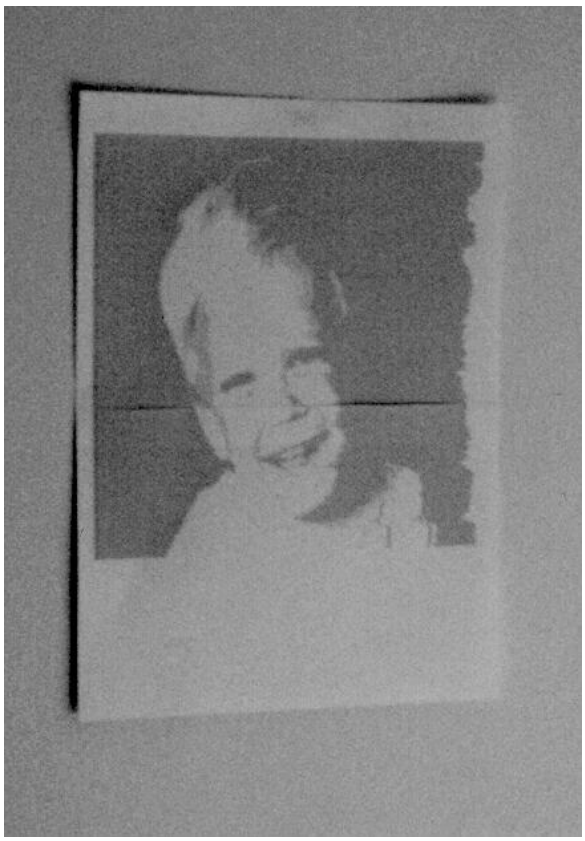


Figure 3.4: Poster Created with the dumpb Program

Chapter 4

Histograms and Equalization

4.1 Introduction

CIPS is almost ready for image processing operators except for one more "preliminary" capability — histograms and histogram equalization. This chapter shows why histogram equalization is a prerequisite to performing other image processing operations and presents source code to implement histogram equalization. It also presents a program that creates an image of an image's histogram and another program that permits combining the two images into one.

4.2 Histograms

A histogram uses a bar graph to profile the occurrences of each gray level present in an image. Figure 4.1 shows a simple histogram. The horizontal axis is the gray-level values. It begins at zero and goes to the number of gray levels (256 in this example). Each vertical bar represents the number of times the corresponding gray level occurred in the image. In Figure 4.1 the bars "peak" at about 70 and 110 indicating that these gray levels occur most frequently in the image.

Among other uses, histograms can indicate whether or not an image was scanned properly. Figure 4.2 shows a histogram of an image that was poorly scanned. The gray levels are grouped together at the dark end of the histogram. This histogram indicates poor contrast. When produced from a normal image, it indicates improper scanning. The scanned image will look

```
              *        *
              *        *               *
   *          **      **      *        *
   *    *     **  *   ***     *        * *      *
   *    * * ***  *   ***     *  *     * * *    *
   *** ** ****** *  ****    *****    *******  **
 * ****** ******** *****   ******* ********* ***
 * ********************** **********************
************************************************
-------------------------------------------------
0        50        100       150       200       255
```

Figure 4.1: Simple Histogram

like a TV picture with the brightness and contrast turned down. (Of course, the same histogram could indicate proper scanning for certain unusual images, such as a black bear at night).

Histograms also help select thresholds for object detection (an object being a house, road, or person). Objects in an image tend to have similar gray levels. For example, in an image of a brick house, all the bricks will usually have similar gray levels. All the roof shingles will share similar gray levels, but differ from the bricks. In Figure 4.1, for example, the valleys between the peaks at about 60 and 190 might indicate that the image contains three major kinds of objects perhaps bricks, roof, and a small patch of sky. Practical object identification is never simply a matter of locating histogram peaks, but histograms have been important to much of the research in object identification.

Figure 4.3 shows the an image with its histogram. The gray levels in the histogram reach across most of the scale, indicating that this image was scanned with good contrast. Figure 4.4 shows a house image with its histogram. Again, the histogram stretches across much of the scale indicating good scanning and contrast.

Because the dark objects and the bright objects in an image with poor

```
      **   ** ***
      **   ** ***
     ***   ** ***
     ***  *** ***
  *  *** **** ***
  * ********* ***
 ****************
----------------------------------------------------
0         50         100        150        200        255
```

Figure 4.2: Histogram of a Poorly Scanned Image

contrast have almost the same gray level, the gray shades from such an image will be grouped too closely together (Figure 4.2). Frequently the human eye will have difficulty distinguishing objects in such an image. Image processing operators will have even less success.

4.3 Histogram Equalization

Figure 4.5 shows an image with poor contrast (a poorly scanned aerial photograph). The rectangles in the center of the picture are house trailers. The areas around the house trailers are roads, parking lots, and lawns. The histogram in the lower right-hand corner shows that the gray levels are grouped in the dark half of the scale. There are trees and bushes in the lawn areas of the image. You cannot see them, however, because their gray levels are too close to the gray levels of the grass.

The cure for low contrast images is "histogram equalization." Equalization causes a histogram with a mountain grouped closely together to "spread out" into a flat or equalized histogram. Spreading or flattening the histogram makes the dark pixels appear darker and the light pixels appear lighter. The

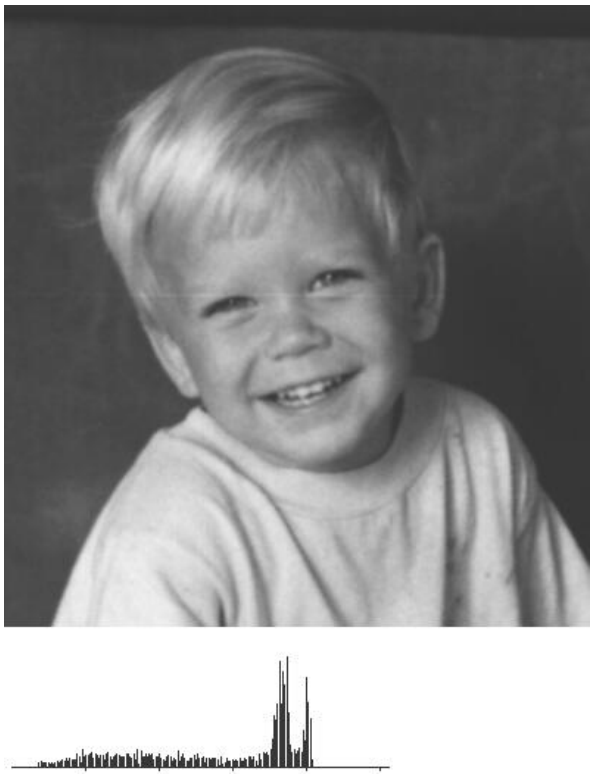


Figure 4.3: Boy Image with Histogram

Figure 4.4: House Image with Histogram

key word is "appear." The dark pixels in Figure 4.5 cannot be any darker. If, however, the pixels that are only slightly lighter become much lighter, then the dark pixels will appear darker. Please note that histogram equalization does not operate on the histogram itself. Rather, histogram equalization uses the results of one histogram to transform the original image into an image that will have an equalized histogram.

The histogram equalization algorithm may make more sense by outlining the derivation of the underlying mathematical transform. (The full derivation is found in Reference [4.1].) Equation 4.1 represents the equalization operation, where c is an image with a poor histogram. The as yet unknown function f transforms the image c into an image b with a flat histogram.

$$b(x, y) = f[c(x, y)] \tag{4.1}$$

Equation 4.2 shows the probability-density function of a pixel value a. $p_1(a)$ is the probability of finding a pixel with the value a in the image. $Area_1$ is the area or number of pixels in the image and $H_1(a)$ is the histogram of the image.

$$p_1(a) = \frac{1}{Area_1} H_1(a) \tag{4.2}$$

Figure 4.5: Image with Poor Contrast

For example, if
$a = 100$
$Area_1 = 10,000$
$H_1(100) = 10$
then
$p_1(100) = 10/10,000 = 0.001$

Equation 4.3 shows the cumulative-density function (cdf) for the pixel value a. The cdf is the sum of all the probability density functions up to the value a.

$$P_1(a) = \frac{1}{Area_1} \sum_{i=0}^{a} H_1(a) \tag{4.3}$$

For example,
$P_1(10) = 1/10,000 * [H(0) + H(1) + ... + H(10)]$

Equation 4.4 shows the form of the desired histogram equalization function $f(a)$. $H_c(a)$ is the histogram of the original image c (the image with the poor histogram). D_m is the number of gray levels in the new image b. $D_m = 1/p(a)$ for all pixel values a in the image b. Note that the image b has a "flat" histogram
$H(0) = H(1) = H(2) = ...$
because the probability of each pixel value is now equal — they all occur the same number of times. So $f(a)$ simply takes the probability density function for the values in image b and multiplies this by the cumulative density function of the values in image c. It is important to realize that histogram equalization reduces the number of gray levels in the image. This seems to be a loss, but it is not.

$$f(a) = D_m \frac{1}{Area_1} \sum_{i=0}^{a} H_c(a) \tag{4.4}$$

The algorithm for performing histogram equalization (see Figure 4.6) is simpler than the equations.

4.4 Equalization Results

Figures 4.7, 4.8, and 4.9 show the result of histogram equalization. The appearance of some images improves after histogram equalization while it

```
1.  calculate histogram
     loop over i ROWS of input image
         loop over j COLS of input image
              k = input_image[i][j]
              hist[k] = hist[k] + 1
         end loop over j
     end loop over i

2.  calculate the sum of hist
     loop over i gray levels
         sum = sum + hist[i]
         sum_of_hist[i] = sum
     end loop over i

3.  transform input image to output image
     area = area of image (ROWS x COLS)
     Dm = number of gray levels in output image
     loop over i ROWS
         loop over j COLS
              k = input_image[i][j]
              out_image[i][j] = (Dm/area) x sum_of_hist[k]
         end loop over j
     end loop over i
```

Figure 4.6: Histogram Equalization Algorithm

degrades with other images. For Figure 4.7 (an equalized version of Figure 4.6) the appearance improves. Note the equalized histogram.

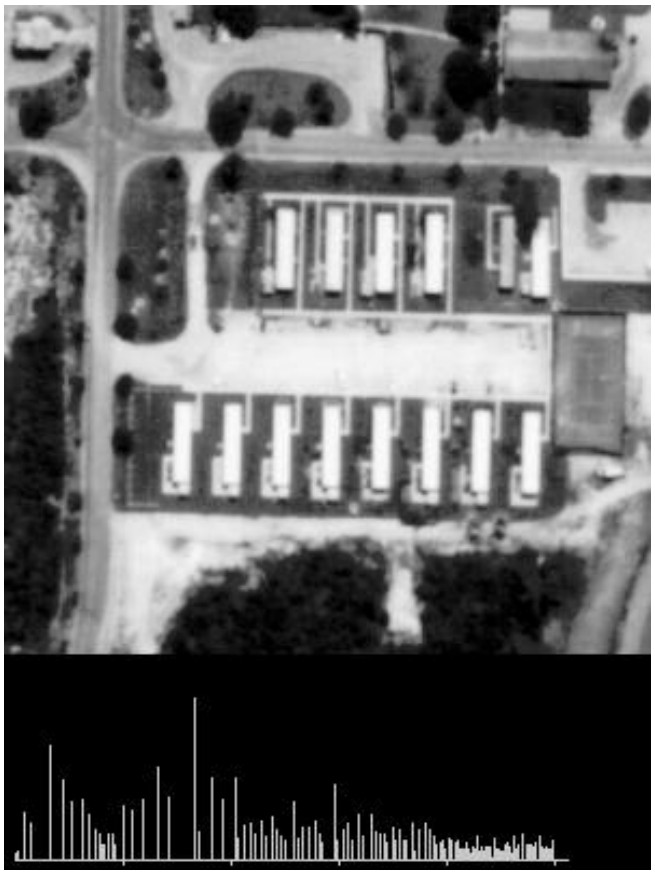


Figure 4.7: Equalized Version of Figure 4.5

The aerial photograph, although fuzzy, has much improved contrast. The dark spots in the lawn areas are trees. If you look closely at Figure 4.7 you may be able to see these trees. Figure 4.8 shows details from these two images together with their histograms. The unequalized image on the left of Figure 4.8 is dark. In the equalized image on the right of Figure 4.8 you can distinguish the trees and bushes from the grass.

With some photographs the equalized image may appear worse than the original image. In a properly scanned image, for example, histogram equalization can introduce "noise" into what were uniform areas of an image. Such "noise" may not be undesirable — in many cases it reflects subtle texture or

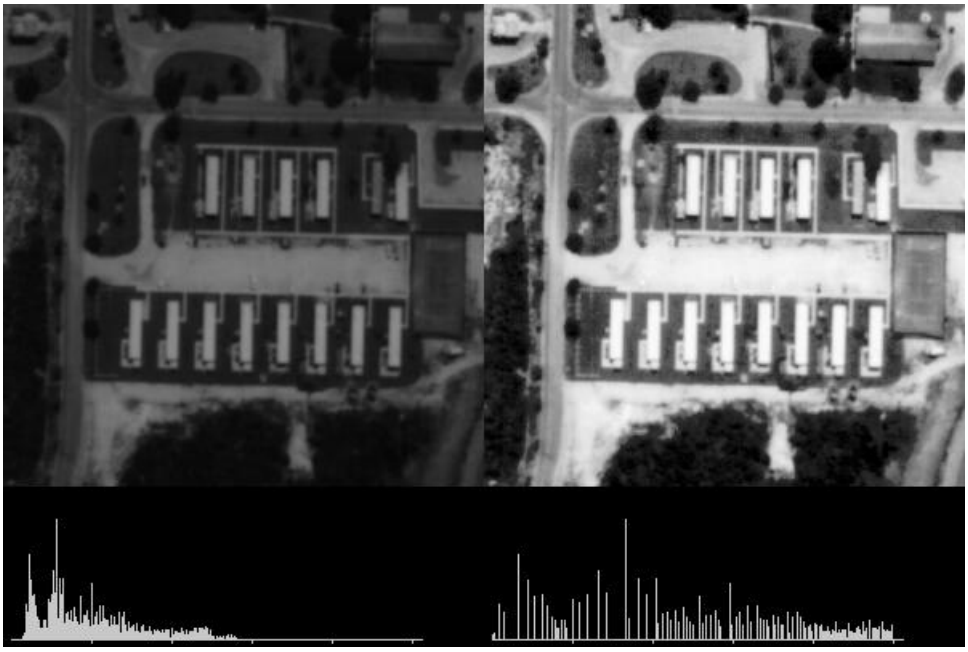


Figure 4.8: Comparing Figures 4.6 and 4.7

detail lost in the more "natural" image.

Figure 4.9 (segments from the house in Figure 4.4) shows how equalization affects a properly scanned image. The histogram for the unequalized image on the top stretches across much of the scale. The bricks, windows, and trees are easy to see. However, in the equalized image on the bottom, the window and the brick appear too bright. While the equalized image does not appear as pleasant, it does have better contrast. The darks appear darker and the lights appear lighter. In this case, however, they are probably too dark and too light.

Since variations in scanning can significantly affect the results of image processing operators, histogram equalization is a prerequisite for further image processing. If you scan an image too bright or too dark, you can remove objects from an image. The result may "improve" the apparent performance of processing and lead you to overestimate the effectiveness of an operator. Consistently pre-processing images with a histogram equalization operation will ensure consistency in all results.

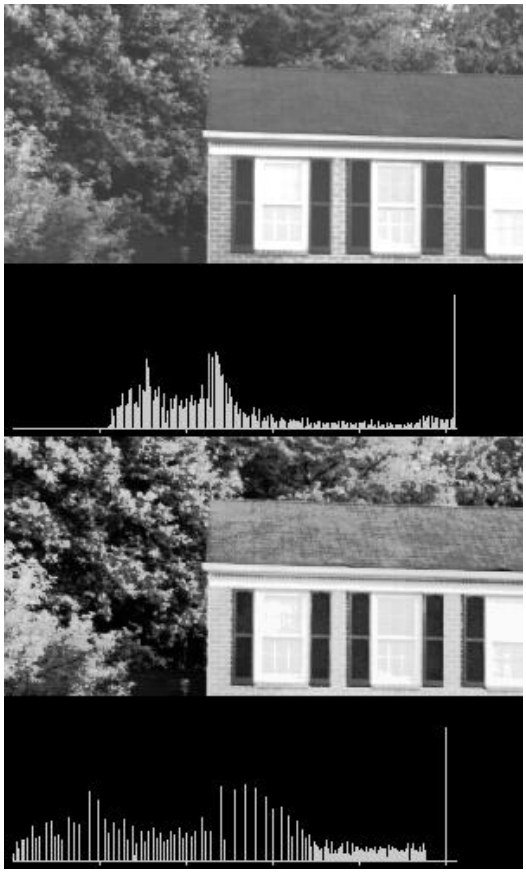


Figure 4.9: Equalizing a Properly Scanned Image

4.5 Implementation

The code in Listing 4.1 implements the histogram operations. The basic data structure is a 256-element array of unsigned longs. The function zero_histogram zeros or clears the histogram array. The function calculate_histogram creates the histogram for an image array. It loops through the image array, takes the pixel value, and increments that element in the histogram array.

The function perform_histogram_equalization implements the algorithm shown in Figure 4.6. The first loop over i calculates the sum_of_h array. The loops over length and width transforms each pixel in the image to a new value using the number of gray levels in the new image, the area of the image, and the sum_of_h array.

The code in Listing 4.2 is the main routine of the *histeq* program. It calls the histogram routines of Listing 4.1 to transform an image by equalizing its histogram. This program produced Figure 4.7 and part of Figure 4.9. Its structure is like the *halftone* program of chapter 3 and the vast majority of programs in this book. It checks the command line, creates arrays, reads input, call the histogram functions, and writes output.

Listing 4.3 shows the source code for the *himage* program. This program reads an image, calculates its histogram, and draws that histogram to another image file. This program produced the histograms that appeared with the images in this chapter. The opening structure of *himage* is similar to *histeq* as it reads the input file and calculates its histogram. The remainder of *himage* draws an axis and tick marks in the output file and draws lines to represent the histogram levels. The code scales the lines to keep them in the image. The user can specify the size of the histogram image or use the default values of L and W. The user must specify a width of more than 256 (the number of gray shades). It is best to have the histogram image the same width as the original image. That allows you to combine the two image with the side program described next.

4.6 The *side* Program

A bonus in this chapter is the *side* program. It combines to images into one image by either pasting them side by side or on top of one another. The *side* program created the images shown in the chapter where the image and its

histogram are shown together. The *himage* program created the image of the histogram, and the *side* program pasted them together. Listing 4.4 show the *side* program. A key to the program is that the two images to be pasted together must have the correct dimensions. If they are to be combined side by side, they must have the same height. If they are to be combined top to bottom, they must have the same width. The program simply reads the two input images, combines them in a large output image array, and writes the result to a large output file. The *side* program is very useful when you want to show two images at once to highlight differences and illustrate how an operator alters an image.

4.7 Conclusions

This chapter has discussed histograms and histogram equalization. Histograms show the occurrences of gray shades in an image as a bar graph. Histogram equalization adjusts the contrast in an image by spreading its histogram. This often improves the appearance of an image. Also presented in this chapter is a program that calculates an images histogram and stores that as a picture in an image file. Another program presented pastes images side by side.

4.8 Reference

4.1 "Digital Image Processing," Kenneth R. Castleman, Prentice-Hall, 1979.

Chapter 5

Basic Edge Detection

5.1 Introduction

Edge detection is one of the fundamental operations in image processing. Many other operations are based on edge detection and much has been written about this subject. This chapter will discuss the subject and explore some basic techniques. The next chapter will discuss some advanced forms of edge detection.

5.2 Edge Detection

This chapter introduces edge detection and shows some basic edge detectors. The next chapter continues with some advanced edge detectors. Detecting edges is a basic operation in image processing. The edges of items in an image hold much of the information in the image. The edges tell you where items are, their size, shape, and something about their texture.

The top part of Figure 5.1 shows the side view of an ideal edge. An edge is where the gray level of the image moves from an area of low values to high values or vice versa. The edge itself is at the center of this transition. An edge detector should return an image with gray levels like those shown in the lower part of Figure 5.1. The detected edge gives a bright spot at the edge and dark areas everywhere else. Calculus fans will note the detected edge is the derivative of the edge. This means it is the slope or rate of change of the gray levels in the edge. The slope of the edge is always positive or zero, and it reaches its maximum at the edge. For this reason, edge detection is often

called image differentiation.

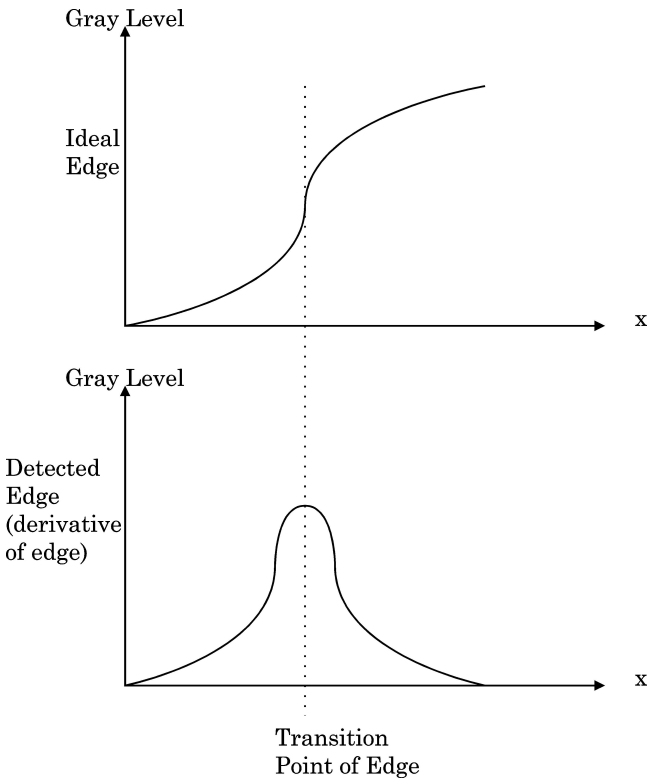


Figure 5.1: Graphs of Gray Scale Values at Edges

The problem in edge detection is how to calculate the derivative (the slope) of an image in all directions? Convolution of the image with masks is the most often used technique of doing this. An article by Wesley Faler in *The C Users Journal* discussed this technique [5.1]. The idea is to take a 3 x 3 array of numbers and multiply it point by point with a 3 x 3 section of the image. You sum the products and place the result in the center point of the image.

The question in this operation is how to choose the 3 x 3 mask. Faler used several masks shown in Figure 5.2. These are basic masks that amplify the slope of the edge. Take the simple one-dimensional case shown in Figure 5.1. Look at the points on the ideal edge near the edge. They could have values such as [3 5 7]. The slope through these three points is $(7 - 3)/2 = 2$. Convolving these three points with [-1 0 1] produces -3 + 7 = 4. The convolution amplified the slope, and the result is a large number at the

```
-1  0  1        1  1  1      -1 -1 -1       0  1  0
-1  0  1        0  0  0      -1  8 -1      -1  0  1
-1  0  1       -1 -1 -1      -1 -1 -1       0 -1  0
```

Figure 5.2: Masks Used by Faler for Edge Detection

transition point in the edge. Convolving [-1 0 1] with a line performs a type of differentiation or edge detection.

The number of masks used for edge detection is almost limitless. Researchers have used different techniques to derive masks and then experimented with them to discover more masks. Figure 5.3 shows four masks used in the source code and examples in this chapter. The first three masks are the Kirsch, Prewitt, and Sobel masks as given in Levine's text [5.2] (there are different masks bearing the same name in the literature) [5.3]. The fourth mask, the "quick" mask, is one I "created" while working on this process (there is no doubt that someone else created this mask before me).

The Kirsch, Prewitt, and Sobel masks are "compass gradient" or directional edge detectors. This means that each of the eight masks detects an edge in one direction. Given a pixel, there are eight directions to travel to a neighboring pixel (above, below, left, right, upper left, upper right, lower left, and lower right). Therefore, there are eight possible directions for an edge. The directional edge detectors can detect an edge in only one of the eight directions. To detect only left to right edges, use only one of the eight masks. To detect all of the edges, perform convolution over an image eight times using each of the eight masks. The "quick mask" is so named because it can detect edges in all eight directions in one convolution. This has obvious speed advantages. There are, however, occasions when detecting one type of edge is desired, so use a directional mask for those occasions.

There are two basic principles for each edge detector mask. The first is that the numbers in the mask sum to zero. If a 3 x 3 area of an image contains a constant value (such as all ones), then there are no edges in that area. The result of convolving that area with a mask should be zero. If the numbers in the mask sum to zero, convolving the mask with a constant area will result in the correct answer of zero. The second basic principle is that the mask should approximate differentiation or amplify the slope of the edge. The simple example [-1 0 1] given earlier showed how to amplify the slope of the edge. The first Kirsch, Prewitt, and Sobel masks use this idea to amplify

```
Kirsch          Prewitt         Sobel           Quick mask
 5  5  5         1  1  1         1  2  1        -1  0 -1
-3  0 -3         1 -2  1         0  0  0         0  4  0
-3 -3 -3        -1 -1 -1        -1 -2 -1        -1  0 -1

-3  5  5         1  1  1         2  1  0
-3  0  5         1 -2 -1         1  0 -1
-3 -3 -3         1 -1 -1         0 -2 -2

-3 -3  5         1  1 -1         1  0 -1
-3  0  5         1 -2 -1         2  0 -2
-3 -3  5         1  1 -1         1  0 -1

-3 -3 -3         1 -1 -1         0 -1 -2
-3  0  5         1 -2 -1         1  0 -1
-3  5  5         1  1  1         2  1  0

-3 -3 -3        -1 -1 -1        -1 -2 -1
-3  0 -3         1 -2  1         0  0  0
 5  5  5         1  1  1         1  2  1

-3 -3 -3        -1 -1  1        -2 -1  0
 5  0 -3        -1 -2  1        -1  0  1
 5  5 -3         1  1  1         0  1  2

 5 -3 -3        -1  1  1        -1  0  1
 5  0 -3        -1 -2  1        -2  0  2
 5 -3 -3        -1  1  1        -1  0  1

 5  5 -3         1  1  1         0  1  2
 5  0 -3        -1 -2  1        -1  0  1
-3 -3 -3        -1 -1  1        -2 -1  0
```

Figure 5.3: Masks for Edge Detection

an edge ramping up from the bottom of an image area to the top.

5.3 Implementing Edge Detectors

Listing 5.1 shows source code that will implement the four edge detectors shown in Figure 5.3. The first section of code declares the masks shown in Figure 5.3. The functions detect_edges and perform_convolution implement the Kirsch, Prewitt, and Sobel edge detectors. The detect_edges function calls perform_convolution to detect the edges. Next, it "fixes" the edges of the output image (more on this later) and writes it to the output image file.

The function perform_convolution does the convolution operation eight times (once for each direction) to detect all the edges. First, it calls setup_masks to copy the correct masks. The parameter detect_type determines which masks to use. The convention is type 1=Prewitt, 2=Kirsch, and 3=Sobel. The function perform_convolution clears the output image, sets several maximum values, and does the convolution eight times over the entire image array. At each point, the code checks to see if the result of convolution is greater than the maximum allowable value or less than zero, and corrects for these cases.

After convolution, there is the option of thresholding the output of edge detection. Edge detectors produce results that vary from zero to the maximum gray level value. This variation shows the strength of an edge. An edge that changes from 10 to 200 will be stronger than one that changes from 10 to 50. The output of convolution will indicate this strength. It is often desirable to threshold the output so strong edges will appear relatively bright (or dark) and weak edges will not appear at all. This lowers the amount of noise in the edge detector output and yields a better picture of the edges. The detect_edges and perform_convolution functions pass a threshold parameter. If threshold == 1, perform_convolution goes through the output image and sets any pixel above the high parameter to the maximum and any pixel below the high parameter to zero.

The quick_edge function performs edge detection using the single 3 x 3 quick_mask. It performs convolution over the image array using the quick_mask. It thresholds the output image if requested, and fixes the edges of the output image. All these operations are the same as in the detect_edges and perform_convolution functions.

Several short utility functions make up the remainder of Listing 5.1. The

setup_masks function copies the desired type of mask (Kirsch, Prewitt, or Sobel) into the mask arrays for the perform_convolution function. The fix_edges function corrects the output image after convolution (fix_edges is shown in this listing, but resides in source code file utility.c). Convolving a 3 x 3 mask over an image array does not process the pixels along on the outer edge of the image. The result is a blank line around the image array. The fix_edges function goes around the edge of the image array and copies valid values out to the edge. This removes the distracting lines.

These edge detectors are called by the main routine of the *medge* program. The *medge* program ties these and the edge detectors described in the next chapter into one convenient program. That program is presented in the next chapter.

5.4 Results

Let's close with examples of the edge detectors in action. Figure 5.4 shows a house image. Figure 5.5 shows the result of applying the Kirsch edge detector masks. Figure 5.6 shows the result of the Prewitt masks and Figure 5.7 shows the result of the Sobel masks. Figures 5.5, 5.6, and 5.7 are outputs that were thresholded. Edge values above a threshold of 33 were set to 255 and all others were set to zero. This gives a clear picture of edges and non-edges. Figure 5.8 shows the result of applying the Sobel masks and not thresholding the result. If you look closely, you can see some variations in gray level indicating some edges are stronger than others. Figure 5.9 shows the result of applying the quick mask. The results of the quick mask are as good as the other masks, and it operates in one-eighth the time.

5.5 Conclusion

This chapter discussed basic edge detection. The next chapter continues the discussion of edge detection. There are many creative methods of detecting edges in images. The next chapter discusses the homogeneity operator, the difference operator, contrast-based edge detection, and edge filtering by varying the size of the convolution mask.

Figure 5.4: The House Image

Figure 5.5: The Result of the Kirsch Masks

Figure 5.6: The Result of the Prewitt Masks

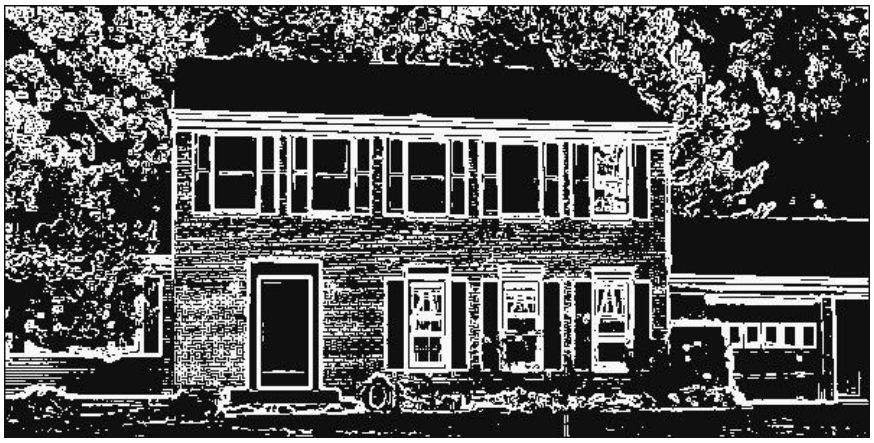


Figure 5.7: The Result of the Sobel Masks

Figure 5.8: The Result of the Sobel Masks Without Thresholding

Figure 5.9: The Result of the Quick Mask

5.6 References

5.1 "Image Manipulation By Convolution," Wesley Faler, The C Users Journal, Vol. 8, No. 8, August 1990, pp. 95-99.
5.2 "Vision in Man and Machine," Martin D. Levine, McGraw-Hill, 1985.
5.3 "Digital Image Processing," Kenneth R. Castleman, Prentice-Hall, 1979.

Chapter 6

Advanced Edge Detection

6.1 Introduction

There are many different methods of edge detection. Chapter 5 discussed some basic techniques. This chapter discusses some unusual and advanced ideas and presents four edge detectors. The first two do not use the convolution operation — they use only subtraction. The third edge detector can vary the level of detail of the edges it will detect. The fourth edge detector will detect edges in unevenly lit images. Finally, an edge detector is used to enhance the appearance of an original image. Figure 6.1 shows the original image used by all the operators.

Figure 6.1: Original House Image

6.2 Homogeneity Operator

The first edge detector is the homogeneity operator [6.1] which uses subtraction to find an edge. Figure 6.2 shows an example of this operator. The operator subtracts each of the pixels next to the center of a 3x3 area from the center pixel. The result is the maximum of the absolute value of these subtractions. Subtraction in a homogeneous region (one that is a solid gray level) produces zero and indicates an absence of edges. A region containing sharp edges, such as area 2 of Figure 6.2, has a large maximum.

The first section of Listing 6.1 shows the homogeneity function. This function is similar in form to the edge detectors discussed in Chapter 5. In the loop over rows and cols, the code performs the subtraction and finds the maximum absolute value of the subtractions. The homogeneity operator requires thresholding (which you can specify). A perfectly homogeneous 3x3 area is rare in an image. If you do not threshold, the result looks like a faded copy of the original. Thresholding at 30 to 50 for a 256 gray level image gives good results.

Figure 6.3 shows the result of the homogeneity operator. This operator gives a good rendition of the edges in the original house image. This is a quick operator that performs only subtraction — eight operations per pixel — and no multiplication.

6.3 Difference Operator

The next edge detector is the difference operator, another simple operator that uses subtraction. Recall that edge detection is often called image differentiation (detecting the slope of the gray levels in the image). The difference operator performs differentiation by calculating the differences between the pixels that surround the center of a 3x3 area.

Figure 6.4 shows an example of the difference operator. The difference operator finds the absolute value of the differences between opposite pixels, the upper left minus lower right, upper right minus lower left, left minus right, and top minus bottom. The result is the maximum absolute value. The results shown in Figure 6.4 are similar but not exactly equal to those from the homogeneity operator in Figure 6.2.

The second part of Listing 6.1 shows the difference_edge function, which is similar to the homogeneity function. The difference_edge function loops

```
Area 1:

      1  2  3
      4  5  6
      7  8  9

     Output of homogeneity edge detector is:
     max of {
          | 5 - 1 |   | 5 - 2 |   | 5 - 3 |
          | 5 - 4 |   | 5 - 6 |   | 5 - 7 |
          | 5 - 8 |   | 5 - 9 |
     } = 4


Area 2:

     10 10 10
     10 10 10
     10 10  1

     Output of homogeneity edge detector is:
     max of {
          | 10 - 10 |   | 10 - 10 |   | 10 - 10 |
          | 10 - 10 |   | 10 - 10 |   | 10 - 10 |
          | 10 - 10 |   | 10 - 1  |
     } = 9


Area 3:

     10  5  3
     10  5  3
     10  5  3

     Output of homogeneity edge detector is:
     max of {
          | 5 - 10|   | 5 - 5 |   | 5 - 3 |
          | 5 - 10|   | 5 - 3 |   | 5 - 10|
          | 5 - 5 |   | 5 - 3 |
     } = 5
```

Figure 6.2: An Example of the Homogeneity Operator

Figure 6.3: Result of Homogeneity Edge Detector

over the input image array and calculates the absolute values of the four differences. As in the homogeneity case, the difference operator requires thresholding.

Figure 6.5 shows the result of the difference edge detector. This result is similar to that shown in Figure 6.3. The difference edge detector did detect more of the brick and mortar lines than the homogeneity operator. The choice between the two edge detectors depending on the desired detail. The difference operator is faster than the homogeneity operator. The difference operator uses only four integer subtractions per pixel, while the homogeneity operator uses eight subtractions per pixel. These compare to the nine multiplications and additions for the convolution-based edge detectors discussed in Chapter 5.

6.4 Difference of Gaussians

The next operator to examine is the difference of Gaussians edge detector, which allows varying the width of a convolution mask and adjusting the detail in the output [6.2, 6.3]. The results in Figures 6.3 and 6.5 are good. Suppose, however, we wanted to detect only the edges of the large objects in the house image (door, windows, etc.) and not detect the small objects (bricks, leaves, etc.).

Varying the width of convolution masks eliminates details. If a mask is wide, convolving an image will smooth out details, much like averaging.

```
Area 1:

      1  2  3
      4  5  6
      7  8  9

     Output of difference edge detector is:
     max of {
          | 1 - 9 |   | 7 - 3 |
          | 4 - 6 |   | 2 - 8 |
     } = 8


Area 2:

     10 10 10
     10 10 10
     10 10  1

     Output of difference edge detector is:
     max of {
          | 10 - 1  |   | 10 - 10 |
          | 10 - 10 |   | 10 - 10 |
     } = 9


Area 3:

     10  5  3
     10  5  3
     10  5  3

     Output of difference edge detector is:
     max of {
          | 10 - 3|   | 10 - 3|
          | 10 - 3|   | 5 - 5 |
     } = 7
```

Figure 6.4: An Example of the Difference Operator

Figure 6.5: Result of Difference Edge Detector

Stock market prices vary greatly by the minute. The variations lesson when the prices are examined hourly. Examining the prices weekly causes the variations to disappear and general trends to appear. Convolving an image with a wide, constant mask, smoothes the image. Narrower, varying masks, permit the details to remain.

Figure 6.6 shows two example masks. These masks are "difference of Gaussians" or "Mexican hat" functions. The center of the masks is a positive peak (16 in the 7x7 masks — 19 in the 9x9 mask). The masks slope downward to a small negative peak (-3 in both masks) and back up to zero. The curve in the 9x9 mask is wider than that in the 3x3 mask. Notice how the 9x9 mask hits its negative peak three pixels away from the center while the 7x7 masks hits its peak two pixels away from the center. Also, notice these masks use integer values. Most edge detectors of this type use floating point numbers that peak at +1. Using integers greatly increases the speed.

Figure 6.7 illustrates how the narrower mask will detect small edges the wide mask misses. Each area in Figure 6.7 has a small pattern similar to the brick and mortar pattern in the house image. This pattern has small objects (bricks) with many edges. Convolving the 7x7 mask in Figure 6.6 with the 7x7 area in Figure 6.7, results in a +40; the 7x7 mask detected an edge at the center of the 7x7 area. Doing the same with the 9x9 mask in Figure 6.6 with the 9x9 area in Figure 6.7, produces a -20; the 9x9 mask did not detect any edges. The "hat" in the 9x9 mask was wide enough to smooth out the edges and not detect them.

```
7x7 mask
        0    0   -1   -1   -1    0    0
        0   -2   -3   -3   -3   -2    0
       -1   -3    5    5    5   -3   -1
       -1   -3    5   16    5   -3   -1
       -1   -3    5    5    5   -3   -1
        0   -2   -3   -3   -3   -2    0
        0    0   -1   -1   -1    0    0
```

```
9x9 mask
      0    0    0   -1   -1   -1    0    0    0
      0   -2   -3   -3   -3   -3   -3   -2    0
      0   -3   -2   -1   -1   -1   -2   -3    0
     -1   -3   -1    9    9    9   -1   -3   -1
     -1   -3   -1    9   19    9   -1   -3   -1
     -1   -3   -1    9    9    9   -1   -3   -1
      0   -3   -2   -1   -1   -1   -2   -3    0
      0   -2   -3   -3   -3   -3   -3   -2    0
      0    0    0   -1   -1   -1    0    0    0
```

Figure 6.6: Gaussian “Mexican Hat” Masks

```
7x7 area with lines

  0 10  0 10  0 10  0
  0  0  0  0  0  0  0
 10  0 10  0 10  0 10
  0  0  0  0  0  0  0
  0 10  0 10  0 10  0
  0  0  0  0  0  0  0
 10  0 10  0 10  0 10

result of convolution with 7x7 mask = 40



9x9 area with lines
  0  0  0  0  0  0  0  0  0
 10  0 10  0 10  0 10  0 10
  0  0  0  0  0  0  0  0  0
  0 10  0 10  0 10  0 10  0
  0  0  0  0  0  0  0  0  0
 10  0 10  0 10  0 10  0 10
  0  0  0  0  0  0  0  0  0
  0 10  0 10  0 10  0 10  0
  0  0  0  0  0  0  0  0  0

result of convolution with 9x9 mask = -20
```

Figure 6.7: Detecting Small Edges

The first section of Listing 6.1 shows the two Gaussian masks and the function gaussian_edge. gaussian_edge has the same form as the other edge detectors. An additional size parameter (either 7 or 9) specifies mask width. The inner loop over a and b varies with this parameter. The processing portion is the same as the other convolution mask edge detectors presented in Chapter 5. With gaussian_edge, thresholding can produce a clear edge image or leave it off to show the strength of the edges.

Figure 6.8 shows the result of edge detection using the narrower 7x7 mask and Figure 6.9 shows the result of the wider 9x9 mask. The narrower mask (Figure 6.8) detected all the edges of the bricks, roof shingles, and leaves. The wider mask (Figure 6.9) did not detect the edges of small objects, only edges of the larger objects. Figure 6.8 might be too cluttered, so use the wider mask. If fine detail is desired, the narrower mask is the one to use.

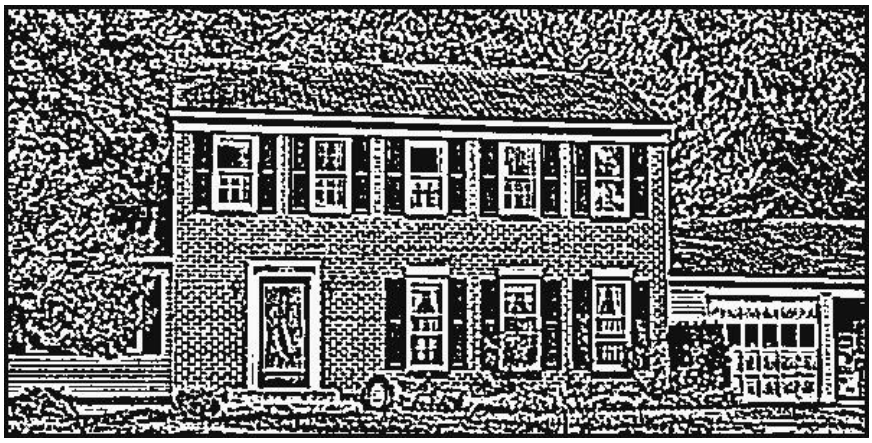


Figure 6.8: Result of Gaussian Edge Detector with 7x7 Mask

6.5 More Differences

The other edge detectors presented so far can detect edges on different size objects. The homogeneity operator can take the difference of the center pixel and a pixel that is two or three pixels away. The difference edge detector can take the difference of opposite pixels in a 5x5 or 7x7 area instead of a 3x3 area. The quick mask in Chapter 5 can change from 3x3 to 5x5 with the center value equal to 4 and the four corners equal to -1. Try these changes as an exercise.

Figure 6.9: Result of Gaussian Edge Detector with 9x9 Mask

6.6 Contrast-based Edge Detector

One problem with detecting edges involves uneven lighting in images. The contrast-based edge detector [6.4] helps take care of this problem. In well lit areas of an image the edges have large differences in gray levels. If the same edge occurs in a poorly lit area of the image, the difference in gray levels is much smaller. Most edge detectors result in a strong edge in the well lit area and a weak edge in the poorly lit area.

The contrast-based edge detector takes the result of any edge detector and divides it by the average value of the area. This division removes the effect of uneven lighting in the image. The average value of an area is available by convolving the area with a mask containing all ones and dividing by the size of the area.

Figure 6.10 illustrates the contrast-based edge detector. Almost any edge detector can be the basis for this operation. Figure 6.10 uses the quick edge detector from Chapter 5. The edge in the well lit area is an obvious and strong edge. Convolving the quick mask with this area yields 100. The edge in the poorly lit area is easy to see, but convolving with the quick mask results in 10, a weak edge that thresholding would probably eliminate. This distinction should be avoided. The well lit and poorly lit edges are very similar; both change from one gray level to another gray level that is twice as bright.

Dividing by the average gray level in the area corrects this discrepancy. Figure 6.10 shows the smoothing mask that calculates the average gray level.

```
Edge Detector Mask
 -1  0 -1
  0  4  0
 -1  0 -1

Edge in well lit area

 50 100 100
 50 100 100             convolution with edge mask yields:
 50 100 100             400 - 50 - 50 - 100 - 100 = 100

Edge in poorly lit area

  5  10  10
  5  10  10             convolution with edge mask yields:
  5  10  10             40 - 5 - 5 - 10 - 10 = 10


Smoothing mask

         1  1  1
  1/9 *  1  1  1
         1  1  1

convolution of smoothing mask with edge in well lit area yields:
50+50+50+100+100+100+100+100+100 / 9 = 750/9 = 83

convolution of smoothing mask with edge in poorly lit area yields:
5+5+5+10+10+10+10+10+10 / 9 = 75/9 = 8

dividing original convolution by the smoothing mask result:

edge in well lit area: 100 / 83 = 1

edge in poorly lit area: 10 / 8 = 1
```

Figure 6.10: Contrast-Based Edge Detector

Convolving the well lit area yields an average value of 83. Convolving the poorly lit area yields an average value of eight. Dividing (integer division) the strong edge in the well lit area by 83 yields one. Dividing the weak edge by eight also gives a result of one. The two edges from unevenly lit areas yield the same answer and you have consistent edge detection.

The next section of Listing 6.1 shows the contrast_edge function that follows the same steps as the other edge detector functions. The difference is in the processing loop over a and b, which calculates two convolutions: sum_n (the numerator or quick edge output) and sum_d (the smoothing output). After the loops over a and b, divide sum_d by nine and divide sum_n by this result. The e_mask at the beginning of Listing 6.1 replaces the quick mask from Chapter 5, with every element in the quick mask multiplied by nine. The larger values are necessary because dividing by the average gray level could reduce the strength of all edges to zero.

Figure 6.11 shows the result of the regular quick edge detector. Figure 6.12 shows the result of dividing the quick edge result by the average value to produce contrast-based edge detection. Notice the inside of the upstairs and downstairs windows. Figure 6.11 (quick edge) shows edges inside the downstairs windows, but not inside the upstairs windows. Figure 6.12 (contrast-based) shows details inside the downstairs and upstairs windows. Refer to the original image (Figure 6.1). The downstairs windows are shaded and the upstairs windows are not. Contrast-based edge detection gives better results in uneven lighting.

Figure 6.11: Result of Quick Edge Detector

Figure 6.12: Result of Contrast-Based Edge Detector

Contrast-based edge detection is possible with any edge detector. As a project, try modifying the homogeneity edge detector by dividing its result by the average gray level — but first multiply the result of homogeneity by a factor (nine or more) so dividing does not reduce edge strengths to zero. Modify any of the other edge detectors in a similar manner.

6.7 Edge Enhancement

A good application of edge detectors is to enhance edges and improve the appearance of an original image. Detect the edges in an image and overlay these edges on top of the original image to accent its edges. The last section of Listing 6.1 shows the enhance_edges function, which repeats the quick_edge function from Chapter 5 with a few added lines of code. Examine the code right after the loops over a and b. If the result of convolution (the sum variable) is greater than a user-chosen threshold, the output image is assigned that value. If not, the output image is assigned the value from the input image. The result is the input image with its strong edges accented.

Figure 6.13 shows the result of enhancing the edges of Figure 6.1. The edges of the bricks, the siding in the left, and the lines on the garage door are all sharper.

Any edge detector can be used to enhance the edges in an input image. Simply add the option of taking the edge detector result or a value from the input image. An interesting project would be to use the 9x9 Gaussian mask

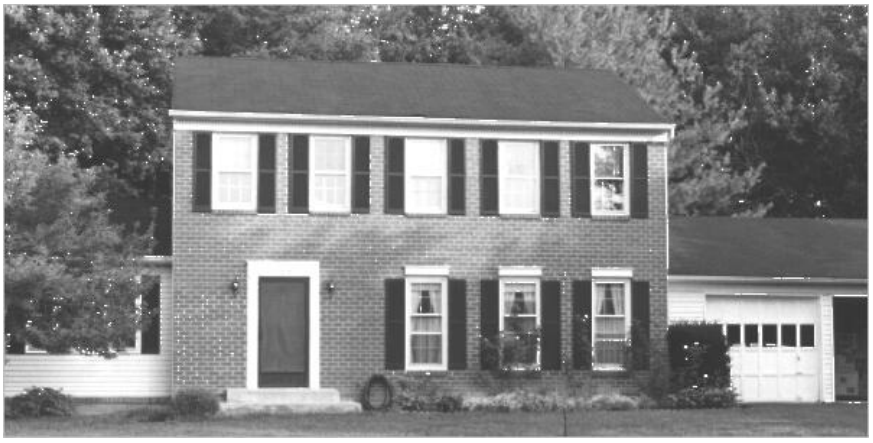


Figure 6.13: Result of Edge Enhancement

to detect the edges of large objects and then use these edges to enhance the input image.

6.8 Variance and Range

The chapter ends with two edge detectors similar to the difference edge detector in that they look at the image numbers inside a small area. The variance operator, examines a 3x3 area and replaces the center pixel with the variance. The variance operator subtracts the pixel next to the center pixel, squares that difference, adds up the squares of the differences from the eight neighbors, and takes the square root. The other edge detector, the range operator, sorts the pixels in an nxn area and subtracts the smallest pixel value from the largest to produce the range.

Figure 6.14 shows the results of applying the variance and range operators to an array of numbers. Figures 6.15 and 6.16 show the outcome of applying these operators.

6.9 Applications

Listing 6.2 shows the *medge* program that ties together all the edge detectors from this and the previous chapter. The user chooses among 11 different edge detectors. Entering the *medge* command without any parameters causes the

```
Input
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20
  5  5  5  5 10 10 10 10 20 20 20 20

Variance Output
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0
  0  0  0  7  7  0  0 14 14  0  0  0

Range Output
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
  0  0  0  5  5  0  0 10 10  0  0  0
```

Figure 6.14: The Results of Applying the Variance and Range Operators to an Array of Numbers

Figure 6.15: Result of Variance Edge Detector

Figure 6.16: Result of Range Edge Detector

usage message to appear and give examples of each operator. Regardless of the operator chosen, the program does the usual creating an output image file, allocating arrays, and reading input data. The program uses the second command line parameter to step into an if statement to interpret the other parameters. It then calls the desired edge detector and writes the result to the output image. The *medge* program serves as a pattern for programs in the following chapters that collect a number of related image processing operations.

6.10 Conclusions

This chapter has continued the discussion of edge detectors. The homogeneity, difference, variance, and range edge detectors work by subtracting pixel values inside a small area around an edge. The Gaussian edge detector convolves an image with a "Mexican hat" image piece. The contrast-based edge detector compensates for differences in brightness levels in different parts of an image. These edge detectors will be used again during the segmentation chapters later in this book.

6.11 References

6.1 "Recognizing Objects in a Natural Environment: A Contextual Vision System (CVS)," Martin A. Fischler, Thomas M. Strat, Proceedings Image Understanding Workshop, pp. 774-796, Morgan Kaufmann Publishers, May 1989.
6.2 "Digital Image Processing," Kenneth R. Castleman, Prentice-Hall, 1979.
6.3 "Vision in Man and Machine," Martin D. Levine, McGraw-Hill, 1985.
6.4. "Contrast-Based Edge Detection," R. P. Johnson, Pattern Recognition, Vol. 23, No. 3/4, pp. 311-318, 1990.

Chapter 7

Spatial Frequency Filtering

7.1 Spatial Frequencies

All images and pictures contain spatial frequencies. Most of us are familiar with some type of frequency such as the 60-cycle, 110-volt electricity in our homes. The voltage varies in time as a sinusoid, and the sinusoid completes a full cycle 60 times a second — a frequency of 60 Hertz.

Images have spatial frequencies. The gray level in the image varies in space (not time), i.e. it goes up and down. Figure 7.1 shows the side view of an image with low spatial frequencies. The gray level is low at the left edge of the figure, stays constant for a while, then shifts to a higher gray level. The gray level is fairly constant throughout (only one change in space) and so the figure has low spatial frequencies.

Figure 7.2 shows the side view of an image with high spatial frequencies. The gray level changes many times in the space of the image. The rate or frequency of change in the space of the image is high, so the image has high spatial frequencies.

7.2 Filtering

Filtering is also a common concept. Adjusting the bass and treble on stereos filters out certain audio frequencies and amplifies others. High-pass filters pass high frequencies and stop low frequencies. Low-pass filters stop high frequencies and pass low frequencies. In the same manner, it is possible to filter spatial frequencies in images. A high-pass filter will amplify or "pass"

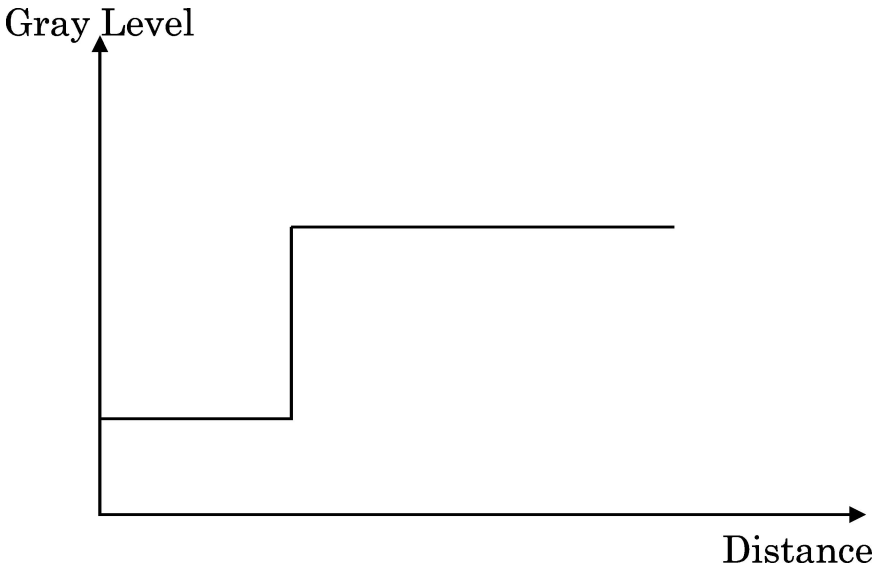


Figure 7.1: Side View of an Image with Low Spatial Frequencies

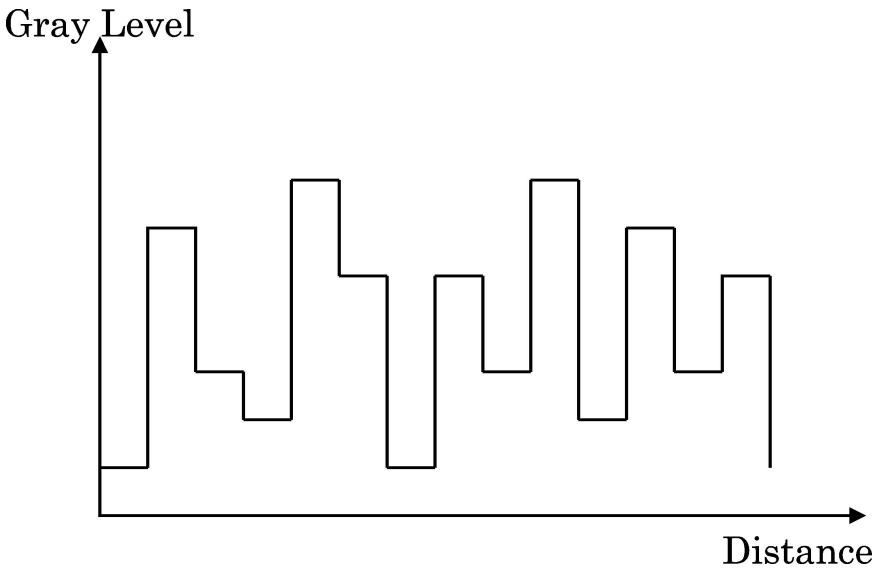


Figure 7.2: Side View of an Image with High Spatial Frequencies

frequent changes in gray levels and a low-pass filter will reduce frequent changes in gray levels.

Consider the nature of a frequent or sharp change in gray level. Figure 7.1 showed an image with only one change in gray level. That change, however, was very sharp — it was an edge. A high-pass filter will pass, amplify, or enhance the edge. A low-pass filter will try to remove the edge. Instead of an instant move from one gray level to another, the low-pass filter will produce a gradual slope between the two levels. The two gray levels will still exist, but the transition will be slower.

7.3 Application of Spatial Image Filtering

Spatial image filtering has several basic applications in image processing. Among these are noise removal, smoothing, and edge enhancement. Noise in an image usually appears as snow (white or black) randomly sprinkled over an image. Spikes, or very sharp, narrow edges in the image cause snow. A low-pass filter smoothes and often removes these sharp edges.

Edge enhancement improves the appearance of an image by sharpening the outlines of objects. Chapter 6 described how an edge detector enhanced edges. The detected edges were overlaid on top of the original image to emphasize the edges. A high-pass filter produces the same result in one operation.

7.4 Frequency vs. Spatial Filtering

Consider sound as noise varying in the time domain, i.e. the pitch of the noise varies with time. A pure sinusoid completing a cycle 1000 times a second is a 1KHz tone. In the frequency domain, this is a single value at 1000. To filter it out, multiply it by a low-pass filter that only passes frequencies below 900 cycles per second. Picture the low-pass filter as an array with the value of one in all places from zero through 900 and the value zero in all places above 900.

Multiplication in the frequency domain is a simple idea, however, there is one problem. People hear sound in the time domain. The signal, however, must be transformed to the frequency domain before multiplication. Fourier transforms do this tranformation [7.1]. Fourier transforms require substantial

```
      0  1  0
1/6 * 1  2  1
      0  1  0

      1  1  1
1/9 * 1  1  1
      1  1  1

       1  1  1
1/10 * 1  2  1
       1  1  1

       1  2  1
1/16 * 2  4  2
       1  2  1
```

Figure 7.3: Low-Pass Filter Convolution Masks

computations, and in some cases is not worth the effort.

Multiplication in the frequency domain corresponds to convolution in the time and the spatial domain (such as in Chapter 5). Using a small convolution mask, such as 3x3, and convolving this mask over an image is much easier and faster than performing Fourier transforms and multiplication.

7.5 Low-Pass Filtering

Low-pass filtering smoothes out sharp transitions in gray levels and removes noise. Figure 7.3 shows four low-pass filter convolution masks. Convolving these filters with a constant gray level area of an image will not change the image. Notice how the second convolution mask replaces the center pixel of the input image with the average gray level of the area. The other three masks have the same general form — a "peak" in the center with small values at the corners.

The next four figures show numerical examples of how a low-pass filter affects an image. Figure 7.4 shows an image segment with low spatial frequencies. The image segment changes gray level once, but with a sharp

```
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
```

Figure 7.4: An Image Segment with Low Spatial Frequencies

```
150 150 150 150 150
  1   1   1   1   1
150 150 150 150 150
  1   1   1   1   1
150 150 150 150 150
  1   1   1   1   1
150 150 150 150 150
  1   1   1   1   1
150 150 150 150 150
  1   1   1   1   1
```

Figure 7.5: An Image Segment with High Spatial Frequencies

transition. Figure 7.5 shows an image segment with higher spatial frequencies. It changes gray level every row of pixels, with every change a sharp transition.

Figure 7.6 shows the result of convolving the first 3x3 low-pass filter mask of Figure 7.3 with the image segment given in Figure 7.4. The high and low gray-level rows remain, but the transition differs. The low-pass filter smoothed the transition from one row to three rows of pixels. In a photograph this would make the edge look fuzzy or blurred.

Figure 7.7 shows the result of convolving the first 3x3 low-pass filter mask of Figure 7.3 with the image segment given in Figure 7.5. The image in Figure 7.7 still has transitions or edges from row to row. The low-pass

```
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
125 125 125 125 125
 25  25  25  25  25
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
```

Figure 7.6: Low-Pass Filtering of Figure 7.4

filter, however, reduced the magnitude of these transitions. In a photograph they would appear dimmer or washed out when compared with the original in Figure 7.5.

7.6 Median Filters

A special type of low-pass filter is the median filter [7.2]. The median filter takes an area of an image (3x3, 5x5, 7x7, etc.), looks at all the pixel values in that area, and replaces the center pixel with the median value. The median filter does not require convolution. It does, however, require sorting the values in the image area to find the median value.

There are two noteworthy features of the median filter. First, it is easy to change the size of the median filter. (The images later will show the effect of using a different size.) Implementing the different size is a simple matter of for loops in the code.

Second, median filters remove noise in images, but change noise-free parts of images minimally. Consider the influence of a 3x3 median filter on the image segments in Figures 7.4 and 7.5. The image in Figure 7.4 would not change. Centering the 3x3 filter on the last row of 150s yields a median value of 150. Centering it the first row of 1s yields a median value of one. The image in Figure 7.5 would change, but the change would not be obvious. The filter would replace the rows of 150s with rows of 1s and would replace the

```
100 100 100 100 100
 50  50  50  50  50
100 100 100 100 100
 50  50  50  50  50
100 100 100 100 100
 50  50  50  50  50
100 100 100 100 100
 50  50  50  50  50
100 100 100 100 100
 50  50  50  50  50
```

Figure 7.7: Low-Pass Filtering of Figure 7.5

rows of 1s with rows of 150s.

7.7 Effects of Low-Pass Filtering

Figure 7.8 is an aerial image spotted with noise. There are two streets running vertically with rows of houses on either sides of the streets. The white dots all over the trees are noise. The noise came from a poor photograph compounded by less than perfect scanning. Figures 7.9, 7.10, 7.11, and 7.12 show the result of applying the low-pass filters to the image in Figure 7.8. The four results are all similar. The filters removed the snow from Figure 7.8 and retained the appearance of the houses and streets. My personal favorite is Figure 7.12, but you should apply all four masks to your image and decide for yourself. The masks are different and produce different results sometimes noticeable, sometimes not.

Figure 7.13 shows the result of applying a 3x3 median filter to Figure 7.8. The filter removed the snow but left the areas of houses and streets unchanged.

Figures 7.15, 7.16, and 7.17 show the result of applying three different median filters (3x3, 5x5, and 7x7) to the house image in Figure 7.14. In the result of the 3x3 filter (Figure 7.15), the trees start to appear fuzzy and the lines between the bricks disappear. In the result of the 5x5 filter (Figure 7.16), the trees are blotches, the bricks only textures, and the other details are disappearing. Finally, the 7x7 filter (Figure 7.17) eliminates all

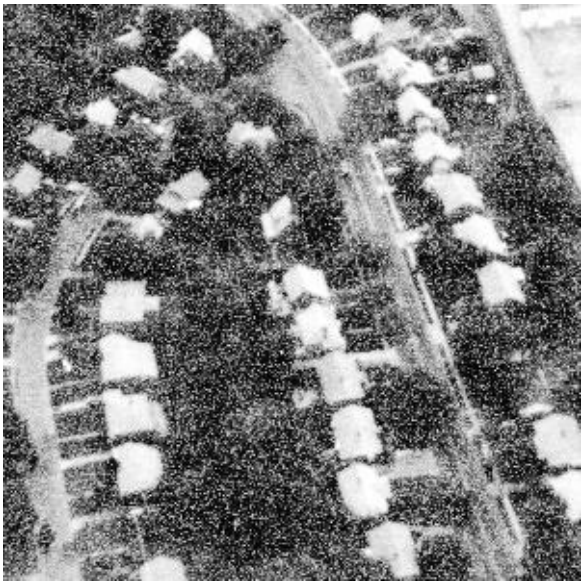


Figure 7.8: Noisy Aerial Image

Figure 7.9: Result of Low-Pass Filter Mask #6

Figure 7.10: Result of Low-Pass Filter Mask #9

Figure 7.11: Result of Low-Pass Filter Mask #10

Figure 7.12: Result of Low-Pass Filter Mask #16

Figure 7.13: Result of 3x3 Median Filter

detail. The "best" filter for this image is probably the 3x3 filter. Images with different size details and noise would require different size filters.

Figure 7.14: House Image

Note how in Figure 7.17 only the large objects are recognizable, such as windows, roof, window frames, door frames, and door. This is an excellent starting point for a part of image processing called segmentation. In segmentation, the computer attempts to find the major objects in the image and separate or segment them from the other objects. Segmentation would be difficult with Figure 7.14 because it contains too many small and insignificant objects, such as bricks and leaves. Figure 7.17 is so fuzzy that only the large objects are recognizable. Later chapters will discuss segmentation.

Although the size and results of median filters are easy to change, the process can be slow. The 3x3 median filter and the 3x3 convolution filters work equally fast. However, when moving to 5x5 and beyond, the median filter slows down noticeably because of the continuous sorting and picking of the median value.

Figure 7.15: Result of 3x3 Median Filter

Figure 7.16: Result of 5x5 Median Filter

Figure 7.17: Result of 7x7 Median Filter

7.8 Implementing Low-Pass Filtering

Listing 7.1 shows the source code for the low-pass and median filters. The first section of code declares the four low-pass filter masks (then three high-pass filter masks which we'll discuss later).

The major filtering function is filter_image. This implements the low-pass (and high-pass) convolution filters. filter_image resembles the convolution-based, edge-detection functions in Chapters 5 and 6.

The d=type statements set up the denominator for later use. The low-pass filter masks should have fractions in front of them (1/6, 1/9, 1/10, and 1/16). Using the fractions in convolution masks would require floating-point arrays and operations. It is much simpler and quicker to use shorts and then divide the final result.

filter_image reads an array from the input image and goes into the for loops to perform the convolution. These loops move through the image array and do the 3x3 multiplication and summing. The sum is divided by the denominator mentioned above and set to the max or min value in case of overrun. filter_image finishes by calling fix_edges to fill the edges of the output and writes the array to the output file.

The next function in Listing 7.1, median_filter, implements the variable-

size median filter. The key to this filter is finding the median pixel value in the nxn area being filtered. This routine does this by creating an array to hold the pixel values, sorting the array, and taking the middle number in the sorted array. First, it allocates the elements array to hold the pixel values by calling malloc.

median_filter goes into a series of loops which cover the entire image array. As it moves through the image, it copies an nxn area of pixel values surrounding each point to the elements array. The output image array is set to the median_of the elements array. median_filter calls fix_edges to fill out the edges of the output and writes it to the output image file.

The next function in Listing 7.1 is median_of. This calls fsort_elements to sort the elements array and returns the middle element of the sorted array. The fsort_elements function (next in Listing 7.1) is a bubble sort. It calls the fswap function (also in Listing 7.1) to swap elements.

7.9 High-Pass Filtering

High-pass filters amplify or enhance a sharp transition (an edge) in an image. Figure 7.18 shows three 3x3 high-pass filter convolution masks. Each will leave a homogenous area of an image unchanged. They all have the same form — a peak in the center, negative values above, below, and to the sides of the center, and corner values near zero. The three masks, however, produce different amplifications to different high spatial frequencies.

7.10 Effects of High-Pass Filtering

Figures 7.19 and 7.20 show the results of applying the first high-pass filter mask to Figures 7.4 and 7.5. In Figure 7.19 (the result of filtering Figure 7.4) the high-pass filter amplified the edge. The transition from 150 to one is now from 255 to zero. In a photograph this would appear as adjacent black and white lines. In Figure 7.20 (the result of filtering Figure 7.5) the high-pass filter amplified the many edges, making the transitions all from 255 to zero. This would appear as alternating black and white lines in a photograph. Notice the differences between Figures 7.19 and 7.20 and Figures 7.5 and 7.6. The low-pass filter (Figures 7.5 and 7.6) smoothed the edges. In contrast, the high-pass filter enhanced them.

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


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


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

Figure 7.18: High-Pass Filter Convolution Masks

```
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
150 150 150 150 150
255 255 255 255 255
  0   0   0   0   0
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
```

Figure 7.19: Result of High-Pass Filter on Figure 7.4

```
255 255 255 255 255
  0   0   0   0   0
255 255 255 255 255
  0   0   0   0   0
255 255 255 255 255
  0   0   0   0   0
255 255 255 255 255
  0   0   0   0   0
255 255 255 255 255
  0   0   0   0   0
```

Figure 7.20: Result of High-Pass Filter on Figure 7.5

Figures 7.21, 7.22, and 7.23 show what the high-pass filters will do to the house image of Fiure 7.14. Figure 7.21 shows the result of the first high-pass filter convolution mask. Look closely at the differences between Figures 7.14 and 7.21. In Figure 7.21 you can see leaves, shingles on the roof, the texture of the screen door, and far greater contrast in the bricks and mortar. This high-pass filter has enhanced the details of the image.

Figure 7.22 shows the result of the second high-pass filter convolution mask. This image is almost black and white (few gray levels between). The details (edges) were enhanced, but probably too much. Figure 7.23 shows the result of the third high-pass filter convolution mask. This image contains many tiny highlights in the leaves of the trees and noise or snow in the remaining parts.

These images show the differences in the filtering properties of the three masks. The third filter mask has little affect on low spatial frequencies and a great affect on areas with relatively high spatial frequencies. So it does not enhance the bricks but does enhance the tiny leaves in the trees. The second filter has a high gain on most high frequencies (edges). So it produced an almost black and white image — all the edges were amplified greatly. The first filter amplifies all edges, but not with a high gain. The filter enhanced, but did not saturate, the edges. Each filter has its own unique properties and you should use them to their advantage. Try all three on an image and choose the enhancement you prefer.

Figure 7.21: Result of High-Pass Filter Mask #1

Figure 7.22: Result of High-Pass Filter Mask #2

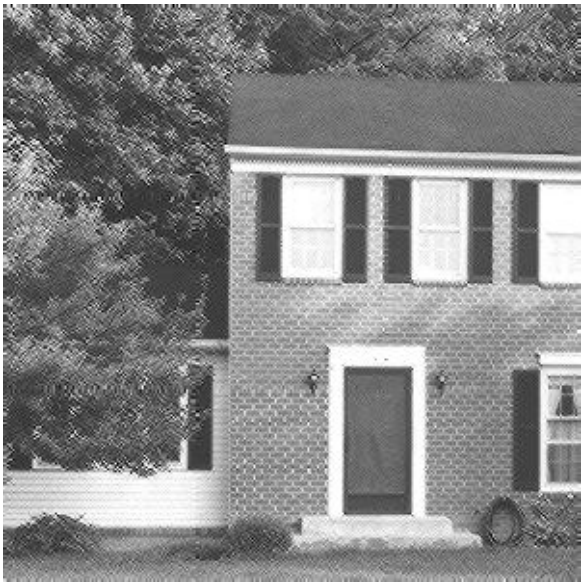


Figure 7.23: Result of High-Pass Filter Mask #3

7.11 Implementing High-Pass Filtering

The high-pass filters use the same function, filter_image, as the low-pass filters, but the user send filter_image it a different filter mask. Listing 7.1, contains the function setup_filters. This copies a filter mask declared at the top of Listing 7.1 into a 3x3 filter array.

Listing 7.2 shows the main routine of the *mfilter* program. This is similar to the *medge* program described in chapter 6. The *mfilter* program interprets the command line input to call one of several types of image filters. It contains the usual calls to create an output image, allocate image arrays, read input, and finally write the result to the output image file.

7.12 Conclusion

This chapter has discussed spatial frequencies in images and how to filter these frequencies. It demonstrated several types of low-pass and high-pass filters on various images. These are not the only filter masks available. Those familiar with Fourier transforms could derive other masks and also experiment with band-pass and band-stop filters.

7.13 References

7.1 "An Introduction to Digital Signal Processing," John H. Karl, Academic Press, 1989.
7.2 "Vision in Man and Machine," Martin D. Levine, McGraw-Hill, 1985.

Chapter 8

Image Operations

8.1 Introduction

This chapter introduces several basic image operations that allow you to have fun with images. These operations are adding and subtracting images and cutting and pasting parts of images. The chapter ends with two utility programs. The first creates blank images, and the second inverts the pixel values in images. As the images will show, these allow editing images by piecing them together.

8.2 Addition and Subtraction

Figure 8.1 illustrates the first operations: image addition and subtraction. Image addition adds each pixel in one image to each pixel in another and places the result in a third image. If the sum is greater than the maximum pixel value, it sets the sum to the maximum value. Image subtraction is the same. If the difference is less than 0, it sets it to zero.

Image addition and subtraction can place objects into and remove objects from images. One application subtracts two images in a sequence to see what has changed. For example, take an aerial photograph of a factory two days in a row, subtract the second from the first, and detect what has moved from day to day.

Image subtraction can also remove features. Figure 8.2 shows a house. Figure 8.3 shows the output of an edge detector applied to Figure 8.2. Figure 8.4 shows the result of subtracting the edges from the original image. Note

```
Image A      Image B
  0 100       50 150
200 255      250 200

A + B        A - B
 50 250        0   0
255 255        0  55
```

Figure 8.1: Addition and Subtraction of Images

how the edges in Figure 8.4 are whitened out or removed.

Figure 8.2: A House Image

Listing 8.1 shows the functions that implement image addition and subtraction. The add_image_array function adds two image arrays. This function shows the simplicity of image operators using the structure of image I/O routines presented in chapter 1. The operators don't do any image I/O — they simply operator on images. The code adds the two image arrays and puts the result in the output image array. If the sum of two pixels is greater than the maximum pixel value, you set it to the maximum value. The subtract_image_array function is the same as add_image_array except

Figure 8.3: Edge Detector Output of Figure 8.2

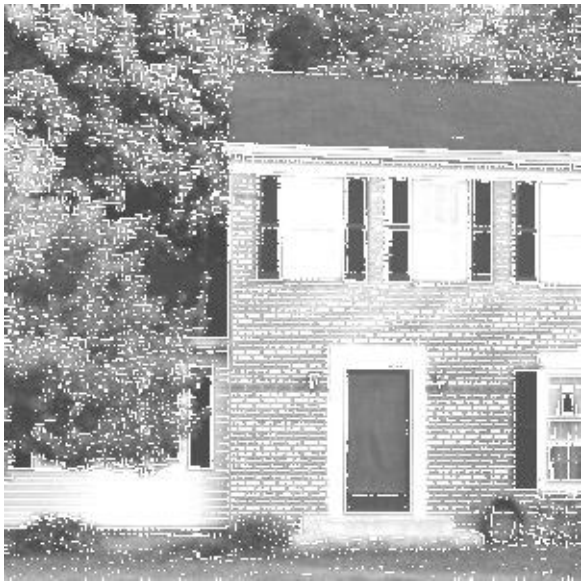


Figure 8.4: Figure 8.2 Minus Figure 8.3 (Edges Subtracted)

```
  Image A          Image B       Cut and Paste Result
 1  2  3  4       0  1  0  1          1  2  3  4
 5  6  7  8       0  1  0  1          5  1  0  1
 9 10 11 12       0  1  0  1          9  1  0  1
13 14 15 16       0  1  0  1         13  1  0  1
```

Figure 8.5: Cutting and Pasting

it subtracts the pixels in one image array from the corresponding pixels in another image array.

These simple functions may seem insignificant. They only add and subtract, but did you ever do anything useful by adding and subtracting numbers? Think of the possibilities and experiment.

Listing 8.2 shows the *mainas* program. It allows a person to call the add_image_array and subtract_image_array routines from the command line. It has the same form as other main routines. Note how it uses the are_not_same_size routine to ensure the two images have the same size.

8.3 Rotation and Flipping

The first edition of this book presented software that could rotate and flip images. This edition covers these topics in chapter 13. The methods used for rotation in this edition are far superior to those given in the first edition.

8.4 Cut and Paste

The next operations are image cut and paste. They allow cutting rectangular areas from one image and pasting them into another. Figure 8.5 shows a cut and paste example where a section of image B was pasted into image A by reading from one image and writing into another one. Figure 8.6 shows the result of pasting parts of the image in Figure 8.3 into the image in Figure 8.2. This demonstrates a method of judging the affect of image processing operators by pasting the processing results back into the original image.

Listing 8.3 shows the function paste_image_piece. This takes in two image arrays and line and element parameters that describe where in the input array to cut a rectangle and where in the output array to paste it. The code that

Figure 8.6: Section of Figure 8.3 Cut and Pasted Into Figure 8.2

performs the pasting comprises simple loops that copy numbers from one array to another.

Much of the cut and paste work is done is the main routine of the *maincp* program shown in listing 8.4. The main program checks the command line parameters, allocates arrays, and reads image arrays. It then calls check_cut_and_paste_limits to ensure that the input rectangle exists and that it will fit in the output image. Listing 8.3 shows the source code for check_cut_and_paste_limits.

8.5 Image Scaling

The first edition of this book presented software that could scale images. This edition covers this topic in chapter 13. The method used for scaling in this edition is far superior to that given in the first edition.

8.6 Blank Images

A handy utility program is one that creates a blank image. A blank image is useful as a bulletin board to paste other images together. Figure 8.7 shows a

composite made of two images pasted onto a blank image. The two images are of a boy with one being the negative of the other (more on this in the next section).

Figure 8.7: Two Images Pasted Onto a Blank Image

Listing 8.5 shows the *create* program that created the blank image. This interprets the command line, sets up the image header, and calls with create_allocate_tiff_file or create_allocate_bmp_file. Those routines fill the blank image with zeros.

8.7 Inverting Images

Another handy utility program inverts the pixels in an image. Some images appear as negatives for certain image viewers. The boy in the upper part of Figure 8.7 is the negative of the boy in the lower part. The *invert* program created on from the other. The *invert* program reads the input image, inverts

the pixels by subtracting them from the number of gray shades (0 becomes 255, 1 becomes 254, etc.), and writes the output image to disk. I don't use *invert* often, but it is essential.

8.8 Conclusion

This chapter described several image operations that provide the ability to edit images by adding and subtracting and cutting and pasting. It described two utility programs that create blank images and invert images. These operations are fun because they allow you to place original images and processing results together in combinations and display them all at once. Enjoy and experiment. These are low-level tools that you can combine in an endless variety of ways.

Chapter 9

Histogram-Based Segmentation

This chapter describes simple image segmentation based on histograms and image thresholding. Image segmentation is the process of dividing an image into regions or objects. It is the first step in the task of image analysis. Image processing displays images and alters them to make them look "better," while image analysis tries to discover what is in the image.

The basic idea of image segmentation is to group individual pixels (dots in the image) together into regions if they are similar. Similar can mean they are the same intensity (shade of gray), form a texture, line up in a row, create a shape, etc. There are many techniques available for image segmentation, and they vary in complexity, power, and area of application.

9.1 Histogram-Based Segmentation

Histogram-based image segmentation is one of the most simple and most often used segmentation techniques. It uses the histogram to select the gray levels for grouping pixels into regions. In a simple image there are two entities: the background and the object. The background is generally one gray level and occupies most of the image. Therefore, its gray level is a large peak in the histogram. The object or subject of the image is another gray level, and its gray level is another, smaller peak in the histogram.

Figure 9.1 shows an image example and Figure 9.2 shows its histogram. The tall peak at gray level 2 indicates it is the primary gray level for the background of the image. The secondary peak in the histogram at gray level 8 indicates it is the primary gray level for the object in the image. Figure

```
22222232221222212222
32222321250132123132
22588897777788888232
12988877707668882122
22888892326669893213
21278221222666665222
22002222220226660225
21221231223266622321
32238852223266821222
21288888342288882232
22328888899888522121
22123988888889223422
23222278888882022122
22232323883212123234
25221212222222222222
22122222320222202102
20222322412212223221
22221212222222342222
21222222221222222142
```

Figure 9.1: An Image Example

9.3 shows the image of Figure 9.1 with all the pixels except the 8s blanked out. The object is a happy face.

This illustrates histogram-based image segmentation. The histogram will show the gray levels of the background and the object. The largest peak represents the background and the next largest peak the object. We choose a threshold point in the valley between the two peaks and threshold the image. Thresholding takes any pixel whose value is on the object side of the point and sets it to one; it sets all others to zero. The histogram peaks and the valleys between them are the keys.

The idea of histogram-based segmentation is simple, but there can be problems. Where is the threshold point for the image of Figure 9.1? Choosing the point mid-way between the two peaks (threshold point = 5), produces the image of Figure 9.4. This is not the happy face object desired. Choosing the valley floor values of 4 or 5 as the threshold point, also produces a poor result. The best threshold point would be 7, but how could anyone know

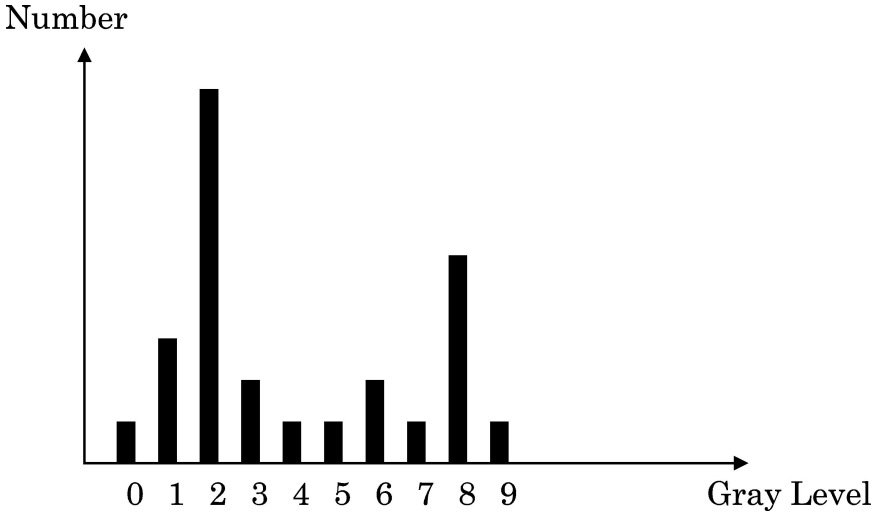


Figure 9.2: A Histogram of the Image of Figure 9.1

```
--------------------
--------------------
---888------88888---
---888-------888----
--8888--------8-----
----8---------------
--------------------
--------------------
----88--------8-----
---88888----8888----
----88888--888------
------8888888-------
-------888888-------
--------88----------
--------------------
--------------------
--------------------
--------------------
--------------------
```

Figure 9.3: The Image in Figure 9.1 with All the Pixels Except the 8s Blanked Out

```
00000000000000000000
00000000000000000000
00011111111111111000
00111111101111110000
00111110001111110000
00011000000111110000
00000000000001110000
00000000000011100000
00001100000011100000
00011111000011110000
00001111111111000000
00000111111111000000
00000011111110000000
00000000110000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Figure 9.4: Figure 9.1 with a Threshold Point of 5

that without using trial and error?

This example is difficult because there are only ten gray levels and the object (happy face) is small. In practice, the techniques discussed below will perform adequately, but there will be problems. Automatic techniques will fail.

9.2 Histogram Preprocessing

Histogram-based segmentation depends on the histogram of the image. Therefore, the image and its histogram may need preprocessing before analyzing them. The first step is histogram equalization, explained in Chapter 4. Histogram equalization attempts to alter an image so its histogram is flat and spreads out over the entire range of gray levels. The result is an image with better contrast.

Figure 9.5 shows an aerial image of several house trailers with its histogram. The contrast is poor and it would be very difficult to find objects based on its histogram. Figure 9.6 shows the result of performing histogram equalization. The contrast is much better and the histogram is spread out over the entire range of gray levels. Figure 9.7 shows the result of performing high-pass filtering on the image of Figure 9.6, explained in Chapter 7. It is easy to see the house trailers, sidewalks, trees, bushes, gravel roads, and parking lots.

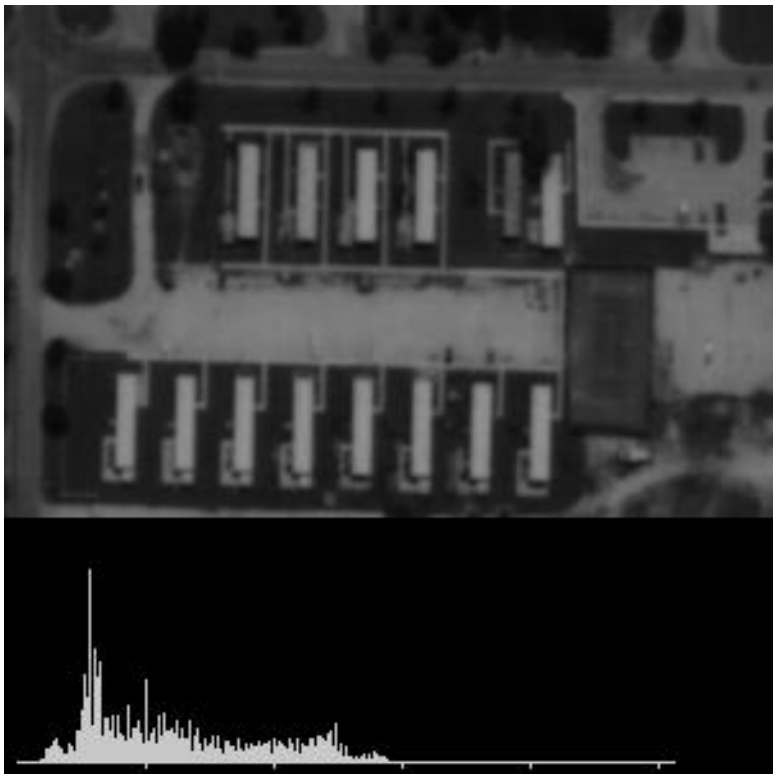


Figure 9.5: Aerial Image with Poor Contrast

The next preprocessing step is histogram smoothing. When examining a histogram, look at the peaks and valleys. Too many tall, thin peaks and deep valleys will cause problems. Smoothing the histogram removes these spikes and fills in empty canyons while retaining the same basic shape of the histogram.

Figure 9.8 shows the result of smoothing the histogram given in Figure 9.2. You can still see the peaks corresponding to the object and background,

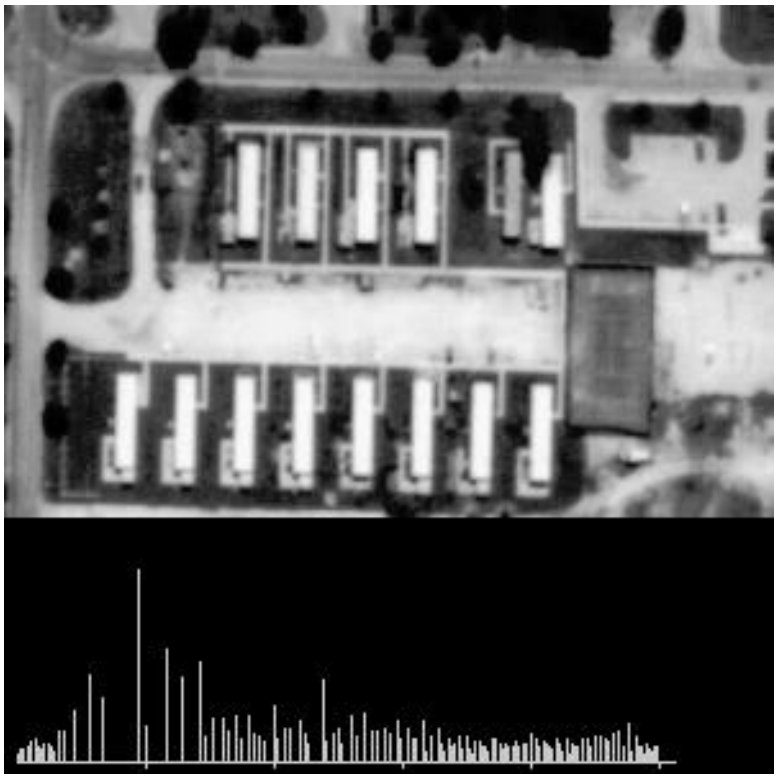


Figure 9.6: Result of Histogram Equalization on Figure 9.5

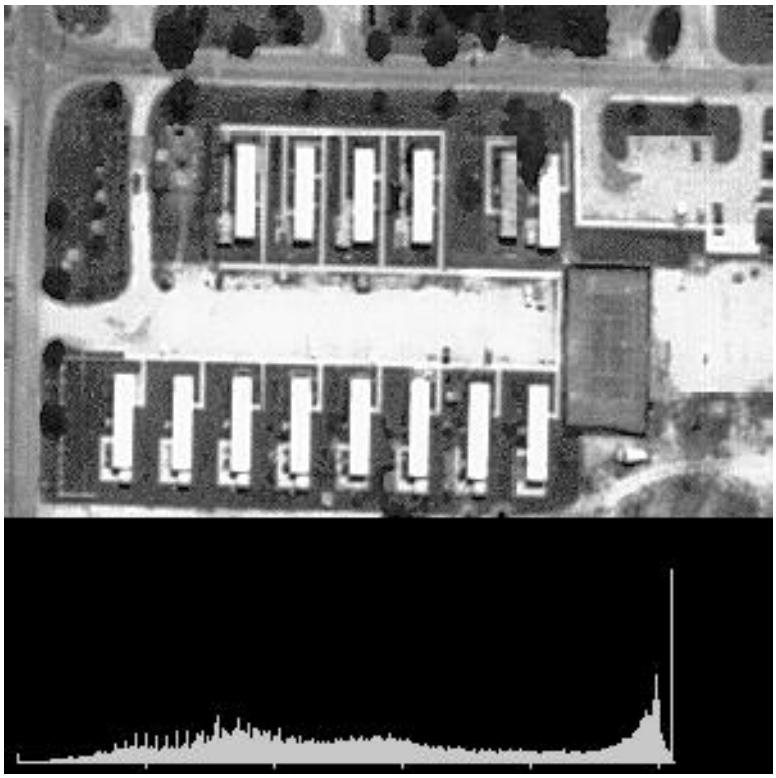


Figure 9.7: Result of High-Pass Filtering on Figure 9.6

but the peaks are shorter and the valleys are filled.

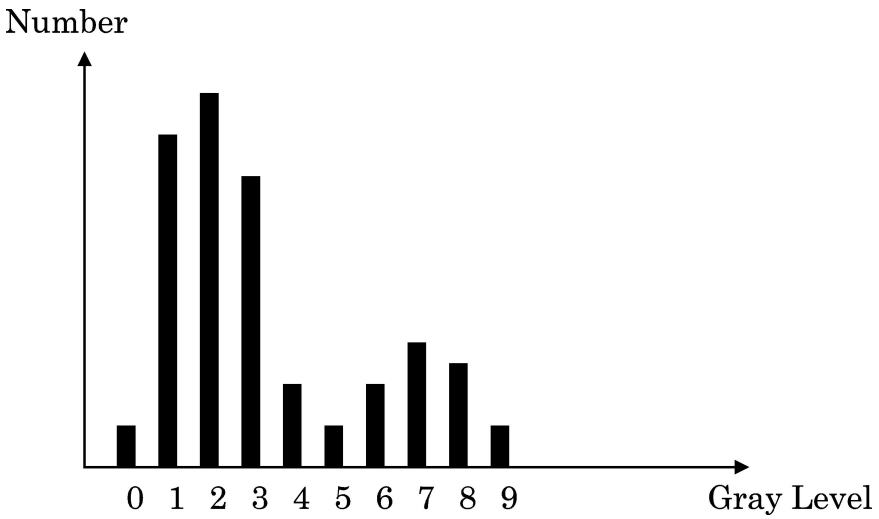


Figure 9.8: The Result of Smoothing the Histogram Given in Figure 9.2

Smoothing a histogram is an easy operation. It replaces each point with the average of it and its two neighbors. Listing 9.1 shows the smooth_histogram function that performs this operation.

9.3 Thresholding and Region Growing

There are two more topics common to all the methods of image segmentation: image thresholding and region growing. Image thresholding sets the pixels in the image to one or zero. Listing 9.2 shows all the subroutines used in this chapter's segmentation techniques. It begins with the routine threshold_image_array that accomplishes this task.

The difficult task is region growing. Figure 9.9 shows the result of thresholding Figure 9.1 correctly. The "object" in Figure 9.9 is a happy face. It comprises three different regions (two eyes and the smile). Region growing takes this image, groups the pixels in each separate region, and gives them unique labels. Figure 9.10 shows the result of region growing performed on Figure 9.9. Region growing grouped and labeled one eye as region 1, the other eye as region 2, and the smile as region 3.

Figure 9.11 shows the algorithm for region growing. It begins with an image array g comprising zeros and pixels set to a value. The algorithm loops

```
00000000000000000000
00000000000000000000
00011100000011111000
00011100000001110000
00111100000000100000
00001000000000000000
00000000000000000000
00000000000000000000
00001100000000100000
00011111000011110000
00001111100111000000
00000011111110000000
00000001111110000000
00000000110000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Figure 9.9: The Result of Correctly Thresholding Figure 9.1

```
00000000000000000000
00000000000000000000
00011100000022222000
00011100000002220000
00111100000000200000
00001000000000000000
00000000000000000000
00000000000000000000
00003300000000300000
00033333000033330000
00003333300333000000
00000033333330000000
00000003333330000000
00000000330000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Figure 9.10: The Result of Region Growing Performed on Figure 9.9

through the image array looking for a g(i,j) == value. When it finds such a pixel, it calls the label_and_check_neighbor routine. label_and_check_neighbor sets the pixel to g_label (the region label) and examines the pixel's eight neighbors. If any of the neighbors equal value, they are pushed onto a stack. When control returns to the main algorithm, each pixel on the stack is popped and sent to label_and_check_neighbor. All the points on the stack equaled value, so set them and check their neighbors. After setting all the pixels in the first region, increment g_label and begin looking for the next region.

Listing 9.2 next shows the functions that implement region growing with grow being the primary routine. grow runs through the region-growing algorithm and calls label_and_check_neighbor (shown next in Listing 9.2). The grow and label_and_check_neighbor functions follow the region- growing algorithm step for step.

9.4 Histogram-Based Techniques

The following pages present four segmentation techniques: manual technique, histogram peak technique, histogram valley technique, and adaptive technique.

9.4.1 Manual Technique

In the manual technique the user inspects an image and its histogram manually. Trial and error comes into play and the result is as good as you want it to be.

Figure 9.12 is the input for all the segmentation examples. Figures 9.13, 9.14, and 9.15 show the result of segmentation using three different thresholds. The result in Figure 9.14 used a high of 255 and a low of 125. The segmentation included the white gravel roads as well as the house trailers and sidewalks. The result in Figure 9.14 used a high of 255 and a low of 175. The gravel roads begin to disappear, but the house trailers and sidewalks remain. Figure 9.15 shows the result using a high of 255 and a low of 225. This segmentation only finds the four dominant house trailers. Which answer is correct? That depends on what you wanted to find.

Note that all image segmentations will appear rough. It is possible to perform additional processing to make the result more pleasing to the eye (see chapter 11 for erosion and dilation techniques), but that is not the

```
1. Given an image g with m rows and n columns
      g(i,j) for i=1,m j=1,n
      g(i,j) = value for object
              = 0 for background

2. set g_label=2  this is the label value

3.    for (i=0; i<m; i++)
            scan ith row
            for (j=0; j<n; j++)
                 check jth element
                 stack_empty = true
                 if g(i,j) == value
                       label_and_check_neighbor(g(i,j),g_label)
                 while stack_empty = false do
                       pop element (i,j) off the stack
                       label_and_check_neighbor(g(i,j),g_label)
                 end while
                 g_label = g_label + 1
            end of checking jth element
      end of scanning ith row

4. The End

                  ---------------------------------------

procedure label_and_check_neighbor(g(r,e), g_label)
      g(r,e) = g_label
      for (R=r-1; r<=r+1; R++)
            for (E=e-1; e<=e+1; e++)
                 if g(R,E) ==  value then
                       push (R,E) onto the stack
                       stack_empty = false
                 end if
            end loop over E
      end loop over R
end procedure label_and_check_neighbor
```

Figure 9.11: Pseudocode for Region Growing

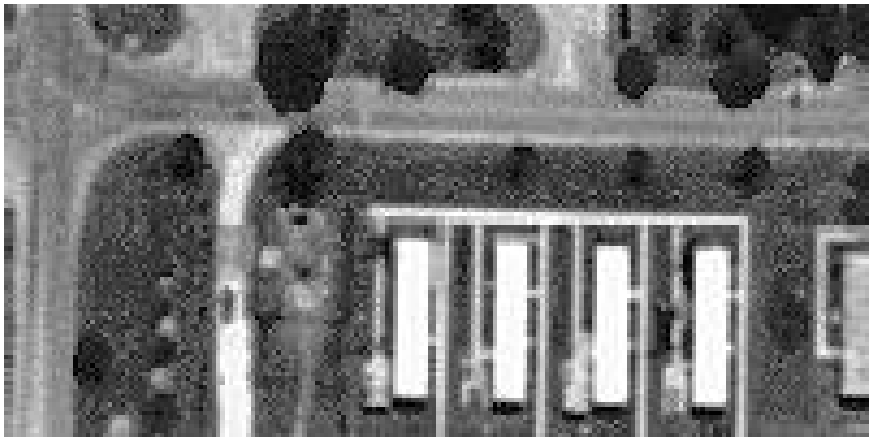


Figure 9.12: Input Image for Segmentation Examples

Figure 9.13: Threshold of Figure 9.12 with High=255 and Low=125

Figure 9.14: Threshold of Figure 9.12 with High=255 and Low=175

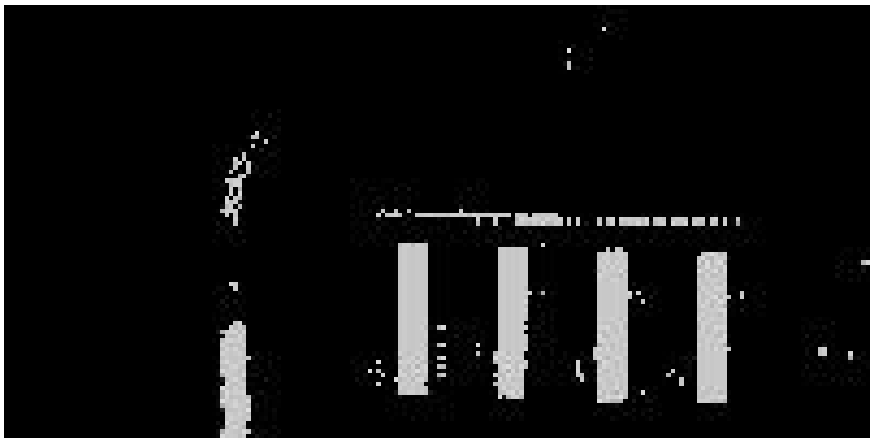


Figure 9.15: Threshold of Figure 9.12 with High=255 and Low=225

purpose of segmentation. The purpose is to break the image into pieces so later computer processing can interpret their meaning. The output is for computer not human consumption. Also note how difficult it is for the computer, even with manual aid, to find objects that are trivial for humans to see. Anyone could trace over the input image and outline the objects better than the segmentation process.

Listing 9.2 next shows the code that implements manual segmentation. The function manual_threshold_segmentation has the same form as the other subroutines.

manual_threshold_segmentation has the usual inputs as well as the high and low threshold values and the value and segment parameters. The value parameter specifies the value at which to set a pixel if it falls between the high and low thresholds. value is usually one since those pixels outside the high-low range are set to zero. The segment parameter specifies whether or not to grow regions after thresholding. Sometimes you only want to threshold an image and not grow regions. The two operations are identical except for the last step. If segment == 1, manual_threshold_segmentation calls the region-growing routines.

Manual segmentation is good for fine tuning and understanding the operation. Its trial-and-error nature, however, makes it time consuming and impractical for many applications. We need techniques that examine the histogram and select threshold values automatically.

9.4.2 Histogram Peak Technique

The first technique to examine the histogram and select threshold values automatically uses the peaks of the histogram. This technique finds the two peaks in the histogram corresponding to the background and object of the image. It sets the threshold halfway between the two peaks. Look back at the smoothed histogram in Figure 9.8. The background peak is at two and the object peak is at seven. The midpoint is four, so the low threshold value is four and the high is nine.

The peak technique is straightforward except for two items. In the histogram in Figure 9.8, you'll note the peak at seven is the fourth highest peak. The peaks at one and three are higher, but they are part of the background mountain of the histogram and do not correspond to the object. Searching the histogram for peaks requires peak spacing to ensure the highest peaks are separated. If the peak technique does not, then it would choose two as

```
---------*----*----
-------*--**--*--*--
-------------------
*--------*-------*--
-----------------*-
-*-----*-----------
--**------*-----*---
-*--*--*-----------*
---------------*---
-*------------------
-----------------*-*
--*-----------------
--------------*--*--
------------*-*-----
----*-*-------------
--*-------*----*-**-
-*-------*--*------*
----*-*-------------
-*--------*------*--
```

Figure 9.16: Result of Incorrect Peak Separation

the background peak and one as the object peak. Figure 9.16 shows the disastrous effect of this.

The second item to watch carefully is determining which peak corresponds to the background and which corresponds to the object. Suppose an image had the histogram shown in Figure 9.17. Which peak corresponds to the background? The peak for gray level eight is the highest, but it corresponds to the object not the background. The reason is the mountain surrounding the peak at gray level two has a much greater area than that next to gray level eight. Therefore, gray levels zero through six occupy the vast majority of the image, and they are the background.

Listing 9.2 next shows the source code to implement the peak technique. peak_threshold_segmentation is the primary function. Next, it calls new functions to find the histogram peaks and determine the high and low threshold values. Finally, it thresholds the image, performs region growing if desired, and writes the result image to the output file.

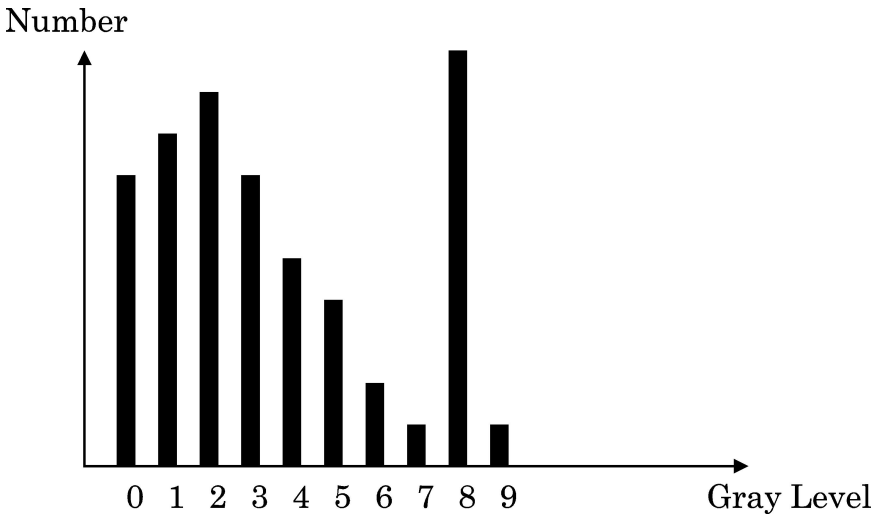


Figure 9.17: A Histogram in which the Highest Peak Does Not Correspond to the Background

The functions find_peaks and insert_into_peaks in Listing 9.2 analyze the histogram to find the peaks for the object and background. These functions build a list of histogram peaks. There are several ways to do this. I used an array of values. find_peaks loops through the histogram and calls insert_into_peaks, which puts the histogram values in the proper places in the array. find_peaks ends by looking at the spacing between the largest peaks to ensure we do not have a disaster such as shown in Figure 9.16.

The function peaks_high_low takes the two peaks from find_peaks and calculates the high-and low-threshold values for segmentation. peaks_high_low examines the mountains next to the peaks as illustrated in Figure 9.17. It then finds the midpoint between the peaks and sets the high and low threshold values.

Figure 9.18 shows the result of applying the peak technique to the image of Figure 9.12. The peak technique found the two peaks at 255 and 77. The midpoint is 166, so the high threshold is 255 and the low threshold is 166. This is a reasonably good segmentation of Figure 9.12.

9.4.3 Histogram Valley Technique

The second automatic technique uses the peaks of the histogram, but concentrates on the valleys between them. Instead of setting the midpoint arbi-

Figure 9.18: Threshold of Figure 9.12 Using Peak Technique (High=255 and Low=166)

trarily halfway between the two peaks, the valley technique searches between the two peaks to find the lowest valley.

Look back at the histogram of Figure 9.17. The peaks are at gray levels two and eight and the peaks technique would set the midpoint at five. In contrast, the valley technique searches from two through eight to find the lowest valley. In this case, the "valley point" is at gray level seven.

Listing 9.2 shows the code that implements the valley technique. The primary function is valley_threshold_segmentation. It calculates and smoothes the histogram and finds the peaks as peak_threshold_segmentation did. It finds the valley point via the functions valley_high_low, find_valley_point, and insert_into_deltas. find_valley_point starts at one peak and goes to the next, inserting the histogram values into a deltas array via the insert_into_deltas function. This uses an array to create a list of deltas in the same manner as insert_into_peaks did. Once it has valley point, valley_high_low checks the mountains around the peaks to ensure it associates the peaks with the background and object correctly.

Figure 9.19 shows the result of applying the valley technique to the image in Figure 9.12. It found the peaks at 77 and 255 and went from 77 up to 255 looking for the lowest valley. It pinpointed the lowest valley at gray level 241.

Figure 9.19: Threshold of Figure 9.12 Using Valley Technique (High=255 and Low=241)

9.4.4 Adaptive Histogram Technique

The final technique uses the peaks of the histogram in a first pass and adapts itself to the objects found in the image in a second pass [9.1]. In the first pass, the adaptive technique calculates the histogram for the entire image. It smoothes the histogram and uses the peak technique to find the high and low threshold values.

In the second pass, the technique segments using the high and low values found during the first pass. Then, it calculates the mean value for all the pixels segmented into background and object. It uses these means as new peaks and calculates new high and low threshold values. Now, it segments that area — again using the new values.

Listing 9.2 next shows the code that implements the adaptive technique with adaptive_threshold_segmentation being the primary function. It is very similar to the peak_threshold_segmentation function in that it uses all that code for its first pass. The second pass starts by calling threshold_and_find_means. This function thresholds the image array into background and object and calculates the mean pixel value for each. The second pass continues by using peaks_high_low to find new threshold values based on the background and object means. Finally, threshold the image using these new threshold values.

Figure 9.20 shows the result of applying the adaptive technique to the image of Figure 9.12. The first pass found the high-and low-threshold values

to be 255 and 166. The second pass thresholded the image array and found the background mean to be 94 and the object mean to be 205. The new threshold values were 255 and 149.

Figure 9.20: Threshold of Figure 9.12 Using Adaptive Technique (High=255 and Low=149)

9.5 An Application Program

Listing 9.3 shows the source code for the *mainseg* program. It has the same form as the other application programs in this text. It interprets the command line parameters, reads the input file, selects the desired segmentation method, and writes the result.

9.6 Conclusions

This chapter introduced image segmentation. This is the first step in locating and labeling the contents of an image. The techniques discussed work on simple images with good contrast and gray level separation between the object and background. We will need additional techniques to attack more complex images.

9.7 Reference

9.1 “Digital Image Processing,” Kenneth R. Castleman, Prentice-Hall, 1979.

Chapter 10

Segmentation via Edges & Gray Shades

10.1 Introduction

This chapter explains image segmentation using edges and gray shades. The previous chapter discussed image segmentation using histograms. That basic technique examined the histogram of an image, transformed the image into a 1-0 image, and "grew" regions. The results were acceptable given the simplicity of the approach.

There are more powerful segmentation techniques that use the edges in an image, grow regions using the gray shades in an image, and use both the edges and gray shades. These techniques work well over a range of images because edges and gray shades are important clues to objects in a scene.

10.2 Segmentation Using Edges & Gray Shades

Figure 10.1 shows the result of using edges to segment an image. The left side shows the output of an edge detector. The right side is the result of grouping the pixels "inside" the edges as objects — a triangle and rectangle. This idea is simple. Detect the edges and group the pixels as objects.

Figure 10.2 illustrates growing objects using the gray shades in an image. Pixels are grouped with a neighboring pixel if their gray shades are close enough. Two pixels are replaced with their average and examination shifts to the neighbors of this two-pixel object. If the gray shades of the neighbors

```
00000000000000000000   --------------------
00000000000000000000   --------------------
00000001000000000000   --------------------
00000010100000000000   -------*------------
00000100010000000000   ------***-----------
00001000001000000000   -----*****----------
00010000000100000000   ----*******---------
00111111111110000000   --------------------
00000000000000000000   --------------------
00000000000000000000   --------------------
00000111111111111000   --------------------
00000100000000001000   ------**********----
00000100000000001000   ------**********----
00000100000000001000   ------**********----
00000100000000001000   ------**********----
00000100000000001000   ------**********----
00000111111111111000   --------------------
00000000000000000000   --------------------
00000000000000000000   --------------------
00000000000000000000   --------------------
```

Figure 10.1: Using Edges to Segment an Image

```
12121212111122211221    11111111111111111111
13121312123121312312    11111111111111111111
12213121212131212122    11111111111111111111
22222218212113112122    11111112111111111111
12111187732121211122    11111122211111111111
12211788872221212211    11111222221111111111
22118778888212212212    11112222222111111111
12121222121222211212    11111111111111111111
21222122111222312123    11111111111111111111
32121321213221322121    11111111111111111111
22121122222121222122    11111111111111111111
22122298989999992212    11111133333333331111
12222298889998992112    11111133333333331111
12212189999898892122    11111133333333331111
12122299998999992122    11111133333333331111
22133389989988982123    11111133333333331111
12122212312321212212    11111111111111111111
13121213212132121321    11111111111111111111
11323212212121112222    11111111111111111111
12212121222222111122    11111111111111111111
```

Figure 10.2: Growing Objects Using Gray Shades

are close enough, they become part of the object and their values adjust the average gray shade of the object. The left side shows the input, and the right side shows the result of growing objects in this manner. The 1s are the background object produced by grouping the 1s, 2s, and 3s. The triangle of 2s is a grouping of the 7s and 8s, and the rectangle of 3s is the 8s and 9s.

Figure 10.3 combines the two techniques. The left side shows a gray shade image with the output of an edge detector (*s) superimposed. The right side shows the result of growing regions using the gray shades while ignoring the detected edges (*s). The result is the three objects produced in Figure 10.2 separated by the edges.

These three simple techniques work well in ideal situations. Most images, however, are not ideal. Real images and image processing routines introduce problems.

```
12121212111122211221   11111111111111111111
13121312123121312312   11111111111111111111
1221312*212131212122   1111111-111111111111
222222*8*12113112122   111111-2-11111111111
12111*877*2121211122   11111-222-1111111111
1221*78887*221212211   1111-22222-111111111
221*8778888*12212212   111-2222222-11111111
12***********2211212   11-----------1111111
21222122111222312123   11111111111111111111
32121321213221322121   11111111111111111111
22121************122   11111------------111
22122*9898999999*212   11111-3333333333-111
12222*9888999899*112   11111-3333333333-111
12212*8999989889*122   11111-3333333333-111
12122*9999899999*122   11111-3333333333-111
22133*8998998898*123   11111-3333333333-111
12122************212   11111------------111
13121213212132121321   11111111111111111111
11323212212121112222   11111111111111111111
12212121222222111122   11111111111111111111
```

Figure 10.3: Growing Objects Using Gray Shades and Edges

10.3 Problems

There are three potential problems using these segmentation techniques: (1) the input image can have too many edges and objects, (2) the edge detectors may not be good enough, and (3) unwanted items ruin region growing.

The input image can be too complicated and have small, unwanted objects. Figure 10.4 shows an aerial image of house trailers, roads, lawns, trees, and a tennis court. The white house trailers are obvious and easy to detect. Other objects (tennis court) have marks or spots that fool the edge detectors and region growing routines. Figure 10.5 shows a house, and Figure 10.6 shows its edges. Segmentation should detect the roof, windows, and door. The bricks, leaves, and shutter slats are real, but small, so unwanted.

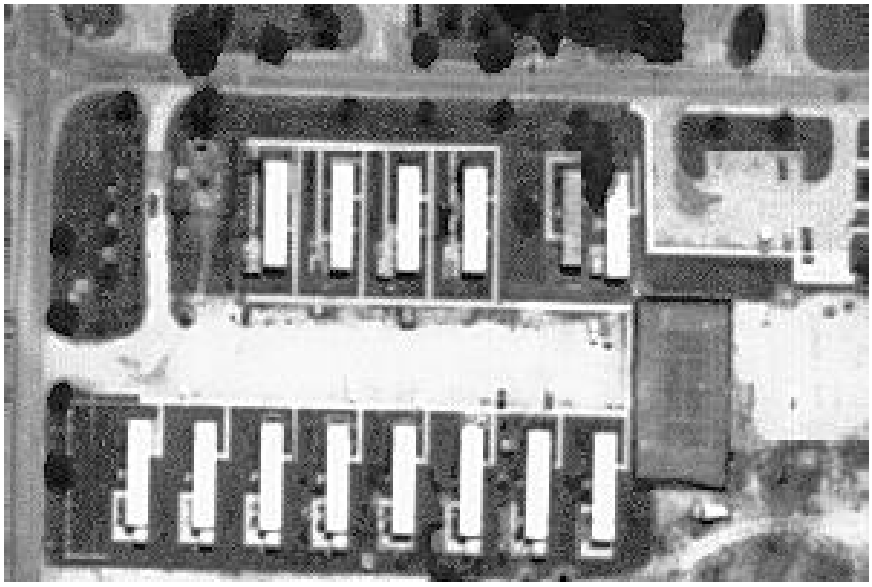


Figure 10.4: Aerial Image of House Trailers

High quality edge detection is essential to use these techniques. Figure 10.8 demonstrates how a small edge detector error leads to a big segmentation error. On the left side of the figure, I poked a small hole in the left edge of the rectangle. The right side shows the terrible segmentation result. Edge detectors do not produce these 1-0 images without thresholding them as explained in Chapter 5. Figure 10.8 shows the result of edge detection on Figure 10.4. Thresholding the strong (bright) and weak (faint) edges produces a clean 1-0 image. This requires a consistent and automatic method

Figure 10.5: House Image

Figure 10.6: Edge Detector Output from Figure 10.5

```
00000000000000000000   ********************
00000000000000000000   ********************
00000001000000000000   *******-************
00000010100000000000   ******-*-***********
00000100010000000000   *****-***-**********
00001000001000000000   ****-*****-*********
00010000000100000000   ***-*******-********
00111111111110000000   **----------*******
00000000000000000000   ********************
00000000000000000000   ********************
00000111111111111000   *****------------***
00000100000000001000   *****-----------***
00000100000000001000   *****-----------***
00000000000000001000   *****************-***
00000100000000001000   *****-----------***
00000100000000001000   *****-----------***
00000111111111111000   *****------------***
00000000000000000000   ********************
00000000000000000000   ********************
00000000000000000000   ********************
```

Figure 10.7: A Small Edge Detector Error Leads to a Big Segmentation Error

to find the threshold point. Detected edges can be too thin and too thick. A surplus of stray, thin edges misleads segmentation, and heavy, extra-thick edges ruin objects. Figure 10.9 shows how the triple thick edges on the left side produce the distorted objects on the right side.

The region-growing algorithm in Chapter 6 must limit the size of objects and exclude unwanted pixels. The house in Figure 10.5 contains objects of widely varying size. "Unwanted" objects may be the bricks or they may be the large objects. There are different situations with different desired results. The left side of Figure 10.10 is a repeat of Figure 10.1 while the right side shows what happens when the region grower mistakes the edges for objects.

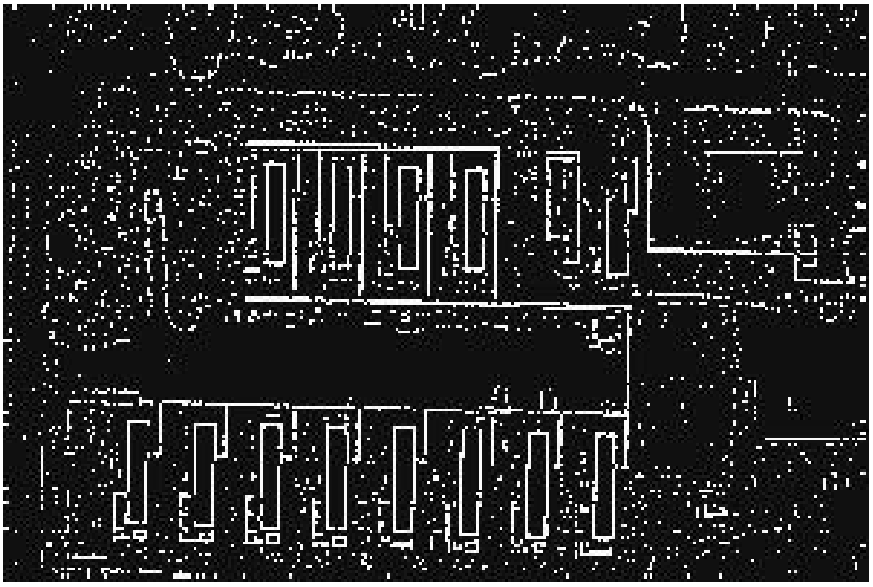


Figure 10.8: Edge Detector Output from Figure 10.4

10.4 Solutions

Some solutions to the object detection problems include preprocessing, better edge detection, and better region growing.

10.4.1 Preprocessing

Preprocessing involves smoothing the input image to remove noise, marks, and unwanted detail. The median filter from Chapter 7, one form of smoothing, sorts the pixels in an nxn area (3x3, 5x5, etc.), and replaces the center pixel with the median value. High- and low-pixel filters, variations of the median filter, sort the pixels in an nxn area and replace the center pixel with either the highest or lowest pixel value.

Figure 10.11 illustrates the median, high-pixel, and low-pixel filters. The left side shows the input — the image section. The right side shows the output for each filter processing a 3x3 area. The median filter removes the spikes of the larger numbers. The high-pixel filter output has many high values because the input has a large number in most of its 3x3 areas. The low-pixel filter output is all 1s because there is a 1 in every 3x3 area of the input.

```
00000000000000000000   --------------------
00000001000000000000   --------------------
00000011100000000000   --------------------
00000111110000000000   --------------------
00001110111000000000   -------*------------
00011100011100000000   ------***-----------
00111111111110000000   --------------------
01111111111111000000   --------------------
11111111111111100000   --------------------
00001111111111111100   --------------------
00001111111111111100   --------------------
00001111111111111100   --------------------
00001110000000011100   -------********-----
00001110000000011100   -------********-----
00001110000000011100   -------********-----
00001111111111111100   --------------------
00001111111111111100   --------------------
00001111111111111100   --------------------
00000000000000000000   --------------------
00000000000000000000   --------------------
```

Figure 10.9: Triple-Thick Edges Distort Objects

```
00000000000000000000   --------------------
00000000000000000000   --------------------
00000001000000000000   -------1------------
00000010100000000000   ------121-----------
00000100010000000000   -----12221----------
00001000001000000000   ----1222221---------
00010000000100000000   ---122222221--------
00111111111110000000   --11111111111-------
00000000000000000000   --------------------
00000000000000000000   --------------------
00000111111111111000   -----333333333333---
00000100000000001000   -----344444444443---
00000100000000001000   -----344444444443---
00000100000000001000   -----344444444443---
00000100000000001000   -----344444444443---
00000100000000001000   -----344444444443---
00000111111111111000   -----333333333333---
00000000000000000000   --------------------
00000000000000000000   --------------------
00000000000000000000   --------------------
```

Figure 10.10: Result of Mistaking Edges for Objects

```
111212163          ---------
111212877          -1112123-
181212123          -1212123-
177212123          -1222122-
116212123          -1222122-     Median Filter Output
111217123          -1112122-
111217123          -1112122-
111217123          -1112122-
111212123          ---------

111212163          ---------
111212877          -8822888-
181212123          -8772888-
177212123          -8872223-
116212123          -7777773-     High-Pixel Filter Output
111217123          -6667773-
111217123          -1227773-
111217123          -1227773-
111212123          ---------

111212163          ---------
111212877          -1111111-
181212123          -1111111-
177212123          -1111111-
116212123          -1111111-     Low-Pixel Filter Output
111217123          -1111111-
111217123          -1111111-
111217123          -1111111-
111212123          ---------
```

Figure 10.11: Output of Median, High-Pixel, and Low-Pixel Filters

Figures 10.12 and 10.13 show how the low-pixel filter reduces the clutter in the edge detector output. Figure 10.12 is the result of the low-pixel filter applied to Figure 10.5. The dark window shutters are larger, and the mortar lines around the bricks are gone. Figure 10.13 is the output of the edge detector applied to Figure 10.12. Compare this to Figure 10.6. The edges around the small objects are gone.

Figure 10.12: Low-Pixel Filtering Performed on Figure 10.5

Listing 10.1 shows the high_pixel and low_pixel subroutines. They loop through the image arrays and place the pixels in the nxn area into the elements array. They sorts the array, and place the highest or lowest pixel value into the output array.

10.4.2 Improved Edge Detection

Accurate edge detectors with automatic thresholding of edges and the ability to thin edges are needed for effective segmentation.

Good edge detection requires a technique for thresholding the edge detector output consistently and automatically. One technique sets the threshold point at a given percentage of pixels in the histogram. This calculates the histogram for the edge detector output and sums the histogram values beginning with zero. When this sum exceeds a given percent of the total, this

Figure 10.13: Edge Detector Output from Figure 10.12

is the threshold value. This method produces consistent results without any manual intervention. A good percentage to use is 50 percent for most edge detectors and images.

Figure 10.14 shows the thresholded edge detector output of Figure 10.4. That is, I processed Figure 10.4 with the edge detector (Figure 10.8 shows the result) and set the threshold at 70 percent. Listing 10.2 shows the find_cutoff_point subroutine that looks through a histogram to find the threshold point. It takes in the histogram and the desired percent and returns the threshold point. This is a simple accumulate-and-compare operation.

The erosion operation [10.1] can solve the final problem with edge detectors, removing extra edges and thinning thick edges. Erosion looks at pixels turned on (edge detector outputs) and turns them off if they have enough neighbors that are turned off. Figures 10.15 and 10.16 illustrate erosion. In Figure 10.15, the left side shows edges (1s) around the triangle and rectangle and several stray edges. The right side shows the result of eroding or removing any 1 that has seven 0 neighbors. In Figure 10.16, the left side shows very thick edges around the triangle and rectangle. The right side shows the result of eroding any 1 that has three 0 neighbors. The edges are thinner, and the objects inside the edges are more accurate.

Listing 10.2 shows the erosion subroutine erode_image_array. The looping

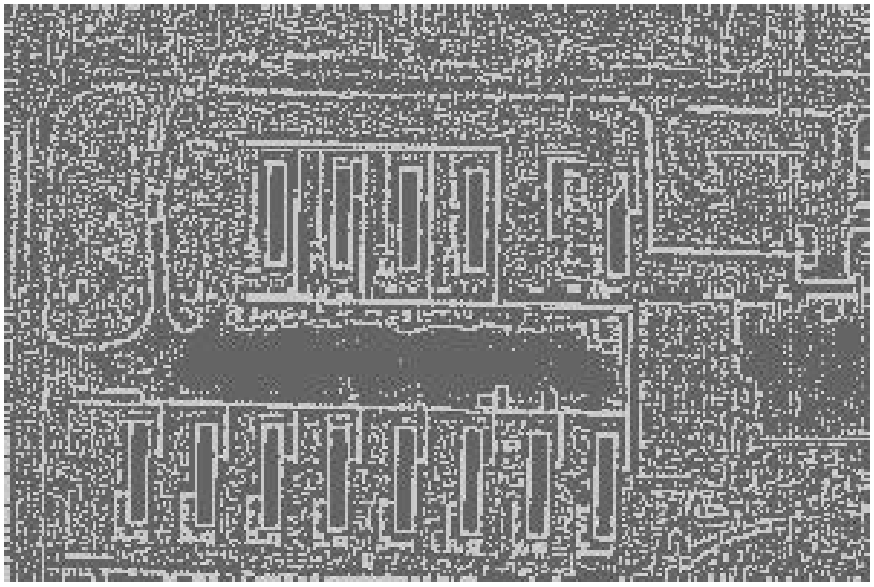


Figure 10.14: Edge Detector Output from Figure 10.4 — Thresholded at 70%

structure examines every pixel in the_image that equals value. It counts the number of neighboring 0 pixels and sets the out_image to zero if this count exceeds the threshold parameter. The threshold parameter controls the erosion. Threshold was six in Figure 10.15 and two in Figure 10.16.

Figure 10.17 shows the result of eroding the thick edges of Figure 10.13. Note how it thinned the thick edges and removed the stray edges in the house, lawn, and tree. The threshold parameter was three for this example.

10.4.3 Improved Region Growing

Accurate region growing is essential to implement the edge and gray shade segmentation techniques. Figure 10.18 shows the region-growing algorithm used in Chapter 9. This worked for binary images containing 0s and a value. If the algorithm found a pixel equal to value, it labeled that pixel and checked its neighbors to see if they also equaled value (step 3).

The region-growing algorithm needs improvements to work with any gray shades, limit the size of regions, and exclude pixels with special values (like edges). Figure 10.19 shows the new region-growing algorithm. The input image g contains gray shades and may contain special pixels equal to FOR-

```
00000000000000000000    00000000000000000000
01000000000000111000    00000000000000010000
00000001000000000000    00000001000000000000
01100010100000000010    00000010100000000000
00000100010000000010    00000100010000000010
00001000001000000010    00001000001000000000
00010000000100000000    00010000000100000000
00111111111110000000    00111111111110000000
00000000000000000110    00000000000000000000
00100000000000000000    00000000000000000000
01000111111111111000    00000111111111111000
00000100000000001000    00000100000000001000
00000100000000001000    00000100000000001000
00000100000000001000    00000100000000001000
00100100000000001000    00000100000000001000
00100100000000001000    00000100000000001000
00000111111111111000    00000111111111111000
00000000000000000000    00000000000000000000
00011000000000000000    00001000000000000000
00000100000000000000    00000000000000000000
```

Figure 10.15: Result of Eroding Stray Edges

```
00000000000000000000    00000000000000000000
00000001000000000000    00000000000000000000
00000011100000000000    00000001000000000000
00000111110000000000    00000011100000000000
00001110111000000000    00000100010000000000
00011100011100000000    00001100011000000000
00111111111110000000    00011110111100000000
01111111111111000000    00111111111110000000
11111111111111100000    00111111111111000000
00001111111111111100    00000111111111111000
00001111111111111100    00000111111111111000
00001111111111111100    00000111000000111000
00001110000000011100    00000110000000011000
00001110000000011100    00000100000000001000
00001110000000011100    00000110000000011000
00001111111111111100    00000110000000111000
00001111111111111100    00000111111111111000
00001111111111111100    00000000000000000000
00000000000000000000    00000000000000000000
00000000000000000000    00000000000000000000
```

Figure 10.16: Eroding Away Thick Edges

Figure 10.17: Result of Eroding the Edges in Figure 10.13

GET_IT. The output image array holds the result. There are three new parameters: diff, min_area, and max_area. diff specifies the allowable difference in gray shade for two pixels to merge into the same object. min_area and max_area specify the limits on the size of objects.

The major differences in the algorithm begin at step 4. Instead of checking if g(i,j) == value, the algorithm performs these checks:
g(i,j) cannot equal FORGET_IT,
output(i,j) must equal zero, and
g(i,j) cannot differ from the target by more than diff.

The first two are simple. The algorithm must exclude certain pixels, so set them to FORGET_IT and ignore them. The output must not be part of an object, so it must be zero.

A third test allows working with gray shade images. In step 3, create a target equal to the average gray shade of the pixels in an object. Group neighboring pixels whose values do not differ by more than the diff parameter. The is_close routine at the bottom of Figure 10.19 tests for this condition. If the pixel g(i,j) is close enough to the target, call the pixel_label_and_check_neighbor routine to add that pixel to the object and check its neighbors. The pixel_label_and_check_neighbor routine updates the target or average gray shade of the object.

```
1. Given an image g with m rows and n columns
      g(i,j) for i=1,m j=1,n
      g(i,j) = value for object
              = 0 for background

2. set g_label=2  this is the label value

3.    for (i=0; i<m; i++)
            scan ith row
            for (j=0; j<n; j++)
                 check jth element
                 stack_empty = true
                 if g(i,j) == value
                       label_and_check_neighbor(g(i,j),g_label)
                 while stack_empty = false do
                       pop element (i,j) off the stack
                       label_and_check_neighbor(g(i,j),g_label)
                 end while
                 g_label = g_label + 1
            end of checking jth element
      end of scanning ith row

4. The End

                  ---------------------------------------

procedure label_and_check_neighbor(g(r,e), g_label)
      g(r,e) = g_label
      for (R=r-1; r<=r+1; R++)
            for (E=e-1; e<=e+1; e++)
                 if g(R,E) ==  value then
                       push (R,E) onto the stack
                       stack_empty = false
                 end if
            end loop over E
      end loop over R
end procedure label_and_check_neighbor
```

Figure 10.18: The Region Growing Algorithm from Chapter 9

```
1. Given:
      Image g with m rows and n columns
      g(i,j) for i=1,m j=1,n
      g(i,j) = gray shades
              = FORGET_IT value is edges are overlayed (optional)
      Image output with m rows and n columns
      output(i,j) for i=1,m j=1,n
      output(i,j) = all zeros
      Parameter diff = allowable difference in gray shades
      Parameter min_area = minimum size of a region allowed
      Parameter max_area = maximum size of a region allowed
2. set g_label=2  this is the label value
3.    for (i=0; i<m; i++)
         scan ith row
         for (j=0; j<n; j++)
           check jth element
           stack_empty = true
           target      = g(i,j)
           sum         = target
           count       = 0
4.         if g(i,j)      != FORGET_IT   AND
              output(i,j) == 0           AND
                 is_close(g(i,j), target, diff)
                 pixel_label_and_check_neighbor(g(i,j), output,
                                      count, sum, target, diff)
                 object_found = 1
           end if
5.         while stack_empty = false do
              pop element (i,j) off the stack
              pixel_label_and_check_neighbor(g(i,j), output,
                                    count, sum, target, diff)
           end while
6.         if(object_found == 1)
              object_found = 0
              if(count >= min_area  AND
                 count <= max_area)
                 g_label = g_label + 1
              else remove object
                 for all output(i,j) = g_label
                       output(i,j) = 0
                       input(i,j)  = FORGET_IT
              end else remove object
              end if
         end of checking jth element
      end of scanning ith row
7. The End
```

Figure 10.19: The Improved Region Growing Algorithm (Part 1)

```
procedure pixel_label_and_check_neighbor(g(r,e), output,
                                         count, sum,
                                         target, diff)

      output(r,e) = g_label
      count       = count+1
      sum         = sum + g(r,e)
      target      = sum/count

      for (R=r-1; r<=r+1; R++)
            for (E=e-1; e<=e+1; e++)

                  if g(R,E)      != FORGET_IT   AND
                     output(R,E) == 0           AND
                     is_close(g(R,E), target, diff)
                        push (R,E) onto the stack
                        stack_empty = false
                  end if

            end loop over E
      end loop over R
end procedure pixel_label_and_check_neighbor

                    ---------------------------------------

procedure is_close(pixel, target, diff)
      if absolute value(pixel - target) < diff
            return 1
      else
            return 0
end procedure is_close
```

Figure 10.19: The Improved Region Growing Algorithm (Part 2)

The new algorithm limits the size of objects in step 6. It tests count (the size of an object) against min_area and max_area. If the object fails the test, you set all pixels of g in the object to FORGET_IT and set all pixels in output to zero. This removes the object from output and eliminates the pixels from any future consideration in g.

I've already discussed how the new algorithm excludes pixels with certain values via the FORGET_IT value. To remove edges from consideration, lay the edge detector output on top of the input image and set to FORGET_IT all pixels corresponding to the edges.

Listing 10.2 shows the source code for the three subroutines outlined in Figure 10.19 (pixel_grow, pixel_label_and_check_neighbor, and is_close). They follow the algorithm closely.

The improved region-growing algorithm is the key to the new techniques. It ignores certain pixels and eliminates objects of the wrong size. These small additions produce segmentation results that are much better than those in Chapter 9.

10.5 The Three New Techniques

Now that I've laid all the groundwork, let's look at the three new techniques.

10.5.1 Edges Only

The edge_region subroutine shown in Listing 10.7 implements this technique. The algorithm is

1. Edge detect the input image
2. Threshold the edge detector output
3. Erode the edges if desired
4. Set the edge values to FORGET_IT
5. Grow the objects while ignoring the edges

Steps 1 through 4 should produce an image like that shown in Figure 10.1. Step 5 grows the objects as outlined by the edges. The edge_region subroutine calls any of the edge detectors from this and previous chapters, the histogram functions from previous chapters, and the find_cutoff_point, erode_image_array, and pixel_grow functions from Listing 10.2.

The edge_type parameter specifies which edge detector to use. min_area and max_area pass through to the pixel_grow routine to constrain the size of

the objects detected. diff passes through to pixel_grow to set the tolerance on gray shades added to an object. diff has little meaning for this technique because the image in which regions are grown contains only 0s and FORGET_IT pixels. The percent parameter passes through to the find_cutoff_point routine to threshold the edge detector output. The set_value parameter is the turned-on pixel in the threshold_image_array and erode_image_array routines. Finally, the erode parameter determines whether to perform erosion on the edge detector output. If erode is not zero, it is the threshold parameter for erode_image_array.

10.5.2 Gray Shades Only

The short gray_shade_region subroutine in Listing 10.2 implements this technique. This subroutine calls the pixel_grow function. pixel_grow does all the work since it handles the gray shade region growing and limits the sizes of the objects. The diff, min_area, and max_area parameters play the same role as in the edge_region routine described above.

10.5.3 Edges and Gray Shade Combined

The technique for combining edges and gray shades is implemented by the edge_gray_shade_region function in Listing 10.2. The algorithm is:

1. Edge detect the input image
2. Threshold the edge detector output
3. Erode the edges if desired
4. Read the input image again
5. Put the edge values on top of the input image setting them to FORGET_IT
6. Grow gray shade regions while ignoring the edges

The differences between edge_region and edge_gray_shade_region are in steps 4 and 5. At this point, edge_gray_shade_region reads the original input image again and overlays it with the detected edges. Step 8 grows gray shade regions while ignoring the detected edges. Steps 1 through 7 generate an image like the left side of Figure 10.3. Step 8 generates the right side of Figure 10.3.

Figures 10.20 through 10.23 illustrate these techniques on the aerial image of Figure 10.4. Figure 10.20 shows the result of the Sobel edge detector after erosion. The edges outline the major objects in the image fairly well.

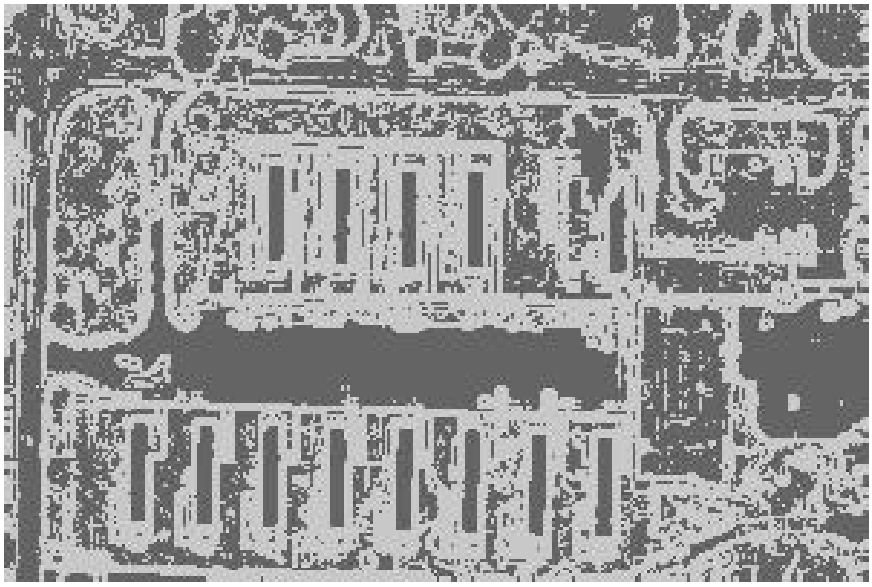


Figure 10.20: Sobel Edge Detector Output from Figure 10.4 (after Erosion)

Figure 10.21 shows the result of the edge-only segmentation of Figure 10.4. It is the result of growing the black regions of Figure 10.20. This is a good segmentation as it denotes the house trailers, roads, trees, and parking lots. This is not just the negative image of Figure 10.20. Regions too small and too large were eliminated.

Figure 10.21 is the result of the gray-shade-only segmentation of Figure 10.4. This segmentation also found the major objects in the image. The combination of edge and gray shade segmentation in Figure 10.26 shows the edges of Figure 10.20 laid on top of the input image of Figure 10.4. Figure 10.23 shows the final result of growing gray shade regions inside these edges. This segmentation has better separation of objects than the gray-shade-only segmentation of Figure 10.21. The edges between the objects caused this spacing.

Which segmentation is best? That is a judgment call. All three segmentations, however, are better than those produced by the simple techniques in Chapter 9.

Figures 10.24, 10.25, and 10.26 show the results of the three techniques applied to the house image of Figure 10.5. The edge-only segmentation of Figure 10.24 is fairly good as it denotes most of the major objects in the image. The gray-shade-only result in Figure 10.25 is not very good because

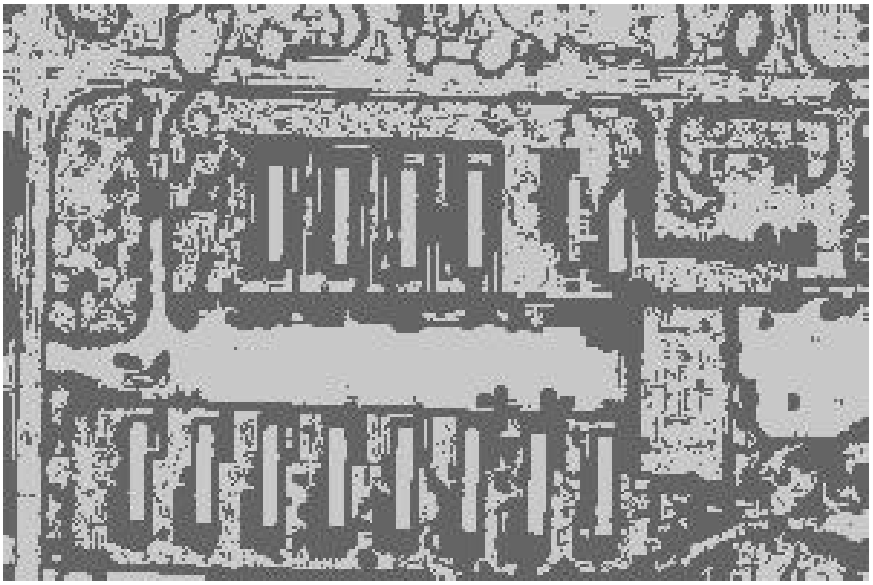


Figure 10.21: Result of Edge-Only Segmentation of Figure 10.4

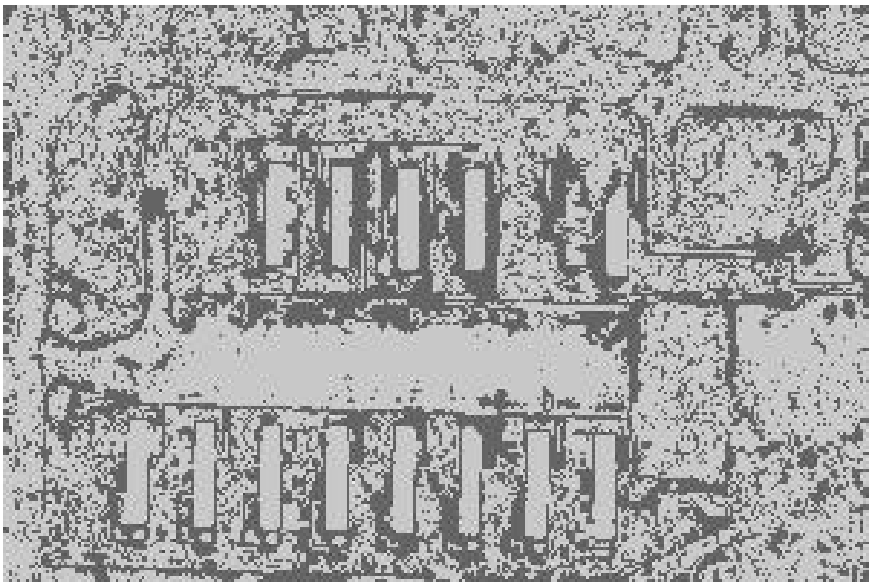


Figure 10.22: Result of Gray-Shade-Only Segmentation of Figure 10.4

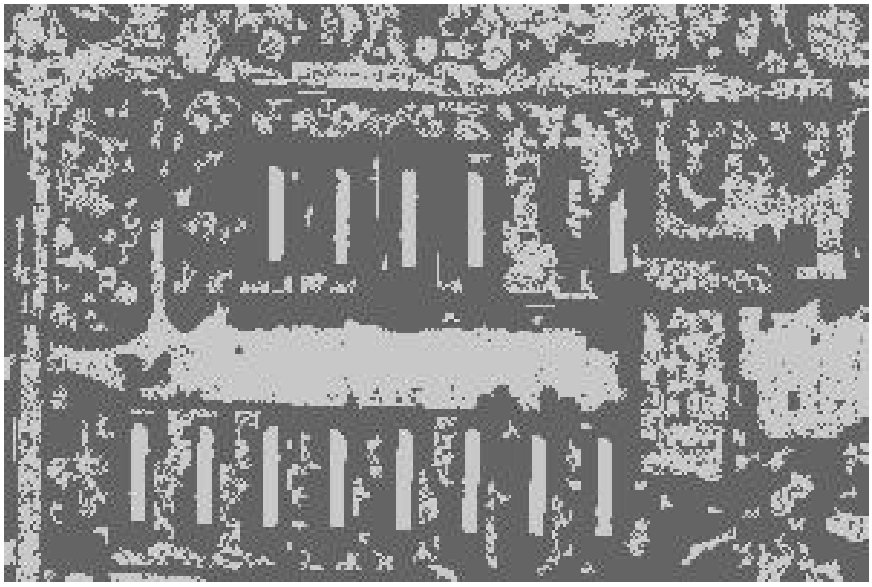


Figure 10.23: Result of Edge and Gray Shade Segmentation of Figure 10.4

all the objects are right next to each other and hard to distinguish. The combination segmentation in Figure 10.26 is an excellent result. It detected objects not found in the edge-only technique and also eliminated many of the unwanted bricks.

10.6 Integrating the New Techniques

Listing 10.3 shows the *main2seg* program for segmenting entire images using the new techniques. It is command-line driven and calls the functions given in the previous listings. It follows the same pattern as the *mainseg* program of chapter 9 and other programs shown in this text.

10.7 Conclusions

This chapter described three powerful image segmentation techniques that work on complicated images. The techniques, however, are only combinations of existing tools and tricks. Given different images, you might have used different combinations of tools. Experiment, try different combinations, and modify existing tools to create new ones.

Figure 10.24: Result of Edge-Only Segmentation of Figure 10.5

Figure 10.25: Result of Gray-Shade-Only Segmentation of Figure 10.5

Figure 10.26: Result of Edge and Gray Shade Segmentation of Figure 10.5

10.8 Reference

10.1 "The Image Processing Handbook, Third Edition," John C. Russ, CRC Press, 1999.

Chapter 11

Manipulating Shapes

11.1 Introduction

This chapter will discuss manipulating shapes. The last two chapters discussed image segmentation. Segmentation took an image and produced a binary output showing the objects of interest. This chapter will describe several techniques for taking those objects and improving their appearance.

11.2 Working with Shapes

A major goal of image processing is to improve the appearance of an image. Figure 11.1 shows an aerial image, and Figure 11.2 a segmentation of it. Figure 11.3 shows a house, and Figure 11.4 a segmentation of it. These are good segmentations, but they have problems.

Segmentation results have "holes" in them. The roof in Figure 11.4 should be solid, but has holes. Larger holes can even break objects. The opposite can also be true, as segmentation can join separate objects. Segmentation results often have little or no meaning. The solid objects resemble blobs and are hard to interpret.

The answer to these problems is morphological filtering or manipulating shapes. Useful techniques include erosion and dilation, opening and closing, outlining, and thinning and skeletonization .

These techniques work on binary images where the object equals a value and the background is zero. Figure 11.5 shows a binary image with the background equal to zero and the object equal to 200. All the figures in the

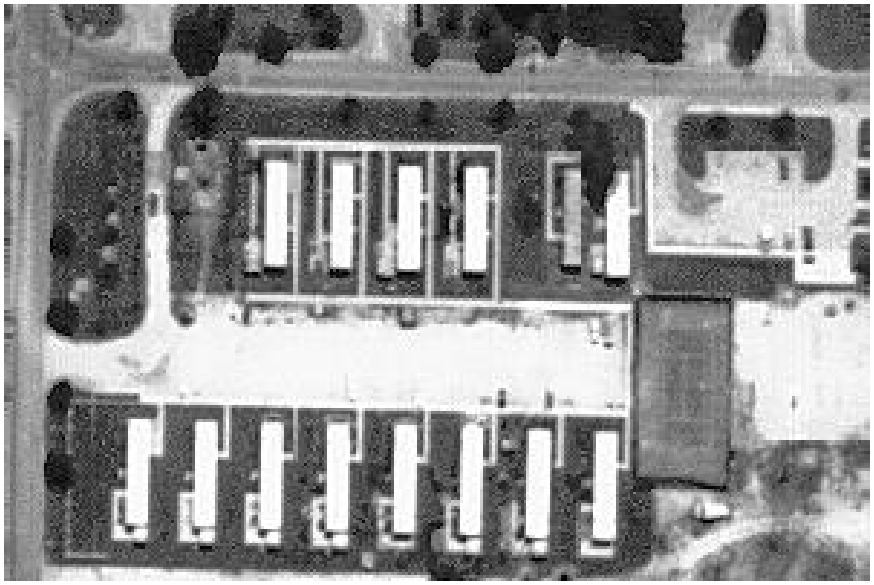


Figure 11.1: Aerial Image

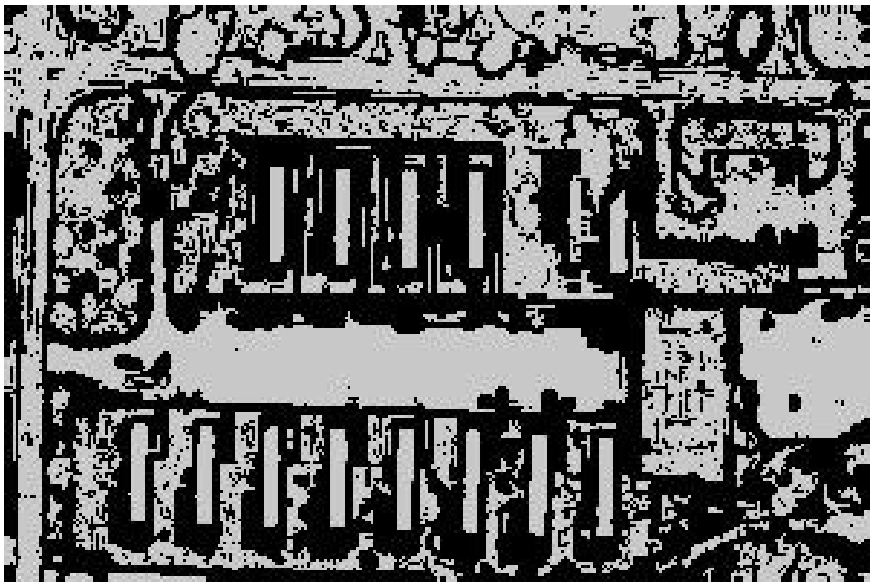


Figure 11.2: Segmentation of Aerial Image

Figure 11.3: House Image

Figure 11.4: Segmentation of House Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 11.5: A Binary Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	200	200	200	200	0	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	0	200	200	200	200	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 11.6: The Result of Eroding Figure 11.5

chapter will use the same format.

11.3 Erosion and Dilation

The erosion and dilation operations make objects smaller and larger. These operations are valuable in themselves and are the foundation for the opening and closing operations. Erosion, discussed briefly in Chapter 10, makes an object smaller by removing or eroding away the pixels on its edges. Figure 11.6 shows the result of eroding the rectangle in Figure 11.5.

Dilation makes an object larger by adding pixels around its edges. Figure

```
0   0   0   0   0   0   0   0   0   0
0 *** *** *** *** *** *** *** ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** 200 200 200 200 200 200 ***   0
0 *** *** *** *** *** *** *** ***   0
0   0   0   0   0   0   0   0   0   0
```

Figure 11.7: The Result of Dilating Figure 11.5

11.7 shows the result of dilating the rectangle in Figure 11.5. I set any zero pixel that was next to a 200 pixel (shown as asterisks).

There are two general techniques for erosion and dilation. The technique introduced in Chapter 10 employs a threshold [11.1]. Another technique uses masks to erode and dilate in desired directions [11.2].

The threshold technique looks at the neighbors of a pixel and changes its state if the number of differing neighbors exceeds a threshold. Listing 11.1 shows the erosion and dilation routines that use this method. The loops in the erosion routine examine every pixel equal to value in the_image. The loops count the number of zero neighbors and set the pixel in question to zero if the count exceeds the threshold parameter. Figure 11.6 used a threshold parameter of three. Compare this to Figure 11.8 (threshold = two).

The loops in the dilation routine do the opposite. They count the value pixels next to a zero pixel. If the count exceeds the threshold parameter, set the zero pixel to value. The dilation in Figure 11.7 used threshold = three, while Figure 11.9 used threshold = two.

The masking technique [11.2] lays an nxn (3x3, 5x5, etc.) array of 1s and 0s on top of an input image and erodes or dilates the input. With masks you can control the direction of erosion or dilation. Figure 11.10 shows four 3x3 masks (5x5, 7x7, etc. masks are other possibilities). The first two masks modify the input image in the vertical or horizontal directions while the second two perform in both directions.

Figure 11.11 shows the results of dilating the object of Figure 11.5 using the four masks of Figure 11.10. The procedure is:

```
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0 200 200 200 200   0   0   0
  0   0   0 200 200 200 200   0   0   0
  0   0   0 200 200 200 200   0   0   0
  0   0   0 200 200 200 200   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
```

Figure 11.8: The Result of Eroding Figure 11.5 Using a Threshold of 2

```
  0   0   0   0   0   0   0   0   0   0
  0   0   0 *** *** *** ***   0   0   0
  0   0 200 200 200 200 200 200   0   0
  0 *** 200 200 200 200 200 200 ***   0
  0 *** 200 200 200 200 200 200 ***   0
  0 *** 200 200 200 200 200 200 ***   0
  0 *** 200 200 200 200 200 200 ***   0
  0   0 200 200 200 200 200 200   0   0
  0   0   0 *** *** *** ***   0   0   0
  0   0   0   0   0   0   0   0   0   0
```

Figure 11.9: The Result of Dilating Figure 11.5 Using a Threshold of 2

```
vertical mask   horizontal mask
   0 1 0           0 0 0
   0 1 0           1 1 1
   0 1 0           0 0 0

horizontal and vertical masks
   0 1 0           1 1 1
   1 1 1           1 1 1
   0 1 0           1 1 1
```

Figure 11.10: Four 3x3 Masks

Vertical Dilation Only

0	0	0	0	0	0	0	0	0	0
0	0	***	***	***	***	***	***	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	***	***	***	***	***	***	0	0
0	0	0	0	0	0	0	0	0	0

Horizontal Dilation Only

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Dilation in Both Directions

0	0	0	0	0	0	0	0	0	0
0	0	***	***	***	***	***	***	0	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	0	***	***	***	***	***	***	0	0
0	0	0	0	0	0	0	0	0	0

Dilation in Both Directions

0	0	0	0	0	0	0	0	0	0
0	***	***	***	***	***	***	***	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	200	200	200	200	200	200	***	0
0	***	***	***	***	***	***	***	***	0
0	0	0	0	0	0	0	0	0	0

Figure 11.11: The Result of Dilating Figure 11.5 with the Four Masks of Figure 11.9

1. Place the 3x3 mask on the object so that the center of the 3x3 array lies on the edge of the object.
2. Place a 200 everywhere a one from the mask lies.

The object in the first part of Figure 11.11 has been dilated, or stretched, vertically. The second result is a horizontal dilation. The third and fourth show dilation in both directions. These last two differ in dilating the corners of the object.

Mask erosion is the same, but opposite. It lays the 3x3 mask on the image so that the center of the array is on top of a zero. If any of the 1s in the mask overlap a 200 in the image, set the 200 to zero. Vertical mask erosion removes the top and bottom rows from an object. Horizontal mask erosion removes the left and right columns, and the other masks remove pixels from all edges.

Listing 11.1 shows the routines for mask erosion and dilation. mask_dilation copies the correct directional mask specified by the mask_type parameter and goes into the looping code. The loop moves through the input image and lays the 3x3 mask on top of every pixel in the image. The inner loops examine those places where the 3x3 mask equals one. If the_image is greater than one (non-zero) at that place, set max to the input image value. After exiting the loop, set the out_image to max. This changes zero pixels to value and enlarges or dilates an object in the_image.

The mask_erosion routine performs the opposite function. Its inner loops look at those places where the 3x3 mask is one and try to find pixels in the_image that are less than min (that are zero). If there are any zeros in this part of the_image, set out_image to zero. This removes value pixels, makes them zeros, and erodes an object in the_image.

Figure 11.12 illustrates directional dilation. The left section shows the segmentation of the house image. The center section shows dilating with a vertical mask, and the right section shows dilating with a horizontal mask.

11.4 Opening and Closing

Opening and closing help separate and join objects. They are powerful operators that are simple combinations of erosion and dilation. opening spaces objects that are too close together, detaches objects that are touching and should not be, and enlarges holes inside objects. The first part of Figure 11.13 shows two objects joined by a "thread." The second part shows how

Figure 11.12: Examples of Masked Vertical and Horizontal Dilations

opening eliminated the thread and separated the two objects. The rest of the figure shows how opening enlarged a desired hole in an object.

Opening involves one or more erosions followed by one dilation. Eroding the object of Figure 11.13 twice erases the thread. A dilation enlarges the two objects back to their original size, but does not re-create the thread. The left side of Figure 11.14 is a segmentation of the house image from Chapter 10. The right side is the result of opening (three erosions followed by one dilation). Although excessive, it shows how opening spaces the major objects.

Closing joins broken objects and fills in unwanted holes in objects. The first part of Figure 11.15 shows two objects that should be joined to make a line. The second part shows how closing removes the break in the line. The last two parts of Figure 11.15 show how closing fills in unwanted holes in objects.

Closing involves one or more dilations followed by one erosion. Dilating the top part of Figure 11.14 twice enlarges the two objects until they join. An erosion thins them back to their original width. Dilating the third part of Figure 11.15 twice makes the box bigger and eliminates the hole. Eroding shrinks the box back to its initial size.

Listing 11.1 shows the routines that perform opening and closing. They call the mask erosion and dilation routines, but calling the threshold erosion and dilation routines would work just as well (homework for the reader). opening calls mask_dilation one or more times and mask_erosion once. closing calls mask_erosion one or more times and mask_dilation once. These are

```
   Two objects joined by a thread
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200

   Opening separates them
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200
200 200 200 200   0   0 200 200 200 200

   An object with a single small hole in it
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200   0 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200

   Opening enlarges the hole
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200   0   0   0 200 200 200 200
200 200 200   0   0   0 200 200 200 200
200 200 200   0   0   0 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
```

Figure 11.13: Two Objects Joined by a Thread, Separated by opening and a Hole Enlarged by opening

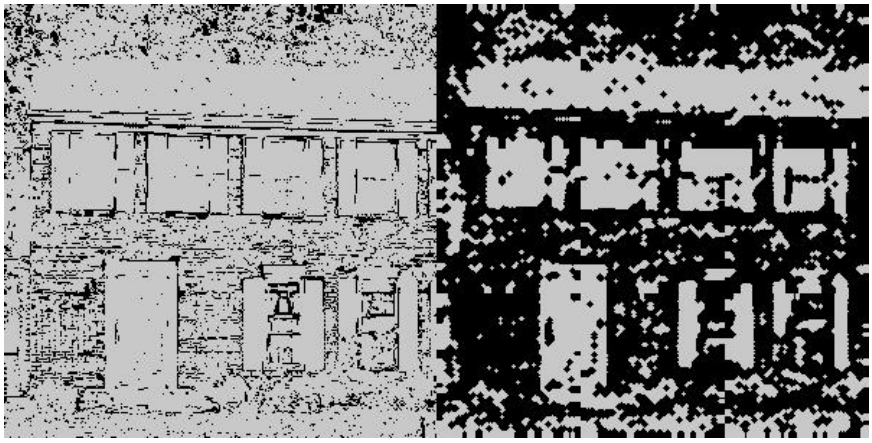


Figure 11.14: A Segmentation and the Result of Opening

simple, yet powerful routines.

11.5 Special Opening and Closing

The opening and closing operators work well, but sometimes produce undesired side effects. closing merges objects, but sometimes merges objects that it shouldn't. Figure 11.16 shows such a case. Figure 11.17 shows the result of closing applied to Figure 11.2. closing closed the holes in the objects, but also joined distinct objects. This distorted the segmentation results. opening enlarges holes in objects, but can break an object. Figure 11.18 shows a case where opening broke an object and eliminated half of it.

The answer is special opening and closing routines that avoid these problems. Figure 11.19 shows the desired result of such special routines that open and close objects, but do not join or break them.

The first difficulty to overcome is what I call the 2-wide problem. In opening, you erode an object more than once, and an object that is an even number of pixels wide can disappear. The first part of Figure 11.20 shows a 2-wide object. The second part shows the object after one erosion, and the third part shows it after two erosions. The object will disappear entirely after several more erosions.

A solution to the 2-wide problem is my own variation of the grass fire wavefront technique [11.3]. My technique scans across the image from left to right looking for a 0 to value transition. When it finds one, it examines the

```
 A broken line
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
200 200 200 200   0 200 200 200 200 200
200 200 200 200   0 200 200 200 200 200
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0

 Closing joins the broken line
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 200 200 200 200 200
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0

 An object with a hole in it
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200   0 200 200 200   0   0
  0   0 200 200   0 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0

 Closing fills the hole
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0 200 200 200 200 200 200   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
```

Figure 11.15: Two Objects that Should be Joined, How closing Removes the Break and Fills Unwanted Holes

```
 Two separate objects
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0

 Closing joins the objects (unwanted)
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0 200 200 200 200 200   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
```

Figure 11.16: An Unwanted Merging of Two Objects

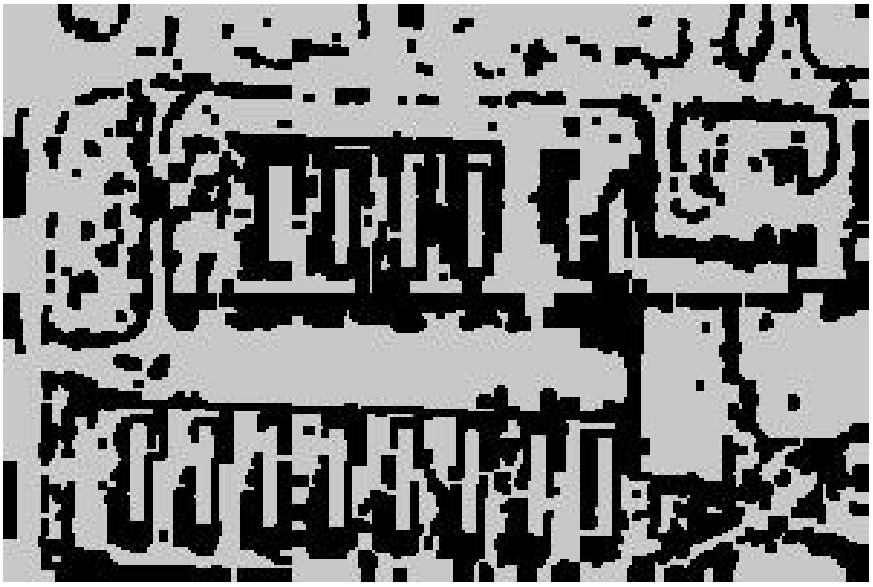


Figure 11.17: Closing of Segmentation in Figure 11.2

value pixel to determine if removing it will break an object. If removal does not break the object, it sets the pixel to 0 and continues scanning. Next, it scans the image from right to left and does the same operation. Then it scans from top to bottom, and finally from bottom to top. The different scans will not erode away an object that is 2-wide.

The key to special opening is not breaking the object. One solution places the pixel in question in the center of a 3x3 array. Find every value pixel next to the center pixel. Do all of those pixels have value neighbors other than the center pixel? If yes, erode or remove the center pixel. If no, removing the center pixel will break the object. The top part of Figure 11.21 shows cases where removing the center pixel will break the object. The bottom part shows cases where removing the center pixel will not break the object. Here, every 200 has a 200 neighbor other than the center pixel.

A similar problem in special closing is not joining two separate objects when dilating or setting a pixel. One solution is to place the pixel in question in the center of a 3x3 array. Grow objects in this array and check if the center pixel has neighbors whose values differ as shown in Chapter 9. If their values differ, do not set the center pixel because this will join different objects. The top part of Figure 11.22 shows 3x3 arrays and the results of growing objects. The center pixel has neighbors that are parts of different objects, so do not

```
 Object with a hole in it
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0 200 200 200 200 200 200   0   0
0   0 200 200 200 200 200 200   0   0
0   0 200 200   0 200 200 200   0   0
0   0 200 200   0 200 200 200   0   0
0   0 200 200 200 200 200 200   0   0
0   0 200 200 200 200 200 200   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0

 Opening removes the hole but breaks the object
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0 200 200 200   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
```

Figure 11.18: An Unwanted Splitting of an Object

```
 Special closing does not join objects
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0 200 200   0 200 200   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0

 Special opening does not break object
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0 200 200 200 200   0   0   0
0   0 200 200 200 200 200 200   0   0
0   0 200 200   0 200 200 200   0   0
0   0 200 200   0 200 200 200   0   0
0   0 200 200 200 200 200 200   0   0
0   0   0 200 200 200 200   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
```

Figure 11.19: Result of Special Routines that Open and Close Objects but do not Join or Break Them

```
  A 2-Wide Object
0   0 200 200   0   0
0   0 200 200   0   0

  The Object After One Erosion
0   0   0 200   0   0
0   0 200 200   0   0

  The Object After Two Erosions
0   0   0   0   0   0
0   0 200 200   0   0
```

Figure 11.20: Result of Opening of a 2-Wide Object

```
  Cases where you cannot erode center pixel
  0 200   0     200   0 200     200 200   0
  0 200   0       0 200   0       0 200   0
  0 200   0       0   0   0       0   0 200

  Cases where you can erode the center pixel
200 200 200     200   0   0       0 200   0
  0 200   0     200 200   0     200 200   0
  0   0   0       0   0   0       0   0   0
```

Figure 11.21: Cases Where Objects Can and Cannot be Eroded

```
  Cases where you cannot dilate the center pixel
200   0 200     200   0   0       0 200   0
200   0 200       0   0   0     200   0   0
200   0 200     200   0   0       0   0 200

  1   0   2       1   0   0       0   1   0
  1   0   2       0   0   0       1   0   0
  1   0   2       1   0   0       0   0   2

  Cases where you can dilate the center pixel
200 200 200       0 200   0       0 200 200
200   0   0     200   0   0     200   0 200
200 200 200       0   0   0       0   0 200

  1   1   1       0   1   0       0   1   1
  1   0   0       1   0   0       1   0   1
  1   1   1       0   0   0       0   0   1
```

Figure 11.22: Cases that do and do not Require a Special Closing Routine

set the center pixel. The bottom part of Figure 11.22 shows another set of 3x3 arrays. Here, the non-zero neighbors of the center pixel all have the same value, so setting the center pixel is alright.

The source code to implement special opening and special closing, shown in Listing 11.2, is basic but long. The special_opening routine calls thinning (instead of erosion — thinning is discussed in a later section) one or more times before calling dilation once. thinning works around the 2-wide problem while performing basic threshold erosion. thinning has four sections — one for each scan (left to right, right to left, top to bottom, and bottom to top) recounted earlier. Whenever thinning finds a pixel to remove, it calls can_thin to prevent breaking an object. can_thin checks the non-zero neighbors of the center pixel. If every non-zero pixel has a non-zero neighbor besides the center pixel, can_thin returns a one, else it returns a zero.

The special_closing routine calls dilate_not_join one or more times before calling erosion once. dilate_not_join uses the basic threshold technique for dilation and calls can_dilate to prevent joining two separate objects. can_dilate grows objects in a 3x3 array and checks if the center pixel has neighbors with

different values. If it does, the neighbors belong to different objects, so it returns a zero. can_dilate grows objects like the routines in Chapters 9 and 10. can_dilate calls little_label_and_check which resembles routines described in those two chapters.

Figure 11.23 shows the result of special closing. Compare this with Figures 11.2 and 11.17. Figure 11.2, the original segmentation, is full of holes. Figure 11.17 closed these holes, but joined objects and ruined the segmentation result. Figure 11.23 closes the holes and keeps the segmentation result correct by not joining the objects.

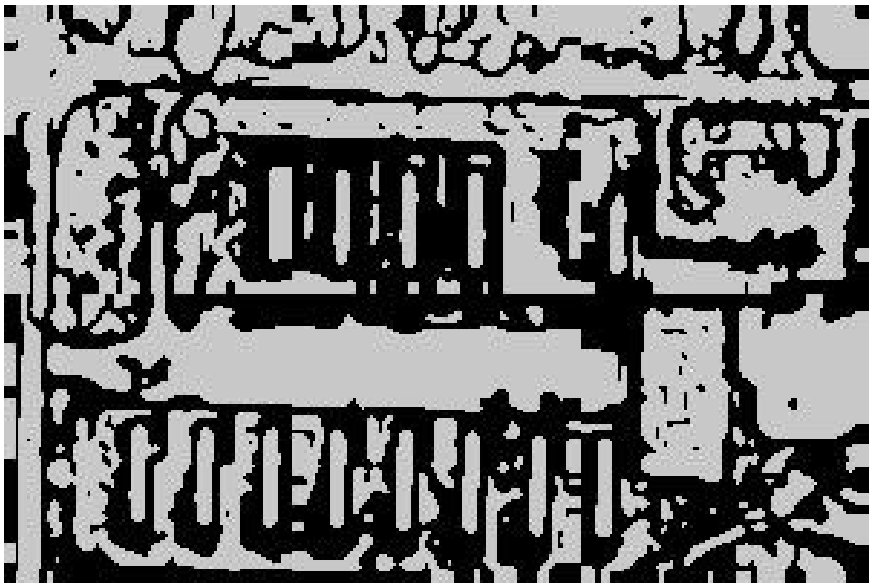


Figure 11.23: Special Closing of Segmentation of Figure 11.2

Figures 11.24 and 11.25 show how to put everything together to improve segmentation results. Figure 11.24 shows the outcome of eroding the segmentation result of Figure 11.4. Applying special closing to Figure 11.24 produces Figure 11.25. Compare Figures 11.4 and 11.25. Figure 11.25 has all the major objects cleanly separated without holes.

11.6 Outlining

Outlining is a type of edge detection. It only works for zero-value images, but produces better results than regular edge detectors. Figure 11.26 shows

Figure 11.24: Erosion of Segmentation in Figure 11.4

Figure 11.25: Special Closing of Figure 11.24

the exterior outline of the objects in Figure 11.4.

Figure 11.26: Outline of Segmentation in Figure 11.4

Outlining helps understand an object. Figures 11.27 and 11.28 show the interior and exterior outlines of objects. Outlining zero-value images is quick and easy with erosion and dilation. To outline the interior of an object, erode the object and subtract the eroded image from the original. To outline the exterior of an object, dilate the object and subtract the original image from the dilated image. Exterior outlining is easiest to understand. Dilating an object makes it one layer of pixels larger. Subtracting the input from this dilated, larger object yields the outline.

Listing 11.1 shows the source code for the interior_outline and exterior_outline operators. The functions call the mask_erosion and mask_dilation routines. They could have called the threshold erosion and dilation routines (homework for the reader). The interior_outline routine erodes the input image and subtracts the eroded image from the original. The exterior_outline routine dilates the input image and subtracts the input image from the dilated image.

200	200	200	200	200	200	200	200	200	200
200	200	200	200	200	200	200	200	200	200
200	200	0	0	0	0	0	0	200	200
200	200	0	0	0	0	0	0	200	200
200	200	0	0	0	0	0	0	200	200
200	200	0	0	0	0	0	0	200	200
200	200	0	0	0	0	0	0	200	200
200	200	0	0	0	0	0	0	200	200
200	200	200	200	200	200	200	200	200	200
200	200	200	200	200	200	200	200	200	200

0	0	0	0	0	0	0	0	0	0
0	200	200	200	200	200	200	200	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	200	200	200	200	200	200	200	0
0	0	0	0	0	0	0	0	0	0

Figure 11.27: The Interior Outline of an Object

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	200	200	200	200	200	200	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	200	200	200	200	200	200	200	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	0	0	0	0	0	0	200	0
0	200	200	200	200	200	200	200	200	0
0	0	0	0	0	0	0	0	0	0

Figure 11.28: The Exterior Outline of an Object

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	200	200	200	200	200	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11.29: Thinning a Rectangle until it is One Pixel Wide

11.7 Thinning and Skeletonization

Thinning is a data reduction process that erodes an object until it is one-pixel wide, producing a skeleton of the object. It is easier to recognize objects such as letters or silhouettes by looking at their bare bones. Figure 11.29 shows how thinning a rectangle produces a line of pixels.

There are two basic techniques for producing the skeleton of an object: basic thinning and medial axis transforms.

Thinning erodes an object over and over again (without breaking it) until it is one-pixel wide. Listing 11.2 contains the thinning routine. The special_opening routine used thinning to erode objects without breaking them.

In that context, the once_only parameter of thinning is one, so that it would erode an image only one time. Setting once_only to zero causes thinning to keep eroding until the objects in the image are all one-pixel wide.

This basic thinning technique works well, but it is impossible to re-create the original object from the result of thinning. Re-creating the original requires the medial axis transform.

The medial axis transform finds the points in an object that form lines down its center, that is, its medial axis. It is easier to understand the medial axis transform if you first understand the Euclidean distance measure (don't you love these big terms that really mean very simple things?). The Euclidean distance measure is the shortest distance from a pixel in an object to the edge of the object. Figure 11.30 shows a square, its Euclidian distance measure (distance to the edge), and its medial axis transform.

The medial axis transform consists of all points in an object that are minimally distant to more than one edge of the object. Every pixel in the bottom of Figure 11.30 was the shortest distance to two edges of the object. The advantage of the medial axis transform is you can re-create the original object from the transform (more homework). Figure 11.31 shows a rectangle (from Figure 11.29) and its medial axis transform. Figure 11.32 shows a block letter A, and going clockwise, the result of exterior outline, medial axis transform, and thinning.

Listing 11.2 shows the source code to implement the Euclidean distance measure and the medial axis transform. edm calculates the Euclidean distance measure. It loops through the image and calls distance_8 to do most of the work. distance_8 has eight sections to calculate eight distances from any value pixel to the nearest zero pixel. distance_8 returns the shortest distance it found.

The functions mat and mat_d calculate the medial axis transform in a similar manner. mat loops through the image and calls mat_d to do the work. mat_d calculates the eight distances and records the two shortest distances. If these two are equal, the pixel in question is minimally distant to two edges, is part of the medial axis transform, and causes mat_d to return value.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	2	2	2	2	2	2	2	2	2	2	2	1	0	0
0	0	1	2	3	3	3	3	3	3	3	3	3	2	1	0	0
0	0	1	2	3	4	4	4	4	4	4	4	3	2	1	0	0
0	0	1	2	3	4	5	5	5	5	5	4	3	2	1	0	0
0	0	1	2	3	4	5	6	6	6	5	4	3	2	1	0	0
0	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	0
0	0	1	2	3	4	5	6	6	6	5	4	3	2	1	0	0
0	0	1	2	3	4	5	5	5	5	5	4	3	2	1	0	0
0	0	1	2	3	4	4	4	4	4	4	4	3	2	1	0	0
0	0	1	2	3	3	3	3	3	3	3	3	3	2	1	0	0
0	0	1	2	2	2	2	2	2	2	2	2	2	2	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11.30: A Square, its Euclidean Distance Measure, and its Medial Axis Transform (Part 1)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	200	0	0	0	0	0	0	0	0	0	0	0	200	0	0
0	0	0	200	0	0	0	0	0	0	0	0	0	200	0	0	0
0	0	0	0	200	0	0	0	0	0	0	0	200	0	0	0	0
0	0	0	0	0	200	0	0	0	0	0	200	0	0	0	0	0
0	0	0	0	0	0	200	0	0	0	200	0	0	0	0	0	0
0	0	0	0	0	0	0	200	0	200	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	200	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	200	0	200	0	0	0	0	0	0	0
0	0	0	0	0	0	200	0	0	0	200	0	0	0	0	0	0
0	0	0	0	0	200	0	0	0	0	0	200	0	0	0	0	0
0	0	0	0	200	0	0	0	0	0	0	0	200	0	0	0	0
0	0	0	200	0	0	0	0	0	0	0	0	0	200	0	0	0
0	0	200	0	0	0	0	0	0	0	0	0	0	0	200	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11.30: A Square, its Euclidean Distance Measure, and its Medial Axis Transform (Part 2)

11.8 A Shape Operations Application Program

Listing 11.3 shows application program *mainsk* that ties together all the routines that manipulate shapes. It can call 14 different operations. The format of *mainsk* is the same as the other applications presented in this text.

11.9 Conclusions

This chapter discussed shape operations or morphological filters. These techniques help you improve the appearance of segmentation results. They are also useful for other situations. As with all the image processing operators in this system, you must experiment. Try the techniques and tools in different combinations until you find what works for the image or class of images at hand.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	200	0	0	0	0	0	0	0	0	0	0	0	200	0	0
0	0	0	200	0	0	0	0	0	0	0	0	0	200	0	0	0
0	0	0	0	200	0	0	0	0	0	0	0	200	0	0	0	0
0	0	0	0	0	200	0	0	0	0	0	200	0	0	0	0	0
0	0	0	0	0	0	200	200	200	200	200	0	0	0	0	0	0
0	0	0	0	0	200	0	0	0	0	0	200	0	0	0	0	0
0	0	0	0	200	0	0	0	0	0	0	0	200	0	0	0	0
0	0	0	200	0	0	0	0	0	0	0	0	0	200	0	0	0
0	0	200	0	0	0	0	0	0	0	0	0	0	0	200	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11.31: A Rectangle and its Medial Axis Transform

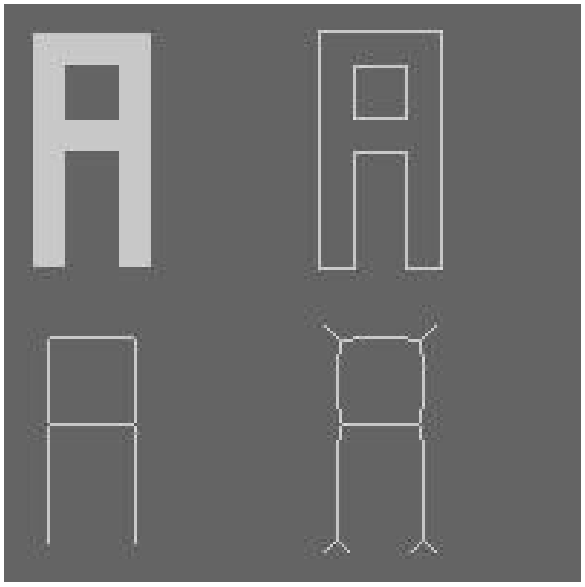


Figure 11.32: (Clockwise from Upper Left) A, Its Outline, Medial Axis Transform, and Thinning

11.10 References

11.1 "The Image Processing Handbook, Third Edition," John C. Russ, CRC Press, 1999.
11.2 "Computer Imaging Recipes in C," Harley R. Myler, and Arthur R. Weeks, Prentice Hall Publishing, Englewood Cliffs, New Jersey, 1993.
11.3 "Vision in Man and Machine," Martin D. Levine, McGraw-Hill, 1985.

Chapter 12

Boolean and Overlay Operations

12.1 Introduction

This chapter will discuss Boolean and overlay operations. These operations are useful for combining images in interesting ways. They are also good for creating special effects in images. The goal is to combine two images to produce a third that has features of the two inputs. The Boolean operations use the functions of Boolean algebra. The overlay operations lay selected pixels from one image on top of another. These are similar to the image addition and subtraction of Chapter 8.

12.2 Boolean Operations

The Boolean operations execute the basic functions from Boolean algebra. Figure 12.1 shows the truth table for these operations. The output of the AND is one when both inputs are one. The output of the OR is one if either of the inputs are one. The output of the exclusive or (XOR) is one if one but not both of the inputs is one. The NAND is the opposite of the AND, the NOR is opposite of the OR, and the NOT reverses the input.

It is a simple matter to extend Boolean operations for gray scale images. Replace the 1s in the A and B columns of the truth table with any non-zero value. Replace the 1s in the output columns with the non-zero value from the A column. For example, if the A image contains all 200s and the B image

a	b	a AND b	a OR b	a XOR b	a NAND b	a NOR b	NOT a
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	1
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	0

Figure 12.1: Existing Standard TIFF Tags

contains all 100s, the output of A AND B will be all 200s.

Listing 12.1 shows the subroutines that implement the Boolean operations. Each of the routines (and_image, or_image, xor_image, nand_image, nor_image, not_image) follows the usual pattern. They combine the input image arrays using the truth table and return the result. These are simple, yet powerful routines.

Listing 12.2 shows the *boolean* program. This program allows the user to apply any of the Boolean operators to images. It follows the same pattern as all main programs in this text.

12.3 Applications of Boolean Operations

Let's look at two applications of the Boolean operations: masking and labeling images. Masking places the gray shades of an image on top of a binary image derived from it. Figure 12.2 shows an aerial image, and Figure 12.3 shows a segmentation of it from Chapter 10. This is a fairly accurate segmentation, but it is difficult to correlate the white shapes to objects in the image. Is the large rectangle to the left grass or a parking lot? One way of determining the source of the objects is to mask the original over the segmentation using the AND operation. Figure 12.4 shows the result of masking (ANDing). It is easy to see that the large rectangle is a tennis court, some of the roads are cement (white), and others are asphalt (gray).

Another use of the Boolean operations is to create and place labels on top of images. Listing 12.3 shows an image labeling program called *ilabel*. This program writes simple 9x7 block letters to an image file. The user calls the program by giving an output image name, a line and element in the image, and the text to go in the image. For example,
ilabel a.tif 10 20 adam
places the letters ADAM in the image a.tif starting in the tenth row, twentieth

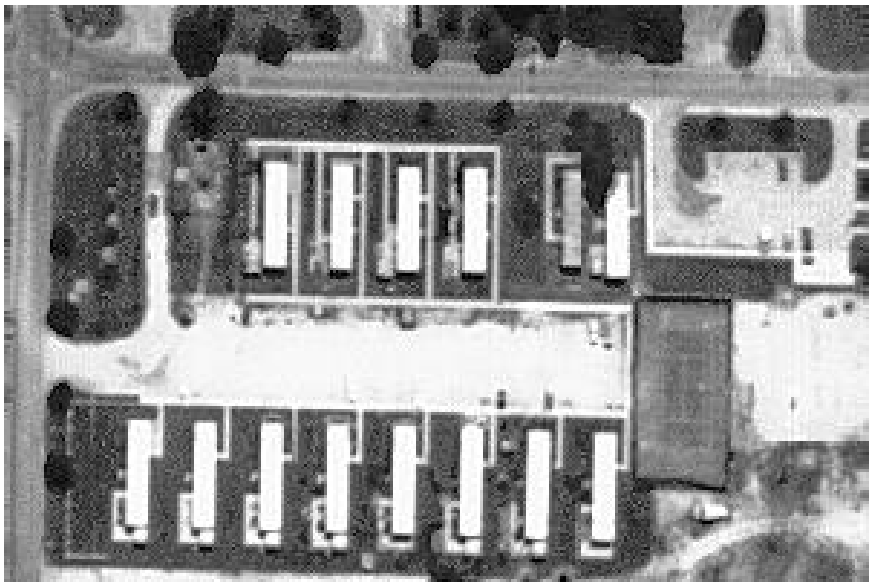


Figure 12.2: Original Aerial Image

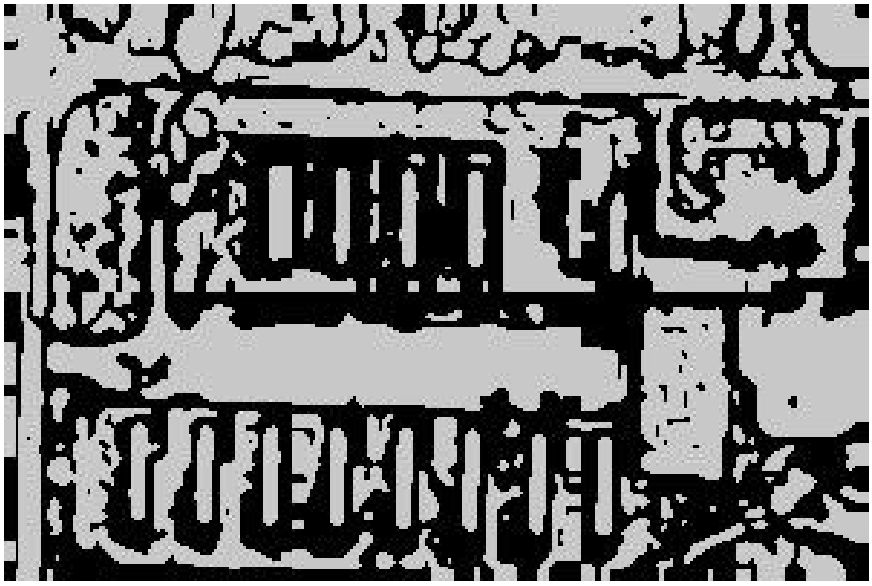


Figure 12.3: Segmentation of Aerial Image (from Chapter 10)

element. Most of the listing is the arrays defining the block letters, numbers, and a few punctuation marks. The program itself loops through the letters in the text and copies each letter's array into the image array.

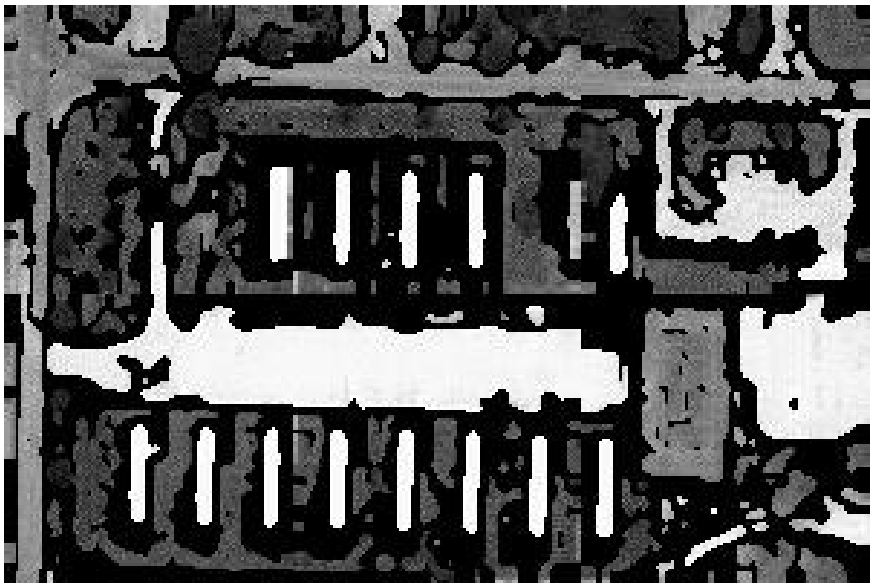


Figure 12.4: Segmented Aerial Image Masked with Original

The left side of the image in Figure 12.5 shows the output of the *ilabel* program. The words ADAM PHILLIPS are clear enough, but they will disappear if laid on top of an image. They need a background. The center section of Figure 12.5 shows the result of dilating the words as in Chapter 11. The right side of Figure 12.5 shows the final label — black letters on a white background. The final label is the result of the exclusive or (XOR) of the letters and their dilation. The output of the XOR is white only where one or the other image is white, but not both.

Figure 12.6 is the outcome of labeling. It is the result of ORing the final label of Figure 12.5 with the boy image. ORing allows us to see through the label to the image underneath. It is also possible to label the image using the greater overlay discussed later. Creating the label, however, is possible only via the XOR operation.

These are only two possible uses of the Boolean operations. There are many more, especially when you start combining them. After all, combining Boolean operations is how people build computers.

Figure 12.5: *ilabel* Output on Left, Dilation in Center, XOR of Both on Right

Figure 12.6: Labeled Boy Image

12.4 Overlay Operations

The overlay operations lay select pixels from one image on top of another and place the output into a third image. This chapter shows five types of pixel overlay operations from image A on top of image B. These are

1. overlay non-zero pixels from A onto B,
2. overlay zero pixels from A onto B,
3. overlay pixels from A onto B if they are greater,
4. overlay pixels from A onto B if they are less, and
5. average the pixels from A and B and use this as the output.

Figures 12.7 through 12.12 illustrate these operations. Figure 12.7 shows two image arrays: A and B. Figure 12.8 shows the result of laying the non-zero pixels of A on top of B. This looks like image A except for the absence of the 2x2 area of 0s in the lower right. Figure 12.9 shows the result of laying the zero pixels of A on top of B. This looks like image B except for the addition of the 2x2 area of 0s. Figure 12.10 shows the result of overlaying the pixels in A that are greater than the corresponding pixels in B. Note the column of 100s to the far right. Figure 12.11 shows the result of overlaying the pixels in A that are less than the corresponding pixels in B. Note the predominance of 50s and the 0s. Figure 12.12 shows the result of averaging images A and B. Figure 12.12 is not easy to perceive or read and is better illustrated with the images discussed below.

Listing 12.4 shows the subroutines that implement the overlay operations. The following routines follow the usual model: non_zero_overlay, zero_overlay, greater_overlay, less_overlay, and average_overlay. They combine the input image arrays and return the result.

12.5 Applications of Overlay Operations

Let's look at two applications of image overlaying. The first is the double exposure. Figure 12.13 shows two images side by side. Figure 12.14 shows the result of averaging the two together. This resembles a double exposure image, as it contains both images.

This technique can also apply a pattern or texture to an image. Figure 12.15 shows a leafy texture, and Figure 12.16 shows a house. Figure 12.17 is the result of averaging the two. It is easy to recognize the house, but parts of it (notably the roof and door) have a texture or pattern to them.

Image A

50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100
50	50	255	255	50	50	50	50	50	100
50	50	255	255	50	50	50	50	50	100
50	50	50	50	50	50	0	0	50	100
50	50	50	50	50	50	0	0	50	100
50	50	50	50	50	50	50	50	50	100
50	100	100	100	100	100	100	100	100	100
50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100

Image B

50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50

Figure 12.7: Images A and B

50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100
50	50	255	255	50	50	50	50	50	100
50	50	255	255	50	50	50	50	50	100
50	50	50	50	50	50	200	200	50	100
50	50	50	50	50	50	200	200	50	100
50	50	50	50	50	50	50	50	50	100
50	100	100	100	100	100	100	100	100	100
50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100

Figure 12.8: Result of Overlay Non-Zero A

50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	0	0	75	75
75	75	200	200	200	200	0	0	75	75
75	75	200	200	200	200	200	200	75	75
75	75	200	200	200	200	200	200	75	75
50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50

Figure 12.9: Result of Overlay Zero A

50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100
75	75	255	255	200	200	200	200	75	100
75	75	255	255	200	200	200	200	75	100
75	75	200	200	200	200	200	200	75	100
75	75	200	200	200	200	200	200	75	100
75	75	200	200	200	200	200	200	75	100
75	100	200	200	200	200	200	200	100	100
50	50	50	50	50	50	50	50	50	100
50	50	50	50	50	50	50	50	50	100

Figure 12.10: Result of Overlay Greater A

50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50
50	50	200	200	50	50	50	50	50	75
50	50	200	200	50	50	50	50	50	75
50	50	50	50	50	50	0	0	50	75
50	50	50	50	50	50	0	0	50	75
50	50	50	50	50	50	50	50	50	75
50	75	100	100	100	100	100	100	75	75
50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50

Figure 12.11: Result of Overlay Less A

50	50	50	50	50	50	50	50	50	75
50	50	50	50	50	50	50	50	50	75
62	62	227	227	125	125	125	125	125	87
62	62	227	227	125	125	125	125	125	87
62	62	125	125	125	125	100	100	125	87
62	62	125	125	125	125	100	100	125	87
62	62	125	125	125	125	125	125	125	87
62	87	150	150	150	150	150	150	87	87
50	50	50	50	50	50	50	50	50	75
50	50	50	50	50	50	50	50	50	75

Figure 12.12: Result of Average Overlay

Figure 12.13: Two Images Side by Side

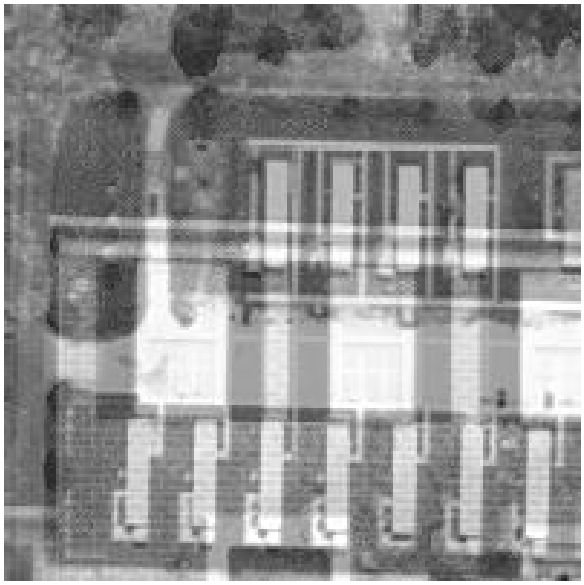


Figure 12.14: Two Images Averaged

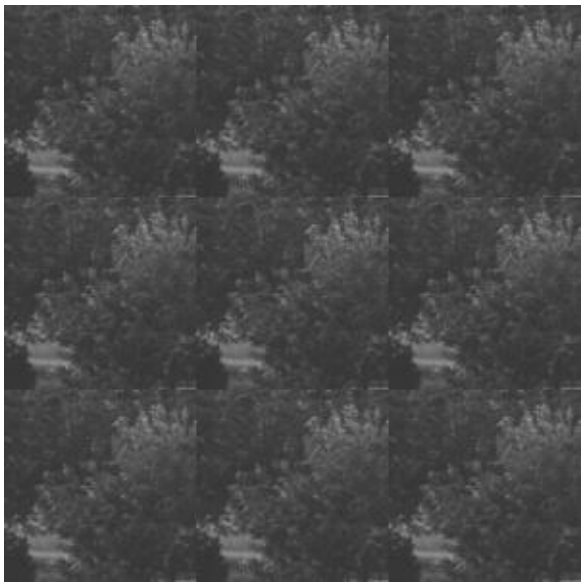


Figure 12.15: Leafy Texture Image

Figure 12.16: House Image

Figure 12.17: Averaging Leafy Texture and House Image

A second application of overlaying is to frame an area of interest in an image. The first step is to create a frame. Figure 12.18 shows a white frame in a blank image. I created this by modifying the *create* program of chapter 8 (left as an exercise for the reader).

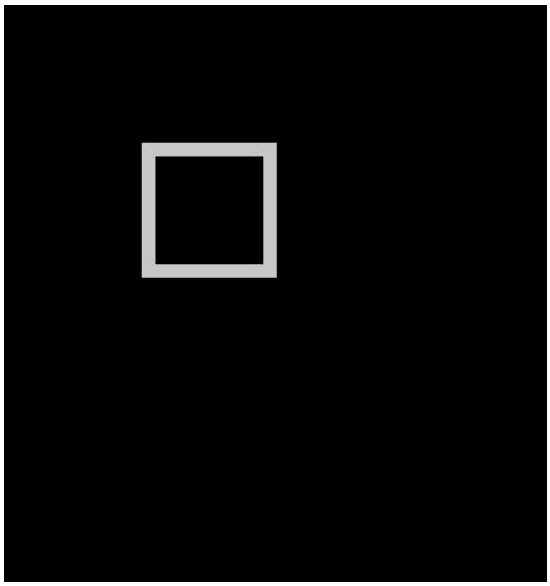


Figure 12.18: White Frame in Blank Image

Laying the rectangle of Figure 12.18 on top of a boy image produces Figure 12.19. The frame draws attention to a spot on the boy image. This is the result of overlaying the pixels in Figure 12.18 that are greater than the pixels in the boy image. The dark pixels inside the frame of Figure 12.18 are all zero so they disappear in the overlaying process. The white frame pixels are all 255 so they show up well in the result.

It is possible to create a frame of all zeros with a small area of 255s in the center. Using the zero_overlay or less_overlay would produce a thick dark frame around an area of interest.

Listing 12.5 shows the *mainover* program. This application allows the user to call any of the overlay programs discussed here and shown in listing 12.4. It follows the same pattern as the other applications discussed in this text.

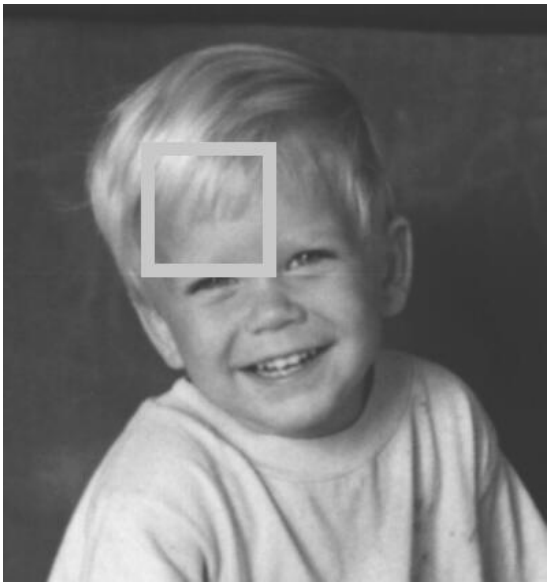


Figure 12.19: Frame Overlaid on Boy Image

12.6 Conclusions

This chapter has discussed Boolean and overlay operations. Though not complicated, these operations allow you to combine images in interesting and creative ways. There are endless possibilities to the combinations. As with all the other operators in this system, you should experiment. Use the operators as building blocks and mix them to fit your needs.

Chapter 13

Geometric Operations

13.1 Introduction

Basic image processing operations include the geometric type that rotate images and scale them (make them bigger and smaller). The first edition of this text included some simple forms of these operations. Those operators are not in this edition. Instead, this chapter discusses a powerful geometric operator that displaces, rotates, stretches, and bends images. It also includes a useful and simple program that stretches images to almost any size.

13.2 Geometric Operations

Geometric operations change the spatial relationships between objects in an image. They do this by moving objects around and changing the size and shape of objects. Geometric operations help rearrange an image so we can see what we want to see a little better.

The three basic geometric operations are displacement, stretching, and rotation. A fourth operation is the cross product (included here to show how to distort an image using higher order terms).

Displacement moves or displaces an image in the vertical and horizontal directions. Stretching enlarges or reduces an image in the vertical and horizontal directions. Rotation turns or rotates an image by any angle. Figure 13.1 shows the basic idea of these three operations.

Equations (13.1) and (13.2) describe mathematically how to perform these operations [13.1]. The first two terms in each equation perform the

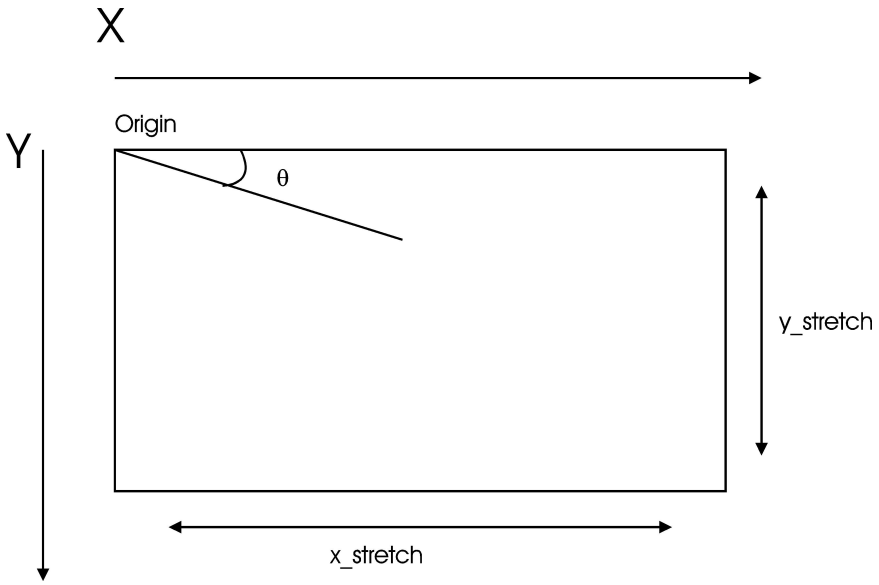


Figure 13.1: The Three Basic Geometric Operations: Displacement, Stretching, and Rotation

rotation by any angle θ . The $x_{displace}$ and $y_{displace}$ terms perform displacement. They shift the image in either direction (shift to the left for $x_{displace}$ greater than zero, shift to the right for less than zero). The x times $x_{stretch}$ enlarges or shrinks the image in the horizontal direction while the y times $y_{stretch}$ does the same in the vertical direction. The x_{cross} and y_{cross} terms distort the image and an example explains them better than words.

$$X = x \cdot cos(\theta) + y \cdot sin(\theta) + x_{displace} + x \cdot x_{stretch} + x \cdot y \cdot x_{cross} \tag{13.1}$$

$$Y = y \cdot cos(\theta) - x \cdot sin(\theta) + y_{displace} + y \cdot y_{stretch} + x \cdot y \cdot y_{cross} \tag{13.2}$$

The power of equations (13.1) and (13.2) is that they do all three (four) operations at one time. Setting the terms accomplishes any or all the operations.

Figures 13.2 through 13.6 illustrate the operations. Figure 13.2 shows displacement. The upper left hand corner shows in the input image. This is a window and brick wall from a house. The upper right hand corner shows

the result of displacing the input image up and to the left. The lower left hand corner shows the result of displacing the input image down and to the right ($x_{displace}$ and $y_{displace}$ are negative values such as -10). The lower right hand corner shows displacement up and to the right.

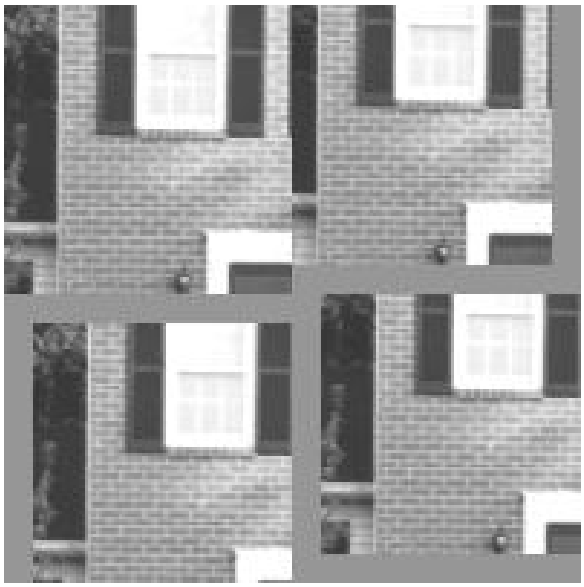


Figure 13.2: Examples of Displacement

Note that when any operator moves an image, blank areas fill in the vacant places.

Figure 13.3 shows stretching. The upper left hand corner is the input. The upper right hand corner is the result of stretching the input image in both directions (set $x_{stretch}$ and $y_{stretch}$ to 2.0). The lower left hand corner is the result of stretching the input image with values less than 1.0. This causes shrinking. The lower right hand corner shows how to combine these effects to enlarge the image in the horizontal direction and shrink it in the vertical direction.

Figure 13.4 shows rotation. The upper left hand corner is the input image. The other areas show the result of rotating the input image by pinning down the upper left hand corner (the origin). The other areas show rotations of θ = 30, 45, and 60 degrees.

Figure 13.5 shows the influence of the cross product terms x_{cross} and y_{cross}. Setting these terms to anything but 0.0 introduces non-linearities (curves). This is because equations (13.1) and (13.2) multiply the terms by both x and

Figure 13.3: Examples of Stretching

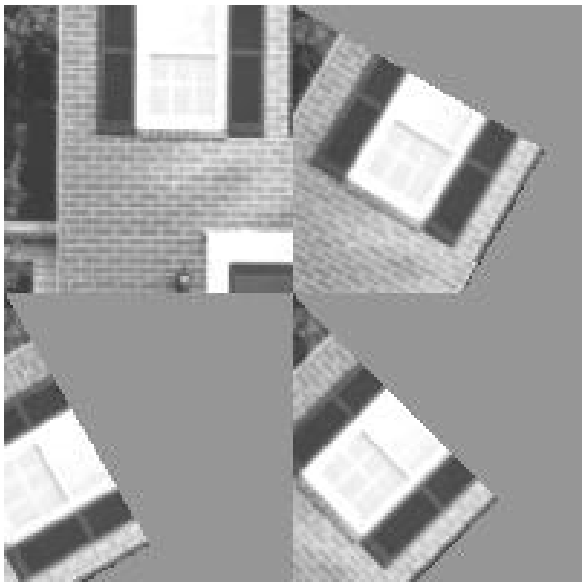


Figure 13.4: Examples of Rotation about the Origin

y. The input image is on the left side of Figure 13.5 with the output shown on the right (x_{cross} and $y_{cross} = 0.01$). Values much bigger than this distort the output image to almost nothing.

Figure 13.5: Examples of Cross Products

Using higher order terms in equations (13.1) and (13.2) can cause greater distortion to the input. You can add a third order term to equation (13.1) ($x \cdot x \cdot y \cdot x_{doublecross}$) and equation (13.2) ($y \cdot y \cdot x \cdot y_{doublecross}$). Try this for homework. It will be easy given the source code.

Figure 13.6 shows the result of using all four operations at once. This is the result of displacing down and to the right, enlarging in both directions, rotating 30 degrees, and using cross products. It is a simple matter of setting the terms in the equations.

Listing 13.1 shows the geometry routine that implements these operations. It has the same form as the other image processing operators in this series. The parameters are from equations (13.1) and (13.2). First, geometry converts the input angle theta from degrees to radians and calculates the sine and cosine. The next section prepares the stretch terms to prevent dividing by zero.

The loops over i and j move through the input image. All the math uses doubles to preserve accuracy. new_i and new_j are the coordinates of the pixels in the input image to copy to the output image.

The final section of geometry sets the output image to the new points in the input image. If bilinear == 1, we will call the bi-linear interpolation function described below. If bilinear == 0, we set the output image directly.

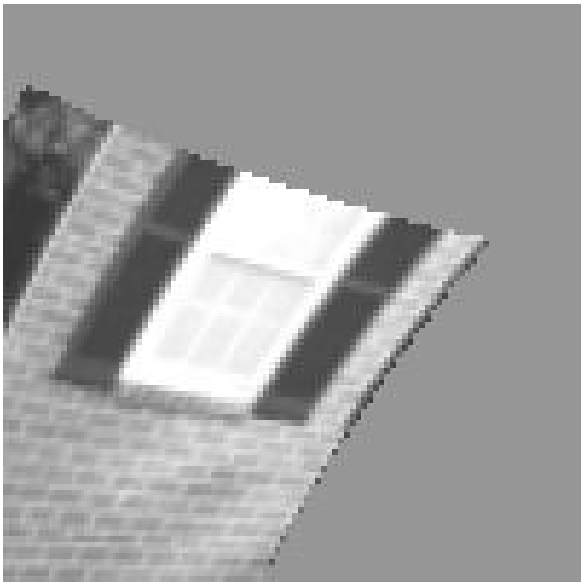


Figure 13.6: Combining All Four Geometric Operations

The compound if statement checks if the new points are inside the image array. If they are not, set the out_image to the FILL value (this fills in vacant areas).

13.3 Rotation About Any Point

The geometric operations above can rotate an image, but only about the origin (upper left hand corner). Another type of rotation allows any point (m, n) in the image to be the center of rotation. Equations (13.3) and (13.4) describe this operation [13.2]. Figure 13.7 illustrates how the input image (the rectangle) revolves about the point (m, n). Figure 13.8 shows several examples. The upper left hand corner is the input image. The other three quadrants show 45 degree rotations about different points in the image. Almost anything is possible by combining the basic geometric operations shown earlier with this type of rotation. For example, you can displace and stretch an image using the earlier operations and rotate that result about any point.

$$X = x \cdot cos(\theta) - y \cdot sin(\theta) - m \cdot cos(theta) + n \cdot sin(\theta) + m \qquad (13.3)$$

$$Y = y \cdot cos(\theta) + x \cdot sin(\theta) - m \cdot sin(theta) - n \cdot sin(\theta) + n \tag{13.4}$$

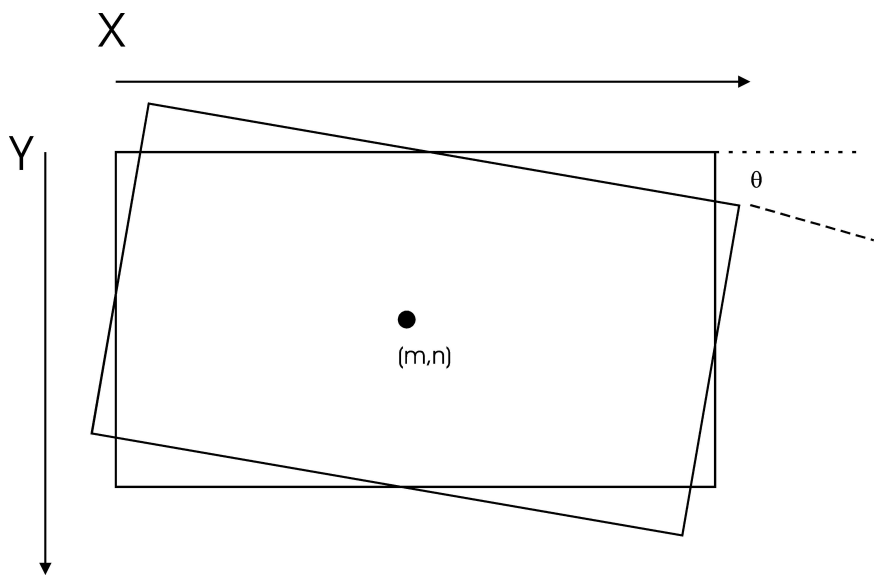


Figure 13.7: Rotation About any Point m,n

Listing 13.1 next shows the routine arotate that performs rotation about any point (m, n). arotate converts the angle of rotation from degrees to radians and calculates the sine and cosine. It loops through the image and calculates the new coordinates tmpx and tmpy using equations (13.3) and (13.4). If bilinear == 1, use bi-linear interpolation (coming up next). If bilinear == 0, check to see if the new coordinates are in the image array. If the are, set the output image to those points in the input image.

13.4 Bi-Linear Interpolation

Now that we have some basics behind us, let's move forward. Critical to making the results of any of the operations look good is bi-linear interpolation. Bi-linear interpolation is present in any good image processing applications performed today in commercials, music videos, and movies. As usual, bi-linear interpolation is a big name for a common sense idea. It fills in holes with gray levels that make sense [13.3] [13.4].

Figure 13.8: Examples of Rotation About Any Point

The bent lines in Figure 13.9 show why bi-linear interpolation is important. The left side did not use bi-linear interpolation. It has jagged lines. The smooth bent lines in the right side illustrate how bi-linear interpolation makes things look so much better.

There is a reason for the jagged lines. In many geometric operations, the resulting pixel lies somewhere between pixels. A pixel's new coordinates could be x=25.38 and y=47.83. Which gray level is assigned to that pixel? Rounding off suggests x=25 and y=48. (That is what happens in the code listings when the parameter bilinear == 0.) Rounding off produces the jagged lines.

Bi-linear interpolation removes jagged lines by finding a gray level between pixels. Interpolation finds values between pixels in one direction (interpolating 2/3's of the way between 1 and 10 returns 7). Bi-linear interpolation finds values between pixels in two directions, hence the prefix " bi."

Figure 13.10 illustrates how to perform bi-linear interpolation. Point P3 (x, y) is somewhere between the pixels at the four corners. The four corners are at integer pixels (x=25, x=26, y=47, y=48). Equations (13.5), (13.6), and (13.7) find a good gray level for point P3. In these equations, x and y are fractions (if x=25.38 and y=47.83, then in the equations x=0.38 and y=0.83).

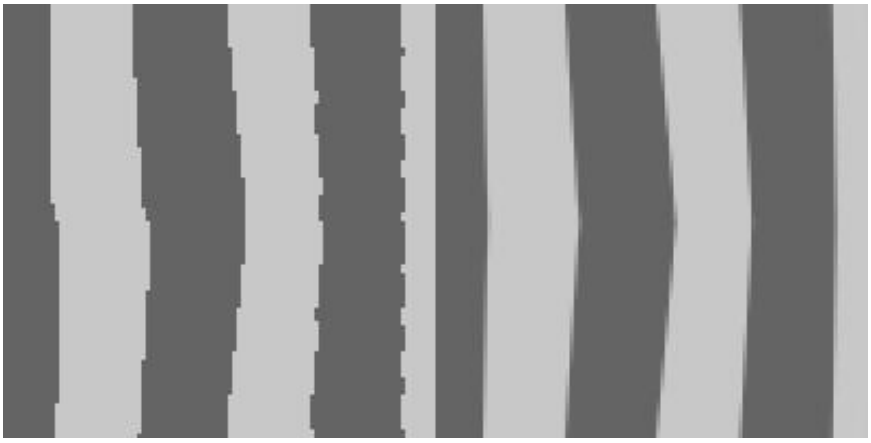


Figure 13.9: A Comparison of Not Using Bi-Linear Interpolation and Using Bi-Linear Interpolation

$$gray(P1) = (1-x) \cdot gray(floor(x), floor(y)) + x \cdot gray(ceiling(x), floor(y)) \tag{13.5}$$

$$gray(P2) = (1-x) \cdot gray(floor(x), ceiling(y)) + x \cdot gray(ceiling(x), ceiling(y)) \tag{13.6}$$

$$gray(P3) = (1-y) \cdot gray(P1) + y \cdot gray(P2) \tag{13.7}$$

Equation (13.5) finds the gray level of point P1 by interpolating between the two upper corners. Equation (13.6) finds the gray level of point P2 by interpolating between the two lower corners. Equation (13.7) finally finds the gray level of P3 by interpolating between points P1 and P2.

Listing 13.1 shows the routine bilinear_interpolate that implements these equations. The input parameters are the image array the_image and the point (x, y) (in their full form x=25.38 and y=47.83). bilinear_interpolate returns the gray level for (x, y). This routine contains slow, double precision floating point math. This is the trade-off between techniques — speed verses good looks.

This first part of bilinear_interpolate checks if x and y are inside the image array. If not, the routine returns a FILL value. The next statements create the floor x and y, ceiling x and y, fractional parts of x and y, and one minus

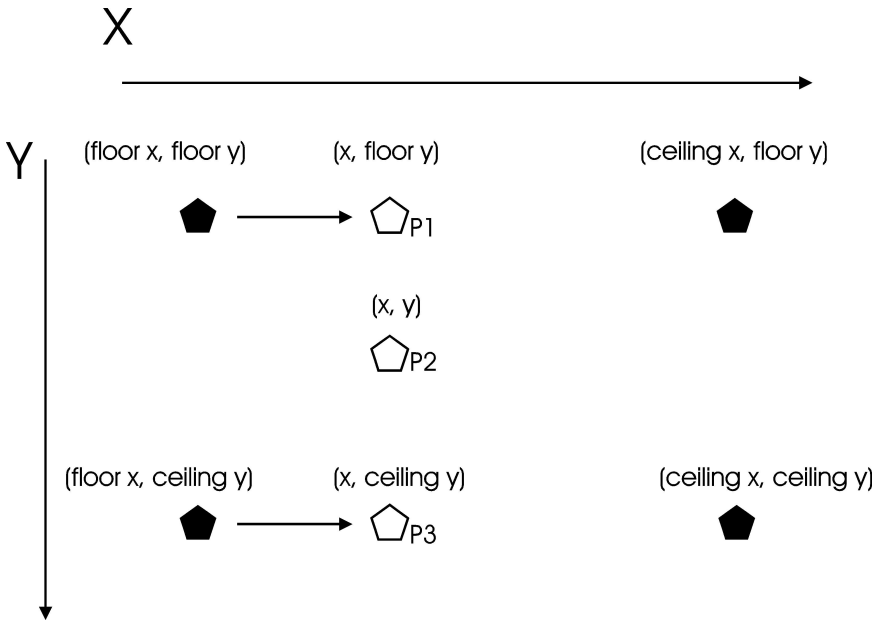


Figure 13.10: Bi-Linear Interpolation

the fractions shown in the figure and needed by the equations. The final statements calculate the gray levels of points P1, P2, and P3. The routine returns the final gray level of P3.

Bi-linear interpolation is a simple idea, uses a simple routine, and makes a world of difference in the output image. The images shown earlier for geometric operations all used the bi-linear option. I recommend the rounding method for quick experiments and bi-linear interpolation for final presentations.

13.5 An Application Program

Listing 13.2 shows the *geometry* program. This program allows the user to either perform the geometric operations of figure 13.1 or the rotation about a point of figure 13.7. *geometry* interprets the command line, loads the parameters depending on the desired operation, and calls the operations. It has the same form as the other applications in this text.

13.6 A Stretching Program

A useful utility for image processing is enlarging and shrinking an entire image. The many uses include making an image fit a display screen for printing or imaging and making two images about the same size for comparisons. The stretching and bi-linear interpolation tools now available permit general stretching.

The main routine and subroutines shown in listing 13.3 make up the *stretch* program. The command line is:

stretch input-image-file output-image-file x-stretch y- stretch bilinear

If the bilinear parameter is 1, *stretch* uses bi-linear interpolation otherwise it uses basic rounding.

stretch has the same form as most applications in this text. It uses the create_resized_image_file because the output file and input file have different sizes. The main routine allocates the image arrays (different sizes), reads the input, calls the stretch subroutine, and writes the output. The stretch subroutine borrows heavily from the geometry subroutine shown in listing 13.1.

Figure 13.11 shows results of the *stretch* program. It demonstrates how *stretch* can enlarge in one direction while shrinking in another. The more you experiment with image processing, the more you will find yourself using *stretch*. It is very handy.

Figure 13.11: The Boy Image Enlarged Horizontally and Shrunk Vertically

13.7 Conclusions

This chapter discussed geometric operations. These powerful and flexible operations change the relationships, size, and shape of objects in images. They allow you to manipulate images for better display, comparison, etc. Keep them handy in your collection of tools.

13.8 References

13.1 "Digital Image Processing," Kenneth R. Castleman, Prentice-Hall, 1979.
13.2 "Mathematical Elements for Computer Graphics," David F. Rogers, J. Alan Adams, McGraw-Hill, New York, New York, 1976.
13.3 "The Image Processing Handbook, Third Edition," John C. Russ, CRC Press, 1999.
13.4. "Modern Image Processing," Christopher Watkins, Alberto Sadun, Stephen Marenka, Academic Press, Cambridge, Mass., 1993.

Chapter 14

Warping and Morphing

14.1 Introduction

This chapter extends the discussion of geometric operations and delves into warping and morphing (Hollywood, here we come). Image warping is a technique that Hollywood discovered in the 1980's. The result is the magic we see every day in commercials, music videos, and movies. Warping (and its cousin morphing) "melts" old cars into new ones and can turn a car into a tiger.

14.2 Image Warping

Image warping is a technique that bends and distorts objects in images. Remember pressing a flat piece of silly putty on a newspaper to copy the image to the silly putty? Grabbing and pulling the silly putty distorted the appearance of the image. Bending and stretching the silly putty caused the objects in the image to assume weird and wonderful shapes. Image warping does the same for digitized images as silly putty did for us as kids.

Using a computer to warp images is not new. It began in the 1960's with early space probes. The pictures of the moon produced by the "cameras" on the probes were distorted. Straight lines appeared bent and the objects were out of proportion. Image processors at the Jet Propulsion Laboratory [14.1] transformed these square images into the shape of pie pieces. The resulting pie piece images had straight lines where straight lines belonged.

The special effects artists in Hollywood discovered warping in the 1980's.

They decided to apply this technique to entertainment. The result is what we see every day in commercials, music videos, and movies.

14.3 The Warping Technique

The basic idea behind warping is to transform a quadrilateral to a rectangle. A quadrilateral is a four-cornered region bounded by straight lines. Transforming a quadrilateral to a rectangle warps the objects inside the quadrilateral.

Figure 14.1 shows a quadrilateral with a point P inside [14.2]. Transforming a quadrilateral to a rectangle requires finding the coordinates of any point P inside the quadrilateral. This is possible given the coordinates of the four corners P1, P2, P3, and P4 and the fractions a and b along the edges. The key to finding P is using bi-linear interpolation. In the last chapter, bi-linear interpolation found the gray level of a pixel between other pixels (gray level bi-linear interpolation). It can also find the location of a pixel between other pixels (spatial bi-linear interpolation).
NOTE: This chapter works with shapes that have four Parts (1, 2, 3, and 4). Part 1 will be in the upper left-hand corner, and parts 2, 3, and 4 will proceed clockwise.

Equations (14.1) through (14.7) show how to find the coordinates of point P. (If mathematical derivation is not for you, skip down to the results in equations (14.6) and (14.7). The source code given later will implement these equations). These equations run through bi-linear interpolation. They interpolate along the top and bottom of the quadrilateral and then along the sides. In the equations, a and b are fractions ($0 < a < 1, 0 < b\ 1$).

Equation (14.1) finds point Q by interpolating between points P1 and P2 using a. Equation (14.2) finds point R by interpolating between points P3 and P4 using a. Equation (14.3) finds point P by interpolating between Q and R using b. Equation (14.4) is the result of substituting the values of Q and R from equations (14.1) and (14.2) into equation (14.3). Equation (14.5) gathers all the terms from (14.4).

$$Q(a) = P_1 + (P_2 - P_1)a \tag{14.1}$$

$$R(a) = P_4 + (P_3 - P_4)a \tag{14.2}$$

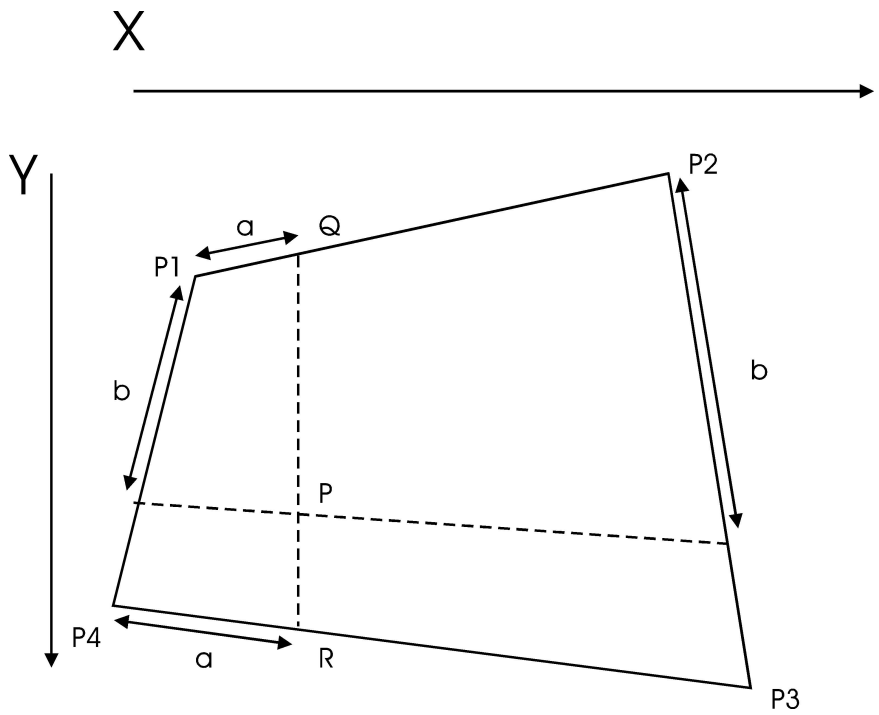


Figure 14.1: Bi-Linear Interpolation of a Quadrilateral

$$P(a,b) = Q + (R - Q)b \tag{14.3}$$

$$P(a,b) = P_1 + (P_2 - P_1)a + [(P_4 + (P_3 - P_4)a) - (P_1 + (P_2 - P_1)a)]b \tag{14.4}$$

$$P(a,b) = P_1 + (P_2 - P_1)a + (P_4 - P_1)b + (P_1 - P_2 + P_3 - P_4)ab \tag{14.5}$$

Equations (14.6) and (14.7) are the final answers. Equation (14.6) shows how to find the x coordinate of any point P given the x coordinates of the four corners and the fractions a and b. Equation (14.7) does the same for the y coordinate. The subroutines described below will implement equations (14.6) and (14.7). Notice the ab term in the equations. This term introduces non-linearities or curves into the results.

$$P(x) = x_1 + (x_2 - x_1)a + (x_4 - x_1)b + (x_1 - x_2 + x_3 - x_4)ab \tag{14.6}$$

$$P(x) = y_1 + (y_2 - y_1)a + (y_4 - y_1)b + (y_1 - y_2 + y_3 - y_4)ab \tag{14.7}$$

14.4 Two Ways to Warp

This chapter implements two kinds of warping. The first is control point warping illustrated in figure 14.2. Divide a section of an image into four smaller squares 1, 2, 3, and 4. Pick a control point anywhere inside the square section. This control point divides the square section into four quadrilaterals as shown in the top part of figure 14.2. Equations (14.6) and (14.7) will transform these four quadrilaterals back into the four squares as shown in the bottom part of figure 14.2. This will warp the objects in the image. The control point will dictate the warping.

Listing 14.1 shows the source code that implements this type of warping. The warp subroutine controls the process and calls the warp_loop and bi_warp_loop subroutines. The inputs to warp include the usual image arrays and line and element coordinates of the image. Specific to warp are the x

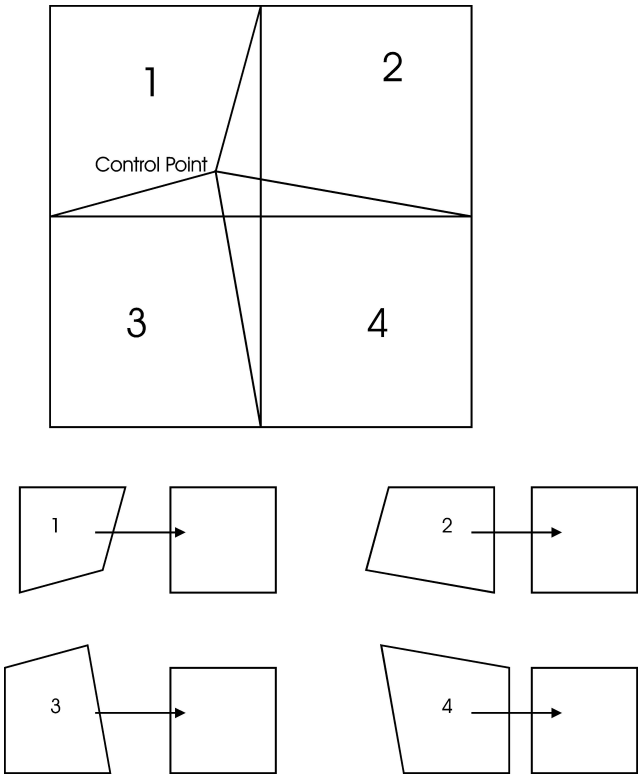


Figure 14.2: The Control Point Warping Process

and y control points and the bilinear parameter. The control points control the warping, and bilinear specifies accuracy. If bilinear == 0, warp calls warp_loop, otherwise it calls bi_warp_loop.

warp works on the four quarters of the input image. For each quarter, warp sets (x1, y1) through (x4, y4) to the coordinates of the corners of the quadrilaterals 1 through 4. warp then sets the extra_x and extra_y variables to the upper left hand corner of the small squares. In doing this, warp is setting the four corners of the quadrilateral. This is necessary to transform the quadrilateral to a square. warp calls a warping routine that will implement equations (14.6) and (14.7) and bend the objects. warp calls warp_loop for quick warping or bi_warp_loop to use bi-linear interpolation for slower but better warping.

warp_loop (next in listing 14.1) implements equations (14.6) and (14.7) to transform the small quadrilateral to a small square. First, it sets up the coefficients for the equations. The variables xa, xb, and xab correspond to a, b, and ab from equation (14.6). The variables ya, yb, and yab correspond to a, b, and ab from equation (14.7). The loops over i and j calculate the coordinates of the pixels in the input image that will be copied to the output image (x_out and y_out). If x_out or y_out lie outside the image array, output_image is set to a FILL value. Otherwise, output_image is set to the proper pixel from the_image.

bi_warp_loop performs the same operations as warp_loop. bi_warp_loop, however, uses floating point math and calls the bilinear_interpolate (last chapter) subroutine to set the final output pixel value. bi_warp_loop produces better results than warp_loop. It also takes more time. Use warp_loop for quick experiments and bi_warp_loop for presentation results.

Figure 14.3 shows some results of control point warping. The upper left quarter is the input image. The other three quarters show the result of picking a control point inside the input image and having the warp routine warp it. The objects in the results are bent out of shape. Repeating the warping can give an object almost any desired shape (windows the shape of circles, triangles, etc.).

A second form of warping is what I call object warping. Instead of picking a control point inside the image array, the user picks the four corners of a quadrilateral as shown in figure 14.4. Object warping transforms this quadrilateral to a square. The four corners of the quadrilateral can be almost anywhere inside or outside the square. The outside the square option is a capability that control point warping did not have.

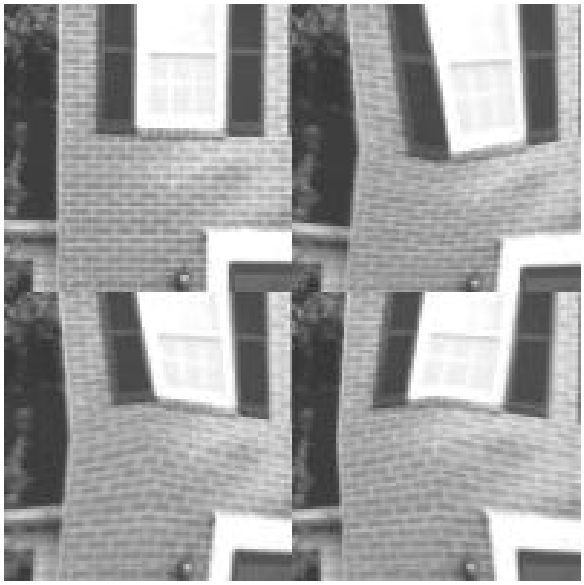


Figure 14.3: Examples of Control Point Warping

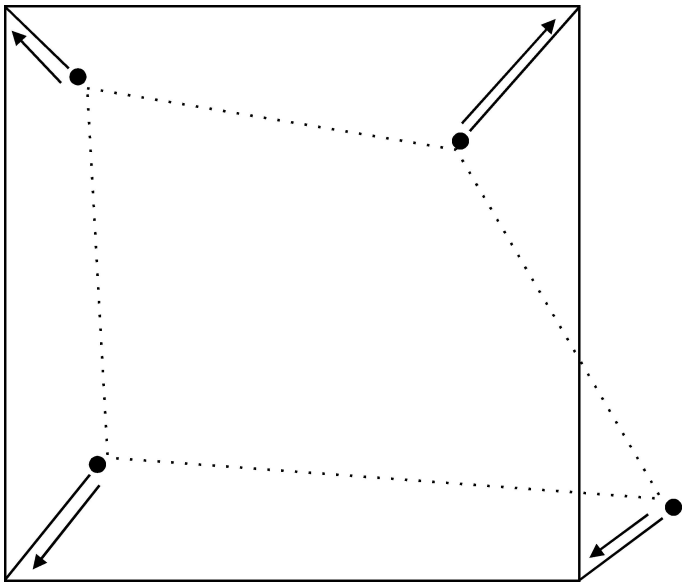


Figure 14.4: The Object Warping Process

Object warping is similar to but simpler than control point warping. Control point warping transformed four quadrilaterals to four squares inside an image array. Object warping transforms one quadrilateral to one square.

Listing 14.1 continues with the source code that implements object warping. The object_warp subroutine controls the process and calls full_warp_loop and bi_full_warp_loop. The inputs to object_warp are the same as with warp except it receives the four corners of the quadrilateral (x1, y1 through x4, y4) instead of the control points. object_warp prepares the calls either full_warp_loop (bilinear == 0) or bi_full_warp_loop (bilinear == 1).

full_warp_loop performs the same function that warp_loop performed for control point warping, i.e., it transforms a quadrilateral to a square. The difference is that the loops over i and j in full_warp_loop go through an entire image array. warp_loop only went through one quarter of the image array.

bi_full_warp_loop performs the same operations as full_warp_loop except it uses floating point math and calls bilinear_interpolate for the final pixel values. bi_full_warp_loop is slower, but it produces better results. Again, use full_warp_loop for experiments and bi_full_warp_loop for presentation results.

Figure 14.5 shows some results of object warping. The upper left quarter shows the input image with the three other quarters showing results. These results are similar to those in Figure 14.3. Notice, however, that each result in Figure 14.5 contains FILL values shifted in from outside the input.

Figures 14.6 and 14.7 show the result of applying warping to complete images and illustrates what warping can do. These images show the outcome of using control point warping several times.

While looking at Figures 14.6 and 14.7, turn on your imagination. The house appears to be in the middle of an earthquake. We could make a series of house images with the warping moved from right to left a little bit in each image. If you put one house image in each frame of motion picture film, the sequence would look like a house rippling in an earthquake. We could make the house look like Jell-O. What we need is a computer fast enough to generate 10,000 images and a motion picture camera. This is how they do it in Hollywood.

14.5 Shearing Images

One application of warping is to shear images. A related article [14.3] showed how to shear images. Object warping and bi-linear interpolation can do the

Figure 14.5: Examples of Object Warping

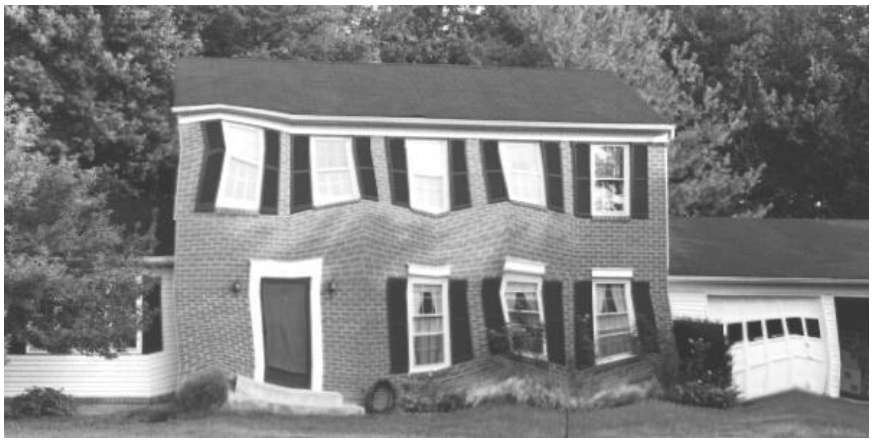


Figure 14.6: Warped House Image

Figure 14.7: Another Warped House Image

same operation without producing jagged lines. Image 14.2 showed how object warping can warp an image and pull in FILL values from outside the image. This is because object warping permits picking quadrilateral corner points outside the image array as shown in figure 14.4.

Figure 14.8 depicts image shearing as produced by object warping. These four image shears are the result of calling the *warp* program (described later) that in turn calls the object_warp routine. The numerical parameters that are outside the input image (i.e., less than 0 or greater than the image size) cause the image to shear or shift over.

14.6 Morphing

Morphing is the term most people use today to describe the melting of one object to another. Michael Jackson "morphed" to a panther. A car "morphed" to a tiger. The Mighty Morphin' Power Rangers — well that's another story.

(Funny, as I sit here revising this in late 1998, many people may not remember Michael Jackson and the Mighty Morphin' Power Rangers. Morphing has become so common that people think it was always there.)

Morphing is a sequence of images — not a single image. A car becomes a tiger by showing a sequence of intermediate images. The transition appears magical when there are dozens of images shown every second in a film sequence.

Morphing can be done as an extension to warping. Suppose we wanted to morph a dark circle to a bright pentagon. The first step is to produce

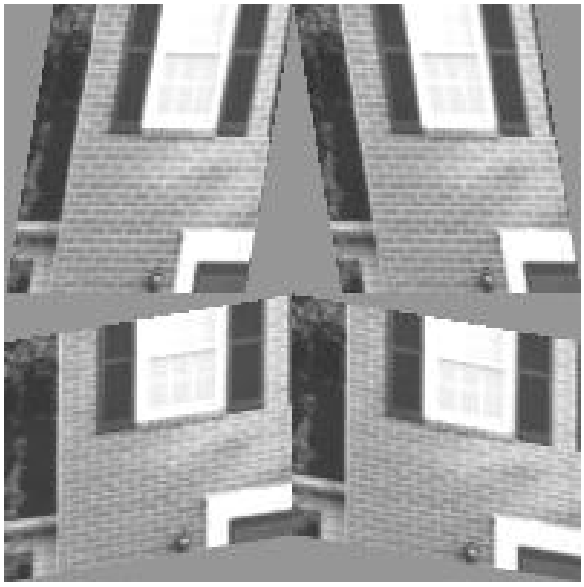


Figure 14.8: Examples of Image Shearing

two sequences of images. One sequence would warp the circle to the triangle shape. The second sequence would warp the triangle to the circle shape. These sequences take care of the transition from one shape to the next. Figure 14.9 shows this process.

The second step is to blend these two sequences into a third sequence. The third sequence would be a weighted average of the first two. This third sequence would take care of the transition from one gray level to the next. Placing one image of the third sequence in each frame of a motion picture film produces a smooth morphing.

Figure 14.10 illustrates the process. The objective here is to morph a window to a door. The far left frame of the middle row shows the original window. The far right of the middle row shows the final door.

The top row of Figure 14.10 shows the sequence to warping the original window up to the size and shape of the door. Object warping produced these by picking a quadrilateral smaller than the image. The output of each step was an enlarged window.

The bottom row of Figure 14.10 shows the sequence to warping the door down to the size and shape of the window. Object warping produced these by picking a quadrilateral larger than the image. The output of each step was a smaller door.

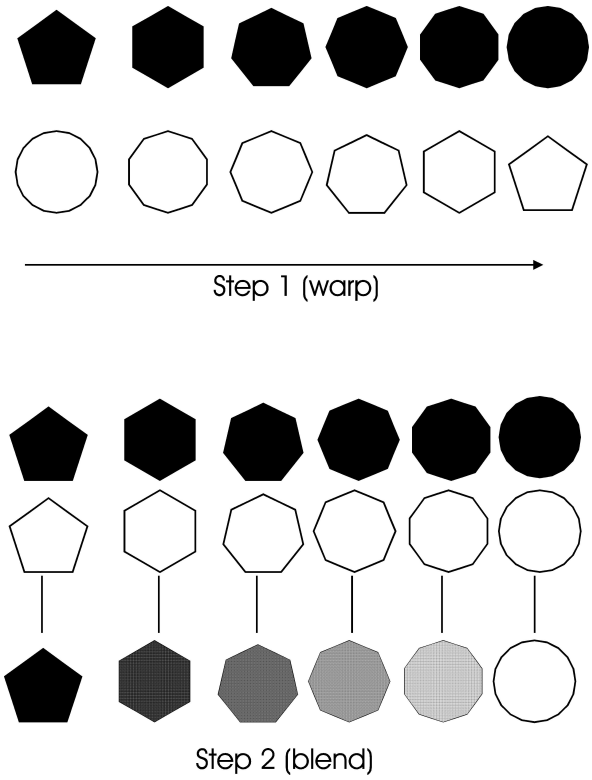


Figure 14.9: Morphing a Black Circle to a White Pentagon

Figure 14.10: A Morphing Sequence

The three frames in the center of the middle row of Figure 14.10 show the result of a weighted average. An average is the result of adding two images together and dividing by two. A weighted average is the result of adding an image to itself several times, adding this result to another image, and dividing.

The frame closest to the original window is the average of two windows from the top row and one door from the bottom row. The frame in the center of the middle row is the average of one window from the top row and one door from the bottom row. The frame closest to the final door is the average of two doors from the bottom row and one window from the top row.

The sequence of frames in the middle row morph the window to the door. It would look much better had I used a sequence of 3000 warps and averages instead of 3. The procedure is the same, but the more steps between the start and the end, the better the effect.

The professional morphing seen on TV and in the movies has hand tuning performed by artists. Those sequences are not as straight forward as discussed above. Artists take the sequences and manipulate individual pixels so they look just right. They also adjust the weighted averages using more complicated algorithms. The results show the tender loving care put into them.

The .bat file shown in listing 14.3 created the image of Figure 14.10. The calls to the *warp* program (described below) created the two warping sequences. The calls to mainover (chapter 12) performed the weighted averages. The calls to side (chapter 4) pasted all the small images together to form the image shown in Figure 14.10.

14.7 A Warping Application Program

Listing 14.3 shows the *warp* program. This is a standalone program that performs control point or object warping on images. This program produced the images shown in this article. *warp* is a command line driven program that calls either the warp or object_warp subroutines.

14.8 Conclusions

This chapter discussed image warping and morphing. Warping bends or warps objects in images. Morphing is an extension of warping that melts or morphs one object into another (just like in the commercials). Warping is an old technique with its roots in the space program of the 1960's. The ever increasing power and decreasing price of computers brought these techniques to Hollywood. They are fun. Experiment with them and turn brick houses into Jell-O.

14.9 References

14.1 "Digital Image Processing," Kenneth R. Castleman, Prentice-Hall, 1979.
14.2. "Modern Image Processing," Christopher Watkins, Alberto Sadun, Stephen Marenka, Academic Press, Cambridge, Mass., 1993.
14.3 "Bitmap Image Transformations," Christopher Dean, The C Users Journal, December 1993, pp. 49-70.

Chapter 15

Basic Textures Operations

15.1 Introduction

This chapter will discuss textures and some basic texture operations. Texture is a characteristic that is difficult to describe, but you know it when you see it. Humans distinguish many objects in images by their textures. The leaves on trees and the shingles on a roof may have similar gray levels, but they have different textures. A capable texture operator could segment that image correctly.

15.2 Textures

Figure 15.1 shows a drawing of different textures from a paint program. Adjectives that describe the textures include dots, lines, random, regular, horizontal, angled, tiled, woven, rough, smooth, etc. Figure 15.2 shows an image containing four textures used in the examples later. These textures are (clockwise starting in the upper left) a fuzzy carpet, a tightly woven straw mat, random grass, and straw.

One way of describing texture is "things" arranged in a "pattern." Textures are a function of things and patterns — mathematically, texture = f(thing, pattern). The thing is a grouping of pixels such as a dot or line. The pattern is the arrangement of the things such as a random or regular cluster of dots and horizontal or vertical lines. Regular patterns are usually man made while random patterns are natural. These ideas about texture all make sense, but a commonly used model of texture is lacking.

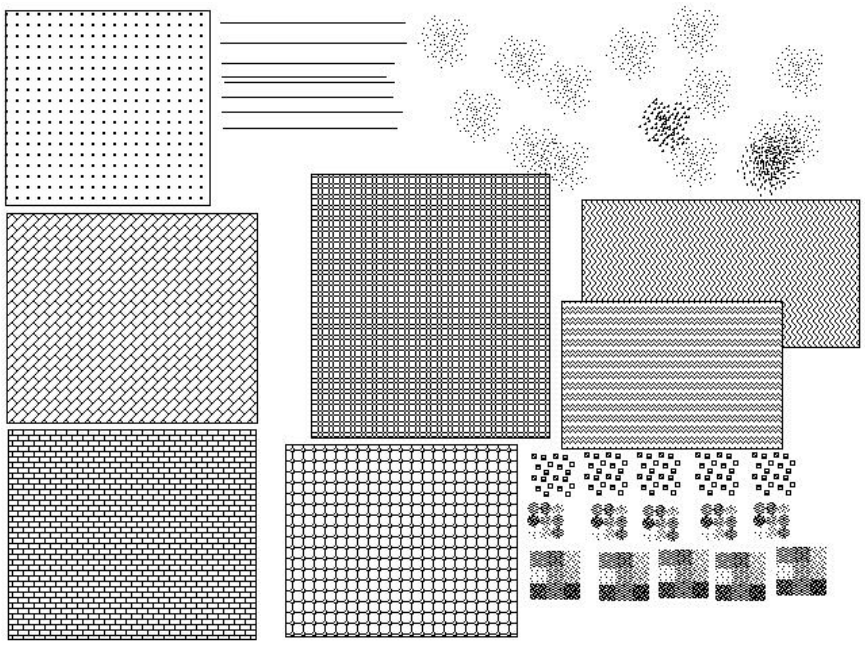


Figure 15.1: Examples of Textures

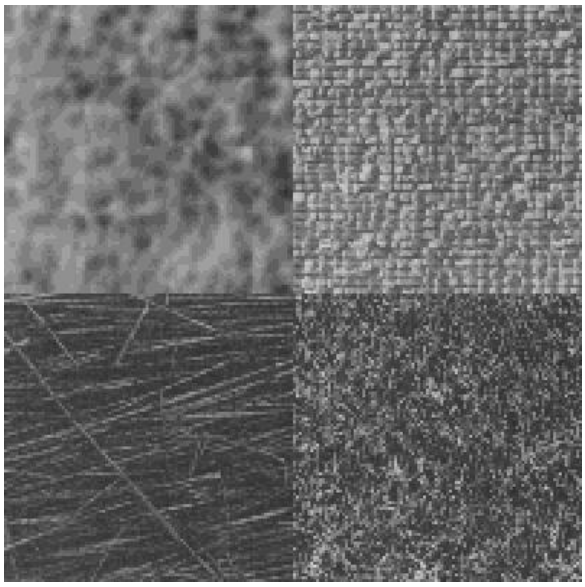


Figure 15.2: Four Textures

What we want is an operator that could characterize the things and patterns that comprise texture. Such an operator would represent a texture by a number just as gray levels represent the lightness or darkness of objects. We could then use the gray level and edge-based segmentation techniques discussed in earlier chapters. Unfortunately, simple operators that work for all textures do not exist. There is a collection of operators that work in some cases. Edge detectors and difference operators work in certain situations.

The Hurst operator [15.2] produces good results in many images, but has a high computational cost. The common sense comparison approach works well in many images, but only isolates one texture per image.

One bad trait of texture operators is their computations. Several of them required many, complex, floating point calculations. There is no way around this problem with texture operators. This was an issue in 1993 when I first tried to write texture operators. It is not an issue in late 1998 as PCs are much more powerful.

15.3 Edge Detectors as Texture Operators

Edge detectors can be used as texture operators. This is because a textured area has many edges compared with a smooth area. Applying an edge detector to a texture produces many strong, bright edges while edge detecting a smooth area yields nothing. Smoothing the edge detector result gives a bright area that can be separated from the dark area that lacked edges.

Not all edge detectors work well with textures. The four parts of Figure 15.3 illustrate how the Sobel edge detector (chapter 5) failed to distinguish two textures. The upper left quarter of the image contains a tightly woven texture. The upper right quarter contains a random grass texture. Beneath each texture is the result of the Sobel edge detector. It detected all the edges in the two distinctly different textures. The result, however, is two areas with similar gray levels.

The range operator is an edge detector that does work well on some textures. It takes the pixels in an nxn area, sorts them by value, and replaces the center pixel with the range (the largest pixel value minus the smallest).

Figure 15.4 shows an example of the range operator (chapter 6) applied to a texture. The upper left quarter is the input image. I placed a small section of the tightly woven texture of Figure 15.2 into a square of the carpet texture from Figure 15.2. The upper right quarter of Figure 15.4 shows the

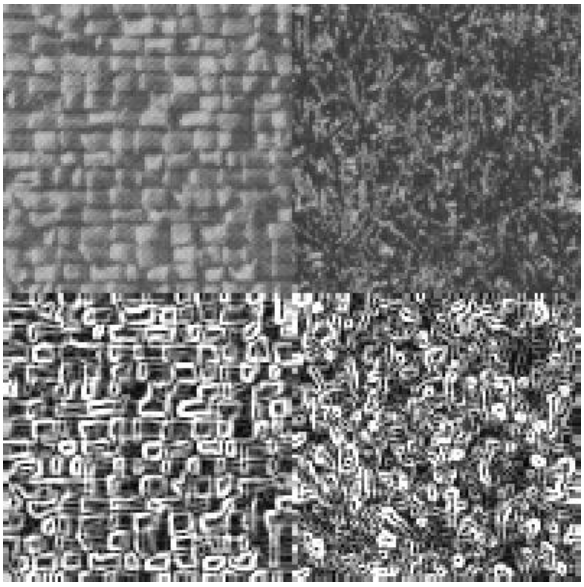


Figure 15.3: An Example of How the Sobel Edge Detector Does Not Work Well with a Texture

result of the range operator applied to the input using a 3x3 area. The range operator produced different gray levels for the different textures. The lower right quarter shows the result of histogram equalization (chapter 4) applied to the upper right quarter. This is not necessary for segmentation, but it does help highlight the result. The lower left quarter shows the result achieved by segmenting the range output using simple threshold segmentation (chapter 9).

Three other related "edge detectors" are the variance, sigma, and skewness (chapter 6). The standard deviation is the basis for these operators. Variance, uses the definition given in [15.1]. That definition of variance takes an nxn area, sums the squares of the center pixel - each neighbor, and takes the square root of this sum. Equation (15.1) shows this definition of variance.

$$variance_{Russ} = \sqrt{\sum(centerpixel - neighbor)^2} \tag{15.1}$$

If the center pixel differs from its neighbors, this variance will produce a large number. This is how it detects edges and produces many edges in a texture area. Figure 15.5 shows an example of the variance operator applied to a texture. This image has the same format as Figure 15.4. The

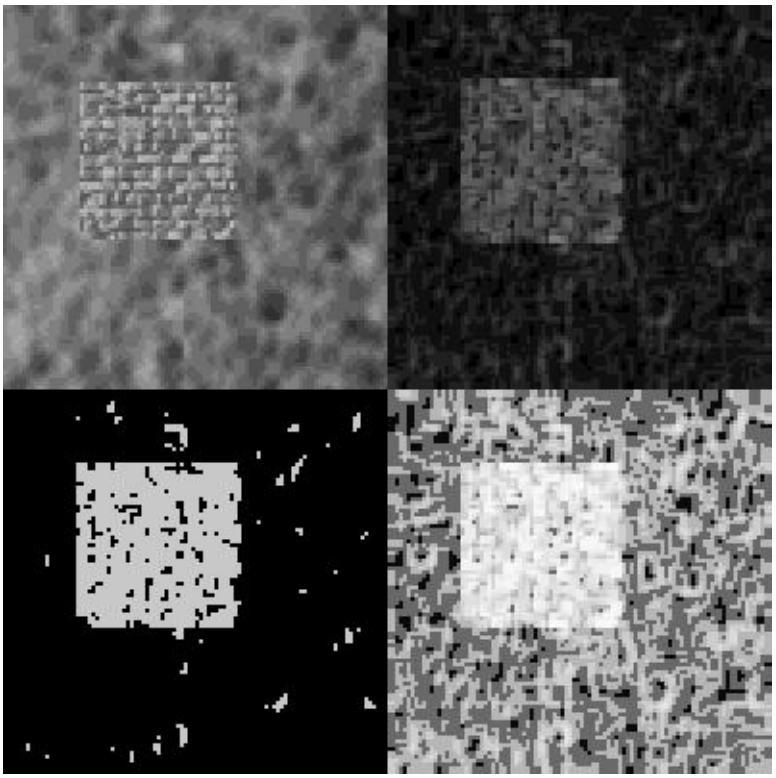


Figure 15.4: The Result of Applying the Range Edge Detector to a Texture

upper left quarter is the input texture and the upper right quarter is the result of the variance operator. Variance produced two different gray levels for the different textures. The histogram equalization result in the lower right quarter highlights the effect. The lower left corner shows the result of segmenting the textures using the variance output.

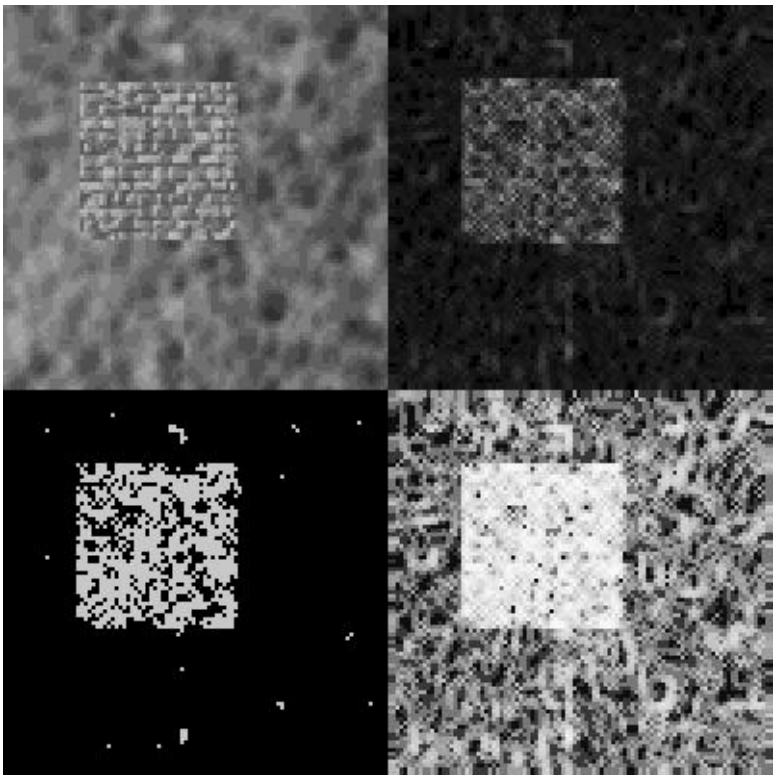


Figure 15.5: The Result of Applying the Variance Edge Detector to a Texture

A different, more classical definition of variance is found in [15.3] and shown in equation (15.2). This definition requires two passes through the nxn area of the image. The first pass calculates the mean or average pixel value of the area. The second pass calculates the variance. This operation takes more time than the variance defined in equation (15.1).

$$variance_{Levine} = \frac{1}{size\ of\ area} \sum (centerpixel - mean)^2 \tag{15.2}$$

After calculating the variance from equation (15.2), take the square root

to find σ as shown in equation (15.3). We can use σ as a texture measure and we will need it to calculate the skewness measure later.

$$\sigma = \sqrt{variance_{Levine}} \tag{15.3}$$

Listing 15.1 starts with the subroutine that implements the sigma operator. This subroutine has the same form as all the operators in this series. sigma works on a sizexsize area (3x3, 5x5, etc.). Once inside the main loop, it calculates the mean of the sizexsize area. Next, it runs through the sizexsize area a second time to sum the square of the difference between each pixel and the mean of the area and place this result in the variance variable. The final answer, σ, is the square root of the variance.

Figure 15.6 shows an example of the sigma operator applied to a texture. Just as in Figures 15.4 and 15.5, the upper right quarter shows the result of the sigma operator on the upper left quarter. It is hard to see anything in the sigma result. The sigma is almost the square root of variance, so its pixel values are much smaller and darker. Histogram equalization, shown in the lower right quarter, is necessary before attempting segmentation. Sigma produced two gray levels to represent two textures. The result in the lower left quarter is the segmentation of the lower right quarter.

The final "edge detector" type of operator is skewness [15.3]. Equation (15.4) shows the formula for skewness. Like the variance of equation (15.2), skewness requires two passes through an area to find the mean and then the σ. After this, calculate skewness using the σ. The skewness measure looks at the histogram of the nxn area of the image. Skewness measures the degree of symmetry in the histogram to see if the out lying points in the histogram favor one side or the other. If the histogram is symmetrical, skewness returns a low number. If the histogram favors one side or is "skewed" to one side, skewness returns a larger number.

$$skewness = \frac{1}{\sigma^3}\frac{1}{size\ of\ area}\sum(centerpixel - mean)^3 \tag{15.4}$$

Listing 15.1 next shows the subroutine that implements the skewness operator. skewness runs through the image array working on sizexsize areas. It makes one pass through the sizexsize area to calculate the mean. The second pass through the area calculates the variance value of equation (15.2) and the cube variable (the sum of the cube of the difference between each

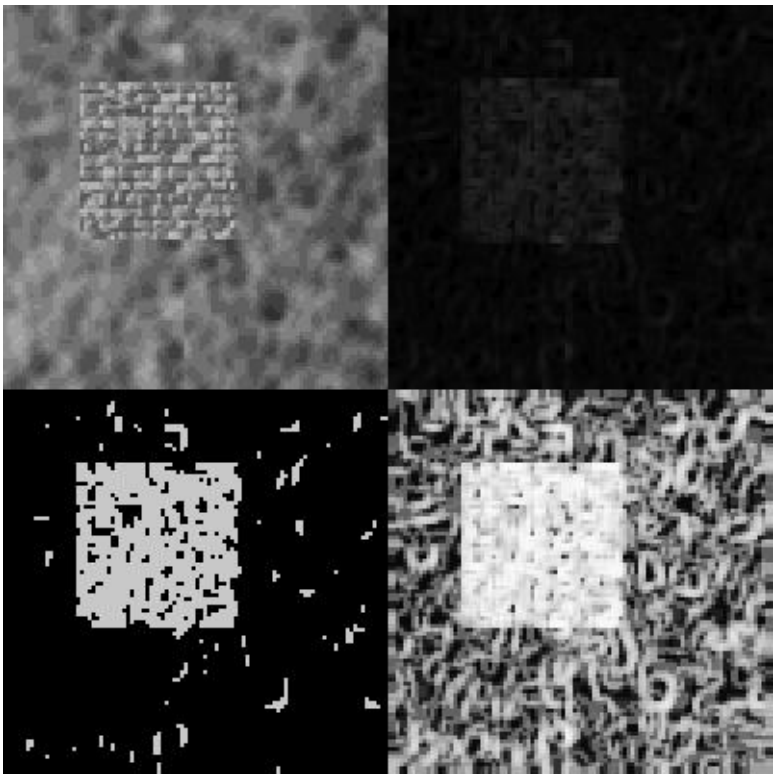


Figure 15.6: The Result of Applying the Sigma Edge Detector to a Texture

pixel and the mean of the area). After the second loop, skewness puts these values together as prescribed in equation (15.4) to form the skew answer.

Figure 15.7 shows the result of the skewness operator. The left half shows two synthetic textures. I created the far left texture by setting each pixel to a random number from the C rand() function. The right texture is a small checkerboard pattern. The two sections on the right half are the results of the skewness operator. The far right result is all zeros. The checkerboard pattern had a perfectly symmetrical histogram, so skewness returned zero everywhere. The histogram of the random pattern was also symmetrical, but skewed enough to return many non zero values. It is easy to segment the right half of the image.

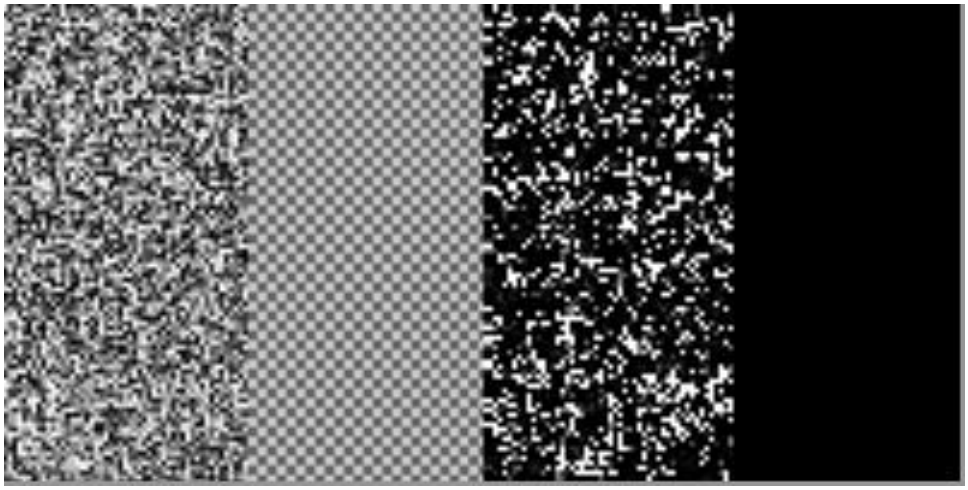


Figure 15.7: The Result of Applying the Skewness Operator to a Texture

15.4 The Difference Operator

The difference operator [15.3] is similar to edge detectors and can be useful in distinguishing textures. Equation (15.5) shows that the difference operator is merely the difference between a pixel and another pixel a given size away. It works on textures if the size specified matches the size of the pattern in a texture. If it matches well, the result is small numbers while other textures return larger numbers. The difference operator runs much faster than the variance, sigma, and skewness operators shown above and is quite effective on certain images.

$$output = absolutevalue(input[i][j]) - input([i + size][j + size]) \quad (15.5)$$

Listing 15.1 next shows the two subroutines that implement the difference operator. The subroutine adifference sets up size parameters while the difference_array subroutine performs the math. difference_array loops through the image array and calculates the difference as stated in equation (15.5). Notice how it is easy to vary the size parameter and look for the size of a texture.

Figure 15.8 shows the result of applying the difference operator on two distinct textures. The upper left quarter is the tightly woven texture shown earlier. The upper right quarter is the loose straw texture from Figure 15.2. The two lower quarters of the image contain the results of the difference operator. The lower left quarter appears brighter because the texture has greater differences in it than the straw texture. The difference operator distinguished these textures by producing areas with different gray levels.

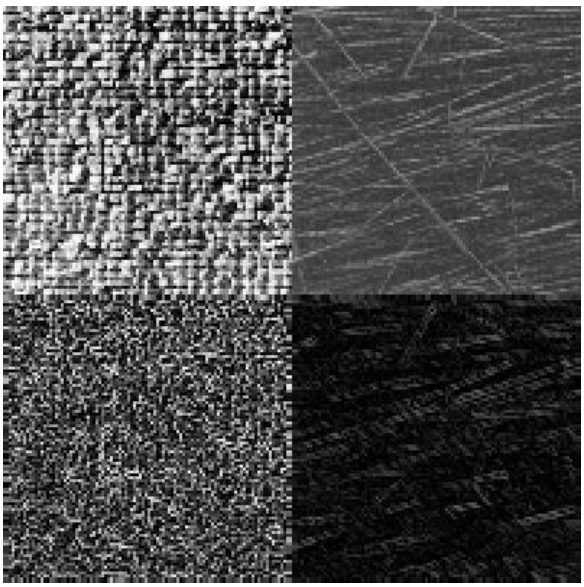


Figure 15.8: The Result of Applying the Difference Operator to a Texture

A variation of the difference operator is its mean operator [15.3]. This first applies the difference operator to an image and then replaces each pixel by the mean or average of the pixels in a sizexsize area. Equation (15.6)

describes the mean operator. The mean operator smoothes the result of the difference operator.

$$mean_{Levine} = \frac{1}{size\ of\ area} \sum (pixels\ of\ difference\ array) \tag{15.6}$$

The amean subroutine implements the mean operator and is the next routine shown in listing 15.1. It calls difference_array to calculate the differences in the input image. amean then smoothes the difference array by replacing each pixel with the average of the pixels in the surrounding sizexsize area.

Figure 15.9 shows the result applying the mean operator to the same textures processed by the difference operator in Figure 15.8. Note how the lower left quarter of Figure 15.9 is fuzzier than the corresponding quarter of Figure 15.8. This is to be expected because the mean is a smoothing operation. The two quarters in the lower half of Figure 15.9 are easily distinguished by their gray levels.

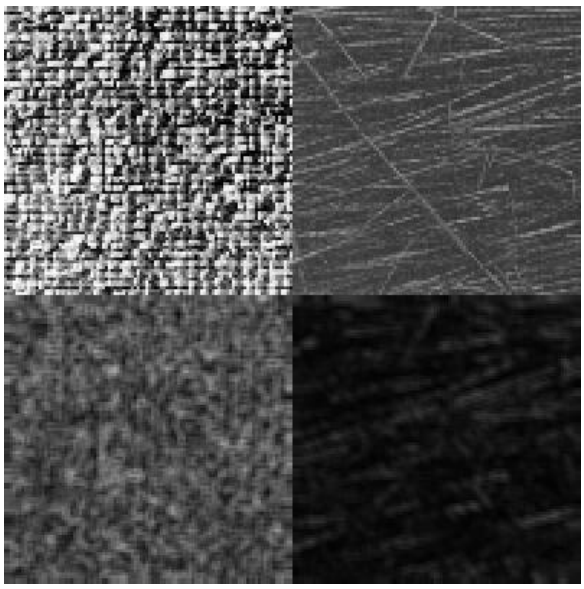


Figure 15.9: The Result of Applying the Mean Operator to the Same Texture as in Figure 15.8

15.5 The Hurst Operator

An excellent, but computationally expensive, texture operator is the Hurst operator [15.1]. The Hurst operator will process a texture area and return a single gray level. The idea is to look at ranges of pixel values in an area, plot them, fit the plot to a straight line, and use the slope of that line to measure the texture.

First, lets look at the ranges of pixels in an area. The range operator discussed earlier produced one range to describe an entire area. The Hurst operator produces several ranges for an area (n ranges for an nxn area). It calculates pixel value ranges for pixels that are an equal distance from the center pixel. Figure 15.10 shows three example size areas (other examples include 9x9, 11x11, etc.). The 7x7 area at the bottom of Figure 15.10 has pixel label 'a' in the center. The pixels labeled 'b' are all one pixel away from the center. The pixels labeled 'c' are the square root of two pixels from the center, the 'd' pixels are two pixels from the center, and so on. The Hurst operator calculates a 'b' range, a 'c' range, a 'd' range, on up to the 'g' range.

Figures 15.11 and 15.12 illustrate the range calculation. Figure 15.11 shows two 7x7 image sections. Image section 1 is smooth while image section 2 is rough. The tables in figure 15.4 show the range calculations. Look at image section 1 of figure 15.12 and examine the pixels that are one pixel away from the center. The largest value is 115, the smallest is 110, and this yields a range of 5. All the range values in the tables were calculated in this manner.

The final phase of the Hurst operator is to plot the distance and range values and find the slope of the line. Plot the natural logarithm of the distances on the vertical axis and the natural log of the ranges on the horizontal axis. This is a Hurst plot. Finally, fit these points to a straight line. The slope of the line is the answer. The notes in figure 15.12 state that Hurst plot for image section 1 had a slope of 0.99 and image section 2's slope was 2.0. Multiply these by a scaling factor of 64 to produce two different gray levels that represent two different textures.

Listing 15.1 next shows the source code that implements the Hurst operator. The hurst subroutine will work for the 3x3, 5x5, and 7x7 cases shown in figure 15.10. The first section of code sets the x array to the natural logarithm of the distances. hurst loops through the image and finds the ranges of pixel values for each pixel class shown in Figure 15.10. Each section of code puts the proper pixels into the elements array, sorts this array by calling

```
  3x3 case
    c b c
  d b a b d
    c b c

  5x5 case
  f e d e f
  e c b c e
  d b a b d
  e c b c e
  f e d e f

  7x7 case
    h g h
  f e d e f
h e c b c e h
g d b a b d g
h e c b c e h
  f e d e f
    h g h
```

Figure 15.10: Three Size Areas for the Hurst Operator

```
Image Section 1

                100 115 105
            105 115 105 110 115
        105 110 110 115 115 110 100
        105 110 110 110 115 105 100
        100 110 115 110 110 105 105
            110 115 100 110 105
                115 100 100


Image Section 2

                120 85  85
            115 110 90  100 115
        130 100 115 100 100 100 120
        120 110 95  80  95  95  125
        145 120 100 100 100 100 120
            130 130 100 100 85
                135 135 105
```

Figure 15.11: Two Example Image Sections

Image Section 1

Pixel Class	b	c	d	e	f	g	h
Distance	1	/2	2	/5	/8	3	/10
Brightest	115	115	110	115	115	115	115
Darkest	110	110	100	105	105	100	100
Range	5	5	10	10	10	15	15

Plot ln(range) vs ln(distance), slope = 0.99

Image Section 2

Pixel Class	b	c	d	e	f	g	h
Distance	1	/2	2	/5	/8	3	/10
Brightest	100	115	110	130	130	135	145
Darkest	95	100	90	100	85	85	85
Range	5	15	20	30	45	50	60

Plot ln(range) vs ln(distance), slope = 2.0

Figure 15.12: Values Calculated by the Hurst Operator

sort_elements, and puts the range in the prange variable. hurst fits the data to a straight line by calling the fit routine. The last step sets the output image to the slope of the line scaled by 64.

fit is a general purpose routine that fits data to a straight line. I took it from chapter 14 of "Numerical Recipes in C." [15.2]

Figure 15.13 shows the result of applying the Hurst operator to the same texture used in Figures 15.4, 15.5, and 15.6. The upper left quarter is the input image and the upper right quarter is the result of the Hurst operator. The lower right is the result of smoothing the Hurst output with a low pass filter. Smoothing blurs the result and makes segmentation easier. The lower left quarter is the final segmentation result.

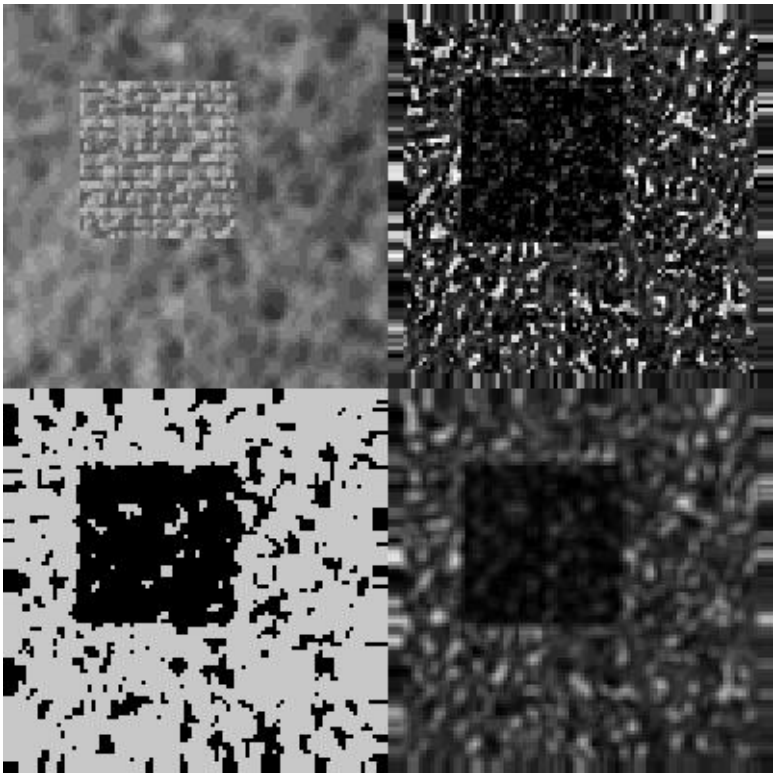


Figure 15.13: The Result of Applying the Hurst Operator to a Texture

Figure 15.14 shows an attempt at using the Hurst operator on a house image. The image in the left half of Figure 15.14 has several distinct textures such as trees, roof shingles, and bricks. The right half of Figure 15.14 shows the result of the Hurst operator. This looks like an edge detector and fails

miserably as a texture segmentation operator. The compare operator in the next section offers some hope.

Figure 15.14: The Failed Result of Applying the Hurst Operator to the House Image

15.6 The Compare Operator

A final texture operator uses the common sense approach of comparing one texture in the image against all textures. Select a small area in an image that contains one sample texture (such as the brick texture in left half of Figure 15.14). Move this small texture area through the entire image. At each pixel, subtract the image from the sample texture and use the average difference as the output. If the texture in the image is similar to the sample texture, the output will be small. If the texture in the image is different from the sample texture, the output will be large.

Listing 15.1 ends by showing the source code that implements the compare operator. The first part of compare mallocs the small array to hold the sample texture. Next, it copies the part of the input image that contains the sample texture into the small array. The main loop of compare sums the absolute value of the difference between the small array and the input image. The output is set to the sum (big) divided by the area (size*size).

Figure 15.15 shows an example of the compare operator. The upper left quarter shows the input image comprising the tightly woven texture next to the straw texture. The upper right quarter shows the result of taking a 3x3

area of the woven texture and comparing it to the input image. The result is dark and difficult to see. The lower right quarter shows the outcome of performing histogram equalization on the dark result. Now we can clearly see how the straw area produced a brighter output. This means that the texture in the straw area is not similar to the sample texture taken from the tightly woven area. The lower left quarter shows the segmentation result. The compare operator successfully produced different gray levels to distinguish two textures.

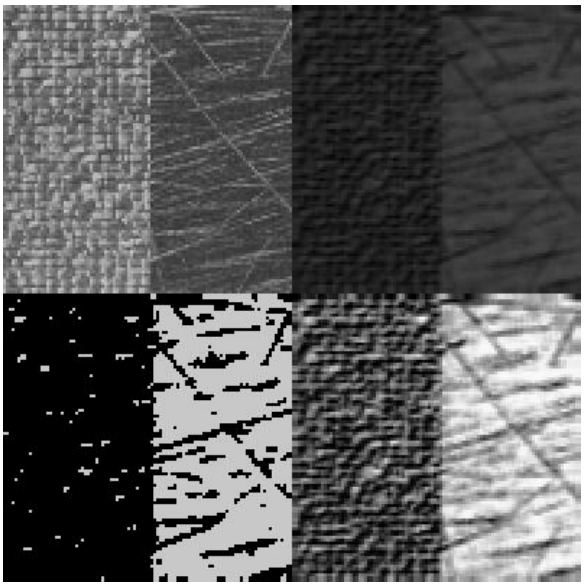


Figure 15.15: The Result of Applying the Compare Operator to a Texture

Figure 15.16 shows another example of the compare operator. This example illustrates both the power and the weakness of the compare operator. The left half of Figure 15.16 shows the house image. The right half shows the result of taking a 5x5 area of the brick texture and comparing it with the house image. The areas corresponding bricks are the darkest because their texture matches the sample brick area. The compare operator did a good job of separating the bricks from the other textures. Note the weakness of the compare operator. It lumped all the other textures (roof, trees, windows, shutters) into one category. The compare operator can only find one texture in an image.

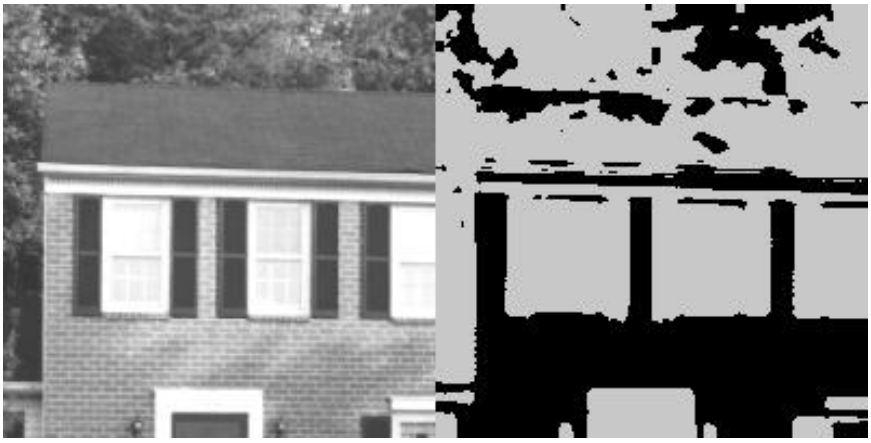


Figure 15.16: The Result of Applying the Compare Operator to the House Image

15.7 An Application Program

Listing 15.2 shows an application program that uses the texture operators with entire image files. It has the same form as the other application programs presented in this text.

15.8 Conclusions

This chapter described textures and several operators that help distinguish textures. We do not have a good definition of texture and any universally applicable texture operators. The operators presented here work well in certain situations. Experiment with them and experiment with the other pre-processing and post-processing operators from the series.

15.9 References

15.1 "The Image Processing Handbook, Third Edition," John C. Russ, CRC Press, 1999.
15.2 "Numerical Recipes in C," Press, William H, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, Cambridge University Press, 1988.
15.3 "Vision in Man and Machine," Martin D. Levine, McGraw-Hill, 1985.

Chapter 16

Random Dot Stereograms

16.1 Introduction

This chapter describes random dot stereograms and provides the source code so you can make your own. Stereograms are those strange 3-D pictures you see on books, calendars, and t-shirts everywhere. You focus past them and all the dots form what appears to be objects with surprising depth. Given different names by different people, these 3-D pictures are all similar in appearance and construction. Stereograms are a part of image processing because making a stereogram involves taking processing an existing image it to give it a new appearance.

16.2 Stereogram Basics

Let's first discuss why stereograms look the way they do. An easy way to do this is with text figures (we'll move on to dots later). One key to stereograms is divergent viewing or focusing on a point behind the image. Figure 16.1 shows that when two eyes (RI=right eye, LI=left eye) focus on a point x behind the picture, they see two different things (LI sees a and RI sees b). The brain mixes these two into a single image. Stereograms use this mixing to produce depth in the mind.

Figure 16.2 shows another basic concept in stereograms — the repeating pattern. The pattern 1234567890 runs from left to right and repeats itself for the width of the image. The repeating pattern has four properties. (1) The pattern runs horizontally, so orientation must be correct (you cannot turn

```
        focus point
            x
            ..
           .  .        picture
----------a----b--------------
         .      .
        .        .
       .          .
      .            .
     LI            RI
```

Figure 16.1: Divergent Viewing

```
        focus point
            x


1234567890123456789012345678 90


     LI            RI
```

Figure 16.2: The Repeating Pattern

the image sideways). (2) There are a fixed set of elements in the pattern. (3) All the elements in the pattern are the same size (fixed font in this case). (4) The width of the pattern must be less than the distance between your eyes. Property (4) is critical when making a stereogram's final output. Most people's eyes are a little more than an inch apart, so one inch is a good width for a repeating pattern. Display screens show 60-70 dots per inch while laser printers give 300 dots per inch. These two output devices require different pattern widths.

Figure 16.3 illustrates how a repeating pattern produces depth. The top part of figure 16.3 shows that a '1' was deleted from the pattern. The left and right eyes, viewing the image divergently, are looking at two different

```
  Deleted a 1 from here
          |
          |
1234567890234567890123456789 0

   so the brain adapts


1234567890
         1
         2345678901234567890 0 0 0
```

```
         Inserted an A here
                  |
                  |
123456789012345678A901234567890

        so the brain adapts


                 901234567890
                 A
123456789012345678




     LI            RI
```

Figure 16.4: Inserting an Element into the Pattern

background. The top part of the figure shows that a '1' was deleted and an 'A' was inserted. The two eyes see different things, so the brain adapts by tucking the '1' behind the '2' and tucking the 'A' behind the '8'. The center section appears farther away than the ends (the object drops back into the background).

Shortening and lengthening the pattern by 2, 3, 4, etc. creates other depth levels (2, 3, 4, etc.). Keep the length of the repeating pattern about twice as big as the number of depth levels.

Figure 16.6 puts these concepts together into a character stereogram using the repeating pattern 0123456789. On the fourth line the pattern is shortened by deleting an '8' and then lengthened by inserting an 'A'. When viewed divergently, you see a rectangle popping out of the center of the image.

Figure 16.7 shows the result of the final step in a random character stereogram. The viewer again sees a rectangle popping out of the background. Figure 16.7 is the result of a line by line random character substitution applied to figure 16.6. For example, take the first line of figure 6, substitute an 'R' for each '0', an 'E' for each '1', and so on using the substitution values shown in figure 16.8 to produce the first line of figure 16.7. A random number

```
Deleted a 1      Inserted an A
          |      |
          |      |
12345678902345678A901234567890

     so the brain adapts



1234567890     901234567890
         1     A
         2345678
```

```
LI            RI
```

Figure 16.5: Deleting and Inserting to Create an Object

```
012345678901234567890123456789012345
012345678901234567890123456789012345
012345678901234567890123456789012345
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345679012345679012345679 0A1234567
012345678901234567890123456789012345
012345678901234567890123456789012345
012345678901234567890123456789012345
012345678901234567890123456789012345
```

Figure 16.6: A Character Stereogram

```
REPGGXRPNRREPGGXRPNRREPGGXRPNRREPGGX
BZCNFWLQIJBZCNFWLQIJBZCNFWLQIJBZCNFW
JBBHAWDYDCJBBHAWDYDCJBBHAWDYDCJBBHAW
WSJRXJHGZWSJRXJHGZWSJRXJHGZWYSJRXJHG
AJQKCKLZAAJQKCKLZAAJQKCKLZAAYJQKCKLZ
SSQCTYDTASSQCTYDTASSQCTYDTASMSQCTYDT
EDRWUDXZFEDRWUDXZFEDRWUDXZFEUDRWUDXZ
RIFSUQHCSRIFSUQHCSRIFSUQHCSRKIFSUQHC
HRWTFDUKFHRWTFDUKFHRWTFDUKFHFRWTFDUK
ZPDPYZKZVZPDPYZKZVZPDPYZKZVZBPDPYZKZ
ISFRFQGVPMISFRFQGVPMISFRFQGVPMISFRFQ
KLASOLWJXPKLASOLWJXPKLASOLWJXPKLASOL
WEAAFJEQIOWEAAFJEQIOWEAAFJEQIOWEAAFJ
WXFAIGAYRUWXFAIGAYRUWXFAIGAYRUWXFAIG
```

Figure 16.7: A Random Character Stereogram

generator created these substitution values. The transition from figure 16.6 to 16.7 used a different set of substitution values for each line.

Figure 16.9 shows another example. First is the depth image with 0 being the background and 2 being closest to the viewer. The bottom is the result of a line by line random character substitution.

Now we have the basics of stereograms. Start with a depth image choose an appropriate pattern length (less than the distance between your eyes and twice as big as the number of depth levels); shorten and lengthen the pattern length according to changes in depth; and produce a stereogram with line by line random substitution. We also know that we can make character stereograms which are easy to e-mail.

Extending these concepts to dots vice characters is simple with the difference being in the random substitution. Dot stereograms have only two values (1 and 0 for white and black). If the output of the random number generator is odd, substitute a 1 and substitute a 0 otherwise. In figure 16.8 the G substitutes for both a 3 and a 4. In dot stereograms, a 1 will substitute for about half the values and a 0 will substitute for the others. Some stereograms have colored dots. If there are four colors, the random number is modulused by 4 (producing 0, 1, 2, and 3) with the result substituting for the four color values in the pattern.

```
0 -> R
1 -> E
2 -> P
3 -> G
4 -> G
5 -> X
6 -> R
7 -> P
8 -> N
9 -> R
```

Figure 16.8: Substitution Values for the First Line of Figures 16.6 and 16.7

16.3 Stereogram Algorithms

The next three figures give the algorithms for turning a depth image into a random dot or character stereogram. Figure 16.10 shows the main processing loop. These 12 steps repeat for every line in the depth file. In step 2, initialize the pattern according to the pattern length. If the pattern length is 100, place 0 through 99 (just one time) in the pattern array. Steps 4 through 8 loop through each element or pixel in a depth line. The values of this_pixel and last_pixel cause the pattern to grow or shrink. We perform the no change (copy the current element of the pattern array to the processed pattern array) on every pixel in the depth line. After saving this line of the processed pattern, perform random substitution and save the result. Processing and substitution occur line by line with each line being independent.

Figure 16.11 shows how to shorten a pattern by deleting size elements (size can by 1, 2, 3, etc.). First, save the input pattern array to a temp_pattern and set new_width to the pattern width less size. Next, increase the pattern's index by size to skip over those elements to delete. Steps 4 through 6 copy the temp_pattern back to the pattern with index skipping over elements. Step 7 shortens the current pattern width.

Figure 16.12 shows how to lengthen a pattern by inserting size elements. First, copy the pattern to a temp_pattern and empty the pattern. Then, put the new element(s) at the front of the pattern, increase the width, and copy the old pattern onto the end of the pattern. For example, if pattern is 0123456789, and size is 2, put AB at the start of the pattern and copy

```
0000000000000000000000000000000000000
0000000000000000000000000000000000000
0000000000000000000000000000000000000
0000000011111111111111110000000000000
0000000011111111111111110000000000000
0000000011111111111111110000000000000
0000000011111111111111110000000000000
0000000011111111122222222222222220000
0000000011111111122222222222222220000
0000000011111111122222222222222220000
0000000000000000022222222222222220000
0000000000000000022222222222222220000
0000000000000000022222222222222220000
0000000000000000022222222222222220000
0000000000000000022222222222222220000
0000000000000000000000000000000000000
0000000000000000000000000000000000000
0000000000000000000000000000000000000
```

```
REPGGXRPNRREPGGXRPNRREPGGXRPNRREPGGX
BZCNFWLQIJBZCNFWLQIJBZCNFWLQIJBZCNFW
JBBHAWDYDCJBBHAWDYDCJBBHAWDYDCJBBHAW
WSJRXJHGZWSJRXJHGZWSJRXYJHGZWSJRXYJH
AJQKCKLZAAJQKCKLZAAJQKCYKLZAAJQKCYKL
SSQCTYDTASSQCTYDTASSQCTMYDTASSQCTMYD
EDRWUDXZFEDRWUDXZFEDRWUUDXZFEDRWUUDX
RIFSUQHCSRIFSUQHCRIFSUQHCRIFSUQHKMCR
HRWTFDUKFHRWTFDUKHRWTFDUKHRWTFDUFJKH
ZPDPYZKZVZPDPYZKZZPDPYZKZZPDPYZKBNZZ
ISFRFQGVPMISFRFQGMISFRFQGMISFRFQVOGM
KLASOLWJXPKLASOLWPKLASOLWPKLASOLTXWP
WEAAFJEQIOWEAAFJEOWEAAFJEOWEAAFJZPEO
WXFAIGAYRUWXFAIGAUWXFAIGAUWXFAIGEIAU
JKZTVJGKWXJKZTVJGXJKZTVJGXJKZTVJKEGX
LRAXGLMTBILRAXGLMTBILRAXGLMTBILRAXGL
BDDNDODXEUBDDNDODXEUBDDNDODXEUBDDNDO
CKVRJBFRJPCKVRJBFRJPCKVRJBFRJPCKVRJB
```

Figure 16.9: A Depth Image and Random Character Stereogram Image

```
1.  Read a line from the depth file

2.  Initialize the pattern

3.  last_pixel = depth_line[0]

4.  Loop through the depth_line j=0, width

   5.  this_pixel = depth_line[j]

   6.  If this_pixel > last_pixel
       shorten the pattern

   7.  If this_pixel < last_pixel
       lengthen the pattern

   8.  Perform no change to the pattern

9.  Save the processed pattern

10. Perform random substitution

11. Save the random pattern

12. go back to 1 until you've read the entire depth file
```

Figure 16.10: The Stereogram Processing Loop

```
1.  Copy the pattern to a temp_pattern

2.  new_width = cuurent_width - size

3.  index = (index + size) modulus current_width

4.  Loop for new_index = 0,new_width

   5.  pattern[new_index] = temp_pattern[index]

   6.  index = (index + 1) modulus current_width

7.  current_width = new_width

8.  index = 0
```

Figure 16.11: The Shorten Pattern Algorithm

0123456789 onto the end producing AB1234567890. The max_width variable points to the last element added to the pattern ('B' in this example). The next time we insert an element into the pattern, max_width will tell us to add a 'C', then a 'D', and so on.

16.4 Source Code and Examples

Listing 16.1 shows the source code for the *cstereo* program that makes random character stereograms like is figure 16.9. The depth, processed pattern, and stereogram images are kept in text files so all I/O is with fgets and fputs. The main while processing loop implements the algorithm of figure 16.10, and the functions shorten_pattern and lengthen_pattern do the same for figures 16.11 and 16.12. The other functions are straightforward. Note how the function get_random_values contains several different ways (ifdef'ed out) to produce random substitution values.

Run *cstereo* with a command line like:
cstereo 10 36 dfile.txt sfile.txt ppfile.txt
where 10 is the pattern width, 36 the width of the text in the files, and the

```
1.  Copy pattern to a temp_pattern

2.  Blank out the pattern

3.  Put 'size' new elements in pattern

4.  new_width = current_width + size

5.  Copy temp_pattern onto the end of pattern

6.  current_width = current_width + size

7.  max_width = max_width + size

8.  index = 0
```

Figure 16.12: The Lengthen Pattern Algorithm

remaining parameters the names of the depth, stereogram, and processed pattern files.

Listing 16.2 shows the source code for the *pstereo* program that makes random dot stereograms and places them in gray scale image files. This program and *cstereo* are the same except for the type of data they process. *pstereo* uses the image I/O routines used by the other image processing programs in this text.

Run *pstereo* with a command line like:
pstereo 100 dfile.tif sfile.tif ppfile.tif
Figure 16.13 is a gray scale depth file. The background is at level 0 while the squares are at levels 2, 4, 5, and 8. Figure 16.14 is a random dot stereogram that *pstereo* produced from Figure 16.13 using a pattern length of 100. The image is 400x400 pixels printed at 4"x4" so the pattern length of 100 pixels translates to 1".

There are several ways to create depth files. Commercial paint and graphics programs can generate gray scale TIFF files. Use these to set the background to all black (0) and the objects to other levels. Take care as some programs make gray shades by setting black pixels next to white ones. These do not work well. I use the *pattern* program discussed briefly earlier in this

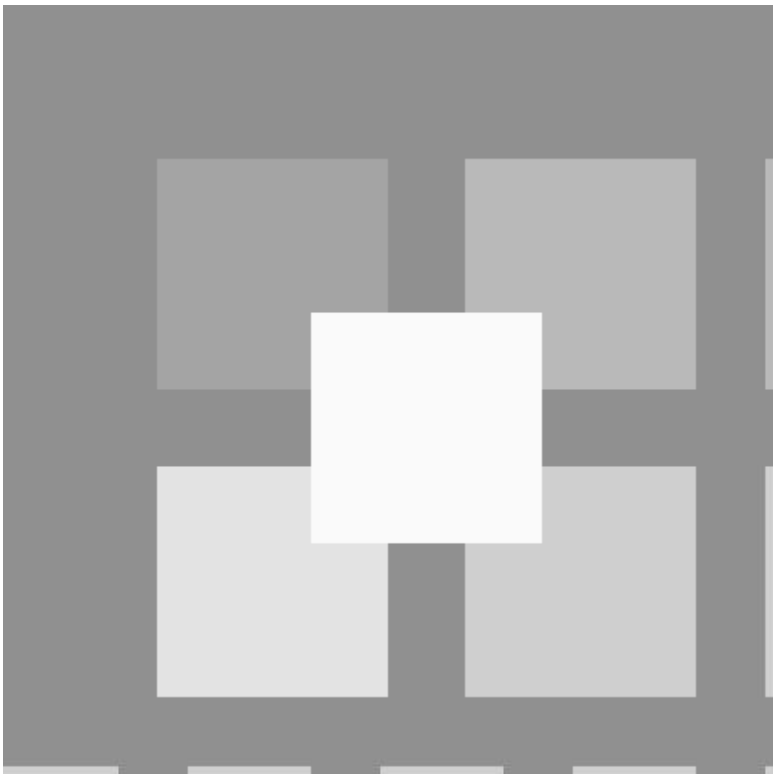


Figure 16.13: A Simple Depth File Image

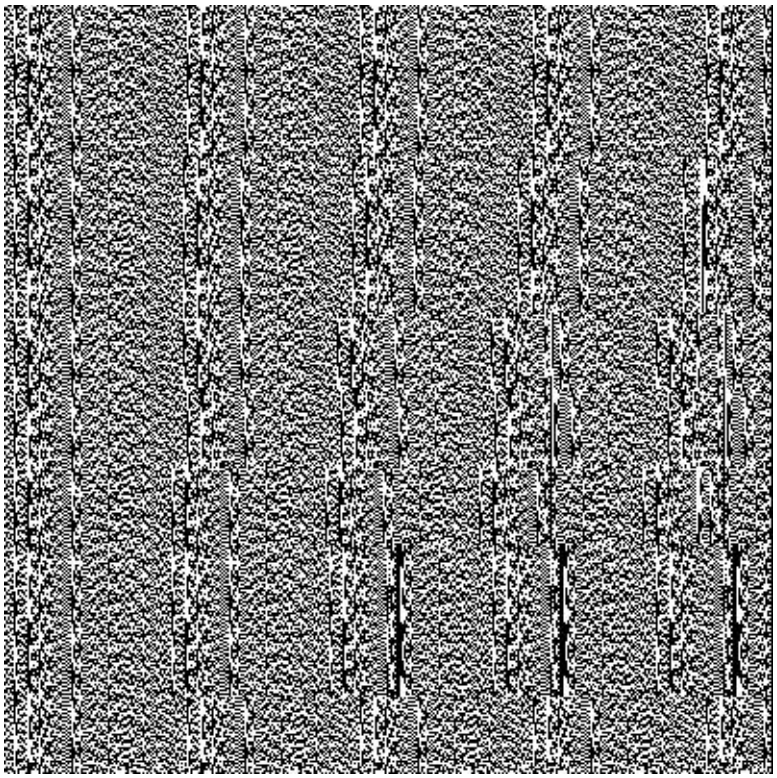


Figure 16.14: A Random Dot Stereogram from Figure 16.13

book.

Figure 16.15 is another random dot stereogram. What's in it?

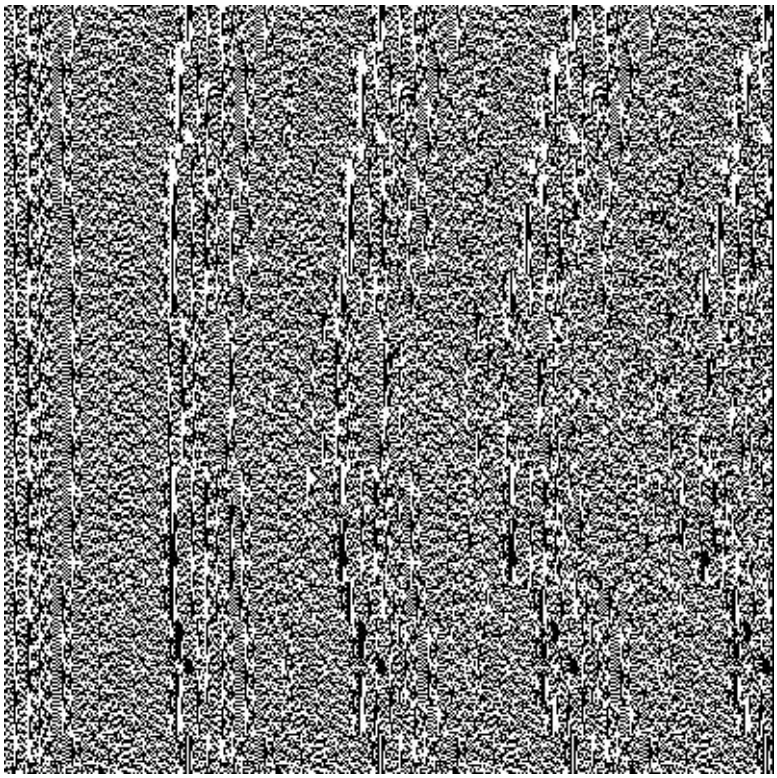


Figure 16.15: A Random Dot Stereogram

16.5 Colorfield Stereograms

A popular variation of the random dot stereograms presented above is the colorfield stereogram. Instead of replacing the processed pattern with random substitutions, colorfield stereograms use special patterns from a colorfield.

Figure 16.16 shows a colorfield image. It has a boy image repeated over and again. The boy image is a special pattern that will make up the substitutions when creating a stereogram. There is nothing special about the the boy image. What matters is that it has a set width like the 0 through 9 sequence shown earlier.

Figure 16.16: A "Colorfield" Image of Boys

Colorfield stereograms are different when the stereogram procedure calls for lengthening the pattern to create the illusion of depth. Items added to the pattern must come from the special pattern — the boy in this case. Figure 16.17 shows a resulting stereogram taken from Figure 16.16.

Figure 16.17: A Colorfield Stereogram from Figure 16.16

Figures 16.18 through 16.19 show another colorfied stereogram example. Figure 16.18 is the input special pattern image. Figure 16.19 is the input depth file, and Figure 16.20 is the final result.

Colorfield stereograms contain the same constraints as random dot stereograms. The pattern width (here the width of the boy and the width of the house) must be a little less than the distance between the viewer's eyes. The creator must consider the output device and the dots per inch in the image, etc.

The principles of colorfield stereograms also apply to the character stereograms shown earlier. Figure 16.21 shows a simple depth file. Figure 16.22 shows the final result of a character colorfield stereogram. This example used

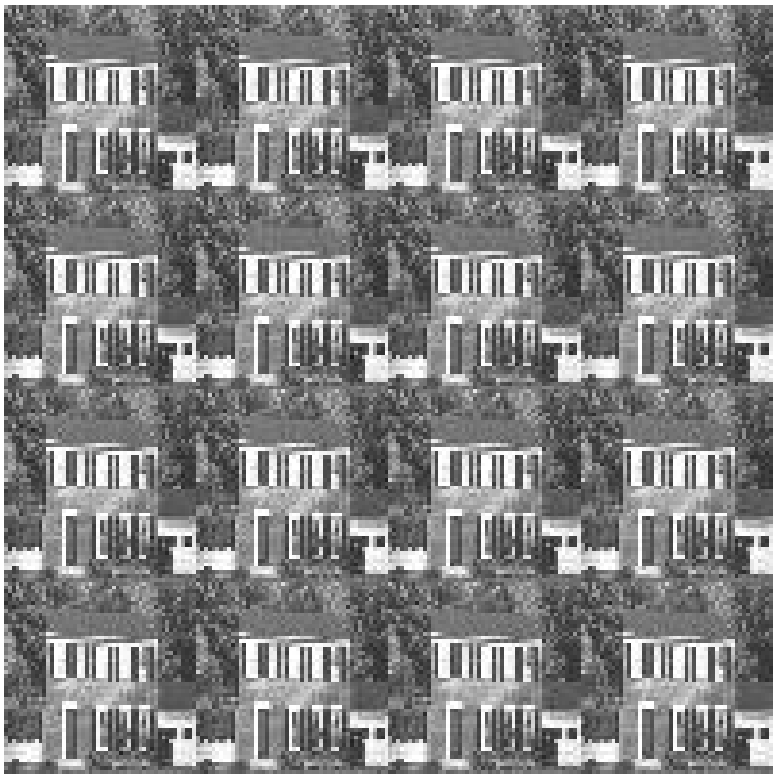


Figure 16.18: A Colorfield Image of Houses

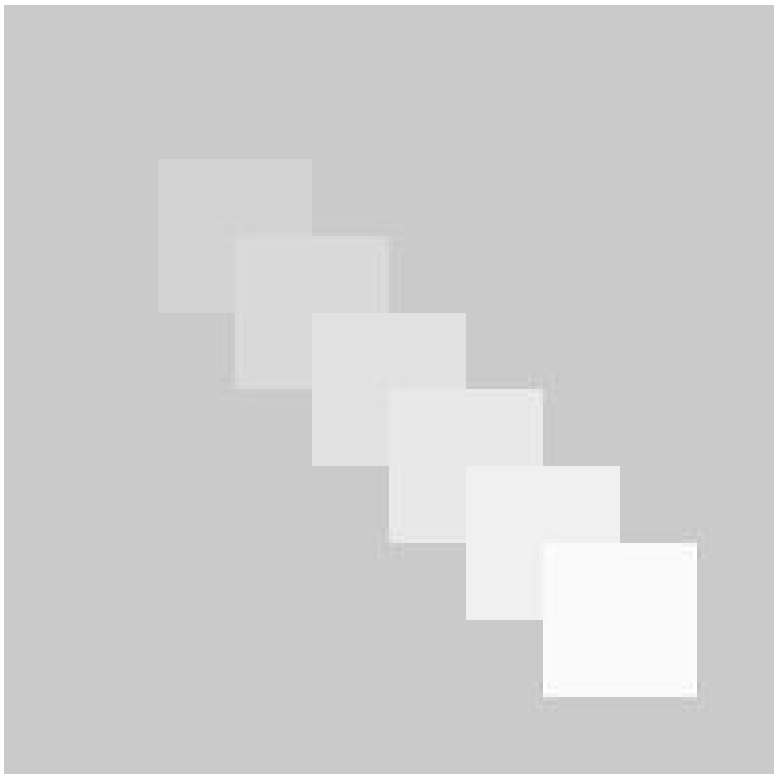


Figure 16.19: A Depth Image

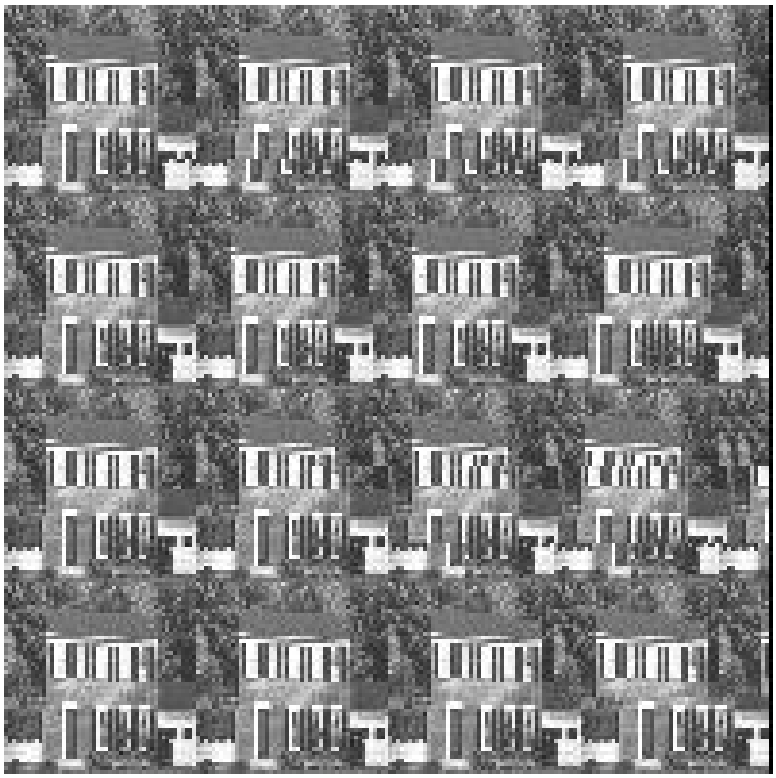


Figure 16.20: The Stereogram from Figures 16.18 and 16.19

```
000000000000011111111111111100000000000000000000000000000000
000000000000011111111111111100000000000000000000000000000000
000000000000011111111111111100000000000000000000000000000000
000000000000011111111111111100000000000000000000000000000000
000000000000011111111122222222222222200000000000000000000000
000000000000011111111122222222222222200000000000000000000000
000000000000011111111122222222222222200000000000000000000000
000000000000000000000022222222222222200000000000000000000000
000000000000000000000022222222222222200000000000000000000000
000000000000000000000022222222222222200000000000000000000000
000000000000000000000022222222222222200000000000000000000000
000000000000000000000022222222222222200000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
```

Figure 16.21: A Character Depth File

the letters "writecode" as the special repeating pattern (just like the boy and house above). This has a pattern length of 9, so if printed at 10 characters per inch, it is easy to view and see the depth.

Figure 16.22 illustrates the differences in colorfield stereograms. Shortening a pattern is simple and is the same as in random dot stereograms. The algorithm removes a letter of the pattern. The change is when lengthening the pattern. The inserted element cannot be just any character (in earlier examples an A was inserted into the 0 though 9 pattern). The inserted element must be from the special pattern (in this case a letter in the pattern writecode).

Listings 16.3 and 16.4 shows the *scstereo* and *spstereo* programs. These differ from *cstereo* and *pstereo* shown earlier only in the routines that lengthen the pattern and make the final substitutions. These new programs use a s_lengthen_patten and a special_substitution routine vice the former lengthen_pattern and random_substitution routines. These new routines ensure that only ele-

```
writecodewritcodewritcodewriitcodewriitcodewriitcodewriitco
writecodewritcodewritcodewriitcodewriitcodewriitcodewriitco
writecodewritcodewritcodewriitcodewriitcodewriitcodewriitco
writecodewritcodewritcodewriitcodewriitcodewriitcodewriitco
writecodewritcodewritcdewritcdewritcdodewritcdodewritcdodew
writecodewritcodewritcdewritcdewritcdodewritcdodewritcdodew
writecodewritcodewritcdewritcdewritcdodewritcdodewritcdodew
writecodewritecodewritodewritodewritocodewritocodewritocode
writecodewritecodewritodewritodewritocodewritocodewritocode
writecodewritecodewritodewritodewritocodewritocodewritocode
writecodewritecodewritodewritodewritocodewritocodewritocode
writecodewritecodewritodewritodewritocodewritocodewritocode
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
writecodewritecodewritecodewritecodewritecodewritecodewrite
```

Figure 16.22: A Character Colorfied Stereogram

ments from the special pattern are used.

16.6 Conclusions

This chapter has described how stereograms work and shown how to make your own. There are many ways to experiment with these. Filter the depth files to blur or round off the edges of objects. Try making depth files with different commercial packages. Use different pattern lengths and output devices. This is a fun topic and making stereograms can become more addictive than playing Doom. The *cstereo* program allows you to make character stereograms that you can e-mail. Send hidden messages through the mail, experiment, and have fun.

16.7 Reference

16.1 "Hidden Images: Making Random Dot Stereograms," Bob Hankinson, Alfonso Hermida, Que Corporation, 1994.

Chapter 17

Steganography: Hiding Information

17.1 Introduction

Steganography is the art of hiding information. It includes techniques to hide an image, a text file, and even an executable program inside a "cover" image without distorting the cover image. This paper will discuss the basic ideas of steganography and show how to hide text on an image via watermarking and hide an image in an image. Source code will be presented to implement these techniques. Extensions to these ideas are also available for those interested in augmenting the code shown. Further information on steganography is available in [17.1] and related web sites.

17.2 Hidden Writing

The word steganography comes from the Greek and literally means "hidden writing." People have used steganography through the centuries to hide messages. The messages are hidden in plain sight, because they are visible to people who know where to look.

Consider the sentence "Where real interesting technical exchanges can overcome dull entertainment."

The first letter of each word spells the message "write code." This is not hidden well. Better hiding methods use the second or third letter of each word or the first letter of the first word, second letter of the second word,

etc.

Steganography and cryptography are closely related. Cryptography scrambles a message to produce something that looks scrambled. The "write code" example could be scrambled to be "xsjuf dpef" (replace each letter with the letter that follows it in the alphabet). The scramble sometimes encourages prying eyes who see it as a challenge to unscramble. Steganography instead hides a message in a cover message. The result looks like something innocent, so prying eyes often dismiss it. Lawyers and libertarians debate if steganography is close enough to cryptography to regulate its use. To date, steganography remains unregulated.

17.3 Watermarking

A watermark adds information to a document or image by placing a logo or seal in plain sight. The watermark protects the owner's rights by showing ownership. TV broadcasters commonly do this by placing their logo in a corner of the broadcast picture. A watermark can be hidden in an image. Hiding the watermark does not change the appearance of the image. This protects the owner's rights, without disturbing the image.

Figures 17.1 through 17.4 show an example of hiding a watermark. Figure 17.1 shows a boy and Figure 17.2 shows a watermark. The watermark is white words on a black background. It is possible to use more complex watermarks, but white on black simplifies the program.

Figure 17.3 is the result of laying the watermark on top of the boy image. A value of 20 was added to each pixel of the boy image where the watermark image was white. This example did not hide the watermark.

Figure 17.4 shows the result of hiding the watermark on the boy image. A value of 2 was added to each pixel of the boy image where the watermark image was white. This small increase is not visible to the casual observer.

It is simple to recover the watermark by subtracting the original boy image (Figure 17.1) from Figure 17.4.

Listing 17.1 shows the source code that hides a watermark in an image and recovers it. The first part of listing 17.1 is the hiding program. After interpreting the command line, the code ensures the images are the same size, and allocates two arrays to hold the images. The hiding operation adds a factor to the image when the watermark image is non-zero. The last few lines of code write the result to a file and free the memory allocated for the

Figure 17.1: The Original Boy Image

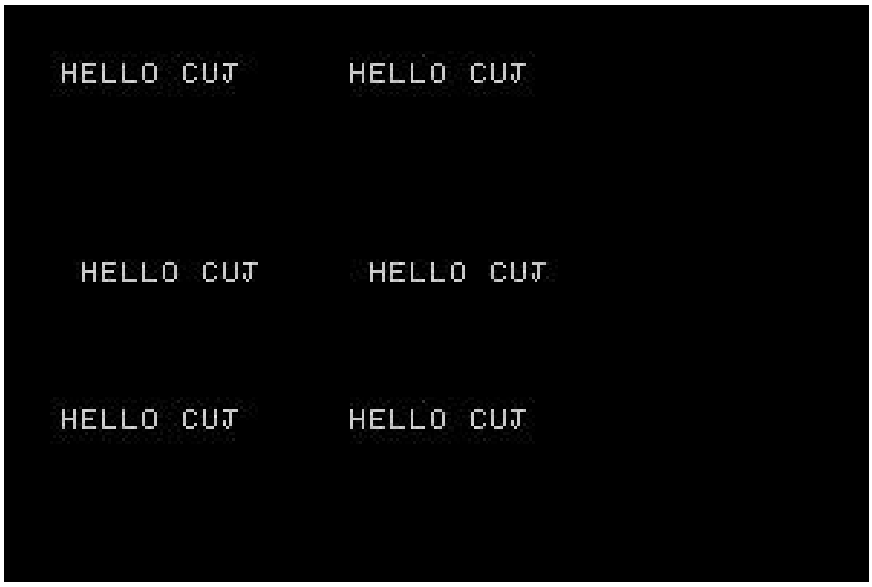


Figure 17.2: The Watermark Image

Figure 17.3: Overlaying the Watermark on the Boy Image

Figure 17.4: Hiding the Watermark on the Boy Image

image arrays.

It is easy to extract the watermark from the image. Use the *mainas* program from chapter 8 to subtract the original from the image with the hidden watermark.

17.4 Hiding Images in Images

Steganography enables hiding an image in another image. A message image hides in a cover image. Hiding alters the cover image, but the alterations are too slight to see. The process permits recovering the message image later, and the recovered message image matches the original exactly.

This is possible because images contain too much information. Common eight-bit gray scale images have 256 shades of gray. People can distinguish only about 40 shades of gray. The extra gray shades are useless. The same is true for color images. Images that use 24 bits per pixel have 16 million possible colors — too many to be useful.

Eight-bit gray scale images have more bits that are needed. Steganography uses the unneeded bits to hide the message image. Steganography stores the bits from the message image in the least significant bits of the cover image. No one can see the difference in the altered cover image, because no one can tell the difference between a 212 and a 213.

Figure 17.5 shows an example of how three pixels from a message image hide in a cover image. The first part of the figure shows the three pixels from the message image. The second part of the figure shows three rows of eight pixels from the cover image. The last part of the figure shows the same three rows of eight pixels after the three message image pixels were hidden. The least significant bits of the cover image are holding the message image pixels.

The pixel 99 from the cover image has bits 0110 0011. To hide the 0 (the first bit) requires clearing the least significant bit of the first pixel of the cover image. The 90 pixel remains 90 because its least significant bit is already a 0. The next two bits of the message image pixel are 1, so set the least significant bit of the next two pixels in the cover image. The 82 becomes an 83, and the 89 remains an 89. This process continues as every pixel in the cover image has its least significant bit cleared or set depending on the bit values of a pixel in the message image.

There is an eight-to-one limitation in this process. Each pixel in the message image has eight bits. Therefore, it needs one bit from eight different

```
Three pixels from message image

- 99
-103
-105

   Three by eight pixels from cover image

- 90- 82- 88-115-148-155-126- 90-
-119-103- 80- 76- 99-131- 91- 43-
-164-120- 85- 63- 59- 80-120- 91-

   Three by eight pixels from cover image
   after hiding the message in the image pixels

- 90- 83- 89-114-148-154-127- 91-
-118-103- 81- 76- 98-131- 91- 43-
-164-121- 85- 62- 59- 80-120- 91-
```

Figure 17.5: Hiding Message Image Pixels in a Cover Image

pixels in the cover image. The cover image must be eight times wider than the message image.

Figures 17.6 through 17.9 illustrate hiding a message image in a cover image. Figure 17.6 is the message image and Figure 17.7 is the original cover image. Figure 17.8 is the cover image after hiding the message image in it. Figures 17.7 and 17.8 are indistinguishable by visual inspection. The difference becomes apparent only when examining the pixel values like in Figure 17.5. Many of the pixel values of Figure 17.8 are one-off those in Figure 17.7.

Figure 17.6: The Cover Image

Figure 17.9 shows the message image after uncovering it from Figure 17.8. Figures 17.6 and 17.9 are exactly alike. The hiding and uncovering process did not alter the message image.

Listing 17.2 shows the source code that produced Figures 17.6 through 17.9. The listing starts with the main program that calls the subroutines to either hide or uncover a message image. This interprets the command line, ensures the input images exists, and checks their dimensions. The dimensions are critical.

Further down in listing 17.2 shows the subroutines hide_image and hide_pixels. The hide_image routine reads the message and cover images, calls the hide_pixels routine, and writes the result to the cover image file. The h_counter loop runs through the width of the message image. The main calling routine ensured that the cover image is eight times wider than the message image.

The hide_pixels routine does most of the work in the hiding operation. It must determine the value of every bit in every pixel in the message image. It must then set or clear the least significant bit of every pixel in the cover image accordingly. The routine uses two mask arrays to determine and alter

Figure 17.7: The Message Image

Figure 17.8: The Cover Image with the Message Image Hidden In It

Figure 17.9: The Unhidden Message Image

bits. The loop over i covers all the rows of the message and cover images. On each row, the loop over j examines each of the eight bits in the message image's pixel. The code then sets or clears the least significant bit of the corresponding pixel of the cover image.

The if(lsb) code is necessary because some images place the least significant bit first while others place it last (the old Intel and Motorola bit order issue). Depending on the bit order, the subroutine uses either mask1 or mask2 to set or clear bits.

Listing 17.2 also shows the subroutines uncover_image and uncover_pixels. These reverse the hiding process, so they are similar to the hiding. The uncover_image routine reads the cover image, calls uncover_pixels for every pixel in the image, and writes the recovered message image to disk.

The uncover_pixels routine does most of the work. It must determine if the least significant bit of each pixel in the cover image is 1 or 0. It then uses these bits to build up the eight bits in every pixel in the message image. The loop over i runs through every row in the images. The loop over j looks at eight pixels in the message image. If a pixel is odd, its least significant bit is 1, so the corresponding bit in the cover image must be set using the mask1 bit mask. Clearing bits is not necessary because the new_message variable was set to 0x00 prior to the loop over j.

17.5 Extensions

There are several extensions to the concepts presented here such as increasing the storage efficiency and hiding executable programs and text in images. The most obvious limitation to the image hiding shown earlier is the cover image must be eight times wider than the message image. This means using a narrow message image (Figure 17.6) and a wide cover image (Figure 17.7).

This ratio can be reduced to three to one. Instead of using the least significant bit of the cover image, use the two least significant bits. The cover image may change from gray shade 128 to 131 when hiding the message image. People cannot see that. The other part of increasing efficiency is to reduce the message image from eight-bit pixels to six-bit pixels. This means 64 shades of gray instead of 256. People can only see 40 shades of gray, so 64 is plenty. The six-bit pixels in the message image are hidden in two bits in the cover image. Hence the three to one ratio. Implementing this scheme would require changes in the routines shown in listing 17.2.

Steganography enables hiding executable programs inside images. In the previous discussion, the message image was a series of eight-bit values. An executable program is also a series of eight-bit values. The least significant bits of the cover image can hold the bits of the executable program. The cover image must contain eight times more pixels than the executable has bytes (four times more pixels if you use the two least significant bits as explained earlier). Uncovering the executable program from the cover image is just like uncovering the message image.

In the same manner, the cover image can hide a text file. The text file is a series of eight-bit bytes. The least significant bits in the cover image can hide the eight-bit text bytes. The cover image must contain eight times more pixels (or four times) than the text message. This use of steganography allows you to hide a message in an image, send the image to a friend (ftp or web site), and have them read it. The whole world can see the image without reading the message or even suspecting a message exists.

17.6 Conclusions

Steganography works as a technique to hide information in plain sight. Watermarks and copyrights can be placed on an image to protect the rights of its owner without altering the appearance of the image. Almost like magic, images, executable programs, and text messages can hide in images. The cover image does not appear altered. People look at the cover image and never suspect something is hidden. Your information is hidden in plain sight.

17.7 Reference

17.1 "Exploring Steganography: Seeing the Unseen," Neil F. Johnson, Sushil Jajodia, Computer, February 1998, pp. 26-34.,
http://patriot.net/~johnson/Steganography.

Chapter 18

Command-Line Programming

18.1 Introduction

This chapter will discuss using DOS .bat files. The previous chapters all included stand-alone application programs that applied image processing operators to entire image files. This chapter will show how to sequence these operators in a .bat file program. The same principles apply to writing UNIX scripts. I do not cover UNIX scripts here, but UNIX users will understand how to apply the same ideas to that fine system.

18.2 Batch Programming with .bat Files

Previous chapters included programs that processed entire image files. These programs were all command-line driven. Command-line programs can be difficult to use because you must remember the obscure command sequence.

A significant advantage of command-line programs is the user can call them from a .bat file. This is good because we often need to perform more than one operation on an image. There is usually pre-processing, processing, and post-processing as well as repetitive and comparative processing.

A .bat or batch file is a unformatted text file containing DOS commands. Batch program are run from the DOS prompt, and they call each of the commands they contains.

Batch files have two advantages. The first is they save typing long commands over and over. This is a physical advantage that applies to all situations. The second advantage is batch files permit performing useful image

```
rem   This is a simple .bat file
echo off
echo Hello World
copy a.doc b.doc
```

Figure 18.1: A .bat File

processing. A batch file can perform the same operation on a series of images, perform a series of similar operations on one image, and perform a long processing string on an image. The examples below will illustrate these advantages.

18.3 Basics of .bat Programming

Before launching into the examples, let's review a few basics of DOS batch file programming. All texts on DOS, such as the DOS manual, discuss .bat file programming. I'll cover the features used in the examples below.

Figure 18.1 shows a basic .bat file. The first line begins with rem and is a remark or comment. The rem places comments anywhere in the file. The second line turns off all echoing or displaying to the screen. Without the echo off, the .bat would display every statement as they executed. The third line displays the words Hello World on the screen. All echo statements except echo off display text to the screen. The final line is the DOS copy command. It copies file a.doc to file b.doc. Saving this file as first.bat and typing the DOS command:
first
would display the message Hello World and copy file a.doc to b.doc.

Figure 18.2 is the same as Figure 18.1 except the last statement. It uses the % sign to allow replaceable parameters on the command line. Saving this file as second.bat and typing the DOS command:
second a.doc b.doc c:
would display the message Hello World and copy the file a.doc to the file c:b.doc. The .bat file would replace %1 with a.doc (the first parameter), %2 with b.doc (the second parameter), and %3 with c: (the third parameter). Notice how %3%2 becomes c:b.doc. You can concatenate %s to make path names.

```
rem   This is a simple .bat file
echo off
echo Hello World
copy %1 %3%2
```

Figure 18.2: Another Simple .bat File

```
rem   This is a simple .bat file
echo off
echo Hello World
goto end
copy %1 %3%2
:end
```

Figure 18.3: A .bat File with Replaceable Parameters

Figure 18.3 is the same as Figure 18.2 with the additional statements goto end and :end. The goto statement transfers control down to the :end label. The result is the copy statement is skipped. Saving this file as third.bat and typing the DOS command:
third a.doc b.doc c:
would display the message Hello World and quit.

Figure 18.4 displays a final feature used in this chapter's .bat listings. The statement if "%3" == "" goto usage checks if the user entered a third parameter on the command line. If the third parameter equals an empty string, the user did not enter it so the if statement executes a goto usage command. Control jumps down to the :usage label, the program displays the usage text message, and ends. If the user entered a third parameter (%3 != ""), the echo and copy statements execute, control jumps down to the :end label, and the program ends. Saving this file as fourth.bat, and typing the DOS command:
fourth or
fourth a.doc or
fourth a.doc b.doc
would display the message
usage fourth source-file destination-file directory
and quit.

```
rem   This is a simple .bat file
echo off
if "%3" == "" goto usage

echo Hello World
copy %1 %3%2
goto end

:usage
echo "usage fourth source-file destination-file directory"

:end
```

Figure 18.4: A .bat File that Checks for Parameters

18.4 Uses and Examples

The first use of .bat files in image processing is repeating operations such as erosions, dilations, and filters. A .bat file could erode an image easier than typing the *mainsk* command three times.

Listing 18.1 shows the erode.bat file. This erodes an image three times. The *erode* command needs the name of the input file, output file, and a working directory. The statement if "%3" == "" goto usage checks to ensure the user entered three parameters. If the user did not, control jumps down to the usage message. The next section of commands runs the *mainsk* program three times to erode the input image using the mask erosion operator and places the result in the output file.

Notice how erode.bat uses the working directory %3 parameter. erode.bat creates two temporary files, tmp1.tif and tmp2.tif, in the working directory and later deletes them. If files named tmp1.tif and tmp2.tif are in the working directory, erode.bat will overwrite and delete them.

Typing the DOS command
erode a.tif b.tif f:
will erode the image file a.tif three times and place the result in b.tif. erode.bat will create and then delete the files f:tmp1.tif and f:tmp2.tif.

Listing 18.2 shows the dilate.bat file. It performs the same operations as erode.bat except it dilates the input image file three times. Listing 18.3

shows the median.bat file. It performs a 3x3 median filter on an input image file three times. It creates and deletes the same temporary files as the first two examples.

A second use of .bat files is running many similar operations to compare their results. Listing 18.4 shows the bedge.bat file that runs ten edge detectors discussed in this book. Running this allows the user to type one command, go away for a while, and return to decide which edge detector is best for this image.

bedge.bat is similar to the previous examples in how it tests for parameters and displays a usage message if necessary. The difference is how bedge.bat names the output files. The user enters an output file directory and an output file prefix. bedge.bat concatenates these with the number of the edge detector to name the output files. Typing the DOS command
bedge a.tif c:results aa
creates output files named c:resultsaa1.tif, c:resultsaa2.tif, etc.

Listing 18.5 shows the lowfilt.bat file that runs five different low-pass filters. It constructs the output file names using the same scheme as bedge.bat. Listing 18.6 shows the final comparison example, med357.bat. It runs the median filter on an input image using 3x3, 5x5, and 7x7 areas, and allows you to compare the results.

Another use of .bat files is to combine images into special images. Listing 18.7 shows the blabel.bat file. It creates a label and lays the label on top of an image. This .bat file requires you to type the message of the label inside the .bat file because I could not figure out a way to put the message on the command line (left as an exercise for the reader). The blabel.bat command line needs the name of the image to label, the output file name, a working directory, and the line and column of the image where you want the label. Before running this .bat file, the user must have created a tmp1.tif image with the same size of the input image. blabel.bat creates a small image file tmp1.tif in the working directory to hold the label message. It then dilates the label twice and exclusive ORs the dilated label with the original label (tmp2.tif, tmp3.tif, and tmp4.tif). Next, it overlays the message tmp4.tif on top of the input file %1 to create the output file %2. It ends by deleting the temporary files.

Listing 18.8 shows another .bat file that combines images. The fourside.bat file uses the side program to paste four images into one big image. This lets you display them all at once for comparisons. fourside.bat needs the names of the four input files, the output file, and a working directory.

It uses the working directory to paste inputs one and two into tmp1.tif and images three and four into tmp2.tif. It then connects the temporary files into the output file and deletes the temporary files.

Another use of .bat files is to string together steps in a processing sequence. Listing 18.9 shows the improve.bat file that performs histogram equalization and applies high-pass filtering. Equalization and high-pass filtering usually improve the appearance of a poorly scanned image.

18.5 Conclusions

This final chapter has discussed using DOS .bat files to call the stand-alone image processing application programs. The .bat files given here only hint at the possible applications. Use .bat files for typing- and time-intensive processes. Most practical and worthwhile image processing requires running experiments with strings of operations. It is not sensible to do this without using .bat files.

You can do the same in UNIX with scripts. My favorite are C shell scripts. The C shell commands are similar to programming in C and are far superior to DOS .bat files.

Chapter 19

A Tcl/Tk Windows Interface

19.1 Introduction

We live in a Windows world. There are still some of us who type command lines, but many people want a Windows interface. This chapter looks at this issue and how to use Tcl/Tk and Visual Tcl to add a portable Windows interface to the image processing programs described in this book.

19.2 The Need for a Windows Interface

The image processing programs in this book are command-line driven, and many people don't like using command-line programs. There are, however, some advantages to command-line programs. They are simple, and programmers can string them together using .bat programming and Unix scripts as discussed in chapter 18. Disadvantages of command-line programs are that the user needs to remember the command-line. The image processing programs all contain hints (if the user enters the command name and return, the program displays a reminder message), but the user still must remember which program.

We live in a Windows world whether that be Windows from Microsoft, Apple, or any one of the popular Unix windows systems. Users expect to be able to fill in a few blanks, point, and click to call processing routines.

A graphical user interface or GUI ("goo-ey") can be helpful to users. A user familiar with image processing should be able to sit in front of the GUI and perform useful tasks with no training. The GUI leads them to the

software by showing them possibilities in a familiar format. The main CIPS program presented in the first edition of this text did the same with a series of text menus. The user selected the option by entering the number next to it and answering questions when prompted. The GUI available now does the same. Instead of picking numbers and answering questions, the user will click on buttons and fill in blanks.

The image processing programs of this book present a special challenge to creating a GUI. These programs already exist. I wrote them over the past eight years in a command-line format. As the prior chapters have shown, the operators exist as subroutines with main routines doing the file I/O. There is structure to the software, but it was not written in a Windows application builder environment.

What I need is a GUI builder that can glue together existing software. Such GUI builders are known as scripting languages or "glueware." They are high-level languages that use the windowing facilities of operating systems. They can call the windowing routines and call the existing programs. Best yet, the programming effort is small compared to traditional programming languages.

19.3 Options

Several excellent scripting languages exist. Perl has been popular since the mid-1990s. It allows programmers to access and manipulate files in entire directories with a couple of statements. It, however, does not help with a GUI (at least not yet). I recommend C and C++ programmers look at Perl. Perl scripts could easily replace the .bat file programs of chapter 18 and be portable among DOS and Unix systems. Perl sources on the Internet and in the book store are too numerous to mention.

Two scripting languages that do help with GUIs are Microsoft's Visual Basic and Tcl/Tk. Visual Basic has been popular since the early 1990s. It is Microsoft's language of choice for creating a GUI quickly. The "programmer" fills a blank window with buttons, entries, and other familiar GUI elements by dragging these items from toolkit areas. Visual Basic writes the code to implement these items. The programmer fills in with specific calls to software where needed.

Visual Basic is an excellent tool. A drawback is that it is tied to Microsoft's operating systems (Windows 95, 98, NT, etc.), so it does not work

in the Unix world. Another negative for this application is the image processing software is in C, not Basic. Visual Basic does allow mixing languages, but I don't like to do that (bad experiences in the past).

Another scripting language that builds GUIs is Tcl/Tk. Tcl (pronounced "tickle") is a tool command language created by John K. Ousterhout [19.1]. Tk (pronounce "tee-kay") is a tool kit built on top of Tcl. Tcl was created as a scripting language in the Unix environment. Tcl code resembles Unix C-Shell and other scripts. It also resembles Perl, although their developments were independent. Tk was created on top of Tcl as an X11 (basic windows in Unix) toolkit.

Tcl/Tk surprised people as it became a great way to build GUIs. A couple of lines of code could create a window with buttons and other GUI elements.

Tcl/Tk is free and it works in both the Unix and Microsoft operating systems worlds. Tcl/Tk has a large following on the Internet and is available at [19.2].

A tool that helps create GUIs with Tcl/Tk is Visual Tcl (available at [19.3]). Tcl/Tk is so flexible that Stewart Allen wrote a GUI builder for Tcl/Tk in Tcl/Tk. Visual Tcl looks much like Visual Basic. The programmer drags GUI elements into a window, and Visual Tcl generates the Tcl/Tk source code. The programmer types a few commands to execute when the user clicks on buttons. The result is a Tcl/Tk script.

I used Tcl/Tk and Visual Tcl to create the GUI for the C Image Processing System. The next section shows a few windows from the GUI and points to the Tcl/Tk script that implements the GUI. The GUI has two dozen individual windows and calls the programs described in this text. I put it together in two dozen hours, and that includes learning how to use the tool.

The concept of scripting languages works, and I encourage all C/C++ programmers to consider them in conjunction with their old favorites.

19.4 The Tcl/Tk Graphical User Interface

The GUI I created for the C Image Processing System is simple, yet sufficient. It comprises windows that pop up and disappear. The user fills in blanks for the image processing parameters and clicks on command buttons to cause the programs to run.

Figure 19.1 shows the first window that appears when running the CIPS

Tcl script. When the user clicks one of the buttons shown, another window appears. For example, if the user clicks on the Stretch button (bottom of right row) the window shown in Figure 19.2 appears. When the user clicks on the exit button in the stretch window, it disappears.

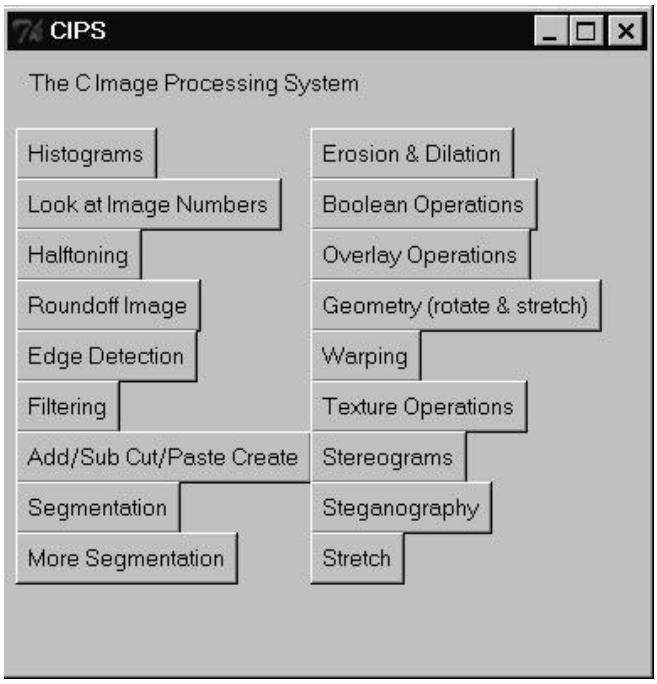


Figure 19.1: The Main CIPS Window

The stretch window in Figure 19.2 shows how the user calls programs. The user clicks on the entry fields (Input File, Output File, etc.) and types in the parameters. Once the parameters are filled, the user clicks on the Stretch button. This causes Tcl/Tk to call the *stretch* program described in chapter 13.

Figure 19.3 shows the window that calls the texture operators described in chapter 15. This window is more complicated than the stretch window. The user can call one of six different related operators. The parameter fields shown are not used the same on all six operators. Therefore, a Help button is available. When the user presses this, another window pops up that explains which fields are necessary for which operator.

This GUI is not fancy. It allows the user to call the same programs with the same parameters as presented throughout the book. The user, however,

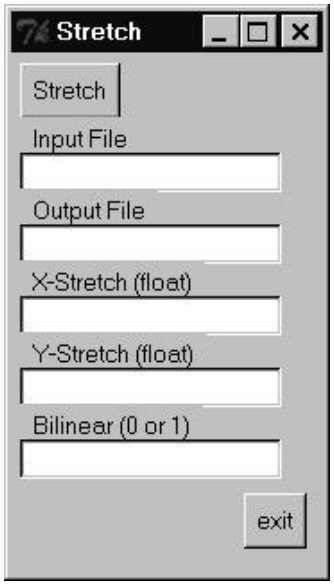


Figure 19.2: The Window for the stretch Program

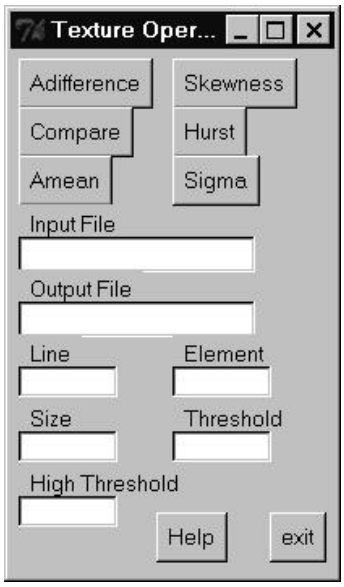


Figure 19.3: The Window for the Various Texture Operators

does not need to open a text window and type command lines.

An excellent attribute of this user interface is it is portable. Any machine that runs Tcl/Tk will run this user interface without any changes. Tcl/Tk runs on all Unix machines as well as Windows 95, 98, NT, etc.

Listing 19.1 shows the Tcl/Tk script generated by the Visual Tcl GUI builder. The only "source code" I entered was the exec statements. Scrolling down through the listing, is a "proc vTclWindow.top11" In this procedure is a statement that beings with "button $base.but28 -command." When the user clicks on button 28, Tcl/Tk executes the command contained in the curly braces that follow. That command calls the *himage* program described in chapter 4. I entered this and all the exec statements inside the curly braces. Visual Tcl did the rest.

19.5 Conclusions

This chapter has discussed adding a Windows interface or GUI to the image processing programs using Tcl/Tk and Visual Tcl. The result is a simple interface that allows the user to fill in blanks and click on buttons to call the image processing programs described in this book. The interface is portable as Tcl/Tk runs without any change in the Microsoft Windows and Unix X world.

19.6 Reference

19.1 "Tcl and the Tk Toolkit," John K. Ousterhout, Addison- Wesley, 1994.
19.2 http://www.scriptics.com
19.3 http://www.neuron.com

Appendix A

The makefile

To help manage the many different programs in the C Image Processing System, I used the GNU make program and a makefile. This kept the software system together, made it easier to make code changes, and made it much easier to distribute the source code to others. This appendix discusses makefiles in general, describes the makefile I used, and gives the commands needed to make the programs described in this book.

A makefile is an ASCII text file that lists files, their dependents, and the commands needed to bring files up to date. GNU's make program (like all make programs) examines the text in the makefile and runs commands if a file's dependents are newer than the file. The most common way to use make and a makefile is to compile .c files and link their .o files into executable programs.

Using a makefile is the easiest way to keep programs up to date and manage the complexity of changes in source code. If you make a change in one .c file, you do not need to worry how many executables depend on that file. The makefile lists the dependencies, and make sorts through them and calls the compiler and linker as needed.

The makefile makes it easy to distribute programs to others. Once the makefile and all the source code files are in one directory, the programmer types one command and comes back in an hour. make and the makefile do all the work.

You may need to make changes in the makefile shown in listing A1.1. It contains commands for the DJGPP C compiler. Different compilers and linkers use different commands. I have tried to make this makefile easy to port.

The best tutorial on make is "Managing Projects with make" [A1.1] which gives a general discussion of make and some very interesting ways to use it. make programs are powerful and useful once you understand how they work.

A.1 The Listings

Listing A1.1 shows the makefile for the CIPS software. There are many good ways to set up a makefile, so do not be afraid to revise this in your favorite style.

The first section defines the macros for the compile and link commands. Macros are abbreviations to use in the makefile. The CC macro is the command to compile a .c file and create a .o file. If you switch to another compiler, you only need to make one change here in the definition of CC. The $(CC) instances throughout the makefile will use the new definition. The same is true for all the macro definitions.

The next section describes three special targets allip, cleanobj, and cleanexe. These special targets do not have any dependent files, so make always executes the subsequent commands.

The allip target causes make to make all the executable programs in CIPS. cleanobj deletes all the .o files in the directory, and cleanexe deletes all the .exe files in the directory. These are good to clean out the directory and start over.

The next section of Listing A1.1 shows how to build each executable program. For each program, I listed the .c files needed, an abbreviation to list the .o files needed, and how to link the .o files into the executable.

The first program listed is texture. The first line
TSRC = imageio.c texture.c txtrsubs.c utility.c fitt.c
states that the source code files needed for this program are imageio.c, texture.c, txtrsubs.c, utility.c, and fitt.c.

The second line "TOBJ = ${TSRC:.c=.o}" uses shorthand to state that the object files needed are the same as the source files with the .c replaced by a .o. The next line "texture.exe: ${TOBJ}" says that the texture executable depends on the object files from the preceding line. The final line "(${LINK}) texture.exe ${TOBJ}" says to create the executable by linking the object files into texture.exe. make uses the LINK command defined at the top of the makefile. GNU make knows by default to create .o files from .c files by running the C compiler defined in the CC abbreviation at the top of the

makefile. All other programs in the makefile work the same as texture.

A.2 Commands to Build The C Image Processing System

To build all the programs in the CIPS software, Type make allip RETURN.

To build a single program, type make program-name RETURN. For example, to build texture.exe, type make texture.exe RETURN.

A.3 Reference

A1.1 "Managing Projects with make," Andrew Oram and Steve Talbott, O'Reilly and Associates, Inc., 1991.

A.4 Code Listings

```
##########################################################
#
#   21 June 1997
#
#   GNU make (the DJGPP version of it)
#
#
# GNU Make version 3.75, by Richard Stallman and Roland McGrath.
# Copyright (C) 1988, 89, 90, 91, 92, 93, 94, 95, 96
#       Free Software Foundation, Inc.
# This is free software; see the source for copying conditions.
# There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE.
#
# Report bugs to <bug-gnu-utils@prep.ai.mit.edu>.

##########################################################
#
#   H E L L O
#
#   This is the basic test program
#
```

```
HELLOSRC = hello.c hello2.c hello3.c
HELLOO   = ${HELLOSRC:.c=.o}

hello.exe: ${HELLOO}
${LINK} hello.exe ${HELLOO}




##########################################################
#
#   These are the commands for using the DJGPP compiler
#
#   to compile and link a single file program
#
#   gcc mfile.c -o myfile.exe -lm
#
#   where the -lm links in math for trig
#
#   to compile a C or C++ source file into an object file
#
#   gcc -o Wall myfile.c
#   gcc -c Wall myfile.cc
#
#   to link several C objects
#
#   gcc -o myprog.exe sub1.o sub2.o sub3.o
#
#   to link several C++ objects
#
#   gxx -o myprog.exe sub1.o sub2.o sub3.o
#
#
#
#   This is how to make a program.
#      list the source files
#   BASSRC = mainas.c addsub.c imageio.c
#      turn the .c files into .o object files
#   BASOBJ  = ${BASSRC:.c=.o}
#   mainas.exe: ${BASOBJ}
#      ${LINK}  mainas.exe ${BASOBJ}
#
#
############################################################
#
```

```
#   Define the basic macros
#
#

LIB      = -lm
CC       = gcc
COMPILE  = gcc -c
LINK     = gcc -o
PLUSLINK = gxx -o
MAKEFILE = -f makegcc


#####################################################




#####################################################
#
#      C I P S   P R O G R A M S
#
#   Special make targets:
#      allip - makes all .exe's - may not work because
#              the compiler runs out of heap space and
#              things like that.  Use the makeall.bat
#              file to do this
#
#      cleanobj - deletes all the .o   files
#      cleanexe - deletes all the .exe files
#


allip :
make -f makegcc medge.exe
make -f makegcc mfilter.exe
make -f makegcc mainas.exe
make -f makegcc maincp.exe
make -f makegcc side.exe
make -f makegcc stretch.exe
make -f makegcc create.exe
make -f makegcc mainseg.exe
make -f makegcc main2seg.exe
make -f makegcc pattern.exe
make -f makegcc boolean.exe
```

```
make -f makegcc mainover.exe
make -f makegcc invert.exe
make -f makegcc mainsk.exe
make -f makegcc ilabel.exe
make -f makegcc hidet.exe
make -f makegcc header.exe
make -f makegcc stega.exe
make -f makegcc texture.exe
make -f makegcc geometry.exe
make -f makegcc warp.exe
make -f makegcc scstereo.exe
make -f makegcc cstereo.exe
make -f makegcc pstereo.exe
make -f makegcc spstereo.exe
make -f makegcc showi.exe
make -f makegcc dumpi.exe
make -f makegcc dumpb.exe
make -f makegcc histeq.exe
make -f makegcc halftone.exe
make -f makegcc stretch.exe
make -f makegcc tif2bmp.exe
make -f makegcc bmp2tif.exe
make -f makegcc himage.exe
make -f makegcc round.exe

cleanobj:
del *.o

cleanexe:
del *.exe


#######################################################
#######################################################
#
#     Define the stand alone application programs



TSRC  = imageio.c texture.c \
        txtrsubs.c utility.c fitt.c
TOBJ  = ${TSRC:.c=.o}
texture.exe: ${TOBJ}
${LINK} texture.exe ${TOBJ}
```

```
GSRC  = geometry.c geosubs.c imageio.c
GOBJ  = ${GSRC:.c=.o}
geometry.exe: ${GOBJ}
${LINK} geometry.exe ${GOBJ}


WSRC  = warp.c warpsubs.c geosubs.c imageio.c
WOBJ  = ${WSRC:.c=.o}
warp.exe: ${WOBJ}
${LINK} warp.exe ${WOBJ}


DBSRC  = dumpb.c imageio.c
DBOBJ  = ${DBSRC:.c=.o}
dumpb.exe: ${DBOBJ}
${LINK}  dumpb.exe ${DBOBJ}


DISRC  = dumpi.c imageio.c
DIOBJ  = ${DISRC:.c=.o}
dumpi.exe: ${DIOBJ}
${LINK}  dumpi.exe ${DIOBJ}


SISRC  = showi.c imageio.c
SIOBJ  = ${SISRC:.c=.o}
showi.exe: ${SIOBJ}
${LINK}  showi.exe ${SIOBJ}


HTSRC  = halftone.c ht.c imageio.c
HTOBJ  = ${HTSRC:.c=.o}
halftone.exe: ${HTOBJ}
${LINK}  halftone.exe ${HTOBJ}



maincp: maincp.exe
MCPSRC  = imageio.c maincp.c cutp.c
MCPOBJ  = ${MCPSRC:.c=.o}
maincp.exe: ${MCPOBJ}
${LINK}  maincp.exe ${MCPOBJ}
```

```
SDSRC  = side.c imageio.c
SDOBJ  = ${SDSRC:.c=.o}
side.exe: ${SDOBJ}
${LINK} side.exe ${SDOBJ}



STSRC  = imageio.c geosubs.c stretch.c
STOBJ  = ${STSRC:.c=.o}
stretch.exe: ${STOBJ}
${LINK}  stretch.exe ${STOBJ}



CRSRC  =  imageio.c create.c
CROBJ  = ${CRSRC:.c=.o}
create.exe: ${CROBJ}
${LINK}  create.exe ${CROBJ}



TBSRC  =  imageio.c tif2bmp.c
TBOBJ  = ${TBSRC:.c=.o}
tif2bmp.exe: ${TBOBJ}
${LINK}  tif2bmp.exe ${TBOBJ}



BTSRC  =  imageio.c bmp2tif.c
BTOBJ  = ${BTSRC:.c=.o}
bmp2tif.exe: ${BTOBJ}
${LINK}  bmp2tif.exe ${BTOBJ}



IHSRC  =  imageio.c himage.c hist.c
IHOBJ  = ${IHSRC:.c=.o}
himage.exe: ${IHOBJ}
${LINK}  himage.exe ${IHOBJ}



PATSRC  = pattern.c imageio.c
PATOBJ  = ${PATSRC:.c=.o}
```

```
pattern.exe: ${PATOBJ}
${LINK}  pattern.exe ${PATOBJ}



MAIN2SRC  = edge2.c edge3.c segment.c \
            edge.c filter.c main2seg.c hist.c segment2.c \
            utility.c imageio.c
MAIN2OBJ  = ${MAIN2SRC:.c=.o}
main2seg.exe: ${MAIN2OBJ}
${LINK} main2seg.exe ${MAIN2OBJ}



SEGSRC = imageio.c hist.c mainseg.c \
         utility.c segment.c
SEGOBJ  = ${SEGSRC:.c=.o}
mainseg.exe: ${SEGOBJ}
${LINK}  mainseg.exe ${SEGOBJ}



BOOLSRC  = boolean.c boole.c imageio.c
BOOLOBJ  = ${BOOLSRC:.c=.o}
boolean.exe: ${BOOLOBJ}
${LINK}  boolean.exe ${BOOLOBJ}



OVERSRC  = mainover.c overlay.c imageio.c
OVEROBJ  = ${OVERSRC:.c=.o}
mainover.exe: ${OVEROBJ}
${LINK}  mainover.exe ${OVEROBJ}



INVSRC  = invert.c imageio.c
INVOBJ  = ${INVSRC:.c=.o}
invert.exe: ${INVOBJ}
${LINK}  invert.exe ${INVOBJ}



SKSRC  = mainsk.c imageio.c \
         skeleton.c ed.c  utility.c
```

```
SKOBJ  = ${SKSRC:.c=.o}
mainsk.exe: ${SKOBJ}
${LINK}  mainsk.exe ${SKOBJ}



ILSRC  = ilabel.c imageio.c
ILOBJ  = ${ILSRC:.c=.o}
ilabel.exe: ${ILOBJ}
${LINK}  ilabel.exe ${ILOBJ}



HESRC  = header.c tiffs.c cips2.c
HEOBJ  = ${HESRC:.c=.o}
header.exe: ${HEOBJ}
${LINK}  header.exe ${HEOBJ}



BMSRC  = medge.c edge.c edge2.c edge3.c imageio.c utility.c
BMOBJ  = ${BMSRC:.c=.o}
medge.exe: ${BMOBJ}
${LINK}  medge.exe ${BMOBJ}



BASSRC = mainas.c addsub.c imageio.c
BASOBJ  = ${BASSRC:.c=.o}
mainas: mainas.exe
mainas.exe: ${BASOBJ}
${LINK}  mainas.exe ${BASOBJ}



BMFSRC  = mfilter.c filter.c imageio.c utility.c
BMFOBJ  = ${BMFSRC:.c=.o}
mfilter.exe: ${BMFOBJ}
${LINK}  mfilter.exe ${BMFOBJ}



BRSRC  = round.c imageio.c
BROBJ  = ${BRSRC:.c=.o}
round.exe: ${BROBJ}
```

```
${LINK}  round.exe ${BROBJ}



HQSRC  = histeq.c hist.c imageio.c
HQOBJ  = ${HQSRC:.c=.o}
histeq.exe: ${HQOBJ}
${LINK}  histeq.exe ${HQOBJ}



BSGSRC  = stega.c imageio.c
BSGOBJ  = ${BSGSRC:.c=.o}
stega.exe: ${BSGOBJ}
${LINK}  stega.exe ${BSGOBJ}



BHDSRC  = hidet.c imageio.c
BHDOBJ  = ${BHDSRC:.c=.o}
hidet.exe: ${BHDOBJ}
${LINK}  hidet.exe ${BHDOBJ}



CSSRC  = cstereo.c
CSOBJ  = ${CSSRC:.c=.o}
cstereo.exe: ${CSOBJ}
 ${LINK} cstereo.exe ${CSOBJ}



PSSRC  = pstereo.c imageio.c
PSOBJ  = ${PSSRC:.c=.o}
pstereo.exe: ${PSOBJ}
 ${LINK} pstereo.exe ${PSOBJ}



SCSSRC  = scstereo.c
SCSOBJ  = ${SCSSRC:.c=.o}
scstereo.exe: ${SCSOBJ}
 ${LINK} scstereo.exe ${SCSOBJ}
```

```
SPSSRC  = spstereo.c imageio.c
SPSOBJ  = ${SPSSRC:.c=.o}
spstereo.exe: ${SPSOBJ}
 ${LINK} spstereo.exe ${SPSOBJ}
```

Listing A1.1 - The CIPS makefile

Appendix B

The Stand-Alone Application Programs

This appendix gives the names, a brief description, and the chapter containing each of the stand-alone application programs in the C Image Processing System.

round Chapter 1
This program saves part of an input image to an output image. It can save any rectangle in the input image.

bmp2tif Chapter 2
This program converts a .bmp image file to a .tif image file.

tif2bmp Chapter 1
This program converts a .tif image file to a .bmp image file.

showi Chapter 2
This program displays the numbers in an image to the screen. The user can move around in the image to see the image numbers.

dumpi Chapter 2
This program dumps the numbers in an image to a text file. The user can then print the image numbers from the text file using a word processor.

halftone Chapter 3
This program changes a gray scale input file into a 1/0 output file using a halftoning algorithm.

dumpb Chapter 3
This program dumps a 1/0 image to a text file. It sends a space for every 0 and an asterisk for every 1. The user can then print that space-asterisk file using a word processor. Combining halftone and dumpb, the user can print posters of images.

histeq Chapter 4
This program performs histogram equalization on an entire input image file and writes the equalized image to an output image file.

himage Chapter 4
This program calculates the histogram of an image and creates a picture of that histogram in another image file.

side Chapter 4
This program puts two entire images next to each other in an output image file. It will either place the input images side by side or one above the other. It creates an output image large enough to hold the two input images.

medge Chapter 6
This program applies any of 11 different edge detectors to an entire input image file and writes the edge detector output to an output image file.

mfilter Chapter 7
This program applies any of 11 different filters to an entire input image file and writes the filtered output to an output image file.

mainas Chapter 8
This program performs addition or subtraction between two entire input image files and writes the result to an output sum or difference file. The output equals either input1 + input2, or input1 - input2.

maincp Chapter 8
This program copies a section from one image file into another image file.

create Chapter 8
This program creates a blank image. You can use this blank image as a bulletin board onto which you can paste parts of other images.

invert Chapter 12
This program inverts the gray shade values of an entire input image file. It creates the output file to hold the inverted result.

mainseg Chapter 9
This program performs image segmentation and related operations on an entire input image file. It can call threshold and region-growing operators as well as histogram peak-based, histogram valley-based, and adaptive histogram- based segmentation. It creates an output image file to hold the result.

main2seg Chapter 10
This program performs image segmentation on an entire input image file. It can perform gray shade, edge only, and combination edge and gray shade segmentation. It creates an output image file to hold the result.

mainsk Chapter 11
This program performs erosion, dilation, outline, thinning, opening, closing, and medial axis transforms on an entire input image file.

boolean Chapter 12
This program performs the Boolean operations of AND, OR, EXCLUSIVE-OR, NAND, NOR, and NOT using two input image files. It creates an output image file to hold the result.

ilabel Chapter 12
This program allows you to write block letter text onto an image. It creates an output image file and writes the input image with the text on it into the output image.

mainover Chapter 12
This program overlays one entire input image file onto another using one of five different overlay operations. These operations are non-zero, zero, greater than, less than, and average. It creates an output image file to hold the result.

geometry Chapter 13
This program performs the geometric operations of displacement, stretching, rotation, and cross products.

stretch Chapter 13
This program stretches and compresses images in the horizontal and vertical directions.

warp Chapter 14
This program allows the user to use either control point or object warping on an image. It is the basis for morphing.

texture Chapter 15
This program allows the user to call any of the texture measures discussed in chapter 15.

cstereo Chapter 16
This program creates random dot stereograms on character "images" (text files of characters).

pstereo Chapter 16
This program creates random dot stereograms on regular gray scale pixel images.

scstereo Chapter 16
This program creates special colorfield stereograms on character "images" (text files of characters).

spstereo Chapter 16
This program creates special colorfield stereograms on regular gray scale pixel images.

hidet Chapter 17
This program hides a watermark image into a gray scale image.

stega Chapter 17
The program hides an image into another image using steganography. It also retrieves the hidden image back to its original format.

Appendix C

Source Code Tables of Contents

This appendix lists the table of contents of all the functions in the C Image Processing System. This appendix tells you which source code file contains which function.

There are two lists below. List A3.1 gives all the function names in alphabetical order next to the name of the corresponding source code file. List A3.2 gives the name of each source code file with the names of the functions it contains.

I used the C-DOC documentation tool to generate these lists. C-DOC is a very helpful tool, and I recommend it highly.

C-DOC
Software Blacksmiths Inc.
6064 St Ives Way
Mississauga, ONT Canada
(416) 858-4466

C.1 Listings

`adaptive_threshold_segmentation`	`segment.c`
`add_image_array`	`addsub.c`
`adifference`	`txtrsubs.c`
`allocate_image_array`	`imageio.c`
`amean`	`txtrsubs.c`
`and_image`	`boole.c`
`are_not_same_size`	`imageio.c`
`arotate`	`geosubs.c`
`average_overlay`	`overlay.c`

bi_full_warp_loop	warpsubs.c
bi_warp_loop	warpsubs.c
bilinear_interpolate	geosubs.c
calculate_histogram	hist.c
calculate_pad	imageio.c
can_dilate	skeleton.c
can_thin	skeleton.c
check_cut_and_paste_limits	cutp.c
closing	ed.c
compare	txtrsubs.c
contrast_edge	edge2.c
convert_matrix	fitt.c
copy_3_x_3	ed.c
copy_array_into_image	ilabel.c
create_allocate_bmp_file	imageio.c
create_allocate_tiff_file	imageio.c
create_bmp_file_if_needed	imageio.c
create_file_if_needed	imageio.c
create_image_file	imageio.c
create_resized_image_file	imageio.c
create_tiff_file_if_needed	imageio.c
cvector	fitt.c
detect_edges	edge.c
difference_array	txtrsubs.c
difference_edge	edge2.c
dilate_not_join	skeleton.c
dilation	ed.c
distance_8	skeleton.c
dmatrix	fitt.c
does_not_exist	imageio.c
dvector	fitt.c
edge_gray_shade_region	segment2.c
edge_region	segment2.c
edm	skeleton.c
enhance_edges	edge3.c
equate_bitmapheaders	imageio.c
equate_bmpfileheaders	imageio.c
equate_tiff_headers	imageio.c
erode_image_array	segment2.c
erosion	ed.c
exterior_outline	ed.c

```
extract_long_from_buffer          imageio.c
extract_short_from_buffer         imageio.c
extract_ulong_from_buffer         imageio.c
extract_ushort_from_buffer        imageio.c

f3tensor                          fitt.c
fill_line                         cstereo.c
filter_image                      filter.c
find_cutoff_point                 segment2.c
find_peaks                        segment.c
find_valley_point                 segment.c
fit                               fitt.c
fix_edges                         utility.c
flip_image_array                  imageio.c
free_convert_matrix               fitt.c
free_cvector                      fitt.c
free_dmatrix                      fitt.c
free_dvector                      fitt.c
free_f3tensor                     fitt.c
free_image_array                  imageio.c
free_imatrix                      fitt.c
free_ivector                      fitt.c
free_lvector                      fitt.c
free_matrix                       fitt.c
free_submatrix                    fitt.c
free_vector                       fitt.c
fsort_elements                    filter.c
fswap                             filter.c
full_warp_loop                    warpsubs.c

gammln                            fitt.c
gammq                             fitt.c
gaussian_edge                     edge3.c
gcf                               fitt.c
geometry                          geosubs.c
get_bitsperpixel                  imageio.c
get_image_size                    imageio.c
get_lsb                           imageio.c
get_random_values                 cstereo.c
gray_shade_region                 segment2.c
greater_overlay                   overlay.c
grow                              segment.c
gser                              fitt.c

half_tone                         ht.c
```

hide_image	stega.c
hide_pixels	stega.c
high_pixel	filter.c
hist_long_clear_buffer	hist.c
hline	himage.c
homogeneity	edge2.c
hurst	txtrsubs.c
imatrix	fitt.c
initialize_pattern	cstereo.c
insert_into_deltas	segment.c
insert_into_peaks	segment.c
insert_long_into_buffer	imageio.c
insert_short_into_buffer	imageio.c
insert_ulong_into_buffer	imageio.c
insert_ushort_into_buffer	imageio.c
interior_outline	ed.c
is_a_bmp	imageio.c
is_a_tiff	imageio.c
is_close	segment2.c
is_in_image	showi.c
is_not_empty	segment.c
is_not_emptyp	segment2.c
ivector	fitt.c
label_and_check_neighbor	segment.c
lengthen_pattern	cstereo.c
less_overlay	overlay.c
little_label_and_check	skeleton.c
low_pixel	filter.c
lvector	fitt.c
main	texture.c
manual_threshold_segmentation	segment.c
mask_dilation	ed.c
mask_erosion	ed.c
mat	skeleton.c
mat_d	skeleton.c
matrix	fitt.c
median_filter	filter.c
median_of	filter.c
nand_image	boole.c
no_change	cstereo.c
non_zero_overlay	overlay.c

nor_image	boole.c
not_image	boole.c
nrerror	fitt.c
object_warp	warpsubs.c
opening	ed.c
or_image	boole.c
paste_image_piece	cutp.c
peak_threshold_segmentation	segment.c
peaks_high_low	segment.c
perform_convolution	edge.c
perform_histogram_equalization	hist.c
pixel_grow	segment2.c
pixel_label_and_check_neighbor	segment2.c
pop	segment.c
popp	segment2.c
print_bm_header	imageio.c
print_bmp_file_header	imageio.c
print_color_table	imageio.c
print_side_usage	side.c
push	segment.c
pushp	segment2.c
quick_edge	edge.c
random_substitution	cstereo.c
range	edge2.c
read_bm_header	imageio.c
read_bmp_file_header	imageio.c
read_bmp_image	imageio.c
read_color_table	imageio.c
read_image_array	imageio.c
read_line	imageio.c
read_tiff_header	imageio.c
read_tiff_image	imageio.c
round_off_image_size	imageio.c
s_lengthen_pattern	scstereo.c
seek_to_end_of_line	imageio.c
seek_to_first_line	imageio.c
setup_filters	filter.c
setup_masks	edge.c
shorten_pattern	cstereo.c
show_edge_usage	medge.c

show_mainsk_usage	mainsk.c
show_screen	showi.c
show_stack	segment.c
show_stackp	segment2.c
show_texture_usage	texture.c
sigma	txtrsubs.c
skewness	txtrsubs.c
smooth_histogram	hist.c
sort_elements	utility.c
special_closing	skeleton.c
special_opening	skeleton.c
special_substitution	scstereo.c
stretch	stretch.c
submatrix	fitt.c
subtract_image_array	addsub.c
swap	utility.c
test_print_line	cstereo.c
thinning	skeleton.c
threshold_and_find_means	segment.c
threshold_image_array	segment.c
valley_high_low	segment.c
valley_threshold_segmentation	segment.c
variance	edge2.c
vector	fitt.c
vline	himage.c
warp	warpsubs.c
warp_loop	warpsubs.c
write_bmp_image	imageio.c
write_image_array	imageio.c
write_line	imageio.c
write_tiff_image	imageio.c
xor_image	boole.c
zero_histogram	hist.c
zero_line	pstereo.c
zero_overlay	overlay.c

Listing A3.1 - Alphabetical List of Functions

add_image_array	addsub.c

subtract_image_array	addsub.c
and_image	boole.c
nand_image	boole.c
nor_image	boole.c
not_image	boole.c
or_image	boole.c
xor_image	boole.c
fill_line	cstereo.c
get_random_values	cstereo.c
initialize_pattern	cstereo.c
lengthen_pattern	cstereo.c
no_change	cstereo.c
random_substitution	cstereo.c
shorten_pattern	cstereo.c
test_print_line	cstereo.c
check_cut_and_paste_limits	cutp.c
paste_image_piece	cutp.c
closing	ed.c
copy_3_x_3	ed.c
dilation	ed.c
erosion	ed.c
interior_outline	ed.c
mask_dilation	ed.c
mask_erosion	ed.c
exterior_outline	ed.c
opening	ed.c
detect_edges	edge.c
perform_convolution	edge.c
quick_edge	edge.c
setup_masks	edge.c
contrast_edge	edge2.c
difference_edge	edge2.c
homogeneity	edge2.c
range	edge2.c

variance	edge2.c
enhance_edges	edge3.c
gaussian_edge	edge3.c
convert_matrix	fitt.c
cvector	fitt.c
dmatrix	fitt.c
dvector	fitt.c
f3tensor	fitt.c
fit	fitt.c
free_convert_matrix	fitt.c
free_cvector	fitt.c
free_dmatrix	fitt.c
free_dvector	fitt.c
free_f3tensor	fitt.c
free_imatrix	fitt.c
free_ivector	fitt.c
free_lvector	fitt.c
free_matrix	fitt.c
free_submatrix	fitt.c
free_vector	fitt.c
gammln	fitt.c
gser	fitt.c
gammq	fitt.c
gcf	fitt.c
imatrix	fitt.c
ivector	fitt.c
lvector	fitt.c
matrix	fitt.c
nrerror	fitt.c
submatrix	fitt.c
vector	fitt.c
filter_image	filter.c
fsort_elements	filter.c
fswap	filter.c
high_pixel	filter.c
low_pixel	filter.c
median_filter	filter.c

```
median_of                          filter.c
setup_filters                      filter.c
high_pixel                         filter.c

arotate                            geosubs.c
bilinear_interpolate               geosubs.c
geometry                           geosubs.c

hline                              himage.c
vline                              himage.c

calculate_histogram                hist.c
hist_long_clear_buffer             hist.c
perform_histogram_equalization     hist.c
smooth_histogram                   hist.c
zero_histogram                     hist.c

half_tone                          ht.c

copy_array_into_image              ilabel.c

create_allocate_bmp_file           imageio.c
calculate_pad                      imageio.c
create_allocate_tiff_file          imageio.c
create_bmp_file_if_needed          imageio.c
allocate_image_array               imageio.c
create_file_if_needed              imageio.c
create_image_file                  imageio.c
create_resized_image_file          imageio.c
are_not_same_size                  imageio.c
create_tiff_file_if_needed         imageio.c
does_not_exist                     imageio.c
equate_bitmapheaders               imageio.c
equate_bmpfileheaders              imageio.c
equate_tiff_headers                imageio.c
extract_long_from_buffer           imageio.c
extract_short_from_buffer          imageio.c
extract_ulong_from_buffer          imageio.c
extract_ushort_from_buffer         imageio.c
flip_image_array                   imageio.c
```

free_image_array	imageio.c
get_bitsperpixel	imageio.c
get_image_size	imageio.c
get_lsb	imageio.c
insert_long_into_buffer	imageio.c
insert_short_into_buffer	imageio.c
insert_ulong_into_buffer	imageio.c
insert_ushort_into_buffer	imageio.c
is_a_bmp	imageio.c
is_a_tiff	imageio.c
print_bm_header	imageio.c
print_bmp_file_header	imageio.c
print_color_table	imageio.c
read_bm_header	imageio.c
read_bmp_file_header	imageio.c
read_bmp_image	imageio.c
read_color_table	imageio.c
read_image_array	imageio.c
read_line	imageio.c
read_tiff_header	imageio.c
read_tiff_image	imageio.c
round_off_image_size	imageio.c
seek_to_end_of_line	imageio.c
seek_to_first_line	imageio.c
write_bmp_image	imageio.c
write_image_array	imageio.c
write_line	imageio.c
write_tiff_image	imageio.c
show_mainsk_usage	mainsk.c
show_edge_usage	medge.c
average_overlay	overlay.c
greater_overlay	overlay.c
zero_overlay	overlay.c
less_overlay	overlay.c
non_zero_overlay	overlay.c
zero_line	pstereo.c

special_substitution	scstereo.c
adaptive_threshold_segmentation	segment.c
find_peaks	segment.c
find_valley_point	segment.c
grow	segment.c
insert_into_deltas	segment.c
insert_into_peaks	segment.c
is_not_empty	segment.c
label_and_check_neighbor	segment.c
manual_threshold_segmentation	segment.c
peak_threshold_segmentation	segment.c
peaks_high_low	segment.c
pop	segment.c
push	segment.c
show_stack	segment.c
threshold_and_find_means	segment.c
threshold_image_array	segment.c
valley_high_low	segment.c
valley_threshold_segmentation	segment.c
edge_gray_shade_region	segment2.c
edge_region	segment2.c
erode_image_array	segment2.c
find_cutoff_point	segment2.c
gray_shade_region	segment2.c
is_close	segment2.c
is_not_emptyp	segment2.c
pixel_grow	segment2.c
pixel_label_and_check_neighbor	segment2.c
popp	segment2.c
pushp	segment2.c
show_stackp	segment2.c
is_in_image	showi.c
show_screen	showi.c
can_dilate	skeleton.c
can_thin	skeleton.c

```
dilate_not_join                    skeleton.c
distance_8                         skeleton.c
edm                                skeleton.c
little_label_and_check             skeleton.c
mat                                skeleton.c
mat_d                              skeleton.c
special_closing                    skeleton.c
thinning                           skeleton.c
special_opening                    skeleton.c

print_side_usage                   side.c

hide_image                         stega.c
hide_pixels                        stega.c

stretch                            stretch.c

show_texture_usage                 texture.c

adifference                        txtrsubs.c
amean                              txtrsubs.c
difference_array                   txtrsubs.c
compare                            txtrsubs.c
hurst                              txtrsubs.c
sigma                              txtrsubs.c
skewness                           txtrsubs.c

fix_edges                          utility.c
sort_elements                      utility.c
swap                               utility.c

bi_full_warp_loop                  warpsubs.c
bi_warp_loop                       warpsubs.c
full_warp_loop                     warpsubs.c
object_warp                        warpsubs.c
warp                               warpsubs.c
warp_loop                          warpsubs.c
```

Listing A3.2 - List of Source Code Files and their Functions

Appendix D

Index of Image Processing Algorithms

This appendix lists the image processing algorithms discussed in this book. It gives the algorithm name and the chapter where you will find the algorithm. Given below are two lists. The first lists the algorithms in the order they appear in the book. The second lists the algorithms sorted alphabetically.

D.1 Algorithms Listed in Order of Appearance

Algorithm	Chapter
Reading BMP Files	1
Reading TIFF Files	1
Writing BMP Files	1
Writing TIFF Files	1
Displaying Image Numbers	2
Dumping Image Numbers	2
Halftoning	3
Calculating Histograms	4

Histogram Equalization	4
Displaying Histograms	4
Pasting Image Side by Side	4
Prewitt Edge Detector	5
Kirsch Edge Detector	5
Sobel Edge Detector	5
Quick Edge Detector	5
Homogeneity Edge Detector	6
Difference Edge Detector	6
Difference of Gaussians Edge Detector	6
Mexican Hat Edge Detector	6
Contrast-based Edge Detector	6
Variance Edge Detector	6
Range Edge Detector	6
Low-pass Filter (4 masks)	7
Median Filter	7
High-pass Filter (3 masks)	7
Image Addition	8
Image Subtraction	8
Image Cutting and Pasting	8
Blank Image Creation	8
Image Thresholding	9
Region Growing	9, 10
Histogram Smoothing	9
Histogram Peak Finding	9
Histogram Valley Finding	9
Histogram-based Segmentation	9
Adaptive Histogram Segmentation	9
Gray Shade Segmentation	10
Edge Based Segmentation	10
Edge and Gray Shade Segmentation	10
High-pixel Filter	10

Low-pixel Filter	10
Automatic Histogram Thresholding	10
Erosion	11
Dilation	11
Mask Erosion	11
Mask Dilation	11
Opening	11
Closing	11
Special Opening	11
Special Closing	11
Thinning	11
Skeletonization	11
Euclidean Distance Measure	11
Medial Axis Transform	11
Interior Outlining	11
Exterior Outlining	11
AND Boolean Operation	12
OR Boolean Operation	12
EXCLUSIVE-OR Boolean Operation	12
NAND Boolean Operation	12
NOR Boolean Operation	12
NOT Boolean Operation	12
Image Labeling	12
Masking	12
Zero Image Overlay	12
Non-zero Image Overlay	12
Greater Image Overlay	12
Less Image Overlay	12
Average Image Overlay	12
Image Diplacement	13
Image Stretching	13
Image Rotation	13
Image Cross Product	13
Bi-linear Interpolation	13

Control Point Warping	14
Object Warping	14
Image Shearing	14
Morphing	14
Hurst Operator	15
Sigma Operator	15
Skewness	15
Difference Operator	15
Compare Operator	15
Random Dot Stereograms	16
Colorfield Stereograms	16
Watermarks	17
Steganography	17

D.2 Algorithms Listed Alphabetical Order

Algorithm	Chapter
Adaptive Histogram Segmentation	9
AND Boolean Operation	12
Automatic Histogram Thresholding	10
Average Image Overlay	12
Bi-linear Interpolation	13
Blank Image Creation	8
Calculating Histograms	4
Closing	11
Colorfield Stereograms	16
Compare Operator	15
Contrast-based Edge Detector	6
Control Point Warping	14

Difference Edge Detector 6
Difference of Gaussians Edge Detector 6
Difference Operator 15
Dilation 11
Displaying Histograms 4
Displaying Image Numbers 2
Dumping Image Numbers 2

Edge and Gray Shade Segmentation 10
Edge Based Segmentation 10
Erosion 11
Euclidean Distance Measure 11
EXCLUSIVE-OR Boolean Operation 12
Exterior Outlining 11

Gray Shade Segmentation 10
Greater Image Overlay 12

Halftoning 3
High-pass Filter (3 masks) 7
High-pixel Filter 10
Histogram-based Segmentation 9
Histogram Equalization 4
Histogram Peak Finding 9
Histogram Smoothing 9
Histogram Valley Finding 9
Homogeneity Edge Detector 6
Hurst Operator 15

Image Addition 8
Image Cutting and Pasting 8
Image Cross Product 13
Image Diplacement 13
Image Labeling 12
Image Thresholding 9
Image Shearing 14
Image Stretching 13
Image Subtraction 8

Image Rotation 13
Interior Outlining 11

Kirsch Edge Detector 5

Less Image Overlay 12
Low-pass Filter (4 masks) 7
Low-pixel Filter 10

Mask Dilation 11
Mask Erosion 11
Masking 12
Medial Axis Transform 11
Median Filter 7
Mexican Hat Edge Detector 6
Morphing 14

NAND Boolean Operation 12
NOR Boolean Operation 12
NOT Boolean Operation 12
Non-zero Image Overlay 12

Object Warping 14
Opening 11
OR Boolean Operation 12

Pasting Image Side by Side 4
Prewitt Edge Detector 5

Quick Edge Detector 5

Random Dot Stereograms 16
Range Edge Detector 6
Reading BMP Files 1
Reading TIFF Files 1
Region Growing 9, 10

Sigma Operator 15

Skeletonization 11
Skewness 15
Sobel Edge Detector 5
Special Closing 11
Special Opening 11
Steganography 17

Thinning 11

Variance Edge Detector 6

Watermarks 17
Writing BMP Files 1
Writing TIFF Files 1

Zero Image Overlay 12

Appendix E

Bibliography

This appendix lists the image processing and computer programming books I used in preparing this book.

E.1 Image Processing Books

Kenneth R. Castleman, "Digital Image Processing," 1979, Prentice- Hall, Englewood Cliffs, N.J. 07632, ISBN 0-13- 212365-7.

This was my first image processing textbook. It was used for senior level courses in image processing. Although dated in its discussion of computer hardware and image processing systems, it discusses the fundamentals with good example images. No source code.

Martin D. Levine, "Vision in Man and Machine," 1985, McGraw- Hill, New York, NY, ISBN 0-07-037446-5.

This is a deep text used in graduate level courses on image processing and computer vision. It discusses many of the biological vision and perception systems as well as computer algorithms. It contains descriptions, equations, and good photograph examples for many image processing applications. No source code.

John H. Karl, "An Introduction to Digital Signal Processing," 1989, Aca-

demic Press, San Diego, California 92101, ISBN 0-12-398420-3

This is not an image processing text, but discusses the basics of digital signal processing. It is written in a narrative vs. mathematical style, so you can read it and learn about how computers process data. It covers several topics of interest to image processors such as Fourier transforms, filters, and two-dimensional signal processing.

Craig A. Lindley, Practical Image Processing in C, 1991, John Wiley & Sons, New York, NY, ISBN 0-471-54377-2.

This is the first of the new breed of image processing books that contains source code written expressly for the personal computer. As the title says, it is a practical not academic book. Much of the book concerns building and using an image digitizer. It covers the TIFF 5.0 specification as well as the PCX file format. It also discusses classic image processing operations. The text lists all source code and the accompanying floppy disk contains the source code, project files, executables, and a few images. The C code was written for the Borland Turbo C compiler.

John C. Russ, "The Image Processing Handbook," Third Edition, 1999, CRC Press, Boca Raton, Florida. ISBN 0-8493-4233- 3.

This is a newer all-encompassing source. It is a very complete text covering many common image processing algorithms and their applications. It discusses techniques and illustrates them with numerous excellent photographs. Although not one of the "practical" texts, it is not full of theory and I would not call it a purely academic work. It does not have any source code. I highly recommend this book.

Harley R. Myler and Arthur R. Weeks, "Imaging Recipes in C," 1993, Prentice-Hall, Englewood Cliffs, New Jersey. ISBN 0-13-189879-5.

This was written by two professors at the University of Central Florida. It is appropriate for both home use and early undergraduate classes. It discusses basic image processing using the UCFImage computer software imaging system. The book lists "code segments" (the computational parts of subroutines) and comes with a disk containing executables and images

not source code. The software can read images in the TIFF, GIF, BMP, and PCX file formats. This book blends theory, practice, and enough source code to make it very worthwhile.

E.2 Programming Books

Jack Purdum, "C Programming Guide," 1983, Que Corp., Indianapolis, Indiana. ISBN 0-88022-022-8.

You need to start somewhere, and this is where I learned how to program in C. This book may be hard to find, but it is very hard to beat as a place to begin.

Brian W. Kernighan and Dennis M. Ritchie, "The C Programming Language," Second Edition, 1988, Prentice-Hall, Englewood Cliffs, New Jersey. ISBN 0-13-110362-8.

You can't program in C without this on the shelf.

Andrew Oram and Steve Talbott, "Managing Projects with make," 1991, O'Reilly & Associates, Sebastopol, California. ISBN 0-937175-90-0.

This is the best tutorial available on makefiles. It is a must if you plan to write any programs that comprise more than one source code file.

Steve Oualline, "C Elements of Style," 1992, M&T Publishing, San Mateo, California. ISBN 1-55851- 291-8.

This is a good text on how to write C programs that you and others can read, understand, and maintain. This is an increasingly important topic in programming. The text also has a chapter on good style for writing makefiles.

P.J. Plauger, "The Standard C Library," 1992, Prentice-Hall, Englewood Cliffs, New Jersey. ISBN 0-13-131509-9.

This text spells out the standard library in great detail. An excellent refer-

ence.

Appendix F

Source Code Listings

The following sections contain the source code listings for chapters 1 through 19.

F.1 Code Listings for Chapter 1

```
   /*******************************************
   *
   *   read_image_array(...
   *
   *   This routine reads the image data from
   *   either a tiff or bmp image.
   *
   *******************************************/

read_image_array(file_name, array)
   char  *file_name;
   short **array;
{
   int ok = 0;

   if(is_a_tiff(file_name)){
      read_tiff_image(file_name, array);
      ok = 1;
   }
```

```
   if(is_a_bmp(file_name)){
      read_bmp_image(file_name, array);
      ok = 1;
   }

   if(ok == 0){
      printf("\nERROR could not read file %s",
             file_name);
      exit(1);
   }

}  /* ends read_image_array */




   /*******************************************
   *
   *   write_image_array(...
   *
   *   This routine writes the image data to
   *   either a tiff or bmp image.
   *
   *******************************************/

write_image_array(file_name, array)
   char  *file_name;
   short **array;
{
   int ok = 0;

   if(is_a_tiff(file_name)){
      write_tiff_image(file_name, array);
      ok = 1;
   }

   if(is_a_bmp(file_name)){
      write_bmp_image(file_name, array);
      ok = 1;
   }

   if(ok == 0){
```

```
      printf("\nERROR could not write file %s",
             file_name);
      exit(1);
   }

}  /* ends write_image_array */




   /*********************************************
   *
   *   create_image_file(...
   *
   *   This function creates an output image file.
   *   It uses the input image file as a pattern.
   *
   *********************************************/

create_image_file(in_name, out_name)
   char *in_name, *out_name;
{
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;
   struct tiff_header_struct tiff_file_header;

   if(is_a_tiff(in_name)){
      read_tiff_header(in_name, &tiff_file_header);
      create_allocate_tiff_file(out_name,
                                &tiff_file_header);
   }

   if(is_a_bmp(in_name)){
      read_bmp_file_header(in_name,
                           &bmp_file_header);
      read_bm_header(in_name, &bmheader);
      create_allocate_bmp_file(out_name,
                               &bmp_file_header,
                               &bmheader);
   }

}  /* ends create_image_file */
```

```
   /*********************************************
   *
   *   get_image_size(...
   *
   *   This function reads the rows and cols
   *   from the header of either a tiff or bmp
   *   image file.
   *
   *   IF IT CANNOT FIND THIS INFORMATION,
   *   it returns a ZERO.
   *
   *********************************************/

int get_image_size(file_name, rows, cols)
   char *file_name;
   long *cols, *rows;
{
   int is_bmp  = 0,
       is_tiff = 0,
       result  = 0;
   struct bitmapheader bmph;
   struct tiff_header_struct tiffh;

   if(is_a_bmp(file_name)){
      is_bmp = 1;
      read_bm_header(file_name, &bmph);
      *rows = bmph.height;
      *cols = bmph.width;
   }  /* ends if is_a_bmp */

   if(is_a_tiff(file_name)){
      is_tiff = 1;
      read_tiff_header(file_name, &tiffh);
      *rows = tiffh.image_length;
      *cols = tiffh.image_width;
   }  /* ends if is_a_bmp */

   if(is_bmp == 1  ||  is_tiff == 1)
      result = 1;

   return(result);
```

```
}  /* ends get_image_size */
```

```
   /****************************************
   *
   *   allocate_image_array(...
   *
   *   This function allocates memory for
   *   a two-dimensional image array.
   *
   ****************************************/

short **allocate_image_array(length, width)
   long  length, width;
{
   int i;
   short **the_array;

   the_array = malloc(length * sizeof(short  *));
   for(i=0; i<length; i++){
      the_array[i] = malloc(width * sizeof(short ));
      if(the_array[i] == '\0'){
         printf("\n\tmalloc of the_image[%d] failed", i);
      }  /* ends if */
   }  /* ends loop over i */
   return(the_array);

}  /* ends allocate_image_array */
```

```
   /****************************************
   *
   *   free_image_array(...
```

```
   *
   *   This function frees up the memory
   *   used by a two-dimensional imaage array.
   *
   ****************************************/

int free_image_array(the_array, length)
   short **the_array;
   long  length;
{
   int i;
   for(i=0; i<length; i++)
      free(the_array[i]);
   return(1);
}  /* ends free_image_array */




   /*********************************************
   *
   *   create_file_if_needed(...
   *
   *   This function creates an output file
   *   if it does not exist.  It can create
   *   the output file as a tiff or a bmp
   *   based on the input file type.
   *
   **********************************************/

create_file_if_needed(in_name, out_name, array)
   char *in_name, *out_name;
   short **array;
{
   if(is_a_tiff(in_name)){
      create_tiff_file_if_needed(in_name,
                                 out_name,
                                 array);
   }  /* ends if is a tiff */

   if(is_a_bmp(in_name)){
      create_bmp_file_if_needed(in_name,
                                out_name,
```

```
                                array);
   }  /* ends if is a tiff */

}  /* ends create_file_if_needed */




   /*********************************************
   *
   *   create_resized_image_file(...
   *
   *   This function creates an output image file.
   *   It uses the input image file as a pattern.
   *
   **********************************************/

create_resized_image_file(in_name, out_name,
                          length, width)
   char *in_name, *out_name;
   long length, width;
{
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;
   struct tiff_header_struct tiff_file_header;

   if(is_a_tiff(in_name)){
      read_tiff_header(in_name, &tiff_file_header);
      tiff_file_header.image_length = length;
      tiff_file_header.image_width  = width;
      create_allocate_tiff_file(out_name,
                                &tiff_file_header);
   }

   if(is_a_bmp(in_name)){
      read_bmp_file_header(in_name,
                           &bmp_file_header);
      read_bm_header(in_name, &bmheader);
      bmheader.height = length;
      bmheader.width  = width;
      create_allocate_bmp_file(out_name,
                               &bmp_file_header,
                               &bmheader);
```

```
   }

}  /* ends create_resided_image_file */




    /**********************************************
    *
    *    does_not_exist(...
    *
    *    This function checks the disk to see if
    *    a file exists.  If the file is there this
    *    function returns a 0, if it does not exist
    *    this function returns a 1.
    *
    **********************************************/

does_not_exist(file_name)
    char file_name[];
{
   FILE *image_file;
   int  result;

   result = 1;
   image_file = fopen(file_name, "rb");
   if(image_file != NULL){
      result = 0;
      fclose(image_file);
   }
   return(result);
}  /* ends does_not_exist */




   /**********************************************
   *
   *   are_not_same_size(...
   *
   *   This function checks the rows and cols
   *   of two images whose names are passed.
```

```
   *   It returns a 1 if the images are not
   *   the same size.
   *
   ********************************************/

int are_not_same_size(file1, file2)
   char *file1, *file2;
{
   int  result = 0;
   long cols1  = 1,
        cols2  = 2,
        rows1  = 3,
        rows2  = 4;

   get_image_size(file1, &rows1, &cols1);
   get_image_size(file2, &rows2, &cols2);

   if(rows1 != rows2  ||  cols1 != cols2)
      result = 1;

   return(result);

}  /* ends are_not_same_size */




   /*********************************************
   *
   *   equate_bitmapheaders(...
   *
   *   This function sets the elements of the
   *   destination header to the values of the
   *   source header.
   *
   ********************************************/

equate_bitmapheaders(src, dest)
   struct bitmapheader *src, *dest;
{
   dest->size         = src->size;
   dest->width        = src->width;
   dest->height       = src->width;
   dest->planes       = src->planes;
   dest->bitsperpixel = src->bitsperpixel;
```

```
   dest->compression  = src->compression;
   dest->sizeofbitmap = src->sizeofbitmap;
   dest->horzres      = src->horzres;
   dest->vertres      = src->vertres;
   dest->colorsused   = src->colorsused;
   dest->colorsimp    = src->colorsimp;
}  /* ends equate_bitmapheader */




      /****************************************
      *
      *   get_bitsperpixel(...
      *
      *   This function reads the bits per pixel
      *   from either a tiff or bmp image file.
      *
      ****************************************/

int get_bitsperpixel(file_name, bitsperpixel)
   char *file_name;
   long *bitsperpixel;
{
   int    is_bmp  = 0,
          is_tiff = 0,
          result  = 0;
   long   temp;
   struct bitmapheader bmph;
   struct tiff_header_struct tiffh;

   if(is_a_bmp(file_name)){
      is_bmp = 1;
      read_bm_header(file_name, &bmph);
      temp = (long)bmph.bitsperpixel;
      *bitsperpixel = temp;
   }  /* ends if is_a_bmp */

   if(is_a_tiff(file_name)){
      is_tiff = 1;
      read_tiff_header(file_name, &tiffh);
      *bitsperpixel = tiffh.bits_per_pixel;
   }  /* ends if is_a_bmp */
```

```
   if(is_bmp == 1  ||  is_tiff == 1)
      result = 1;

   return(result);
}  /* ends get_image_size */


      /*****************************************
      *
      *   get_lsb(...
      *
      *   This function reads the lsb flag
      *   from a tiff image file.
      *
      *****************************************/

int get_lsb(name)
   char *name;
{
   int   result = 0;
   struct tiff_header_struct tiff_header;

   if(is_a_bmp(name))
      result = 1;

   if(is_a_tiff(name)){
      read_tiff_header(name, &tiff_header);
      if(tiff_header.lsb == 1)
         result = 1;
   }  /* ends if is a tiff */

   return(result);

}  /*  ends get_lsb */
```

Listing 1.1 - The High-Level I/O Routines

```
       /***********************************************
       *
       *   read_tiff_header(...
       *
       *   This function reads the header of a TIFF
       *   file and places the needed information into
       *   the struct tiff_header_struct.
       *
       ***********************************************/

read_tiff_header(file_name, image_header)
   char file_name[];
   struct tiff_header_struct *image_header;
{
   char buffer[12], response[80];

   FILE *image_file;

   int  bytes_read,
        closed,
        i,
        j,
        lsb,
        not_finished,
        position;

   long bits_per_pixel,
        image_length,
        image_width,
        length_of_field,
        offset_to_ifd,
        strip_offset,
        subfile,
        value;

   short entry_count,
         field_type,
         s_bits_per_pixel,
         s_image_length,
         s_image_width,
         s_strip_offset,
         tag_type;
```

```
image_file = fopen(file_name, "rb");
if(image_file != NULL){

     /*************************************
     *
     *   Determine if the file uses MSB
     *   first or LSB first
     *
     *************************************/

bytes_read = fread(buffer, 1, 8, image_file);

if(buffer[0] == 0x49)
   lsb = 1;
else
   lsb = 0;

     /*************************************
     *
     *   Read the offset to the IFD
     *
     *************************************/

extract_long_from_buffer(buffer, lsb, 4,
                         &offset_to_ifd);

not_finished = 1;
while(not_finished){

     /*************************************
     *
     *   Seek to the IFD and read the
     *   entry_count, i.e. the number of
     *   entries in the IFD.
     *
     *************************************/

   position   = fseek(image_file, offset_to_ifd,
                      SEEK_SET);
   bytes_read = fread(buffer, 1, 2, image_file);
   extract_short_from_buffer(buffer, lsb, 0,
                             &entry_count);

     /***************************************
     *
```

```
  *   Now loop over the directory entries.
  *   Look only for the tags we need.  These
  *   are:
  *     ImageLength
  *     ImageWidth
  *     BitsPerPixel(BitsPerSample)
  *     StripOffset
  *
  ****************************************/

 for(i=0; i<entry_count; i++){
  bytes_read = fread(buffer, 1, 12, image_file);
  extract_short_from_buffer(buffer, lsb, 0,
                            &tag_type);

  switch(tag_type){

     case 255: /* Subfile Type */
        extract_short_from_buffer(buffer, lsb, 2,
                                  &field_type);
        extract_short_from_buffer(buffer, lsb, 4,
                                &length_of_field);
        extract_long_from_buffer(buffer, lsb, 8,
                                 &subfile);
        break;

     case 256: /* ImageWidth */
        extract_short_from_buffer(buffer, lsb, 2,
                                  &field_type);
        extract_short_from_buffer(buffer, lsb, 4,
                                &length_of_field);
        if(field_type == 3){
         extract_short_from_buffer(buffer, lsb, 8,
                                 &s_image_width);
         image_width = s_image_width;
        }
        else
         extract_long_from_buffer(buffer, lsb, 8,
                                  &image_width);
        break;

     case 257: /* ImageLength */
        extract_short_from_buffer(buffer, lsb, 2,
                                  &field_type);
        extract_short_from_buffer(buffer, lsb, 4,
```

```
                                &length_of_field);
         if(field_type == 3){
          extract_short_from_buffer(buffer, lsb, 8,
                                &s_image_length);
          image_length = s_image_length;
         }
         else
          extract_long_from_buffer(buffer, lsb, 8,
                                  &image_length);
         break;

      case 258: /* BitsPerSample */
         extract_short_from_buffer(buffer, lsb, 2,
                                  &field_type);
         extract_short_from_buffer(buffer, lsb, 4,
                                &length_of_field);
         if(field_type == 3){
          extract_short_from_buffer(buffer, lsb, 8,
                              &s_bits_per_pixel);
          bits_per_pixel = s_bits_per_pixel;
         }
         else
          extract_long_from_buffer(buffer, lsb, 8,
                                &bits_per_pixel);
         break;

      case 273: /* StripOffset */
         extract_short_from_buffer(buffer, lsb, 2,
                                  &field_type);
         extract_short_from_buffer(buffer, lsb, 4,
                                &length_of_field);
         if(field_type == 3){
          extract_short_from_buffer(buffer, lsb, 8,
                                &s_strip_offset);
          strip_offset = s_strip_offset;
         }
         else
          extract_long_from_buffer(buffer, lsb, 8,
                                  &strip_offset);
         break;

      default:
         break;

   }  /* ends switch tag_type */
```

```
      }  /* ends loop over i directory entries */

      bytes_read = fread(buffer, 1, 4, image_file);
      extract_long_from_buffer(buffer, lsb, 0,
                               &offset_to_ifd);
      if(offset_to_ifd == 0) not_finished = 0;

   }  /* ends while not_finished */


   image_header->lsb                = lsb;
   image_header->bits_per_pixel = bits_per_pixel;
   image_header->image_length       = image_length;
   image_header->image_width        = image_width;
   image_header->strip_offset       = strip_offset;

   closed = fclose(image_file);
   }  /* ends if file opened ok */
   else{
      printf("\n\nTIFF.C> ERROR - could not open "
             "tiff file");
   }
}  /* ends read_tiff_header */




   /****************************************
   *
   *   read_tiff_image(...
   *
   *   This function reads the image data
   *   from a tiff image file.
   *
   *   It only works for 8-bit gray scale
   *   images.
   *
   ****************************************/

read_tiff_image(image_file_name, the_image)
      char   image_file_name[];
      short   **the_image;
{
```

```
char  *buffer,  /* CHANGED */
      rep[80];
int   bytes_read,
      closed,
      position,
      i,
      j;
FILE  *image_file;
float a;
long  line_length, offset;

struct tiff_header_struct image_header;

read_tiff_header(image_file_name, &image_header);

   /***********************************************
   *
   *   Procedure:
   *   Seek to the strip offset where the data begins.
   *   Seek to the first line you want.
   *   Loop over the lines you want to read:
   *      Seek to the first element of the line.
   *      Read the line.
   *      Seek to the end of the data in that line.
   *
   ***********************************************/

image_file = fopen(image_file_name, "rb");
if(image_file != NULL){
   position = fseek(image_file,
                    image_header.strip_offset,
                    SEEK_SET);

   for(i=0; i<image_header.image_length; i++){

      bytes_read   = read_line(image_file, the_image,
                               i, &image_header,
                               0, image_header.image_width);
   }  /* ends loop over i  */

   closed = fclose(image_file);
}  /* ends if file opened ok */
else{
   printf("\nRTIFF.C> ERROR - cannot open "
          "tiff file");
```

```
   }

}  /*  ends read_tiff_image */




   /*********************************************
   *
   *   read_line(...
   *
   *   This function reads bytes from the TIFF
   *   file into a buffer, extracts the numbers
   *   from that buffer, and puts them into a
   *   ROWSxCOLS array of shorts. The process
   *   depends on the number of bits per pixel used
   *   in the file (4 or 8).
   *
   *********************************************/

read_line(image_file, the_image, line_number,
          image_header, ie, le)
   FILE   *image_file;
   int    ie, le, line_number;
   short  **the_image;
   struct tiff_header_struct *image_header;
{
   char  *buffer, first, second;
   float a, b;
   int bytes_read, i;
   unsigned int bytes_to_read;
   union short_char_union scu;


   buffer = (char  *) malloc(image_header->image_width * sizeof(char ));
   for(i=0; i<image_header->image_width; i++)
      buffer[i] = '\0';

        /*********************************************
        *
        *   Use the number of bits per pixel to
        *   calculate how many bytes to read.
        *
        *********************************************/
```

```
   bytes_to_read = (le-ie)/
                   (8/image_header->bits_per_pixel);
   bytes_read    = fread(buffer, 1, bytes_to_read,
                         image_file);

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

        /*********************************************
        *
        *   Use unions defined in cips.h to stuff bytes
        *   into shorts.
        *
        **********************************************/

      if(image_header->bits_per_pixel == 8){
       scu.s_num          = 0;
       scu.s_alpha[0]        = buffer[i];
       the_image[line_number][i] = scu.s_num;
      }  /* ends if bits_per_pixel == 8 */

      if(image_header->bits_per_pixel == 4){

       scu.s_num             = 0;
       second                = buffer[i] & 0X000F;
       scu.s_alpha[0]        = second;
       the_image[line_number][i*2+1] = scu.s_num;

       scu.s_num             = 0;
       first                 = buffer[i] >> 4;
       first                 = first & 0x000F;
       scu.s_alpha[0]        = first;
       the_image[line_number][i*2] = scu.s_num;

      }  /* ends if bits_per_pixel == 4 */

   }  /*  ends loop over i  */

   free(buffer);
   return(bytes_read);

}  /* ends read_line  */
```

```
   /***********************************************
   *
   *   create_tiff_file_if_needed(...
   *
   *   This function creates a tiff file on disk
   *   if it does not exist.  The out file is
   *   patterned after the in file.
   *
   ***********************************************/

create_tiff_file_if_needed(in_name, out_name, out_image)
   char in_name[], out_name[];
   short **out_image;
{
   int    length, width;
   struct tiff_header_struct image_header;

   if(does_not_exist(out_name)){
      printf("\n\n output file does not exist %s",
               out_name);
      read_tiff_header(in_name, &image_header);
      create_allocate_tiff_file(out_name, &image_header);
      printf("\nBFIN> Created %s", out_name);
   }  /* ends if does_not_exist */
}  /* ends create_tiff_file_if_needed */




   /***********************************************
   *
   *   create_alllocate_tiff_file(...
   *
   *   This function creates a file on disk that will be
   *   large enough to hold a tiff image.  The input
   *   tiff_header_struct describes the desired tiff file.
   *   This function writes the tiff header and then
   *   writes a blank image array out to disk the proper
   *   number of times.  This has the effect of allocating
   *   the correct number of bytes on the disk.
   *
   *   There will be 18 entries in the IFD.
```

```
   *   The image data will begin at byte 296.
   *   I will use LSB first data.
   *   I will have one strip for the entire image.
   *   Black is zero.
   *   The component values for the image are CHUNKY
   *      (Planer configuration = 1).
   *
   ****************************************************/


create_allocate_tiff_file(file_name,
                          image_header)
   char   file_name[];
   struct tiff_header_struct *image_header;
{
   char  buffer[12], *image_buffer, long_buffer[50];
   FILE  *image_file;
   int   bytes_written,
         i,
         j,
         l,
         w;

   long  k;



      /****************************************
      *
      *   Create the image file in binary mode
      *   for both reading and writing.
      *
      ****************************************/

   image_file = fopen(file_name, "wb");

      /****************************************
      *
      *   Write out the first 8 bytes of the
      *   header.  The meaning of the
      *   bytes (HEX) is:
      *      0-1 = 49 49 - LSB first
      *      2-3 = 2A 00 - version #
      *      4-7 = 08 00 00 00 - go to offset
      *           8 for the first
```

```
   *           Image File
   *           Directory
   *
   ****************************************/

buffer[0] = 0x49;
buffer[1] = 0x49;
buffer[2] = 0x2A;
buffer[3] = 0x00;
buffer[4] = 0x08;
buffer[5] = 0x00;
buffer[6] = 0x00;
buffer[7] = 0x00;

bytes_written = fwrite(buffer, 1, 8, image_file);

printf("\n wrote %d bytes", bytes_written);

   /****************************************
   *
   *   Write out the first 2 bytes of the
   *   Image File Directory.  These tell
   *   the number of entries in the IFD.
   *
   ****************************************/

buffer[0] = 0x12;
buffer[1] = 0x00;
bytes_written = fwrite(buffer, 1, 2, image_file);

printf("\n wrote %d bytes", bytes_written);

   /****************************************
   *
   *   Write out the entries into the
   *   Image File Directory.
   *
   ****************************************/


   /* New Subfile Type */
buffer[0]  = 0xFE;
buffer[1]  = 0x00;
buffer[2]  = 0x03;
buffer[3]  = 0x00;
```

```
buffer[4]  = 0x01;
buffer[5]  = 0x00;
buffer[6]  = 0x00;
buffer[7]  = 0x00;
buffer[8]  = 0x00;
buffer[9]  = 0x00;
buffer[10] = 0x00;
buffer[11] = 0x00;
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Subfile Type */
buffer[0]  = 0xFF;
buffer[1]  = 0x00;
buffer[2]  = 0x03;
buffer[3]  = 0x00;
buffer[4]  = 0x01;
buffer[5]  = 0x00;
buffer[6]  = 0x00;
buffer[7]  = 0x00;
buffer[8]  = 0x01;
buffer[9]  = 0x00;
buffer[10] = 0x00;
buffer[11] = 0x00;
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Image Width */
insert_short_into_buffer(buffer, 0, 256);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8,
                  image_header->image_width);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Image Length */
insert_short_into_buffer(buffer, 0, 257);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8,
                  image_header->image_length);
```

```
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Bits Per Sample */
insert_short_into_buffer(buffer, 0, 258);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8,
                  image_header->bits_per_pixel);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Compression - None */
insert_short_into_buffer(buffer, 0, 259);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 1);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Photometric Interpretation */
   /* set to 1 because BLACK is ZERO */
insert_short_into_buffer(buffer, 0, 262);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 1);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Strip Offset */
   /* start after software name at 296 */
insert_short_into_buffer(buffer, 0, 273);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 296);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Samples per Pixel */
insert_short_into_buffer(buffer, 0, 277);
```

```
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 1);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* clear buffer */
for(i=0; i<12; i++) buffer[i] = 0x00;

   /* Rows Per Strip 1 strip for the entire image */
   /* use 2E32 - 1, which is max */
insert_short_into_buffer(buffer, 0, 278);
insert_short_into_buffer(buffer, 2, 4);
insert_short_into_buffer(buffer, 4, 1);
insert_long_into_buffer(buffer, 8, 4294967295);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Strip Byte Counts */
   /* this = image width times length */
insert_short_into_buffer(buffer, 0, 279);
insert_short_into_buffer(buffer, 2, 4);
insert_short_into_buffer(buffer, 4, 1);
insert_long_into_buffer(buffer, 8,
 (long)(image_header->image_length *
        image_header->image_width));
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Min Sample Value */
insert_short_into_buffer(buffer, 0, 280);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 0);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Max Sample Value */
insert_short_into_buffer(buffer, 0, 281);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
```

```
if(image_header->bits_per_pixel == 8)
   insert_short_into_buffer(buffer, 8, 255);
else
   insert_short_into_buffer(buffer, 8, 15);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* X Resolution */
   /* Store the 8 bytes for this value
      starting at 230 */
insert_short_into_buffer(buffer, 0, 282);
insert_short_into_buffer(buffer, 2, 5);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 230);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Y Resolution */
   /* Store the 8 bytes for this value
      starting at 238 */
insert_short_into_buffer(buffer, 0, 283);
insert_short_into_buffer(buffer, 2, 5);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 238);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);



   /* clear buffer */
for(i=0; i<12; i++) buffer[i] = 0x00;

   /* Planer Configuration */
   /* chunky */
insert_short_into_buffer(buffer, 0, 284);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 1);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Resolution Unit */
```

```
   /* inches */
insert_short_into_buffer(buffer, 0, 296);
insert_short_into_buffer(buffer, 2, 3);
insert_short_into_buffer(buffer, 4, 1);
insert_short_into_buffer(buffer, 8, 2);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Software */
   /* Put this a 246, 50 bytes */
insert_short_into_buffer(buffer, 0, 305);
insert_short_into_buffer(buffer, 2, 2);
insert_short_into_buffer(buffer, 4, 50);
insert_short_into_buffer(buffer, 8, 246);
bytes_written = fwrite(buffer, 1, 12, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Offset to next IFD (0 means no more IFD's) */
for(i=0; i<12; i++) buffer[i] = 0x00;
bytes_written = fwrite(buffer, 1, 4, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* clear buffer */
for(i=0; i<12; i++) buffer[i] = 0x00;

   /* Now store the X Resolution
      first long is numerator
      second long is denominator */
insert_long_into_buffer(buffer, 0, 300L);
insert_long_into_buffer(buffer, 4, 1L);
bytes_written = fwrite(buffer, 1, 8, image_file);
printf("\n wrote %d bytes", bytes_written);


   /* Now store the Y Resolution
      first long is numerator
      second long is denominator */
insert_long_into_buffer(buffer, 0, 300L);
insert_long_into_buffer(buffer, 4, 1L);
bytes_written = fwrite(buffer, 1, 8, image_file);
printf("\n wrote %d bytes", bytes_written);
```

```
   /* Now store the software tag */
for(i=0; i<50; i++) long_buffer[i] = '\0';
strcpy(long_buffer,
"Dwayne Phillips C Image Processing System 1993");
long_buffer[46] = '\0';
long_buffer[47] = '\0';
long_buffer[48] = '\0';
long_buffer[49] = '\0';
bytes_written = fwrite(long_buffer, 1, 50,
                       image_file);
printf("\n wrote %d bytes", bytes_written);
printf("\n%s", long_buffer);



   /****************************************
   *
   *   Now write the image data.
   *
   ****************************************/

printf("\n length is %ld",
       image_header->image_length);
printf("\n width is %ld",
       image_header->image_width);

k = image_header->image_width;

if(image_header->bits_per_pixel == 4)
   k = k/2;

image_buffer = (char  *) malloc(k * sizeof(char ));

for(i=0; i<k; i++)
   image_buffer[i] = 0x00;

for(i=0; i<image_header->image_length; i++){
   bytes_written = fwrite(image_buffer, 1, k, image_file);
   /***printf("\n wrote %d bytes", bytes_written);***/
}

fclose(image_file);
free(image_buffer);
```

```
}  /* ends create_allocate_tiff_file */




       /**********************************************
       *
       *   write_tiff_image(...
       *
       *   This function takes an array of shorts and
       *   writes them into an existing tiff image file.
       *
       **********************************************/

write_tiff_image(image_file_name, array)

        char    image_file_name[];
        short   **array;
{

   FILE  *image_file;
   int   bytes_written,
         closed,
         i,
          position,
         written;

   float a;

   long  line_length,
         offset;

   struct tiff_header_struct image_header;

   read_tiff_header(image_file_name, &image_header);

      /**********************************************
      *
      *   Procedure:
      *   Seek to the strip offset where the data begins.
      *   Seek to the first line you want.
      *   Loop over the lines you want to write.
      *      Seek to the first element of the line.
      *      Write the line.
```

```
      *      Seek to the end of the data in that line.
      *
      ************************************************/

   image_file = fopen(image_file_name, "rb+");
   position   = fseek(image_file,
                      image_header.strip_offset,
                      SEEK_SET);

   for(i=0; i<image_header.image_length; i++){
      bytes_written = write_line(image_file, array,
                                 i, &image_header,
                                 0, image_header.image_width);
   }  /* ends loop over i  */

   closed = fclose(image_file);

}  /*  ends write_tiff_image */




       /*********************************************
       *
       *   write_line(...
       *
       *   This function takes an array of shorts,
       *   extracts the numbers and puts them into
       *   a buffer, then writes this buffer into a
       *   tiff file on disk. The process depends on
       *   the number of bits per pixel used in the
       *   file (4 or 8).
       *
       *********************************************/

write_line(image_file, array, line_number,
           image_header, ie, le)
   FILE   *image_file;
   int    ie, le, line_number;
   short  **array;
   struct tiff_header_struct *image_header;
{
   char     *buffer, first, second;
   float    a, b;
```

```
int      bytes_written, i;
unsigned int bytes_to_write;
union    short_char_union scu;

buffer = (char  *) malloc(image_header->image_width * sizeof(char ));
for(i=0; i<image_header->image_width; i++)
   buffer[i] = '\0';

bytes_to_write = (le-ie)/
                 (8/image_header->bits_per_pixel);

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

     /***********************************************
     *
     *   Use unions defined in cips.h to stuff shorts
     *   into bytess.
     *
     ***********************************************/

   if(image_header->bits_per_pixel == 8){
    scu.s_num = 0;
    scu.s_num = array[line_number][i];
    buffer[i] = scu.s_alpha[0];
   }  /* ends if bits_per_pixel == 8 */


   if(image_header->bits_per_pixel == 4){

    scu.s_num = 0;
    scu.s_num = array[line_number][i*2];
    first     = scu.s_alpha[0] << 4;

    scu.s_num = 0;
    scu.s_num = array[line_number][i*2];
    second    = scu.s_alpha[0] & 0X000F;

    buffer[i] = first | second;
   }  /* ends if bits_per_pixel == 4 */

}  /*  ends loop over i  */


bytes_written = fwrite(buffer, 1, bytes_to_write,
                       image_file);
```

```
   free(buffer);
   return(bytes_written);

}  /* ends write_line  */




   /***********************************************
   *
   *   is_a_tiff(...
   *
   *   This function looks at a file to see if it
   *   is a tiff file.  First look at the file
   *   extension.  Next look at the first four
   *   bytes of the header.
   *
   ***********************************************/


int is_a_tiff(file_name)
   char *file_name;
{
   char   *cc;
   char   buffer[4];
   FILE   *fp;
   int    ok     = 0,
          result = 0;

   cc = strstr(file_name, ".tif");
   if(cc == NULL)
      return(result);

   fp = fopen(file_name, "rb");
   fread(buffer, 1, 4, fp);
   fclose(fp);

   if(buffer[0] == 0x49  &&
      buffer[1] == 0x49  &&
      buffer[2] == 0x2a  &&
      buffer[3] == 0x00)
      ok = 1;
```

```
   if(buffer[0] == 0x4d  &&
      buffer[1] == 0x4d  &&
      buffer[2] == 0x00  &&
      buffer[3] == 0x2a)
      ok = 1;

   if(ok == 0)
      return(result);

   result = 1;
   return(result);
}  /* ends is_a_tiff */




   /**********************************************
   *
   *   equate_tiff_headers(...
   *
   *   This function sets the elements of the
   *   destination header to the values of the
   *   source header.
   *
   **********************************************/

equate_tiff_headers(src, dest)
   struct tiff_header_struct *src, *dest;
{
   dest->lsb            = src->lsb;
   dest->bits_per_pixel = src->bits_per_pixel;
   dest->image_length   = src->image_length;
   dest->image_width    = src->image_width;
   dest->strip_offset   = src->strip_offset;
}  /* ends equate_tiff_headers */




   /****************************************
   *
   *   extract_long_from_buffer(...
   *
```

```
   *   This takes a four byte long out of a
   *   buffer of characters.
   *
   *   It is important to know the byte order
   *   LSB or MSB.
   *
   ****************************************/

extract_long_from_buffer(buffer, lsb, start, number)
   char  buffer[];
   int       lsb, start;
   long  *number;
{
   int i;
   union long_char_union lcu;

   if(lsb == 1){
      lcu.l_alpha[0] = buffer[start+0];
      lcu.l_alpha[1] = buffer[start+1];
      lcu.l_alpha[2] = buffer[start+2];
      lcu.l_alpha[3] = buffer[start+3];
   }  /* ends if lsb = 1 */

   if(lsb == 0){
      lcu.l_alpha[0] = buffer[start+3];
      lcu.l_alpha[1] = buffer[start+2];
      lcu.l_alpha[2] = buffer[start+1];
      lcu.l_alpha[3] = buffer[start+0];
   }  /* ends if lsb = 0      */

   *number = lcu.l_num;


}  /* ends extract_long_from_buffer */




   /****************************************
   *
   *   extract_ulong_from_buffer(...
   *
   *   This takes a four byte unsigned long
   *   out of a buffer of characters.
```

```
   *
   *   It is important to know the byte order
   *   LSB or MSB.
   *
   ****************************************/

extract_ulong_from_buffer(buffer, lsb, start, number)
   char  buffer[];
   int       lsb, start;
   unsigned long  *number;
{
   int i;
   union ulong_char_union lcu;

   if(lsb == 1){
      lcu.l_alpha[0] = buffer[start+0];
      lcu.l_alpha[1] = buffer[start+1];
      lcu.l_alpha[2] = buffer[start+2];
      lcu.l_alpha[3] = buffer[start+3];
   }  /* ends if lsb = 1 */

   if(lsb == 0){
      lcu.l_alpha[0] = buffer[start+3];
      lcu.l_alpha[1] = buffer[start+2];
      lcu.l_alpha[2] = buffer[start+1];
      lcu.l_alpha[3] = buffer[start+0];
   }  /* ends if lsb = 0      */

   *number = lcu.l_num;
}  /* ends extract_ulong_from_buffer */




   /****************************************
   *
   *   extract_short_from_buffer(...
   *
   *   This takes a two byte short out of a
   *   buffer of characters.
   *
   *   It is important to know the byte order
   *   LSB or MSB.
   *
```

```
   ****************************************/

extract_short_from_buffer(buffer, lsb, start, number)
   char  buffer[];
   int       lsb, start;
   short *number;
{

   int i;
   union short_char_union lcu;

   if(lsb == 1){
      lcu.s_alpha[0] = buffer[start+0];
      lcu.s_alpha[1] = buffer[start+1];
   }  /* ends if lsb = 1 */

   if(lsb == 0){
      lcu.s_alpha[0] = buffer[start+1];
      lcu.s_alpha[1] = buffer[start+0];
   }  /* ends if lsb = 0      */

   *number = lcu.s_num;


}  /* ends extract_short_from_buffer */




   /****************************************
   *
   *   extract_ushort_from_buffer(...
   *
   *   This takes a two byte unsiged short
   *   out of a buffer of characters.
   *
   *   It is important to know the byte order
   *   LSB or MSB.
   *
   ****************************************/

extract_ushort_from_buffer(buffer, lsb, start, number)
   char  buffer[];
   int       lsb, start;
```

```
   unsigned short *number;
{

   int i;
   union ushort_char_union lcu;

   if(lsb == 1){
      lcu.s_alpha[0] = buffer[start+0];
      lcu.s_alpha[1] = buffer[start+1];
   }  /* ends if lsb = 1 */

   if(lsb == 0){
      lcu.s_alpha[0] = buffer[start+1];
      lcu.s_alpha[1] = buffer[start+0];
   }  /* ends if lsb = 0      */

   *number = lcu.s_num;
}  /* ends extract_ushort_from_buffer */




   /****************************************
   *
   *   insert_short_into_buffer(...
   *
   *   This inserts a two byte short into a
   *   buffer of characters.  It does this
   *   is LSB order.
   *
   ****************************************/


insert_short_into_buffer(buffer, start, number)
    char  buffer[];
    int   start;
    short number;
{
    union short_char_union lsu;

    lsu.s_num       = number;
    buffer[start+0] = lsu.s_alpha[0];
    buffer[start+1] = lsu.s_alpha[1];
```

```
}  /* ends insert_short_into_buffer */




   /****************************************
   *
   *   insert_ushort_into_buffer(...
   *
   *   This inserts a two byte unsigned
   *   short into a buffer of characters.
   *   It does this is LSB order.
   *
   ****************************************/

insert_ushort_into_buffer(buffer, start, number)
    char  buffer[];
    int   start;
    unsigned short number;
{
    union ushort_char_union lsu;

    lsu.s_num       = number;
    buffer[start+0] = lsu.s_alpha[0];
    buffer[start+1] = lsu.s_alpha[1];

}  /* ends insert_short_into_buffer */




   /****************************************
   *
   *   insert_long_into_buffer(...
   *
   *   This inserts a four byte long into a
   *   buffer of characters.  It does this
   *   is LSB order.
   *
   ****************************************/
```

```
insert_long_into_buffer(buffer, start, number)
    char buffer[];
    int  start;
    long number;
{
    union long_char_union lsu;

    lsu.l_num       = number;
    buffer[start+0] = lsu.l_alpha[0];
    buffer[start+1] = lsu.l_alpha[1];
    buffer[start+2] = lsu.l_alpha[2];
    buffer[start+3] = lsu.l_alpha[3];

}  /* ends insert_short_into_buffer */




   /****************************************
   *
   *   insert_ulong_into_buffer(...
   *
   *   This inserts a four byte unsigned
   *   long into a buffer of characters.
   *   It does this is LSB order.
   *
   ****************************************/

insert_ulong_into_buffer(buffer, start, number)
    char buffer[];
    int  start;
    unsigned long number;
{
    union ulong_char_union lsu;

    lsu.l_num       = number;
    buffer[start+0] = lsu.l_alpha[0];
    buffer[start+1] = lsu.l_alpha[1];
    buffer[start+2] = lsu.l_alpha[2];
    buffer[start+3] = lsu.l_alpha[3];

}  /* ends insert_ulong_into_buffer */
```

Listing 1.2 - The TIFF I/O Routines

```
      /*****************************************
      *
      *   read_bmp_file_header(...
      *
      *   This function reads the bmpfileheader
      *   structure from the top of a bmp
      *   image file.
      *
      *****************************************/

read_bmp_file_header(file_name,
                     file_header)
   char *file_name;
   struct bmpfileheader *file_header;
{
   char  buffer[10];
   long  ll;
   short ss;
   unsigned long  ull;
   unsigned short uss;
   FILE     *fp;

   fp = fopen(file_name, "rb");

   fread(buffer, 1, 2, fp);
   extract_ushort_from_buffer(buffer, 1, 0, &uss);
   file_header->filetype = uss;

   fread(buffer, 1, 4, fp);
   extract_ulong_from_buffer(buffer, 1, 0, &ull);
   file_header->filesize = ull;

   fread(buffer, 1, 2, fp);
   extract_short_from_buffer(buffer, 1, 0, &ss);
   file_header->reserved1 = ss;

   fread(buffer, 1, 2, fp);
   extract_short_from_buffer(buffer, 1, 0, &ss);
```

```
   file_header->reserved2 = ss;

   fread(buffer, 1, 4, fp);
   extract_ulong_from_buffer(buffer, 1, 0, &ull);
   file_header->bitmapoffset = ull;

   fclose(fp);

}  /* ends read_bmp_file_header */




      /*****************************************
      *
      *   read_bm_header(...
      *
      *   This function reads the bitmapheader
      *   structure from the top of a bmp
      *   image file.
      *
      *****************************************/

read_bm_header(file_name,
               bmheader)
   char *file_name;
   struct bitmapheader *bmheader;
{
   char  buffer[10];
   long  ll;
   short ss;
   unsigned long  ull;
   unsigned short uss;
   FILE *fp;

   fp = fopen(file_name, "rb");

      /****************************************
      *
      *   Seek past the first 14 byte header.
      *
      ****************************************/

   fseek(fp, 14, SEEK_SET);
```

```
fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->size = ull;

fread(buffer, 1, 4, fp);
extract_long_from_buffer(buffer, 1, 0, &ll);
bmheader->width = ll;

fread(buffer, 1, 4, fp);
extract_long_from_buffer(buffer, 1, 0, &ll);
bmheader->height = ll;

fread(buffer, 1, 2, fp);
extract_ushort_from_buffer(buffer, 1, 0, &uss);
bmheader->planes = uss;

fread(buffer, 1, 2, fp);
extract_ushort_from_buffer(buffer, 1, 0, &uss);
bmheader->bitsperpixel = uss;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->compression = ull;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->sizeofbitmap = ull;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->horzres = ull;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->vertres = ull;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->colorsused = ull;

fread(buffer, 1, 4, fp);
extract_ulong_from_buffer(buffer, 1, 0, &ull);
bmheader->colorsimp = ull;
```

```
   fclose(fp);

}  /* ends read_bm_header */




      /****************************************
      *
      *   read_color_table(...
      *
      *   This function reads the color table
      *   from a bmp image file.
      *
      ****************************************/

read_color_table(file_name, rgb, size)
   char   *file_name;
   struct ctstruct *rgb;
   int    size;
{
   int  i;
   char buffer[10];
   FILE *fp;

   fp = fopen(file_name, "rb");

   fseek(fp, 54, SEEK_SET);

   for(i=0; i<size; i++){
      fread(buffer, 1, 1, fp);
      rgb[i].blue = buffer[0];
      fread(buffer, 1, 1, fp);
      rgb[i].green = buffer[0];
      fread(buffer, 1, 1, fp);
      rgb[i].red = buffer[0];
      fread(buffer, 1, 1, fp);
         /* fourth byte nothing */
   }  /* ends loop over i */

   fclose(fp);

}  /* ends read_color_table */
```

```
      /****************************************
      *
      *   read_bmp_image(...
      *
      *   This function reads the image array
      *   from a bmp file.
      *
      *   It only works for 8-bit images.
      *
      ****************************************/

read_bmp_image(file_name, array)
   char  *file_name;
   short **array;
{
   FILE   *fp;
   int    i, j;
   int    negative = 0,
          pad      = 0,
          place    = 0;
   long   colors   = 0,
          height   = 0,
          position = 0,
          width    = 0;
   struct bmpfileheader file_header;
   struct bitmapheader  bmheader;
   struct ctstruct rgb[GRAY_LEVELS+1];
   unsigned char uc;

   read_bmp_file_header(file_name, &file_header);
   read_bm_header(file_name, &bmheader);
   if(bmheader.bitsperpixel != 8){
      printf("\nCannot read image when bits per"
      "pixel is not 8");
      exit(1);
   }

   if(bmheader.colorsused == 0)
      colors = GRAY_LEVELS + 1;
   else
      colors = bmheader.colorsused;
```

```
   read_color_table(file_name, &rgb, colors);

   fp = fopen(file_name, "rb");
   fseek(fp, file_header.bitmapoffset, SEEK_SET);

   width = bmheader.width;
   if(bmheader.height < 0){
      height   = bmheader.height * (-1);
      negative = 1;
   }
   else
      height = bmheader.height;

   pad = calculate_pad(width);

   for(i=0; i<height; i++){
      for(j=0; j<width; j++){
         place = fgetc(fp);
         uc = (place & 0xff);
         place = uc;
         array[i][j] = rgb[place].blue;
      }  /* ends loop over j */
      if(pad != 0){
         position = fseek(fp, pad, SEEK_CUR);
      }  /* ends if pad 1= 0 */
   }  /* ends loop over i */

   if(negative == 0)
      flip_image_array(array, height, width);

}  /* ends read_bmp_image */


      /*****************************************
      *
      *   create_bmp_file_if_needed(...
      *
      *   This function allocates a bmp image
      *   file it it does not exist.  It uses
      *   the header information from the input
      *   image name.
      *
```

```
      ****************************************/

create_bmp_file_if_needed(in_name, out_name, out_image)
   char in_name[], out_name[];
   short **out_image;
{
   int    length, width;
   struct bmpfileheader file_header;
   struct bitmapheader  bmheader;

   if(does_not_exist(out_name)){
      printf("\n\n output file does not exist %s",
               out_name);
      read_bm_header(in_name, &bmheader);
      create_allocate_bmp_file(out_name, &file_header, &bmheader);
      printf("\nBFIN> Created %s", out_name);
   }  /* ends if does_not_exist */
}  /* ends bmp_file_if_needed */




   /*********************************************
   *
   *   create_allocate_bmp_file(...
   *
   *   The calling routine must set the
   *   height and width.  This routine will set
   *   everything else.
   *
   **********************************************/

create_allocate_bmp_file(file_name,
                         file_header,
                         bmheader)
   char  *file_name;
   struct bmpfileheader *file_header;
   struct bitmapheader  *bmheader;
{
   char buffer[100];
   int  i, pad = 0;
   FILE *fp;

   pad = calculate_pad(bmheader->width);
```

```
bmheader->size         =  40;
bmheader->planes       =   1;
bmheader->bitsperpixel =   8;
bmheader->compression  =   0;
bmheader->sizeofbitmap = bmheader->height *
                         (bmheader->width + pad);
bmheader->horzres      = 300;
bmheader->vertres      = 300;
bmheader->colorsused   = 256;
bmheader->colorsimp    = 256;

file_header->filetype     = 0x4D42;
file_header->reserved1    =  0;
file_header->reserved2    =  0;
file_header->bitmapoffset = 14 +
                            bmheader->size +
                            bmheader->colorsused*4;
file_header->filesize     = file_header->bitmapoffset +
                            bmheader->sizeofbitmap;

if((fp = fopen(file_name, "wb")) == NULL){
   printf("\nERROR Could not create file %s",
          file_name);
   exit(2);
}

   /*********************************************
   *
   *   Write the 14-byte bmp file header.
   *
   *********************************************/

insert_ushort_into_buffer(buffer, 0, file_header->filetype);
fwrite(buffer, 1, 2, fp);

insert_ulong_into_buffer(buffer, 0, file_header->filesize);
fwrite(buffer, 1, 4, fp);

insert_short_into_buffer(buffer, 0, file_header->reserved1);
fwrite(buffer, 1, 2, fp);

insert_short_into_buffer(buffer, 0, file_header->reserved2);
fwrite(buffer, 1, 2, fp);
```

```
insert_ulong_into_buffer(buffer, 0, file_header->bitmapoffset);
fwrite(buffer, 1, 4, fp);


   /*******************************************
   *
   *   Write the 40-byte bit map header.
   *
   *******************************************/

insert_ulong_into_buffer(buffer, 0, bmheader->size);
fwrite(buffer, 1, 4, fp);

insert_long_into_buffer(buffer, 0, bmheader->width);
fwrite(buffer, 1, 4, fp);

insert_long_into_buffer(buffer, 0, bmheader->height);
fwrite(buffer, 1, 4, fp);

insert_ushort_into_buffer(buffer, 0, bmheader->planes);
fwrite(buffer, 1, 2, fp);

insert_ushort_into_buffer(buffer, 0, bmheader->bitsperpixel);
fwrite(buffer, 1, 2, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->compression);
fwrite(buffer, 1, 4, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->sizeofbitmap);
fwrite(buffer, 1, 4, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->horzres);
fwrite(buffer, 1, 4, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->vertres);
fwrite(buffer, 1, 4, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->colorsused);
fwrite(buffer, 1, 4, fp);

insert_ulong_into_buffer(buffer, 0, bmheader->colorsimp);
fwrite(buffer, 1, 4, fp);

   /*******************************************
   *
```

```
      *   Write a blank color table.
      *   It has 256 entries (number of colors)
      *   that are each 4 bytes.
      *
      *********************************************/

   buffer[0] = 0x00;

   for(i=0; i<(256*4); i++)
      fwrite(buffer, 1, 1, fp);

      /*********************************************
      *
      *   Write a zero image.
      *
      *********************************************/

   buffer[0] = 0x00;

   for(i=0; i<bmheader->sizeofbitmap; i++)
      fwrite(buffer, 1, 1, fp);

   fclose(fp);

}  /* ends create_allocate_bmp_file */




      /*****************************************
      *
      *   write_bmp_image(...
      *
      *   This function writes an image array
      *   to a bmp image file.
      *
      *****************************************/

write_bmp_image(file_name, array)
   char   *file_name;
   short  **array;
{
   char   *buffer, c;
   FILE   *image_file;
```

```
int    pad = 0,
       position;
int    bytes, i, j;
long   height = 0, width = 0;
struct bitmapheader  bmheader;
struct bmpfileheader file_header;
struct ctstruct rgb[GRAY_LEVELS+1];
union  short_char_union scu;

read_bmp_file_header(file_name, &file_header);
read_bm_header(file_name, &bmheader);

height = bmheader.height;
width  = bmheader.width;
if(height < 0) height = height*(-1);

buffer = (char  *) malloc(width * sizeof(char ));
for(i=0; i<width; i++)
   buffer[i] = '\0';

image_file = fopen(file_name, "rb+");

   /****************************************
   *
   *   Write the color table first.
   *
   ****************************************/

position   = fseek(image_file, 54, SEEK_SET);
for(i=0; i<GRAY_LEVELS+1; i++){
   rgb[i].blue  = i;
   rgb[i].green = i;
   rgb[i].red   = i;
}  /* ends loop over i */

for(i=0; i<bmheader.colorsused; i++){
   buffer[0] = rgb[i].blue;
   fwrite(buffer , 1, 1, image_file);
   buffer[0] = rgb[i].green;
   fwrite(buffer , 1, 1, image_file);
   buffer[0] = rgb[i].red;
   fwrite(buffer , 1, 1, image_file);
   buffer[0] = 0x00;
   fwrite(buffer , 1, 1, image_file);
}  /* ends loop over i */
```

```
   position   = fseek(image_file,
                      file_header.bitmapoffset,
                      SEEK_SET);

   pad = calculate_pad(width);

   for(i=0; i<height; i++){
      for(j=0; j<width; j++){

         if(bmheader.bitsperpixel == 8){
            scu.s_num = 0;
            if(bmheader.height > 0)
               scu.s_num = array[height-1-i][j];
            else
               scu.s_num = array[i][j];
            buffer[j] = scu.s_alpha[0];
         }  /* ends if bits_per_pixel == 8 */
         else{
            printf("\nERROR bitsperpixel is not 8");
            exit(1);
         }
      }  /* ends loop over j */

      bytes = fwrite(buffer, 1, width, image_file);

      if(pad != 0){
         for(j=0; j<pad; j++)
            buffer[j] = 0x00;
         fwrite(buffer, 1, pad, image_file);
      }  /* ends if pad != 0 */

   }  /* ends loop over i */

   fclose(image_file);
   free(buffer);
}  /* ends write_bmp_image */




   /***********************************************
   *
   *   is_a_bmp(...
```

```
   *
   *   This function looks at a file to see if it
   *   is a bmp file.  First look at the file
   *   extension.  Next look at the filetype to
   *   ensure it is 0x4d42.
   *
   **********************************************/

int is_a_bmp(file_name)
   char *file_name;
{
   char   *cc;
   int    result = 0;
   struct bmpfileheader file_header;

   cc = strstr(file_name, ".bmp");
   if(cc == NULL)
      return(result);

   read_bmp_file_header(file_name, &file_header);
   if(file_header.filetype != 0x4d42)
      return(result);

   result = 1;
   return(result);
}  /* ends is_a_bmp */




      /*****************************************
      *
      *   calculate_pad(...
      *
      *   This function calculates the pad needed
      *   at the end of each row of pixels in a
      *   bmp image.
      *
      *****************************************/

int calculate_pad(width)
   long width;
{
   int pad = 0;
```

```
   pad = ( (width%4) == 0) ? 0 : (4-(width%4));
   return(pad);
}  /* ends calculate_pad */




   /*********************************************
   *
   *   equate_bmpfileheaders(...
   *
   *   This function sets the elements of the
   *   destination header to the values of the
   *   source header.
   *
   *********************************************/

equate_bmpfileheaders(src, dest)
   struct bmpfileheader *src, *dest;
{
   dest->filetype     = src->filetype;
   dest->filesize     = src->filesize;
   dest->reserved1    = src->reserved1;
   dest->reserved2    = src->reserved2;
   dest->bitmapoffset = src->bitmapoffset;
}  /* ends equate_bmpfileheaders */




      /*******************************************
      *
      *   flip_image_array(...
      *
      *   This function flips an image array
      *   about its horizontal mid-point.
      *
      *******************************************/

flip_image_array(the_image, rows, cols)
   long   cols, rows;
   short  **the_image;
{
```

```
   int  i, j;
   long rd2;
   short **temp;

   temp = allocate_image_array(rows, cols);
   rd2  = rows/2;
   for(i=0; i<rd2; i++){
      for(j=0; j<cols; j++){
         temp[rows-1-i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

   for(i=rd2; i<rows; i++){
      for(j=0; j<cols; j++){
         temp[rows-1-i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] = temp[i][j];

   free_image_array(temp, rows);
}  /* ends flip_image_array */
```

Listing 1.3 - The BMP I/O Routines

```
/*************************************************************
*
*   file cips.h
*
*       Functions: This file contains no functions. It
*          contains declarations of the data structures used
*          by the C Image Processing Systems CIPS.
*
*   Purpose:
*      To declare data structures.
*
*   Modifications:
```

```
  *      June 1990 = created
  *
  ************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include <sys\types.h>
#include <sys\stat.h>

#define MAX_NAME_LENGTH       80
#define ROWS                 100
#define COLS                 100
#define GRAY_LEVELS          255
#define PREWITT                1
#define PEAK_SPACE            50
#define PEAKS                 30
#define KIRSCH                 2
#define SOBEL                  3
#define STACK_SIZE         40000
#define STACK_FILE_LENGTH    500
#define FORGET_IT            -50
#define STACK_FILE     "c:stack"


#define OTHERC 1
#undef  MSC

      /**************************************************
      *
      *   The following struct defines the information
      *   you need to read from the tiff file
      *   header.
      *
      **************************************************/

struct tiff_header_struct{
   short lsb;
   long  bits_per_pixel;
   long  image_length;
```

```
   long  image_width;
   long  strip_offset;
};

      /**************************************************
      *
      *   The following struct defines the information
      *   you need to read from the bmp file
      *   header.
      *
      ***************************************************/


struct bmpfileheader{
   unsigned short  filetype;
   unsigned long   filesize;
   short  reserved1;
   short  reserved2;
   unsigned long   bitmapoffset;
};

struct bitmapheader{
   unsigned long   size;
   long   width;
   long   height;
   unsigned short  planes;
   unsigned short  bitsperpixel;
   unsigned long   compression;
   unsigned long   sizeofbitmap;
   unsigned long   horzres;
   unsigned long   vertres;
   unsigned long   colorsused;
   unsigned long   colorsimp;
};

struct ctstruct{
   unsigned char blue;
   unsigned char green;
   unsigned char red;
};



    /*******************************************
    *
```

```
    *       The following unions are used
    *       to put the bytes from the header
    *       into either an integer or a floating
    *       point number.
    *
    **********************************************/

union short_char_union{
        short s_num;
        char  s_alpha[2];
};

union int_char_union{
        int  i_num;
   char i_alpha[2];
};

union long_char_union{
        long  l_num;
        char  l_alpha[4];
};

union float_char_union{
        float f_num;
        char  f_alpha[4];
};

union ushort_char_union{
        short s_num;
        char  s_alpha[2];
};

union uint_char_union{
        int  i_num;
   char i_alpha[2];
};

union ulong_char_union{
        long  l_num;
        char  l_alpha[4];
};
```

Listing 1.4 - The cips.h Include File

```
/**********************************************
*
*       file c:\cips\round.c
*
*       Functions: This file contains
*          main
*
*       Purpose:
*          This program takes an image file and
*          rounds if off by copying a part of it
*          to another file.
*
*       External Calls:
*         imageio.c - does_not_exit
*                     get_image_size
*                     allocate_image_array
*                     read_image_array
*                     is_a_tiff
*                     is_a_bmp
*                     create_allocate_tiff_file
*                     read_bmp_file_header
*                     read_bm_header
*                     create_allocate_bmp_file
*                     write_image_array
*                     free_image_array
*
*
*       Modifications:
*          31 March 1991 - created
*          8  May 1993 - Made this program
*                        command line driven.
*          6 August 1998 - Made this work with
*              entire image arrays at once.
*         18 September 1998 - modified to work with
*               all I O routines in imageio.c.
*
**********************************************/

#include "cips.h"


main(argc, argv)
   int  argc;
   char *argv[];
```

```
{
char response[80];
   char     name[80], name2[80];
   int      i          = 0,
            ie         = 0,
            il         = 0,
            j          = 0,
            in_length  = 0,
            out_length = 0,
            in_width   = 0,
            out_width  = 0;
   short    **the_image, **out_image;

   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;
   struct tiff_header_struct tiff_file_header;

      /*****************************************
      *
      *   Ensure the command line is correct.
      *
      *****************************************/

   if(argc < 5   ||
     (argc > 5  &&  argc < 7)){
      printf("\nusage: roundoff in-image out-image"
          " length width [il ie]"
          "\n"
          "\n       If you do not specify il ie"
          " they will be set to 1 1."
          "\n       ll le will always be"
          " il+length and ie+width"
          "\n");
      exit(0);
   }

   strcpy(name,  argv[1]);
   strcpy(name2, argv[2]);
   out_length = atoi(argv[3]);
   out_width  = atoi(argv[4]);

   if(argc > 5){
      il = atoi(argv[5]);
      ie = atoi(argv[6]);
   }
```

```
if(does_not_exist(name)){
 printf("\nERROR input file %s does not exist",
          name);
 exit(0);
}

get_image_size(name, &in_length, &in_width);
the_image = allocate_image_array(in_length,
                                 in_width);
read_image_array(name, the_image);

   /****************************************
   *
   *   Create the output image and allocate
   *   the output image array.
   *
   ****************************************/

if(is_a_tiff(name)){
   read_tiff_header(name, &tiff_file_header);
   tiff_file_header.image_length = out_length;
   tiff_file_header.image_width  = out_width;
   create_allocate_tiff_file(name2,
                             &tiff_file_header);
}

if(is_a_bmp(name)){
   read_bmp_file_header(name,
                        &bmp_file_header);
   read_bm_header(name, &bmheader);
   bmheader.height = out_length;
   bmheader.width  = out_width;
   create_allocate_bmp_file(name2,
                            &bmp_file_header,
                            &bmheader);
}

out_image = allocate_image_array(out_length, out_width);

   /****************************************
   *
   *   Copy the input image array to the output
   *   image array per the input parameters.
   *
```

```
      ****************************************/

   for(i=0; i<out_length; i++)
      for(j=0; j<out_width; j++)
         out_image[i][j] = the_image[i+il][j+ie];

   write_image_array(name2, out_image);

   free_image_array(out_image, out_length);
   free_image_array(the_image, in_length);

}  /* ends main */
```

Listing 1.5 - The round Program

```
   /*************************************************
   *
   *       file tif2bmp.c
   *
   *       Functions: This file contains
   *           main
   *
   *       Purpose:
   *          This program creates a bmp file
   *          that is just like the input tiff file.
   *
   *       External Calls:
   *          imageio.c
   *             does_not_exist
   *             get_image_size
   *             read_image_array
   *             write_image_array
   *             free_image_array
   *             create_allocate_bmp_file
   *
   *       Modifications:
   *         27 September 1998 - created
   *
   ***************************************************/
```

```
#include "cips.h"


main(argc, argv)
   int  argc;
   char *argv[];
{
   char   *cc;
   int    l, w;
   int    ok = 0;
   short  **the_image;
   struct tiff_header_struct image_header;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;

   if(argc < 3 || argc > 3){
      printf(
      "\nusage: tif2bmp tif-file-name bmp-file-name\n");
      exit(-1);
   }

   if(does_not_exist(argv[1])){
    printf("\nERROR input file %s does not exist",
             argv[1]);
    exit(0);
   }

   cc = strstr(argv[1], ".tif");
   if(cc == NULL){
      printf("\nERROR %s must be a tiff file",
             argv[1]);
    exit(0);
   }  /* ends tif */

   cc = strstr(argv[2], ".bmp");
   if(cc == NULL){  /* create a bmp */
      printf("\nERROR %s must be a bmp file name",
             argv[2]);
      exit(0);
   }

   get_image_size(argv[1], &l, &w);
   the_image       = allocate_image_array(l, w);
   bmheader.height = l;
```

```
   bmheader.width  = w;
   create_allocate_bmp_file(argv[2],
                            &bmp_file_header,
                            &bmheader);

   read_image_array(argv[1], the_image);
   write_image_array(argv[2], the_image);
   free_image_array(the_image, l);

}  /* ends main */
```

Listing 1.6 - The tif2bmp Program

```
   /**************************************************
   *
   *       file bmp2tif.c
   *
   *       Functions: This file contains
   *           main
   *
   *       Purpose:
   *          This program creates a tiff file
   *          that is just like the input bmp file.
   *
   *       External Calls:
   *          imageio.c
   *             does_not_exist
   *             get_image_size
   *             read_image_array
   *             write_image_array
   *             free_image_array
   *             create_allocate_tif_file
   *
   *       Modifications:
   *         27 September 1998 - created
   *
   **************************************************/

#include "cips.h"
```

```
main(argc, argv)
   int  argc;
   char *argv[];
{
   char   *cc;
   int    l, w;
   int    ok = 0;
   short  **the_image;
   struct tiff_header_struct image_header;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;

   if(argc < 3 || argc > 3){
      printf(
      "\nusage: bmp2tif bmp-file-name tif-file-name\n");
      exit(-1);
   }

   if(does_not_exist(argv[1])){
    printf("\nERROR input file %s does not exist",
             argv[1]);
    exit(0);
   }

   cc = strstr(argv[1], ".bmp");
   if(cc == NULL){
      printf("\nERROR %s must be a bmp file",
             argv[1]);
    exit(0);
   }  /* ends tif */

   cc = strstr(argv[2], ".tif");
   if(cc == NULL){  /* create a bmp */
      printf("\nERROR %s must be a tiff file name",
             argv[2]);
      exit(0);
   }

   get_image_size(argv[1], &l, &w);
   the_image = allocate_image_array(l, w);
   image_header.lsb            = 1;
   image_header.bits_per_pixel = 8;
   image_header.image_length   = l;
   image_header.image_width    = w;;
```

```
   image_header.strip_offset   = 1000;
   create_allocate_tiff_file(argv[2],
                             &image_header);

   read_image_array(argv[1], the_image);
   write_image_array(argv[2], the_image);
   free_image_array(the_image, l);

}  /* ends main */
```

Listing 1.7 - The bmp2tif Program

F.2 Code Listings for Chapter 2

```
   /***************************************************
   *
   *   file showi.c
   *
   *   Functions: This file contains
   *      main
   *      show_screen
   *      is_in_image
   *
   *   Purpose:
   *      This file contains the program
   *      that shows image numbers on the screen.
   *
   *   External Calls:
   *      imageio.c - get_image_size
   *                  read_image_array
   *                  allocate_image_array
   *                  free_image_array
   *
   *   Modifications:
   *      1 October 1998 - created to work with
   *           all I O routines in imageio.c.
   *
   ***************************************************/

#include "cips.h"
```

```
#define  SHEIGHT 20
#define  SWIDTH  15

main(argc, argv)
   int argc;
   char *argv[];
{
   char  in_name[MAX_NAME_LENGTH];
   char  response[MAX_NAME_LENGTH];
   int   ie, il, not_done, temp_ie, temp_il;
   long  height, width;
   short **the_image;

      /*****************************************
      *
      *   Ensure the command line is correct.
      *
      *****************************************/

   if(argc != 4){
    printf("\nusage: showi input-image il ie");
    exit(0);
   }

   strcpy(in_name,  argv[1]);
   il = atoi(argv[2]);
   ie = atoi(argv[3]);

      /*****************************************
      *
      *   Ensure the input image exists.
      *   Allocate an image array.
      *   Read the image and show it on the
      *   screen.
      *
      *****************************************/

   if(does_not_exist(in_name)){
      printf("\nERROR input file %s does not exist",
             in_name);
      exit(0);
   }  /* ends if does_not_exist */

   get_image_size(in_name, &height, &width);
   the_image = allocate_image_array(height, width);
```

```
   read_image_array(in_name, the_image);

   temp_il  = il;
   temp_ie  = ie;
   not_done = 1;

   while(not_done){
      if(is_in_image(temp_il, temp_ie, height, width)){
         il = temp_il;
         ie = temp_ie;
         show_screen(the_image, il, ie);
      }  /* ends if is_in_image */

      printf("\n\n x=quit j=down k=up h=left l=right"
             "\nEnter choice and press Enter:  ");
      gets(response);

      if(response[0] == 'x' || response[0] == 'X')
         not_done = 0;
      if(response[0] == 'j' || response[0] == 'J')
         temp_il = temp_il + ((3*SHEIGHT)/4);
      if(response[0] == 'k' || response[0] == 'K')
         temp_il = temp_il - ((3*SHEIGHT)/4);
      if(response[0] == 'h' || response[0] == 'H')
         temp_ie = temp_ie - ((3*SWIDTH)/4);
      if(response[0] == 'l' || response[0] == 'L')
         temp_ie = temp_ie + ((3*SWIDTH)/4);
   }  /* ends while not_done */

   free_image_array(the_image, height);

}  /* ends main */



int is_in_image(il, ie, height, width)
   int il, ie;
   long height, width;
{
   int result = 1;

   if(il < 0){
      printf("\nil=%d tool small", il);
      result = 0;
   }
```

```
   if(ie < 0){
      printf("\nie=%d tool small", ie);
      result = 0;
   }

   if((il+SHEIGHT) > height ){
      printf("\nll=%d tool big", il+SHEIGHT);
      result = 0;
   }

   if((ie+SWIDTH) > width ){
      printf("\nle=%d tool big", ie+SWIDTH);
      result = 0;
   }

   return(result);


}  /* ends is_in_image */




show_screen(the_image, il, ie)
   int  il, ie;
   short **the_image;
{
   int i, j;

   printf("\n     ");
   for(i=ie-1; i<ie-1+SWIDTH; i++)
      printf("-%3d", i);

   for(i=il-1; i<il-1+SHEIGHT; i++){
      printf("\n%4d>", i);
      for(j=ie-1; j<ie-1+SWIDTH; j++){
         printf("-%3d", the_image[i][j]);
      }
   }

}  /* ends show_screen */
```

Listing 2.1 - The Showi Program

```
   /***********************************************
   *
   *   file dumpi.c
   *
   *   Functions: This file contains
   *      main
   *
   *   Purpose:
   *      This file contains a program that
   *      dumps the number values of an image
   *      to an ascii text file.
   *
   *   External Calls:
   *      imageio.c - get_image_size
   *                  read_image_array
   *                  allocate_image_array
   *                  free_image_array
   *
   *   Modifications:
   *      1 October 1998 - created to work with
   *           all I O routines in imageio.c.
   *
   ***********************************************/

#include "cips.h"

main(argc, argv)
   int argc;
   char *argv[];
{
   char  in_name[MAX_NAME_LENGTH];
   char  out_name[MAX_NAME_LENGTH];
   char  *line, buffer[10];
   int   i, j;
   long  height, width;
   short **the_image;
   FILE  *out_file;


      /*****************************************
      *
      *   Ensure the command line is correct.
```

```
   *
   ****************************************/

if(argc != 3){
 printf("\nusage: dumpi input-image output-file");
 exit(0);
}

strcpy(in_name,  argv[1]);
strcpy(out_name, argv[2]);

   /****************************************
   *
   *   Ensure the input image exists.
   *   Create the output text file.
   *   Allocate an image array.
   *   Read the image and dump the nubmers
   *   to a text file.
   *
   ****************************************/

if(does_not_exist(in_name)){
   printf("\nERROR input file %s does not exist",
          in_name);
   exit(0);
}  /* ends if does_not_exist */

if((out_file = fopen(out_name, "wt")) == NULL){
   printf("\nERROR Could not open file %s",
          out_name);
   exit(2);
}


get_image_size(in_name, &height, &width);
the_image = allocate_image_array(height, width);
read_image_array(in_name, the_image);

line = malloc( ((width*4)+7) * sizeof(char  *));

sprintf(line, "      ");
for(i=0; i<width; i++){
   sprintf(buffer, "%4d", i);
   strcat(line, buffer);
}
```

```
   strcat(line, "\n");
   fputs(line, out_file);

   for(i=0; i<height; i++){
      sprintf(line, "%5d>", i);
      for(j=0; j<width; j++){
         sprintf(buffer, "-%3d", the_image[i][j]);
         strcat(line, buffer);
      }
      strcat(line, "\n");
      fputs(line, out_file);
   }

   free_image_array(the_image, height);
   fclose(out_file);

}  /* ends main */
```

Listing 2.2 - The Dumpi Program

F.3 Code Listings for Chapter 3

```
#include "cips.h"

       /**********************************************
       *
       *   half_tone(...
       *
       *   ep[m][n] = sum of erros propogated
       *               to position (m,n).
       *   eg[m][n] = total error generated at
       *               location (m,n).
       *
       **********************************************/

half_tone(in_image, out_image,
          threshold,
          one, zero,
          rows, cols)
   long  rows, cols;
   short threshold, one, zero;
```

```
   short **in_image, **out_image;
{
   float **eg, **ep;
   float c[2][3],
         sum_p,
         t;
   int   i, j, m, n, xx, yy;

   c[0][0] = 0.0;
   c[0][1] = 0.2;
   c[0][2] = 0.0;
   c[1][0] = 0.6;
   c[1][1] = 0.1;
   c[1][2] = 0.1;

      /**********************************************
      *
      *   Calculate the total propogated error
      *   at location(m,n) due to prior
      *   assignment.
      *
      *   Go through the input image.  If the output
      *   should be one then display that pixel as such.
      *   If the output should be zero then display it
      *   that way.
      *
      *   Also set the pixels in the input image array
      *   to 1's and 0's in case the print option
      *   was chosen.
      *
      **********************************************/

   eg = malloc(rows * sizeof(float  *));
   for(i=0; i<rows; i++){
      eg[i] = malloc(cols * sizeof(float ));
      if(eg[i] == '\0'){
         printf("\n\tmalloc of eg[%d] failed", i);
      }  /* ends if */
   }  /* ends loop over i */

   ep = malloc(rows * sizeof(float  *));
   for(i=0; i<rows; i++){
      ep[i] = malloc(cols * sizeof(float ));
      if(ep[i] == '\0'){
         printf("\n\tmalloc of ep[%d] failed", i);
```

```
      }  /* ends if */
   }  /* ends loop over i */

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         eg[i][j] = 0.0;
         ep[i][j] = 0.0;
      }
   }


       /**********************************************
       *
       *   29 February 1988 - Fix to remove a solid
       *   line at the bottom of each region. Loop
       *   over ROWS-1 and then draw an extra line.
       *
       **********************************************/

   for(m=0; m<rows; m++){
      for(n=0; n<cols; n++){

         sum_p = 0.0;
         for(i=0; i<2; i++){
            for(j=0; j<3; j++){

               xx = m-i+1;
               yy = n-j+1;
               if(xx < 0)     xx = 0;
               if(xx >= rows) xx = rows-1;
               if(yy < 0)     yy = 0;
               if(yy >= cols) yy = cols-1;

               sum_p = sum_p + c[i][j] * eg[xx][yy];
            }  /* ends loop over j */
         }     /* ends loop over i */

         ep[m][n] = sum_p;
         t = in_image[m][n] + ep[m][n];

               /**********************************
               *
               *    Here set the point [m][n]=one
               *
               ***********************************/
```

```
         if(t > threshold){
            eg[m][n]        = t - threshold*2;
            out_image[m][n] = one;
         }  /* ends if t > threshold  */


               /*********************************
               *
               *    Here set the point [m][n]=zero
               *
               **********************************/

         else{  /* t <= threshold */
            eg[m][n]        = t;
            out_image[m][n] = zero;
         }  /* ends else t <= threshold  */

      }  /* ends loop over n columns */
   }         /* ends loop over m rows */


   for(i=0; i<rows; i++){
      free(eg[i]);
      free(ep[i]);
   }


}  /*  ends half_tone  */
```

Listing 3.1 - The half_tone Subroutine

```
   /**********************************************
   *
   *   file halftone.c
   *
   *   Functions: This file contains
   *      main
   *
```

```
   *   Purpose:
   *      This file contains the main calling
   *      routine that performs histogram
   *      equalization.
   *
   *   External Calls:
   *      imageio.c - create_image_file
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
   *                  does_not_exist
   *      ht.c - half_tone
   *
   *   Modifications:
   *     30 September 1998 - created to work with
   *         all I O routines in imageio.c.
   *
   ************************************************/

#include "cips.h"

main(argc, argv)
   int argc;
   char *argv[];
{
   char  in_name[MAX_NAME_LENGTH];
   char  out_name[MAX_NAME_LENGTH];
   int   i;
   long  height, width;
   short **the_image, **out_image;
   short threshold;


      /*****************************************
      *
      *   Ensure the command line is correct.
      *
      *****************************************/

   if(argc != 4){
    printf(
    "\nusage: halftone input-image output-image threshold");
    exit(0);
```

```
   }

   strcpy(in_name,  argv[1]);
   strcpy(out_name, argv[2]);
   threshold = atoi(argv[3]);

      /*****************************************
      *
      *   Ensure the input image exists.
      *   Create the output image file.
      *   Allocate an image array, read the input
      *   image, half_tone it, and write
      *   the result.
      *
      *****************************************/

   if(does_not_exist(in_name)){
      printf("\nERROR input file %s does not exist",
             in_name);
      printf("\n      "
             "usage: histeq input-image output-image");
      exit(0);
   }  /* ends if does_not_exist */

   create_image_file(in_name, out_name);
   get_image_size(in_name, &height, &width);
   the_image = allocate_image_array(height, width);
   out_image = allocate_image_array(height, width);
   read_image_array(in_name, the_image);

   half_tone(the_image, out_image,
             threshold, 200, 0,
             height, width);

   write_image_array(out_name, out_image);
   free_image_array(the_image, height);
   free_image_array(out_image, height);

}  /* ends main */
```

Listing 3.2 - The main Routine for the Halftone Program

```
   /***********************************************
   *
   *   file dumpi.c
   *
   *   Functions: This file contains
   *      main
   *
   *   Purpose:
   *      This file contains a program that
   *      is very similar to dumpi.  Dumpi dumps
   *      the number values of an image
   *      to an ascii text file.
   *      This program sends a space to a text
   *      file for zeros in the image and an
   *      asterisk for non-zeros in the image.
   *
   *   External Calls:
   *      imageio.c - read_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
   *                  does_not_exist
   *
   *   Modifications:
   *      3 October 1998 - created to work with
   *           all I O routines in imageio.c.
   *
   ************************************************/

#include "cips.h"

main(argc, argv)
   int argc;
   char *argv[];
{
   char  in_name[MAX_NAME_LENGTH];
   char  out_name[MAX_NAME_LENGTH];
   char  *line, buffer[10];
   int   i, j;
   long  height, width;
   short **the_image;
   FILE  *out_file;


      /*****************************************
```

```
   *
   *   Ensure the command line is correct.
   *
   ******************************************/

if(argc != 3){
 printf("\nusage: dumpb input-image output-file");
 exit(0);
}

strcpy(in_name,  argv[1]);
strcpy(out_name, argv[2]);

   /******************************************
   *
   *   Ensure the input image exists.
   *   Read the input image and dump the
   *   1s and 0s to an output text file.
   *
   ******************************************/

if(does_not_exist(in_name)){
   printf("\nERROR input file %s does not exist",
          in_name);
   exit(0);
}  /* ends if does_not_exist */

if((out_file = fopen(out_name, "wt")) == NULL){
   printf("\nERROR Could not open file %s",
          out_name);
   exit(2);
}


get_image_size(in_name, &height, &width);
the_image = allocate_image_array(height, width);
read_image_array(in_name, the_image);

line = malloc( (width+7) * sizeof(char  *));

sprintf(line, "      ");
for(i=0; i<width; i++){
   sprintf(buffer, "%4d", i);
   strcat(line, buffer);
}
```

```
   strcat(line, "\n");
   fputs(line, out_file);

   for(i=0; i<height; i++){
      sprintf(line, "%5d>", i);
      for(j=0; j<width; j++){
         if(the_image[i][j] == 0)
            strcat(line, " ");
         else
            strcat(line, "*");
      }
      strcat(line, "\n");
      fputs(line, out_file);
   }

   free_image_array(the_image, height);
   fclose(out_file);

}  /* ends main */
```

Listing 3.3- The dumpb Program

F.4 Code Listings for Chapter 4

```
   /****************************************************
   *
   *   file hist.c
   *
   *   Functions: This file contains
   *       calculate_histogram
   *       perform_histogram_equalization
   *       zero_histogram
   *       smooth_histogram
   *
   *   Purpose: These functions calculate
   *      the histogram of an input image array.
   *      They also modify an image by equalizing
   *      its histogram.
   *
   *   Modifications:
```

```
   *       July 86 - ported to IBM-PC
   *       August 1990 - modified for use in the
   *           C Image Processing System
   *       March 1992 - removed the hardwired values
   *           of 100 and replaced them with ROWS
   *           and COLS.  There are still some
   *           hardwired numbers in this file, but
   *           they deal with displaying a histogram.
   *       October 4, 1992 - added the smooth histogram
   *           function.
   *       April 22, 1998 - modified routines to work
   *           with an entire image in one array.
   *
   ************************************************/

#include "cips.h"


#define PRINT_WIDTH  80
#define FORMFEED     '\014'



        /****************************************
        *
        *   zero_histogram(...
        *
        *   This function clears or zeros a
        *   histogram array.
        *
        ****************************************/

zero_histogram(histogram, gray_levels)
   int      gray_levels;
   unsigned long histogram[];
{
   int i;
   for(i=0; i<gray_levels; i++)
      histogram[i] = 0;
}  /* ends zero_histogram */



        /****************************************
```

```
        *
        *   calculate_histogram(...
        *
        *   This function calculates the histogram
        *   for an input image array.
        *
        *******************************************/

calculate_histogram(image, histogram, length, width)
   int    length, width;
   short  **image;
   unsigned long histogram[];
{
   long  i,j;
   short k;
   for(i=0; i<length; i++){
      for(j=0; j<width; j++){
         k = image[i][j];
         histogram[k] = histogram[k] + 1;
      }
   }
}  /* ends calculate_histogram */



      /*********************************************
      *
      *   smooth_histogram(...
      *
      *   This function smoothes the input histogram
      *   and returns it.  It uses a simple averaging
      *   scheme where each point in the histogram
      *   is replaced by the average of itself and
      *   the two points on either side of it.
      *
      **********************************************/

smooth_histogram(histogram, gray_levels)
   int      gray_levels;
   unsigned long histogram[];
{
   int i;
   unsigned long new_hist[gray_levels];
```

```
   zero_histogram(new_hist, gray_levels);

   new_hist[0] = (histogram[0] + histogram[1])/2;
   new_hist[gray_levels] =
      (histogram[gray_levels] +
       histogram[gray_levels-1])/2;

   for(i=1; i<gray_levels-1; i++){
      new_hist[i] = (histogram[i-1] +
                     histogram[i]   +
                     histogram[i+1])/3;
   }

   for(i=0; i<gray_levels; i++)
      histogram[i] = new_hist[i];

}  /* ends smooth_histogram */



         /****************************************
         *
         *    perform_histogram_equalization(...
         *
         *    This function performs histogram
         *    equalization on the input image array.
         *
         ****************************************/

perform_histogram_equalization(image,
                               histogram,
                               gray_levels,
                               new_grays,
                               length,
                               width)
   int   gray_levels, new_grays;
   long  length, width;
   short **image;
   unsigned long histogram[];
{
   int i,
       j,
       k;
   unsigned long sum,
```

```
            sum_of_h[gray_levels];

   double constant;

   sum = 0;
   for(i=0; i<gray_levels; i++){
      sum         = sum + histogram[i];
      sum_of_h[i] = sum;
   }

      /* constant = new # of gray levels div by area */
   constant = (float)(new_grays)/(float)(length*width);
   for(i=0; i<length; i++){
      for(j=0; j<width; j++){
         k           = image[i][j];
         image[i][j] = sum_of_h[k] * constant;
      }
   }
}  /* ends perform_histogram_equalization */




hist_long_clear_buffer(string)
   char string[];
{
   int i;
   for(i=0; i<300; i++)
      string[i] = ' ';
}
```

Listing 4.1 - The Histogram Routines

```
   /***********************************************
   *
   *   file histeq.c
   *
   *   Functions: This file contains
   *      main
   *
```

```
   *   Purpose:
   *      This file contains the main calling
   *      routine that performs histogram
   *      equalization.
   *
   *   External Calls:
   *      imageio.c - create_image_file
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
   *      hist.c - calculate_histogram
   *               perform_histogram_equalization
   *
   *   Modifications:
   *     18 September 1998 - created to work with
   *           all I O routines in imageio.c.
   *
   ************************************************/

#include "cips.h"

main(argc, argv)
   int argc;
   char *argv[];
{
   char  in_name[MAX_NAME_LENGTH];
   char  out_name[MAX_NAME_LENGTH];
   char  response[MAX_NAME_LENGTH];
   int   i;
   long  height, width;
   short **the_image;
   unsigned long histogram[GRAY_LEVELS+1];


      /****************************************
      *
      *   Ensure the command line is correct.
      *
      ****************************************/

   if(argc < 3){
    printf("\n   usage: histeq input-image output-image");
    exit(0);
```

```
   }

   strcpy(in_name,  argv[1]);
   strcpy(out_name, argv[2]);

      /*****************************************
      *
      *   Ensure the input image exists.
      *   Create the output image file.
      *   Allocate an image array and call the
      *   histogram operators.
      *
      *****************************************/

   if(does_not_exist(in_name)){
      printf("\nERROR input file %s does not exist",
             in_name);
      printf("\n      "
             "usage: histeq input-image output-image");
      exit(0);
   }  /* ends if does_not_exist */

   create_image_file(in_name, out_name);
   get_image_size(in_name, &height, &width);
   the_image = allocate_image_array(height, width);
   read_image_array(in_name, the_image);

   for(i=0; i<GRAY_LEVELS+1; i++) histogram[i] = 0;

   calculate_histogram(the_image, histogram,
                       height, width);

   perform_histogram_equalization(
                       the_image, histogram,
                       256, 250,
                       height, width);

   write_image_array(out_name, the_image);
   free_image_array(the_image, height);

}  /* ends main */
```

Listing 4.2 - The main Routine of the histeq Program

```
/**************************************************
*
*       file himage.c
*
*       Functions: This file contains
*           main
*           vline
*           hline
*
*       Purpose:
*          This program calculates the histogram
*          of an image and puts the picture of
*          that histogram in an output image.
*
*       External Calls:
*          imageio.c
*             does_not_exist
*             create_allocate_tiff_file
*             create_allocate_bmp_file
*             get_image_size
*             allocate_image_array
*             free_image_array
*             read_image_array
*             write_image_array
*          hist.c
*             calculate_histogram
*
*       Modifications:
*          7 Arpil 1992 - created
*         15 August 1998 - modified to work with
*              an entire image array at once.
*         22 September 1998 - modified to work with
*              all I O routines in imageio.c.
*
**************************************************/

#include "cips.h"

#define W    300
#define L    300
#define UP     5
#define LEFT   5
```

```
#define SPOT 200


main(argc, argv)
   int  argc;
   char *argv[];
{
   char   *cc;
   int    i, j, amount;
   long   l, w;
   int    ok = 0;
   long   length, width;
   short  **image, **hist;
   struct tiff_header_struct image_header;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;
   unsigned long histogram[GRAY_LEVELS+1];
   int     count;
   unsigned long max, scale;

   max   = 0;
   count = 0;

   if(argc < 3 ){
      printf(
      "\nusage: himage image-file histogram-file [length width]\n");
      exit(-1);
   }

   if(does_not_exist(argv[1])){
    printf("\nERROR input file %s does not exist",
             argv[1]);
    exit(0);
   }

   if(argc >= 4)
      l = atoi(argv[3]);
   else
      l = L;

   if(argc >= 5)
      w = atoi(argv[4]);
   else
      w = W;
```

```
cc = strstr(argv[2], ".tif");
if(cc != NULL){  /* create a tif */
   ok = 1;
   image_header.lsb            = 1;
   image_header.bits_per_pixel = 8;
   image_header.image_length   = l;
   image_header.image_width    = w;;
   image_header.strip_offset   = 1000;
   create_allocate_tiff_file(argv[2],
                              &image_header);
}  /* ends tif */

cc = strstr(argv[2], ".bmp");
if(cc != NULL){  /* create a bmp */
   ok = 1;
   bmheader.height = l;
   bmheader.width  = w;
   create_allocate_bmp_file(argv[2],
                            &bmp_file_header,
                            &bmheader);
}  /* ends tif */

if(ok == 0){
   printf("\nERROR input file neither tiff nor bmp");
   exit(0);
}

get_image_size(argv[1], &length, &width);
image = allocate_image_array(length, width);
hist  = allocate_image_array(l, w);
read_image_array(argv[1], image);

for(i=0; i<l; i++)
   for(j=0; j<w; j++)
      hist[i][j] = 0;

for(i=0; i<GRAY_LEVELS+1; i++) histogram[i] = 0;

calculate_histogram(image, histogram,
                    length, width);


hline(hist, l-UP, LEFT, LEFT+GRAY_LEVELS+1);
vline(hist, LEFT+ 50, l-UP+2, l-UP);
vline(hist, LEFT+100, l-UP+2, l-UP);
```

```
   vline(hist, LEFT+150, l-UP+2, l-UP);
   vline(hist, LEFT+200, l-UP+2, l-UP);
   vline(hist, LEFT+250, l-UP+2, l-UP);

   for(i=0; i<GRAY_LEVELS+1; i++)
      if(histogram[i] > max)
         max = histogram[i];

   if(max > (l-UP-UP))
      scale = max/(l-5*UP);
   else
      scale = 1;

   printf("\n max=%ld scale=%ld",max, scale);

   for(i=0; i<GRAY_LEVELS+1; i++){
      amount = histogram[i]/scale;
      if(amount > 0){
         vline(hist, i+LEFT, l-UP, l-UP-amount);
      }  /* ends if not zero */
   }  /* ends loop over i GRAY_LEVELS */

   write_image_array(argv[2], hist);

   free_image_array(image, length);
   free_image_array(hist, l);

}  /* ends main */




vline(image, ie, il, ll)
   int ie, il, ll;
   short **image;
{
   int i, j;

   for(i=il; i>=ll; i--)
      image[i][ie] = SPOT;

}  /* ends vline */
```

```
hline(image, il, ie, le)
   int il, ie, le;
   short **image;
{
   int i, j;

   for(i=ie; i<=le; i++)
      image[il][i] = SPOT;

}  /* ends hline */
```

Listing 4.3 - The himage Program

```
   /**********************************************
   *
   *   file d:\cips\side.c
   *
   *   Functions: This file contains
   *      main
   *      print_side_usage
   *
   *   Purpose:
   *      This file contains the main calling
   *      routine for a program which
   *      takes two images and pastes them
   *      together side by side or top to bottom
   *      into a new image file.
   *
   *      There are three files: two input files
   *      (file1 and file2), and one output
   *      file (file3).
   *
   *   External Calls:
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
```

```
   *                  create_allocate_tiff_file
   *                  create_allocate_bmp_file
   *                  read_bm_header
   *                  read_bmp_file_header
   *                  read_bm_header
   *
   *   Modifications:
   *      19 April 1992 - created
   *      13 August 1998 - modified to work on an
   *           entire image at one time.
   *      19 September 1998 - modified to work with
   *            all I O routines in imageio.c.
   *
   ***************************************************/

#include "cips.h"



main(argc, argv)
   int  argc;
   char *argv[];
{

   char     method[80], name1[80], name2[80], name3[80];
   int      i, j;
   long     length1, length2, length3,
            width1, width2, width3;
   short    **image1, **image2, **image3;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;
   struct tiff_header_struct tiff_file_header;

       /*****************************************
       *
       *  Interpret the command line parameters.
       *
       *****************************************/

   if(argc != 5){
      print_side_usage();
      exit(0);
   }

   strcpy(name1,  argv[1]);
```

```
strcpy(name2,  argv[2]);
strcpy(name3,  argv[3]);
strcpy(method, argv[4]);

if(method[0] != 't'   &&
   method[0] != 'T'   &&
   method[0] != 's'   &&
   method[0] != 'S'){
   printf("\nERROR: Did not choose a valid method");
   print_side_usage();
   exit(4);
}

if(does_not_exist(name1)){
   printf("\nERROR: Input file %s does not exist",
          name1);
   print_side_usage();
   exit(2);
}

if(does_not_exist(name2)){
   printf("\nERROR: Input file %s does not exist",
          name2);
   print_side_usage();
   exit(3);
}

    /*****************************************
    *
    *   Look at the sizes of the two input
    *   files.  Ensure they are the correct
    *   dimensions and set the dimensions
    *   of the output image.
    *
    *****************************************/

get_image_size(name1, &length1, &width1);
get_image_size(name2, &length2, &width2);

if(method[0] == 'T' || method[0] == 't'){
   if(width1 != width2){
      printf("\nERROR: input images are not "
             "the same width");
      exit(4);
   }  /* ends if widths are unequal */
```

```
      else{
         width3  = width1;
         length3 = length1 + length2;
      }  /* ends else widths are ok */
   }  /* ends if method is T */

   if(method[0] == 'S' || method[0] == 's'){
      if(length1 != length2){
         printf("\nERROR: input images are not "
                "the same length");
         exit(4);
      }  /* ends if lengths are unequal */
      else{
         width3  = width1 + width2;
         length3 = length1;
      }  /* ends else lengths are ok */
   }  /* ends if method is S */

       /*******************************************
       *
       *   Create the output image to be the same
       *   type as the first input image.
       *
       *******************************************/

   if(is_a_tiff(name1)){
      read_tiff_header(name1, &tiff_file_header);
      tiff_file_header.image_length = length3;
      tiff_file_header.image_width  = width3;
      create_allocate_tiff_file(name3,
                                 &tiff_file_header);
   }

   if(is_a_bmp(name1)){
      read_bmp_file_header(name1,
                           &bmp_file_header);
      read_bm_header(name1, &bmheader);
      bmheader.height = length3;
      bmheader.width  = width3;
      create_allocate_bmp_file(name3,
                               &bmp_file_header,
                               &bmheader);
   }
```

```
    /*****************************************
    *
    *   Allocate the image arrays and read the
    *   two input images.
    *
    *****************************************/

image1 = allocate_image_array(length1, width1);
image2 = allocate_image_array(length2, width2);
image3 = allocate_image_array(length3, width3);

read_image_array(name1, image1);
read_image_array(name2, image2);

    /*****************************************
    *
    *   First do the side by side option.
    *
    *****************************************/

if(method[0] == 'S' || method[0] == 's'){

   for(i=0; i<length1; i++)
      for(j=0; j<width1; j++)
         image3[i][j] = image1[i][j];

   for(i=0; i<length2; i++)
      for(j=0; j<width2; j++)
         image3[i][j+width1] = image2[i][j];

}  /* ends if side-by-side method */


    /*****************************************
    *
    *   Now do the top to bottom option.
    *
    *****************************************/

if(method[0] == 'T' || method[0] == 't'){

   for(i=0; i<length1; i++)
      for(j=0; j<width1; j++)
         image3[i][j] = image1[i][j];
```

```
      for(i=0; i<length2; i++)
         for(j=0; j<width2; j++)
            image3[i+length1][j] = image2[i][j];

   }  /* ends top-to-bottom method */

   write_image_array(name3, image3);

   free_image_array(image1, length1);
   free_image_array(image2, length2);
   free_image_array(image3, length3);

}  /* ends main */



print_side_usage()
{
   printf(
    "\n"
    "\n usage: side in-file-1 in-file-2 "
    "out-file method"
    "\n        where method is Top-to-bottom "
    "or Side-by-side"
    "\n");
}
```

Listing 4.4 - The side Program

F.5 Code Listings for Chapter 5

```
/***********************************************
*
*       file edge.c
*
*       Functions: This file contains
*          detect_edges
*          setup_masks
*          get_edge_options
*          perform_convolution
```

```
       *          quick_edge
       *
       *       Purpose:
       *          These functions implement several
       *          types of basic edge detection.
       *
       *       External Calls:
       *          utility.c - fix_edges
       *
       *       Modifications:
       *          27 January 1991 - created
       *          27 December 1992 - Fixed an error in
       *              how I did the 8 direction edge
       *              detectors.  I was only detecting
       *              edges in the last (the 7)
       *              direction.  I fixed this by
       *              setting the out_image to the sum
       *              only if the sum was greater than
       *              the out_image.  This is in the
       *              function perform_convolution.
       *          22 April 1998 - added capability to
       *              work an entire image at one time.
       *
       ************************************************/

#include "cips.h"

short quick_mask[3][3] =  {
       {-1,  0, -1},
       { 0,  4,  0},
       {-1,  0, -1} };


   /***************************
   *
   *   Directions for the masks
   *  3 2 1
   *  4 x 0
   *  5 6 7
   *
   ***************************/

   /* masks for kirsch operator */
short kirsch_mask_0[3][3] =  {
       { 5,  5,  5},
```

```
       {-3,  0, -3},
       {-3, -3, -3} };

short kirsch_mask_1[3][3] =  {
       {-3,  5,  5},
       {-3,  0,  5},
       {-3, -3, -3} };

short kirsch_mask_2[3][3] =  {
       {-3, -3,  5},
       {-3,  0,  5},
       {-3, -3,  5} };

short kirsch_mask_3[3][3] =  {
       {-3, -3, -3},
       {-3,  0,  5},
       {-3,  5,  5} };

short kirsch_mask_4[3][3] =  {
       {-3, -3, -3},
       {-3,  0, -3},
       { 5,  5,  5} };

short kirsch_mask_5[3][3] =  {
       {-3, -3, -3},
       { 5,  0, -3},
       { 5,  5, -3} };

short kirsch_mask_6[3][3] =  {
       { 5, -3, -3},
       { 5,  0, -3},
       { 5, -3, -3} };

short kirsch_mask_7[3][3] =  {
       { 5,  5, -3},
       { 5,  0, -3},
       {-3, -3, -3} };


   /* masks for prewitt operator */
```

```
short prewitt_mask_0[3][3] =  {
       { 1,  1,  1},
       { 1, -2,  1},
       {-1, -1, -1} };

short prewitt_mask_1[3][3] =  {
       { 1,  1,  1},
       { 1, -2, -1},
       { 1, -1, -1} };

short prewitt_mask_2[3][3] =  {
       { 1,  1, -1},
       { 1, -2, -1},
       { 1,  1, -1} };

short prewitt_mask_3[3][3] =  {
       { 1, -1, -1},
       { 1, -2, -1},
       { 1,  1,  1} };

short prewitt_mask_4[3][3] =  {
       {-1, -1, -1},
       { 1, -2,  1},
       { 1,  1,  1} };

short prewitt_mask_5[3][3] =  {
       {-1, -1,  1},
       {-1, -2,  1},
       { 1,  1,  1} };

short prewitt_mask_6[3][3] =  {
       {-1,  1,  1},
       {-1, -2,  1},
       {-1,  1,  1} };

short prewitt_mask_7[3][3] =  {
       { 1,  1,  1},
       {-1, -2,  1},
       {-1, -1,  1} };
```

```
   /* masks for sobel operator */

short sobel_mask_0[3][3] =  {
       { 1,  2,  1},
       { 0,  0,  0},
       {-1, -2, -1} };

short sobel_mask_1[3][3] =  {
       { 2,  1,  0},
       { 1,  0, -1},
       { 0, -1, -2} };

short sobel_mask_2[3][3] =  {
       { 1,  0, -1},
       { 2,  0, -2},
       { 1,  0, -1} };
short sobel_mask_3[3][3] =  {
       { 0, -1, -2},
       { 1,  0, -1},
       { 2,  1,  0} };

short sobel_mask_4[3][3] =  {
       {-1, -2, -1},
       { 0,  0,  0},
       { 1,  2,  1} };

short sobel_mask_5[3][3] =  {
       {-2, -1,  0},
       {-1,  0,  1},
       { 0,  1,  2} };

short sobel_mask_6[3][3] =  {
       {-1,  0,  1},
       {-2,  0,  2},
       {-1,  0,  1} };

short sobel_mask_7[3][3] =  {
       { 0,  1,  2},
       {-1,  0,  1},
       {-2, -1,  0} };
```

```
  /**************************************************
  *
  *   detect_edges(...
  *
  *   This function detects edges in an area of one
  *   image and sends the result to another image
  *   on disk.  It reads the input image from disk,
  *   calls a convolution function, and then writes
  *   the result out to disk.  If needed, it
  *   allocates space on disk for the output image.
  *
  **************************************************/



detect_edges(the_image, out_image,
             detect_type, threshold, high,
             rows, cols, bits_per_pixel)
   int    detect_type, high, threshold;
   long   rows, cols, bits_per_pixel;
   short  **the_image, **out_image;

{
   perform_convolution(the_image, out_image,
                       detect_type, threshold,
                       rows, cols,
                       bits_per_pixel,
                       high);
   fix_edges(out_image, 1, rows, cols);
}  /* ends detect_edges */




     /*************************************************************
     *
     *   perform_convolution(...
     *
     *   This function performs convolution between the input
     *   image and 8 3x3 masks.  The result is placed in
```

```
     *   the out_image.
     *
     ********************************************************/

perform_convolution(image, out_image,
                    detect_type, threshold,
                    rows, cols, bits_per_pixel, high)
   short **image,
         **out_image;
   int   detect_type, high, threshold;
   long  rows, cols, bits_per_pixel;
{

   char response[80];
   int a,
       b,
       i,
       is_present,
       j,
       sum;

   short  mask_0[3][3],
          mask_1[3][3],
          mask_2[3][3],
          mask_3[3][3],
          mask_4[3][3],
          mask_5[3][3],
          mask_6[3][3],
          mask_7[3][3],
          max,
          min,
          new_hi,
          new_low;


   setup_masks(detect_type, mask_0, mask_1,
               mask_2, mask_3, mask_4, mask_5,
               mask_6, mask_7);

   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }
```

```
   min = 0;
   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

     /* clear output image array */
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = 0;


   printf("\n ");

   for(i=1; i<rows-1; i++){
if( (i%10) == 0){ printf("%4d", i); }
      for(j=1; j<cols-1; j++){


         /* Convolve for all 8 directions */

         /* 0 direction */

      sum = 0;
      for(a=-1; a<2; a++){
         for(b=-1; b<2; b++){
            sum = sum + image[i+a][j+b] *
                  mask_0[a+1][b+1];
         }
      }
         if(sum > max) sum = max;
         if(sum < 0)   sum = 0;
            /* Correction 12-27-92
               see file header for
               details. */
      if(sum > out_image[i][j])
         out_image[i][j]   = sum;


         /* 1 direction */

      sum = 0;
      for(a=-1; a<2; a++){
         for(b=-1; b<2; b++){
            sum = sum + image[i+a][j+b] * mask_1[a+1][b+1];
```

```
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
         details. */
if(sum > out_image[i][j])
   out_image[i][j]   = sum;


   /* 2 direction */

sum = 0;
for(a=-1; a<2; a++){
   for(b=-1; b<2; b++){
      sum = sum + image[i+a][j+b] * mask_2[a+1][b+1];
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
         details. */
if(sum > out_image[i][j])
   out_image[i][j]   = sum;


   /* 3 direction */

sum = 0;
for(a=-1; a<2; a++){
   for(b=-1; b<2; b++){
      sum = sum + image[i+a][j+b] * mask_3[a+1][b+1];
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
         details. */
if(sum > out_image[i][j])
   out_image[i][j]   = sum;
```

```
   /* 4 direction */

sum = 0;
for(a=-1; a<2; a++){
   for(b=-1; b<2; b++){
      sum = sum + image[i+a][j+b] * mask_4[a+1][b+1];
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
         details. */
if(sum > out_image[i][j])
   out_image[i][j]   = sum;


   /* 5 direction */

sum = 0;
for(a=-1; a<2; a++){
   for(b=-1; b<2; b++){
      sum = sum + image[i+a][j+b] * mask_5[a+1][b+1];
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
         details. */
if(sum > out_image[i][j])
   out_image[i][j]   = sum;


   /* 6 direction */
sum = 0;
for(a=-1; a<2; a++){
   for(b=-1; b<2; b++){
      sum = sum + image[i+a][j+b] * mask_6[a+1][b+1];
   }
}
   if(sum > max) sum = max;
   if(sum < 0)   sum = 0;
      /* Correction 12-27-92
         see file header for
```

```
            details. */
      if(sum > out_image[i][j])
         out_image[i][j]   = sum;


         /* 7 direction */

      sum = 0;
      for(a=-1; a<2; a++){
         for(b=-1; b<2; b++){
            sum = sum + image[i+a][j+b] * mask_7[a+1][b+1];
         }
      }
         if(sum > max) sum = max;
         if(sum < 0)   sum = 0;
            /* Correction 12-27-92
               see file header for
               details. */
      if(sum > out_image[i][j])
         out_image[i][j]   = sum;


      }  /* ends loop over j */
   }  /* ends loop over i */

     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

}  /* ends perform_convolution */
```

```
     /*****************************************
     *
     *   quick_edge(...
     *
     *   This function finds edges by using
     *   a single 3x3 mask.
     *
     *****************************************/


quick_edge(the_image, out_image,
           threshold, high, rows, cols, bits_per_pixel)
   int    high, threshold;
   long   rows, cols, bits_per_pixel;
   short  **the_image, **out_image;

{
   short  a, b, i, j, k,
          length, max, new_hi, new_low,
          sum, width;


   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

         /* Do convolution over image array */
   printf("\n");
   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=1; j<cols-1; j++){
         sum = 0;
         for(a=-1; a<2; a++){
            for(b=-1; b<2; b++){
               sum = sum +
                     the_image[i+a][j+b] *
```

```
                    quick_mask[a+1][b+1];
            }
        }
        if(sum < 0)   sum = 0;
        if(sum > max) sum = max;
        out_image[i][j]   = sum;

      }  /* ends loop over j */
   }  /* ends loop over i */

     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

   fix_edges(out_image, 1,
             rows-1, cols-1);

}  /* ends quick_edge */
```

```
   /***********************************************
    *
    *    setup_masks(...
    *
    *    This function copies the mask values defined
    *    at the top of this file into the mask
    *    arrays mask_0 through mask_7.
    *
    ***********************************************/
```

```
setup_masks(detect_type, mask_0, mask_1, mask_2, mask_3,
            mask_4, mask_5, mask_6, mask_7)
   int    detect_type;
   short  mask_0[3][3],
          mask_1[3][3],
          mask_2[3][3],
          mask_3[3][3],
          mask_4[3][3],
          mask_5[3][3],
          mask_6[3][3],
          mask_7[3][3];
{
   int i, j;

   if(detect_type == KIRSCH){
      for(i=0; i<3; i++){
        for(j=0; j<3; j++){
          mask_0[i][j] = kirsch_mask_0[i][j];
          mask_1[i][j] = kirsch_mask_1[i][j];
          mask_2[i][j] = kirsch_mask_2[i][j];
          mask_3[i][j] = kirsch_mask_3[i][j];
          mask_4[i][j] = kirsch_mask_4[i][j];
          mask_5[i][j] = kirsch_mask_5[i][j];
          mask_6[i][j] = kirsch_mask_6[i][j];
          mask_7[i][j] = kirsch_mask_7[i][j];
        }
      }
   }  /* ends if detect_type == KIRSCH */


   if(detect_type == PREWITT){
      for(i=0; i<3; i++){
        for(j=0; j<3; j++){
          mask_0[i][j] = prewitt_mask_0[i][j];
          mask_1[i][j] = prewitt_mask_1[i][j];
          mask_2[i][j] = prewitt_mask_2[i][j];
          mask_3[i][j] = prewitt_mask_3[i][j];
          mask_4[i][j] = prewitt_mask_4[i][j];
          mask_5[i][j] = prewitt_mask_5[i][j];
          mask_6[i][j] = prewitt_mask_6[i][j];
          mask_7[i][j] = prewitt_mask_7[i][j];
        }
```

```
      }
   }  /* ends if detect_type == PREWITT */


   if(detect_type == SOBEL){
      for(i=0; i<3; i++){
        for(j=0; j<3; j++){
          mask_0[i][j] = sobel_mask_0[i][j];
          mask_1[i][j] = sobel_mask_1[i][j];
          mask_2[i][j] = sobel_mask_2[i][j];
          mask_3[i][j] = sobel_mask_3[i][j];
          mask_4[i][j] = sobel_mask_4[i][j];
          mask_5[i][j] = sobel_mask_5[i][j];
          mask_6[i][j] = sobel_mask_6[i][j];
          mask_7[i][j] = sobel_mask_7[i][j];
        }
      }
   }  /* ends if detect_type == SOBEL */

}  /* ends setup_masks */




   /************************************************
    *
    *    fix_edges(...
    *
    *    This function fixes the edges of an image
    *    array after convolution was performed.
    *    It copies the points near the edge of the
    *    array out to the edge of the array.
    *
    ************************************************/

fix_edges(im, w, rows, cols)
      int   w;
      short **im;
      long  rows, cols;
{
   int i, j;

printf("\nFIX> rows=%ld cols=%ld w=%d",rows,cols,w);
```

```
      /* four corners */
   for(i=w; i>0; i--){
    im[i-1][i-1] = im[i][i];
    im[i-1][cols-(i-1)] = im[i][cols-1-(i-1)];
    im[rows-(i-1)][i-1] = im[rows-1-(i-1)][i];
    im[rows-(i-1)][cols-(i-1)] = im[rows-1-(i-1)][cols-1-(i-1)];
   }  /* ends four corners loop */

   for(i=0; i<rows; i++){
      for(j=w; j>0; j--){
       im[i][j-1] = im[i][j];
       im[i][cols-j] = im[i][cols-j-1];
      }
   }

   for(j=0; j<cols; j++){
      for(i=w; i>0; i--){
       im[i-1][j] = im[i][j];
       im[rows-i][j] = im[rows-i-1][j];
      }
   }

}  /* ends fix_edges */
```

Listing 5.1 - The Edge Detector Subroutines

F.6 Code Listings for Chapter 6

```
#include "cips.h"


short g7[7][7] = {
     {  0,  0, -1, -1, -1,  0,  0},
     {  0, -2, -3, -3, -3, -2,  0},
     { -1, -3,  5,  5,  5, -3, -1},
     { -1, -3,  5, 16,  5, -3, -1},
     { -1, -3,  5,  5,  5, -3, -1},
     {  0, -2, -3, -3, -3, -2,  0},
     {  0,  0, -1, -1, -1,  0,  0}};

short g9[9][9] = {
   {  0,  0,  0,  -1, -1, -1,  0,  0,  0},
```

```
   {  0, -2, -3,  -3, -3, -3, -3, -2,  0},
   {  0, -3, -2,  -1, -1, -1, -2, -3,  0},
   { -1, -3, -1,   9,  9,  9, -1, -3, -1},
   { -1, -3, -1,   9, 19,  9, -1, -3, -1},
   { -1, -3, -1,   9,  9,  9, -1, -3, -1},
   {  0, -3, -2,  -1, -1, -1, -2, -3,  0},
   {  0, -2, -3,  -3, -3, -3, -3, -2,  0},
   {  0,  0,  0,  -1, -1, -1,  0,  0,  0}};

short e_mask[3][3] = {
       {-9,  0, -9},
       { 0, 36,  0},
       {-9,  0, -9} };

short contrast[3][3] = {
   {  1,  1,  1},
   {  1,  1,  1},
   {  1,  1,  1}};


short enhance_mask[3][3] =  {
       {-1,  0, -1},
       { 0,  4,  0},
       {-1,  0, -1} };



   /**************************************************
   *
   *   homogeneity(...
   *
   *   This function performs edge detection by looking
   *   for the absence of an edge.  The center of a
   *   3x3 area is replaced by the absolute value of
   *   the max difference between the center point
   *   and its 8 neighbors.
   *
   **************************************************/


homogeneity(the_image, out_image,
            rows, cols, bits_per_pixel,
            threshold, high)
   int    high, threshold;
   short  **the_image, **out_image;
```

```
   long   rows, cols, bits_per_pixel;
{
   int a, b, absdiff, absmax, diff, i, j,
       length, max, max_diff, new_hi, new_low, width;


   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         out_image[i][j] = 0;
      }
   }

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%4d", i);
      for(j=1; j<cols-1; j++){

          max_diff = 0;
          for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){

                diff = the_image[i][j] -
                        the_image[i+a][j+b];
                absdiff = abs(diff);
                if(absdiff > max_diff)
                   max_diff = absdiff;

             }  /* ends loop over b */
          }  /* ends loop over a */

          out_image[i][j] = max_diff;
      }  /* ends loop over j */
   }  /* ends loop over i */
```

```
     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

} /* ends homogeneity */



   /****************************************************
   *
   *   difference_edge(...
   *
   *   This function performs edge detection by looking
   *   at the differences in the pixels that surround
   *   the center point of a 3x3 area.  It replaces the
   *   center point with the absolute value of the
   *   max difference of:
   *      upper left - lower right
   *      upper right - lower left
   *      left - right
   *      top - bottom
   *
   ****************************************************/

difference_edge(the_image, out_image,
                rows, cols, bits_per_pixel,
                threshold, high)
   int    high, threshold;
   short  **the_image, **out_image;
   long   rows, cols, bits_per_pixel;
{
   int a, b, absdiff, absmax, diff, i, j,
       length, max, max_diff, new_hi, new_low, width;

   new_hi  = 250;
```

```
new_low = 16;
if(bits_per_pixel == 4){
    new_hi  = 10;
    new_low = 3;
}

max = 255;
if(bits_per_pixel == 4)
   max = 16;

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      out_image[i][j] = 0;

for(i=1; i<rows-1; i++){
   if( (i%10) == 0) printf("%4d", i);
   for(j=1; j<cols-1; j++){

       max_diff = 0;
       absdiff = abs(the_image[i-1][j-1] -
                       the_image[i+1][j+1]);
       if(absdiff > max_diff) max_diff = absdiff;

       absdiff = abs(the_image[i-1][j+1] -
                       the_image[i+1][j-1]);
       if(absdiff > max_diff) max_diff = absdiff;

       absdiff = abs(the_image[i][j-1] -
                       the_image[i][j+1]);
       if(absdiff > max_diff) max_diff = absdiff;

       absdiff = abs(the_image[i-1][j] -
                       the_image[i+1][j]);
       if(absdiff > max_diff) max_diff = absdiff;


       out_image[i][j] = max_diff;

   }  /* ends loop over j */
}  /* ends loop over i */



  /* if desired, threshold the output image */
if(threshold == 1){
```

```
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

} /* ends difference_edge */




   /***************************************************
   *
   *   gaussian_edge(...
   *
   *
   ****************************************************/

gaussian_edge(the_image, out_image,
              rows, cols, bits_per_pixel,
              size, threshold, high)
   int    high, size, threshold;
   short  **the_image,
          **out_image;
   long   rows, cols, bits_per_pixel;
{
   char response[80];
   long sum;
   int  a, b, absdiff, absmax, diff, i, j,
        length, lower, max, new_hi, new_low,
        scale, starti, stopi, startj, stopj,
        upper, width;

   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
```

```
    new_low = 3;
}

max = 255;
if(bits_per_pixel == 4)
   max = 16;


if(size == 7){
   lower = -3;
   upper =  4;
   starti =  3;
   startj =  3;
   stopi  =  rows-3;
   stopj  =  cols-3;
   scale =  2;
}

if(size == 9){
   lower = -4;
   upper =  5;
   starti =  4;
   startj =  4;
   stopi  =  rows-4;
   stopj  =  cols-4;
   scale =  2;
}


for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      out_image[i][j] = 0;


for(i=starti; i<stopi; i++){
   if ( (i%10) == 0) printf(" i=%d", i);
   for(j=startj; j<stopj; j++){

   sum = 0;

   for(a=lower; a<upper; a++){
      for(b=lower; b<upper; b++){
         if(size == 7)
            sum = sum + the_image[i+a][j+b] *
                  g7[a+3][b+3];
```

```
            if(size == 9)
               sum = sum + the_image[i+a][j+b] *
                     g9[a+4][b+4];
         } /* ends loop over a */
      }  /* ends loop over b */

      if(sum < 0) sum = 0;
      if(sum > max) sum = max;
      out_image[i][j] = sum;


      }  /* ends loop over j */
   }  /* ends loop over i */

     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

} /* ends gaussian_edge */



   /**************************************************
   *
   *   contrast_edge(...
   *
   *   The edge detector uses the basic quick edge
   *   detector mask and then divides the result
   *   by a contrast smooth mask.  This implements
   *   Johnson's contrast based edge detector.
   *
   **************************************************/

contrast_edge(the_image, out_image,
              rows, cols, bits_per_pixel,
```

```
            threshold, high)
   int    high, threshold;
   short  **the_image, **out_image;
   long   rows, cols, bits_per_pixel;
{
   int ad, d;
   int a, b, absdiff, absmax, diff, i, j,
       length, max, new_hi, new_low,
       sum_d, sum_n, width;

   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = 0;

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%4d", i);
      for(j=1; j<cols-1; j++){

         sum_n = 0;
         sum_d = 0;

         for(a=-1; a<2; a++){
            for(b=-1; b<2; b++){
               sum_n = sum_n + the_image[i+a][j+b] *
                       e_mask[a+1][b+1];
               sum_d = sum_d + the_image[i+a][j+b] *
                       contrast[a+1][b+1];
            }
         }

         d = sum_d / 9;
         if(d == 0)
            d = 1;
```

```
         out_image[i][j] = sum_n/d;

         if(out_image[i][j] > max)
            out_image[i][j] = max;
         if(out_image[i][j] < 0)
            out_image[i][j] = 0;


      }  /* ends loop over j */
   }  /* ends loop over i */



     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

} /* ends contrast_edge */




     /*****************************************
     *
     *   range(..
     *
     *   This edge detector performs the
     *   range operation.
     *   It replaces the pixel at the center of a
     *   3x3, 5x5, etc. area with the max - min
     *   for that area.
     *
     *****************************************/
```

```
range(the_image, out_image,
      rows, cols, bits_per_pixel,
      size, threshold, high)
   int    high, threshold, size;
   short  **the_image,
          **out_image;
   long   rows, cols, bits_per_pixel;
{
   int    a, b, count, i, j, k,
          new_hi, new_low, length,
          sd2, sd2p1, ss, width;
   short  *elements;

   sd2   = size/2;
   sd2p1 = sd2 + 1;

      /**********************************************
      *
      *   Allocate the elements array large enough
      *   to hold size*size shorts.
      *
      **********************************************/

   ss       = size*size;
   elements = (short *) malloc(ss * sizeof(short));

   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }

      /**************************
      *
      *   Loop over image array
      *
      ***************************/

   printf("\n");
   for(i=sd2; i<rows-sd2; i++){
      if( (i%10) == 0) printf("%4d ", i);
      for(j=sd2; j<cols-sd2; j++){
         count = 0;
         for(a=-sd2; a<sd2p1; a++){
```

```
            for(b=-sd2; b<sd2p1; b++){
               elements[count] = the_image[i+a][j+b];
               count++;
            }
         }
         sort_elements(elements, &ss);
         out_image[i][j] = elements[ss-1]-elements[0];
      }  /* ends loop over j */
   }  /* ends loop over i */

     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

   free(elements);

}  /* ends range */




   /*****************************************************
   *
   *   variance(...
   *
   *   This function replaces the pixel in the center
   *   of a 3x3 area with the square root of the sum
   *   of squares of the differences between the
   *   center pixel and its eight neighbors.
   *
   *****************************************************/

variance(the_image, out_image,
         rows, cols, bits_per_pixel,
```

```
        threshold, high)
   int    high, threshold;
   short  **the_image,
          **out_image;
   long   rows, cols, bits_per_pixel;
{
   int      a, b, i, j, length,
            max, new_hi, new_low, width;
   long     diff;
   unsigned long sum, tmp;

   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
   }

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%4d", i);
      for(j=1; j<cols-1; j++){
         sum = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                if( a!=0  &&  b!=0){
                 diff = 0;
                 diff = the_image[i][j] -
                        the_image[i+a][j+b];
                 tmp = diff*diff;
                 sum = sum + tmp;
                }
             }
         }
         if(sum < 0)
            printf("\nWHAT? sum < 0, %ld ,diff=%d ", sum, diff);
         sum = sqrt(sum);
         if(sum > max) sum = max;
         out_image[i][j] = sum;
      }  /* ends loop over j */
   }  /* ends loop over i */
```

```
      /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

} /* ends variance */




     /*******************************************
     *
     * enhance_edges(...
     *
     * This function enhances the edges in an
     * input image and writes the enhanced
     * result to an output image.  It operates
     * much the same way as detect_edges
     * except it uses only one type of mask.
     *
     * The threshold and high parameters perform
     * a different role in this function.  The
     * threshold parameter does not exist.  The
     * high parameter determines if the edge is
     * strong enough to enhance or change the
     * input image.
     *
     *******************************************/


enhance_edges(the_image, out_image,
              rows, cols, bits_per_pixel, high)
   int    high;
   short  **the_image,
```

```
          **out_image;
   long   rows, cols, bits_per_pixel;

{
   int    a, b, i, j, k,
          length, max, new_hi,
          new_lo, sum, width;

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;


         /* Do convolution over image array */
   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=1; j<cols-1; j++){
         sum = 0;
         for(a=-1; a<2; a++){
            for(b=-1; b<2; b++){
               sum = sum +
                     the_image[i+a][j+b] *
                     enhance_mask[a+1][b+1];
            }
         }
         if(sum < 0)   sum = 0;
         if(sum > max) sum = max;
         if(sum > high)
            out_image[i][j] = max;
         else
            out_image[i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends enhance_edges */
```

Listing 6.1 - Edge Detectors

```
   /**********************************************
   *
```

```
   *   file medge.c
   *
   *   Functions: This file contains
   *      main
   *
   *   Purpose:
   *      This file contains the main calling
   *      routine that performs edge
   *      detection.
   *
   *   External Calls:
   *      imageio.c - create_image_file
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
   *      edge.c -
   *          detect_edges
   *          setup_masks
   *          get_edge_options
   *          perform_convolution
   *          quick_edge
   *      edge2.c -
   *          homogeneity
   *          difference_edge
   *          contrast_edge
   *          range
   *          variance
   *      edge3.c -
   *          gaussian_edge
   *          enhance_edges
   *
   *   Modifications:
   *     18 September 1998 - created to work with
   *           all I O routines in imageio.c.
   *
   ***************************************************/


#include "cips.h"


main(argc, argv)
```

```
   int argc;
   char *argv[];
{
   char  image_name[MAX_NAME_LENGTH];
   char  image_name2[MAX_NAME_LENGTH];
   char  response[MAX_NAME_LENGTH];
   int   i, j;
   int   high, size, threshold, type;
   long  bits_per_pixel, height, width;
   short **the_image, **out_image;
   struct   tiff_header_struct image_header;


      /*****************************************
      *
      *   Ensure the command line is correct.
      *
      *****************************************/

   if(argc < 4   || argc > 7){
      show_edge_usage();
      exit(0);
   }

   strcpy(image_name,  argv[2]);
   strcpy(image_name2, argv[3]);

   if(does_not_exist(image_name)){
    printf("\nERROR input file %s does not exist",
             image_name);
    exit(0);
   }

   create_image_file(image_name, image_name2);
   get_image_size(image_name, &height, &width);
   get_bitsperpixel(image_name, &bits_per_pixel);
   the_image = allocate_image_array(height, width);
   out_image = allocate_image_array(height, width);
   read_image_array(image_name, the_image);

   if(argv[1][0] == 'q' || argv[1][0] == 'Q'){
      threshold = atoi(argv[4]);
      high      = atoi(argv[5]);
      quick_edge(the_image, out_image,
                 threshold, high,
```

```
                height, width,
                bits_per_pixel);
}  /* ends if q */

if(argv[1][0] == 'b' || argv[1][0] == 'B'){
   threshold = atoi(argv[4]);
   high      = atoi(argv[5]);
   type      = atoi(argv[6]);
   perform_convolution(
                the_image, out_image,
                type, threshold,
                height, width,
                bits_per_pixel, high);
}  /* ends if b */

if(argv[1][0] == 'h' || argv[1][0] == 'H'){
   threshold = atoi(argv[4]);
   high      = atoi(argv[5]);
   homogeneity(the_image, out_image,
                height, width,
                bits_per_pixel,
                threshold, high);
}  /* ends if h */

if(argv[1][0] == 'd' || argv[1][0] == 'D'){
   threshold = atoi(argv[4]);
   high      = atoi(argv[5]);
   difference_edge(the_image, out_image,
                    height, width,
                    bits_per_pixel,
                    threshold, high);
}  /* ends if d */

if(argv[1][0] == 'c' || argv[1][0] == 'C'){
   threshold = atoi(argv[4]);
   high      = atoi(argv[5]);
   contrast_edge(the_image, out_image,
                  height, width,
                  bits_per_pixel,
                  threshold, high);
}  /* ends if c */

if(argv[1][0] == 'r' || argv[1][0] == 'R'){
   threshold = atoi(argv[4]);
   high      = atoi(argv[5]);
```

```
      size      = atoi(argv[6]);
      range(the_image, out_image,
            height, width,
            bits_per_pixel,
            size, threshold, high);
   }  /* ends if r */

   if(argv[1][0] == 'v' || argv[1][0] == 'V'){
      threshold = atoi(argv[4]);
      high      = atoi(argv[5]);
      variance(the_image, out_image,
               height, width,
               bits_per_pixel,
               threshold, high);
   }  /* ends if v */

   if(argv[1][0] == 'g' || argv[1][0] == 'G'){
      threshold = atoi(argv[4]);
      high      = atoi(argv[5]);
      size      = atoi(argv[6]);
      gaussian_edge(the_image, out_image,
               height, width,
               bits_per_pixel,
               size, threshold, high);
   }  /* ends if g */

   if(argv[1][0] == 'q' || argv[1][0] == 'Q'){
      high      = atoi(argv[4]);
      enhance_edges(the_image, out_image,
               height, width,
               bits_per_pixel, high);
   }  /* ends if q */

   write_image_array(image_name2, out_image);
   free_image_array(the_image, height);
   free_image_array(out_image, height);

}  /* ends main */


show_edge_usage()
{
printf("\nusage of medge"
"\n  Quick edge detector"
"\nmedge Q in-file out-file threshold (1/0) high"
```

```
"\n  Sobel Kirsch Prewitt edge detectors"
"\nmedge B in-file out-file threshold (1/0) high type (1,2,3)"
"\n  Homogeneity edge detector"
"\nmedge H in-file out-file threshold (1/0) high"
"\n  Difference edge detector"
"\nmedge D in-file out-file threshold (1/0) high"
"\n  Contrast edge detector"
"\nmedge C in-file out-file threshold (1/0) high"
"\n  Range edge detector"
"\nmedge R in-file out-file threshold (1/0) high size(3,5,7...)"
"\n  Variance edge detector"
"\nmedge V in-file out-file threshold (1/0) high"
"\n  Guassian edge detector"
"\nmedge G in-file out-file threshold (1/0) high size(7 or 9)"
"\n  Enhance edges"
"\nmedge E in-file out-file high "
"\n");
}  /* ends show_edge_usage */
```

Listing 6.2 - The medge Program

F.7 Code Listings for Chapter 7

```
/***********************************************
*
*    file filter.c
*
*    Functions: This file contains
*       filter_image
*       median_filter
*       setup_filters
*       get_filter_options
*       median_of
*       fsort_elements
*       fswap
*
*    Purpose:
*       These functions implement several
*       types of basic spatial frequency
*       filters.
*
```

```
    *    External Calls:
    *       utility.c - fix_edges
    *
    *    Modifications:
    *       15 February 1992 - created
    *       22 April 1998 - added capability to
    *          work an entire image at one time.
    *
    ************************************************/

#include "cips.h"


     /*******************************************
     *
     *   Define the filter masks.
     *
     *******************************************/

short lpf_filter_6[3][3] =
   { {0, 1, 0},
     {1, 2, 1},
     {0, 1, 0}};

short lpf_filter_9[3][3] =
   { {1, 1, 1},
     {1, 1, 1},
     {1, 1, 1}};

short lpf_filter_10[3][3] =
   { {1, 1, 1},
     {1, 2, 1},
     {1, 1, 1}};

short lpf_filter_16[3][3] =
   { {1, 2, 1},
     {2, 4, 2},
     {1, 2, 1}};

short lpf_filter_32[3][3] =
   { {1,  4, 1},
     {4, 12, 4},
     {1,  4, 1}};

short hpf_filter_1[3][3] =
```

```
   { { 0, -1,  0},
     {-1,  5, -1},
     { 0, -1,  0}};

short hpf_filter_2[3][3] =
   { {-1, -1, -1},
     {-1,  9, -1},
     {-1, -1, -1}};

short hpf_filter_3[3][3] =
   { { 1, -2,  1},
     {-2,  5, -2},
     { 1, -2,  1}};





     /*******************************************
     *
     *   filter_image(...
     *
     *   This function filters an image by using
     *   a single 3x3 mask.
     *
     *******************************************/


filter_image(the_image, out_image,
             rows, cols, bits_per_pixel,
             filter, type, low_high)
   int    type;
   short  filter[3][3],
          **the_image,
          **out_image;
   char   low_high[];
   long   rows, cols, bits_per_pixel;

{
   int    a, b, d, i, j, k,
          length, max, sum, width;

   setup_filters(type, low_high, filter);

   d = type;
```

```
   if(type == 2 || type == 3) d = 1;

   max = 255;
   if(bits_per_pixel == 4)
      max = 16;

         /* Do convolution over image array */
   printf("\n");
   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=1; j<cols-1; j++){
         sum = 0;
         for(a=-1; a<2; a++){
            for(b=-1; b<2; b++){
               sum = sum +
                     the_image[i+a][j+b] *
                     filter[a+1][b+1];
            }
         }
         sum               = sum/d;
         if(sum < 0)   sum = 0;
         if(sum > max) sum = max;
         out_image[i][j]   = sum;

      }  /* ends loop over j */
   }  /* ends loop over i */
   fix_edges(out_image, 1, rows-1, cols-1);
}  /* ends filter_image */




     /*******************************************
     *
     *   median_filter(..
     *
     *   This function performs a median filter
     *   on an image using a size (3x3, 5x5, etc.)
     *   specified in the call.
     *
     *******************************************/

median_filter(the_image, out_image,
              rows, cols, size)
   int    size;
```

```
   short  **the_image,
          **out_image;
   long   rows, cols;

{
   int    a, b, count, i, j, k,
          length, sd2, sd2p1, ss, width;
   short  *elements;

   sd2   = size/2;
   sd2p1 = sd2 + 1;

      /***********************************************
      *
      *   Allocate the elements array large enough
      *   to hold size*size shorts.
      *
      ***********************************************/

   ss       = size*size;
   elements = (short *) malloc(ss * sizeof(short));

      /***************************
      *
      *   Loop over image array
      *
      ****************************/

   printf("\n");
   for(i=sd2; i<rows-sd2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=sd2; j<cols-sd2; j++){
         count = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               elements[count] = the_image[i+a][j+b];
               count++;
            }
         }
         out_image[i][j] = median_of(elements, &ss);
      }  /* ends loop over j */
   }  /* ends loop over i */

   free(elements);
   fix_edges(out_image, sd2, rows-1, cols-1);
```

```
}  /* ends median_filter */


    /**********************************************
    *
    *    median_of(...
    *
    *    This function finds and returns the
    *    median value of the elements array.
    *
    *    As a side result, it also sorts the
    *    elements array.
    *
    **********************************************/

median_of(elements, count)
   int   *count;
   short elements[];
{
   short median;

   fsort_elements(elements, count);
   median = elements[*count/2];
   return(median);
}  /* ends median_of */




    /**********************************************
    *
    *    fsort_elements(...
    *
    *    This function performs a simple bubble
    *    sort on the elements from the median
    *    filter.
    *
    **********************************************/

fsort_elements(elements, count)
   int   *count;
   short elements[];
```

```
{
   int i, j;
   j = *count;
   while(j-- > 1){
      for(i=0; i<j; i++){
         if(elements[i] > elements[i+1])
            fswap(&elements[i], &elements[i+1]);
      }
   }
}  /* ends fsort_elements */
```

```
    /************************************************
    *
    *    fswap(...
    *
    *    This function swaps two shorts.
    *
    ************************************************/

fswap(a, b)
   short *a, *b;
{
   short temp;
   temp  = *a;
   *a    = *b;
   *b    = temp;
}  /* ends swap */
```

```
    /*************************************************
    *
    *    setup_filters(...
    *
    *    This function copies the filter mask
    *    values defined at the top of this file
    *    into the filter array.
    *
    *************************************************/
```

```
setup_filters(filter_type, low_high, filter)
   char   low_high[];
   int    filter_type;
   short  filter[3][3];
{
   int i, j;

   if(low_high[0] == 'l'  || low_high[0] =='L'){
      if(filter_type == 6){
         for(i=0; i<3; i++){
           for(j=0; j<3; j++){
             filter[i][j] = lpf_filter_6[i][j];
           }
         }
      }  /* ends if filter_type == 6 */

      if(filter_type == 9){
         for(i=0; i<3; i++){
           for(j=0; j<3; j++){
             filter[i][j] = lpf_filter_9[i][j];
           }
         }
      }  /* ends if filter_type == 9 */

      if(filter_type == 10){
         for(i=0; i<3; i++){
           for(j=0; j<3; j++){
             filter[i][j] = lpf_filter_10[i][j];
           }
         }
      }  /* ends if filter_type == 10 */


      if(filter_type == 16){
         for(i=0; i<3; i++){
           for(j=0; j<3; j++){
             filter[i][j] = lpf_filter_16[i][j];
           }
         }
      }  /* ends if filter_type == 16 */


      if(filter_type == 32){
```

```
        for(i=0; i<3; i++){
          for(j=0; j<3; j++){
            filter[i][j] = lpf_filter_32[i][j];
          }
        }
      }  /* ends if filter_type == 32 */
   }  /* ends low pass filter */




   if(low_high[0] == 'h'  || low_high[0] =='H'){
      if(filter_type == 1){
        for(i=0; i<3; i++){
          for(j=0; j<3; j++){
            filter[i][j] = hpf_filter_1[i][j];
          }
        }
      }  /* ends if filter_type == 1 */

      if(filter_type == 2){
        for(i=0; i<3; i++){
          for(j=0; j<3; j++){
            filter[i][j] = hpf_filter_2[i][j];
          }
        }
      }  /* ends if filter_type == 2 */

      if(filter_type == 3){
        for(i=0; i<3; i++){
          for(j=0; j<3; j++){
            filter[i][j] = hpf_filter_3[i][j];
          }
        }
      }  /* ends if filter_type == 3 */
   }  /* ends high pass filter */

}  /* ends setup_filters */
```

Listing 7.1 - Image Filter Operators

```
/**********************************************
*
*   file mfilter.c
*
*   Functions: This file contains
*      main
*
*   Purpose:
*      This is the main calling program for
*      a set of spatial filtering routines.
*
*   External Calls:
*      imageio.c - create_image_file
*                  read_image_array
*                  write_image_array
*                  get_image_size
*                  get_bitsperpixel
*                  allocate_image_array
*                  free_image_array
*      filter.c -  filter_image
*                  median_filter
*                  high_pixel
*                  low_pixel
*
*   Modifications:
*      15 February 1992 - created
*      01 January 1993 - added calls to
*          high_pixel and low_pixel.
*      18 September 1998 - modified to work with
*           all I O routines in imageio.c.
*
**********************************************/


#include "cips.h"


main(argc, argv)
   int argc;
   char *argv[];
{
   char     name1[MAX_NAME_LENGTH];
```

```
char     name2[MAX_NAME_LENGTH];
char     low_high[MAX_NAME_LENGTH];
int      i, j, size, type;
long     bits_per_pixel, length, width;
short    **the_image, **out_image, filter[3][3];


   /*****************************************
   *
   *   Ensure the command line is correct.
   *
   *****************************************/

if(argc < 5  ||  argc > 6){
 printf(
 "\nusage 1: mfilter in-image out-image g Low-High type]"
 "\n                 OR"
 "\nusage 2: mfilter in-image out-image High-Low-Median size"
 "\n                   h - high pixel"
 "\n                   l - low pixel"
 "\n                   m - median pixel"
 "\n                  "
 "\n Using the General type requires entering "
 "\n the type per the following table"
 "\n L -  6"
 "\n L -  9"
 "\n L - 10"
 "\n L - 16"
 "\n L - 32"
 "\n H -  1"
 "\n H -  2"
 "\n H -  3"
 "\n Using the High-Low-Median type requirs entering "
 "\n the size of the filtered area 3 (3x3) 5, 7, 9, etc.");
 exit(0);
}

strcpy(name1, argv[1]);
strcpy(name2, argv[2]);

if(does_not_exist(name1)){
   printf("\nERROR input file %s does not exist",
          name1);
   exit(0);
}  /* ends if does_not_exist */
```

```
   /*****************************************
   *
   *   Read the input image header, allocate
   *   the image arrays, create the output
   *   image, and read the input image.
   *
   *****************************************/

create_image_file(name1, name2);
get_image_size(name1, &length, &width);
get_bitsperpixel(name1, &bits_per_pixel);
the_image = allocate_image_array(length, width);
out_image = allocate_image_array(length, width);
read_image_array(name1, the_image);

   /*****************************************
   *
   *   Call the proper image filter function
   *   per the command line.
   *
   *****************************************/


  /* General filtering case */
if(argc == 6){
   strcpy(low_high, argv[4]);
   type = atoi(argv[5]);
   filter_image(the_image, out_image,
                length,
                width,
                bits_per_pixel,
                filter, type, low_high);
}  /* ends if argc == 6 */


  /* High, Low, and Median filtering cases */
if(argc == 5){
   strcpy(low_high, argv[3]);
   size = atoi(argv[4]);
   if(low_high[0] == 'h' || low_high[0] == 'H')
      high_pixel(the_image, out_image,
                 length,
                 width,
                 size);
```

```
      if(low_high[0] == 'l' || low_high[0] == 'L')
         low_pixel(the_image, out_image,
                   length,
                   width,
                   size);
      if(low_high[0] == 'm' || low_high[0] == 'M')
         median_filter(the_image, out_image,
                       length,
                       width,
                       size);
   }  /* ends if argc == 5 */

      /*****************************************
      *
      *   Write the output image and free the
      *   image arrays: THE END
      *
      *****************************************/

   write_image_array(name2, out_image);
   free_image_array(the_image, length);
   free_image_array(out_image, length);

}  /* ends main */
```

Listing 7.2 - The mfilter Program

F.8 Code Listings for Chapter 8

```
   /***********************************************
   *
   *       file addsub.c
   *
   *       Functions: This file contains
   *          add_image_array
   *          subtract_image_array
   *
   *       Purpose:
   *          These functions implement
   *          image addition and subtraction.
```

```
   *
   *       External Calls:
   *          none
   *
   *       Modifications:
   *          1 April 1992 - created
   *
   ************************************************/

#include "cips.h"

     /*****************************************
     *
     *   add_image_array(...
     *
     *   This function adds two image arrays.
     *   The image array out_image will hold
     *   the result.
     *
     *****************************************/

add_image_array(the_image, out_image, rows, cols, max)
   int    rows, cols;
   short  **the_image,
          **out_image,
          max;
{
   int    i, j;

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         out_image[i][j] = the_image[i][j] +
                           out_image[i][j];
         if(out_image[i][j] > max)
            out_image[i][j] = max;
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends add_image_array */




     /*****************************************
```

```
     *
     *   subtract_image_array(...
     *
     *   This function subtracts two image arrays.
     *   The image array out_image will hold
     *   the result.
     *
     *****************************************/

subtract_image_array(the_image, out_image, rows, cols)
   int    rows, cols;
   short  **the_image,
          **out_image;
{
   int    i, j, length, width;

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         out_image[i][j] = the_image[i][j] -
                           out_image[i][j];
         if(out_image[i][j] < 0)
            out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends subtract_image_array */
```

Listing 8.1 - Image Addition and Subtraction Routines

```
   /***********************************************
   *
   *   file mainas.c
   *
   *   Functions: This file contains
   *      main
   *
   *   Purpose:
   *      This file contains the main calling
   *      routine in an image addition and subtraction
   *      program.
   *
   *   External Calls:
```

```
   *      imageio.c - create_image_file
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  get_bitsperpixel
   *                  allocate_image_array
   *                  free_image_array
   *                  does_not_exist
   *                  are_not_same_size
   *      addsub.c -  add_image_array
   *                  subtract_image_array
   *
   *   Modifications:
   *      1 April 1992 - created
   *     10 August 1998 - modified to work on entire
   *         images at one time.
   *     18 September 1998 - modified to work with
   *           all I O routines in imageio.c.
   *
   ************************************************/

#include "cips.h"



main(argc, argv)
   int  argc;
   char *argv[];
{

   char     name1[80], name2[80], name3[80];
   long     bits_per_pixel, length, width;
   short    **image1, **image2;
   short    max;

       /*****************************************
       *
       *  Interpret the command line parameters.
       *
       *****************************************/

   if(argc != 5){
    printf(
     "\n"
     "\n usage: mainas in1-file in2-file "
```

```
    "out_file add-subtract"
    "\n"
    "\n   recall add-subtract a=add s=subtract\n");
   exit(0);
  }

  strcpy(name1, argv[1]);
  strcpy(name2, argv[2]);
  strcpy(name3, argv[3]);

  if(does_not_exist(name1)){
   printf("\nERROR input file %s does not exist",
            name1);
   exit(0);
  }

  if(does_not_exist(name2)){
   printf("\nERROR input file %s does not exist",
            name2);
   exit(0);
  }

      /*****************************************
      *
      *  Ensure the two input images have the
      *  same sizes.
      *
      *****************************************/

  if(are_not_same_size(name1, name2)){
     printf(
     "\nERROR Image files %s and %s are not same size",
     name1, name2);
     exit(0);
  }

      /*****************************************
      *
      *  Allocate the two image arrays
      *
      *****************************************/

  get_image_size(name1, &length, &width);
  get_bitsperpixel(name1, &bits_per_pixel);
  image1 = allocate_image_array(length, width);
```

```
   image2 = allocate_image_array(length, width);

       /*****************************************
       *
       *  Create the output file and read the
       *  two input images.
       *
       *****************************************/

   create_image_file(name1, name3);
   read_image_array(name1, image1);
   read_image_array(name2, image2);

       /*******************************************
       *
       *   Add or subtract the input images and
       *   write the result to the output image.
       *
       *******************************************/

   if(argv[4][0] == 'a' || argv[4][0] == 'A'){
      if(bits_per_pixel == 4)
         max = 16;
      else
         max = 255;
      add_image_array(image1, image2,
         length, width, max);
   }  /* ends if add */

   if(argv[4][0] == 's' || argv[4][0] == 'S')
      subtract_image_array(image1, image2,
         length, width);

   write_image_array(name3, image2);

   free_image_array(image1, length);
   free_image_array(image2, length);

}  /* ends main  */
```

Listing 8.2 - The mainas Program

```
    /***********************************************
    *
    *       file cutp.c
    *
    *       Functions: This file contains
    *          paste_image_piece
    *          check_cut_and_paste_limits
    *
    *       Purpose:
    *          These functions paste a part of one
    *          image into another image.
    *
    *       External Calls:
    *          none
    *
    *       Modifications:
    *          3 April 1992 - created
    *         12 August 1998 - modified to work
    *             with an entire image array.
    *
    ***************************************************/

#include "cips.h"


     /*******************************************
     *
     *   paste_image_piece(...
     *
     *   This function pastes a rectangular
     *   piece of an image into another image.
     *   The rectangle to be pasted into the image
     *   is described by the il1, ie1, ll1, le1
     *   parameters for the input image.
     *
     *******************************************/

paste_image_piece(the_image, out_image,
                  il1, ie1, ll1, le1,
                  il2, ie2)
   int    il1, ie1, ll1, le1, il2, ie2;
   short  **the_image,
          **out_image;
```

```
{
   int i, j, limit1, limit2;

   limit1 = ll1-il1;
   limit2 = le1-ie1;

   for(i=0; i<limit1; i++){
      for(j=0; j<limit2; j++){
         out_image[il2+i][ie2+j] = the_image[il1+i][ie1+j];
      }
   }

}  /* ends paste_image_piece */



     /*****************************************
     *
     *   check_cut_and_paste_limits(...
     *
     *   This function looks at the line and
     *   element parameters and ensures that they
     *   are not bigger than ROWS and COLS.  If
     *   they are bigger, the last element or
     *   last line parameters are reduced.
     *
     *****************************************/


check_cut_and_paste_limits(
      il1, ie1,
      ll1, le1,
      il2, ie2,
      image1_length,
      image1_width,
      image2_length,
      image2_width,
      is_ok)
   int il1, ie1, ll1, le1, il2, ie2,
       image1_length, image1_width,
       image2_length, image2_width,
       *is_ok;
{
   int result = 1;
```

```
   if(  il1 < 0 ||
        ie1 < 0){
      printf("\nCheck> il1=%d ie1=%d", il1, ie1);
      result = 0;
   }

   if(  il2 < 0 ||
        ie2 < 0){
      printf("\nCheck> il2=%d ie2=%d", il2, ie2);
      result = 0;
   }

   if(ll1 > image1_length){
      printf("\nCheck> ll1=%d length=%d",
         ll1, image1_length);
      result = 0;
   }

   if(le1 > image1_width){
      printf("\nCheck> le1=%d width=%d",
         le1, image1_width);
      result = 0;
   }

   if((il2+(ll1-il1)) > image2_length){
      printf("\nCheck> il2=%d length=%d",
         il2+(ll1-il1), image2_length);
      result = 0;
   }

   if((ie2+(le1-ie1)) > image2_width){
      printf("\nCheck> ie2=%d width=%d",
         ie2+(le1-ie1), image2_width);
      result = 0;
   }

   *is_ok = result;

}  /* ends check_cut_and_paste_limits */
```

Listing 8.3 - Cut and Paste Routines

```
    /***********************************************
    *
    *    file maincp.c
    *
    *    Functions: This file contains
    *       main
    *
    *    Purpose:
    *       This file contains the main calling
    *       routine for a program which
    *       cuts a piece from one image and pastes
    *       it into another.
    *
    *    External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *      cutp.c - paste_image_piece
    *               check_cut_and_paste_limits
    *
    *    Modifications:
    *       8 April 1992 - created
    *      12 August 1998 - modified to work on
    *           entire image array at once.
    *      18 September 1998 - modified to work with
    *            all I O routines in imageio.c.
    *
    ***********************************************/

#include "cips.h"



main(argc, argv)
   int  argc;
   char *argv[];
{

   char     name1[80], name2[80];
   int      i, is_ok, il1, ie1, ll1, le1,
            il2, ie2, ll2, le2;
   long     length1, length2, width1, width2;
```

```
short    **the_image, **out_image;

    /*****************************************
    *
    *  Interpret the command line parameters.
    *
    *****************************************/

if(argc != 9){
 printf(
  "\n"
  "\n usage: maincp in-file out_file "
  "in-il in-ie in-ll in-le out-il out-ie"
  "\n"
  "\n The image portion is pasted from the "
  "\n in-file into the out-file"
  "\n");
 exit(0);
}

strcpy(name1, argv[1]);
strcpy(name2, argv[2]);

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

if(does_not_exist(name2)){
 printf("\nERROR input file %s does not exist",
          name2);
 exit(0);
}

il1 = atoi(argv[3]);
ie1 = atoi(argv[4]);
ll1 = atoi(argv[5]);
le1 = atoi(argv[6]);
il2 = atoi(argv[7]);
ie2 = atoi(argv[8]);

   /*****************************************
   *
   *   Read the input image sizes, allocate
```

```
      *   the image array and read the image
      *   for both images.
      *
      *****************************************/

   get_image_size(name1, &length1, &width1);
   get_image_size(name2, &length2, &width2);

   the_image = allocate_image_array(length1, width1);
   out_image = allocate_image_array(length2, width2);

   read_image_array(name1, the_image);
   read_image_array(name2, out_image);


       /************************
       *
       *   Paste
       *
       *************************/

   check_cut_and_paste_limits(
      il1, ie1,
      ll1, le1,
      il2, ie2,
      length1, width1,
      length2, width2,
      &is_ok);

printf("\nMAIN> is_ok=%d", is_ok);

   if(is_ok)
      paste_image_piece(the_image, out_image,
                        il1, ie1, ll1, le1,
                        il2, ie2);

   write_image_array(name2, out_image);
   free_image_array(out_image, length2);
   free_image_array(the_image, length1);

}  /* ends main */
```

Listing 8.4 - The maincp Program

```
   /**************************************************
   *
   *       file create.c
   *
   *       Functions: This file contains
   *           main
   *
   *       Purpose:
   *          This program creates an 8 bit tiff file
   *          of size l*ROWS by w*COLS.
   *
   *       External Calls:
   *          imageio.c
   *             create_allocate_tif_file
   *             create_allocate_bmp_file
   *
   *       Modifications:
   *          7 Arpil 1992 - created
   *         15 August 1998 - modified to work with
   *              an entire image array at once.
   *         18 September 1998 - modified to work with
   *              all I O routines in imageio.c.
   *
   **************************************************/

#include "cips.h"


main(argc, argv)
   int  argc;
   char *argv[];
{
   char   *cc;
   int    l, w;
   int    ok = 0;
   struct tiff_header_struct image_header;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;

   if(argc < 4 || argc > 4){
      printf(
      "\nusage: create file-name length width\n");
      exit(-1);
```

```
   }

   l = atoi(argv[2]);
   w = atoi(argv[3]);

   cc = strstr(argv[1], ".tif");
   if(cc != NULL){  /* create a tif */
      ok = 1;
      image_header.lsb            = 1;
      image_header.bits_per_pixel = 8;
      image_header.image_length   = l;
      image_header.image_width    = w;;
      image_header.strip_offset   = 1000;
      create_allocate_tiff_file(argv[1],
                                &image_header);
   }  /* ends tif */

   cc = strstr(argv[1], ".bmp");
   if(cc != NULL){  /* create a bmp */
      ok = 1;
      bmheader.height = l;
      bmheader.width  = w;
      create_allocate_bmp_file(argv[1],
                               &bmp_file_header,
                               &bmheader);
   }  /* ends tif */

   if(ok == 0){
      printf("\nERROR input file neither tiff nor bmp");
      exit(0);
   }

}
```

Listing 8.5 - The create Program

```
   /***********************************************
   *
   *   file invert.c
   *
   *   Functions: This file contains
   *      main
```

```
   *
   *   Purpose:
   *      This program takes an image file and
   *      inverts it. It works with 8 bit images
   *      only.
   *
   *   External Calls:
   *         imageio.c
   *            create_allocate_tif_file
   *            create_allocate_bmp_file
   *            get_image_size
   *            allocate_image_array
   *            free_image_array
   *            read_image_array
   *            write_image_array
   *
   *   Modifications:
   *      6 March 1993 - created
   *      22 August 1998 - modified to work on entire
   *           images at once.
   *      19 September 1998 - modified to work with
   *           all I O routines in imageio.c.
   *
   ************************************************/

#include "cips.h"




main(argc, argv)
   int argc;
   char *argv[];
{
   char     name1[80], name2[80];
   char     *cc;
   int      a, b;
   int      ok = 0;
   long     length, width;
   short    **the_image;
   struct tiff_header_struct image_header;
   struct bmpfileheader      bmp_file_header;
   struct bitmapheader       bmheader;

   if(argc != 3){
```

```
      printf("\nusage: invert in-file out-file\n");
      exit(1);
   }

   strcpy(name1, argv[1]);
   strcpy(name2, argv[2]);

   get_image_size(name1, &length, &width);

   the_image = allocate_image_array(length, width);

   cc = strstr(argv[1], ".tif");
   if(cc != NULL){  /* create a tif */
      ok = 1;
      image_header.lsb            = 1;
      image_header.bits_per_pixel = 8;
      image_header.image_length   = length;
      image_header.image_width    = width;
      image_header.strip_offset   = 1000;
      create_allocate_tiff_file(argv[2],
                                &image_header);
   }  /* ends tif */

   cc = strstr(argv[1], ".bmp");
   if(cc != NULL){  /* create a bmp */
      ok = 1;
      bmheader.height = length;
      bmheader.width  = width;
      create_allocate_bmp_file(argv[2],
                               &bmp_file_header,
                               &bmheader);
   }  /* ends tif */

   if(ok == 0){
      printf("\nERROR input file neither tiff nor bmp");
      exit(0);
   }

   read_image_array(name1, the_image);

   for(a=0; a<length; a++)
      for(b=0; b<width; b++)
         the_image[a][b] = GRAY_LEVELS-the_image[a][b];

   write_image_array(name2, the_image);
```

```
   free_image_array(the_image, length);

}  /* ends main */
```

Listing 8.6 - The invert Program

F.9 Code Listings for Chapter 9

```
      /*********************************************
      *
      *   smooth_histogram(...
      *
      *   This function smoothes the input histogram
      *   and returns it.  It uses a simple averaging
      *   scheme where each point in the histogram
      *   is replaced by the average of itself and
      *   the two points on either side of it.
      *
      **********************************************/

smooth_histogram(histogram, gray_levels)
   int      gray_levels;
   unsigned long histogram[];
{
   int i;
   unsigned long new_hist[gray_levels];

   zero_histogram(new_hist, gray_levels);

   new_hist[0] = (histogram[0] + histogram[1])/2;
   new_hist[gray_levels] =
      (histogram[gray_levels] +
       histogram[gray_levels-1])/2;

   for(i=1; i<gray_levels-1; i++){
      new_hist[i] = (histogram[i-1] +
                     histogram[i]   +
                     histogram[i+1])/3;
```

```
   }

   for(i=0; i<gray_levels; i++)
      histogram[i] = new_hist[i];

}  /* ends smooth_histogram */
```

Listing 9.1 - The smooth_histogram function

```
   /*********************************************
   *
   *       file segment.c
   *
   *       Functions: This file contains
   *           adaptive_threshold_segmentation
   *           find_peaks
   *           find_valley_point
   *           grow
   *           insert_into_peaks
   *           insert_into_deltas
   *           label_and_check_neighbors
   *           manual_threshold_segmentation
   *           peak_threshold_segmentation
   *           peaks_high_low
   *           push
   *           pop
   *           is_not_empty
   *           threshold_image_array
   *           valley_high_low
   *           valley_threshold_segmentation
   *
   *       Purpose:
   *           These functions are part of histogram
   *           based image segmentation.
   *
   *       External Calls:
   *           none
   *
   *       Modifications:
   *           October 1992 - created
```

```
   *           15 August 1998 - modified to work on
   *                images at once.
   *
   ************************************************/


#include "cips.h"



struct stacks{
   short         x;
   short         y;
   struct stacks *next;
};

struct stacks *stack;





   /***************************************************
   *
   *   threshold_image_array(...
   *
   *   This function thresholds an input image array
   *   and produces a binary output image array.
   *   If the pixel in the input array is between
   *   the hi and low values, then it is set to value.
   *   Otherwise, it is set to 0.
   *
   ****************************************************/


threshold_image_array(in_image, out_image,
                      hi, low, value,
                      rows, cols)
   short hi, low, **in_image,
         **out_image, value;
   int   rows, cols;
{
   int   i, j;
   unsigned long counter = 0L;
```

```
   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         if(in_image[i][j] >= low  &&
            in_image[i][j] <= hi){
            out_image[i][j] = value;
            counter++;
         }
         else
            out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */
   printf("\n\tTIA> set %ld points", counter);
}  /* ends threshold_image_array */




    /**********************************************
    *
    *  grow(...
    *
    *  This function is an object detector.
    *  Its input is an binary image array
    *  containing 0's and value's.
    *  It searches through the image and connects
    *  the adjacent values.
    *
    **********************************************/

grow(binary, value, rows, cols)
   short **binary,
         value;
{
   int   first_call,
         i,
         j,
         object_found;
   short g_label,
         pop_i,
         pop_j;


            /************************************
            *
```

```
            *   Now begin the process of growing
            *   regions.
            *
            ************************************/

   g_label       = 2;
   object_found  = 0;
   first_call    = 1;

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){

         stack = NULL;

               /**********************************
               *
               *  Search for the first pixel of
               *  a region.
               *
               ***********************************/

         if(binary[i][j] == value){
/*printf("\nGROW> Hit value at %d %d number %d",i,j,g_label);*/
            label_and_check_neighbor(
                      binary,
                      g_label,
                      i, j, value,
                      &first_call,
                      rows, cols);
            object_found = 1;
         }  /* ends if binary[i]j] == value */

               /******************************
               *
               *  If the stack is not empty,
               *  pop the coordinates of
               *  the pixel off the stack
               *  and check its 8 neighbors.
               *
               *******************************/

         while(is_not_empty(stack)){
            pop(&pop_i, &pop_j);
            label_and_check_neighbor(
                        binary,
```

```
                        g_label,
                        pop_i,
                        pop_j, value,
                        &first_call,
                        rows, cols);
         }  /* ends while stack_empty == 0 */

         if(object_found == 1){
            object_found = 0;
            ++g_label;
         }  /* ends if object_found == 1 */

      }   /* ends loop over j */
   }  /* ends loop over i */

   printf("\nGROW> found %d objects", g_label);

} /* ends grow  */




   /*******************************************
   *
   *  label_and_check_neighbors(...
   *
   *  This function labels a pixel with an object
   *  label and then checks the pixel's 8
   *  neighbors.  If any of the neigbors are
   *  set, then they are also labeled.
   *
   ************************************************/

label_and_check_neighbor(binary_image,
                         g_label,
                         r, e, value,
                         first_call,
                         rows, cols)
int   cols,
      e,
      *first_call,
      r,
      rows;
short **binary_image,
```

```
      g_label,
      value;
{
char rr[80];
   int already_labeled = 0,
       i, j;
   struct stacks *temp;

   if (binary_image[r][e] == g_label)
      already_labeled = 1;

   binary_image[r][e] = g_label;

      /****************************************
      *
      *   Look at the 8 neighors of the
      *   point r,e.
      *
      *   Ensure the points you are checking
      *   are in the image, i.e. not less
      *   than zero and not greater than
      *   rows-1 or cols-1.
      *
      ****************************************/

   for(i=(r-1); i<=(r+1); i++){
      for(j=(e-1); j<=(e+1); j++){

            /********************************
            *
            *   Ensure i and j are not
            *   outside the boundary of the
            *   image.
            *
            ********************************/

         if((i>=0)   &&
            (i<=rows-1)  &&
            (j>=0)   &&
            (j<=cols-1)){

            if(binary_image[i][j] == value){
               push(i, j);
            }  /* end of if binary_image == value */
         }  /* end if i and j are on the image */
```

```
      }  /* ends loop over i rows           */
   }  /* ends loop over j columns        */
}  /* ends label_and_check_neighbors  */
```

```
   /**************************************************
   *
   *   manual_threshold_segmentation(...
   *
   *   This function segments an image using thresholding
   *   given the hi and low values of the threshold
   *   by the calling routine.  It reads in an image
   *   and writes the result to the output image.
   *
   *   If the segment parameter is 0, you only
   *   threshold the array - you do not segment.
   *
   **************************************************/

manual_threshold_segmentation(the_image, out_image,
                              hi, low, value, segment,
                              rows, cols)
   int    rows, cols, segment;
   short  hi, low, **the_image,
          **out_image, value;
{
   threshold_image_array(the_image, out_image,
            hi, low, value, rows, cols);
   if(segment == 1)
      grow(out_image, value, rows, cols);

}  /* ends manual_threshold_segmentation */
```

```
   /***************************************************
   *
   *   peak_threshold_segmentation(...
   *
   *   This function segments an image using
   *   thresholding.  It uses the histogram peaks
   *   to find the hi and low values of the
   *   threshold.
   *
   *   If the segment parameter is 0, you only
   *   threshold the array - you do not segment.
   *
   ***************************************************/

peak_threshold_segmentation(the_image, out_image,
                            value, segment,
                            rows, cols)
   int    rows, cols, segment;
   short  **the_image, **out_image, value;
{
   int      peak1, peak2;
   short    hi, low;
   unsigned long histogram[GRAY_LEVELS+1];

   zero_histogram(histogram, GRAY_LEVELS+1);
   calculate_histogram(the_image, histogram,
                       rows, cols);
   smooth_histogram(histogram, GRAY_LEVELS+1);
   find_peaks(histogram, &peak1, &peak2);
   peaks_high_low(histogram, peak1, peak2,
                  &hi, &low);
   threshold_image_array(the_image,
                         out_image,
                         hi, low, value,
                         rows, cols);
   if(segment == 1)
      grow(out_image, value, rows, cols);

}  /* ends peak_threshold_segmentation */




   /*********************************************
```

```
   *
   *   find_peaks(...
   *
   *   This function looks through the histogram
   *   array and finds the two highest peaks.
   *   The peaks must be separated, cannot be
   *   next to each other, by a spacing defined
   *   in cips.h.
   *
   *   The peaks array holds the peak value
   *   in the first place and its location in
   *   the second place.
   *
   *********************************************/

find_peaks(histogram, peak1, peak2)
   unsigned long histogram[];
   int *peak1, *peak2;
{
   int distance[PEAKS], peaks[PEAKS][2];
   int i, j=0, max=0, max_place=0;

   for(i=0; i<PEAKS; i++){
      distance[i] =  0;
      peaks[i][0] = -1;
      peaks[i][1] = -1;
   }

   for(i=0; i<=GRAY_LEVELS; i++){
      max       = histogram[i];
      max_place = i;
      insert_into_peaks(peaks, max, max_place);
   }  /* ends loop over i */

   for(i=1; i<PEAKS; i++){
      distance[i] = peaks[0][1] - peaks[i][1];
      if(distance[i] < 0)
         distance[i] = distance[i]*(-1);
   }

   *peak1 = peaks[0][1];
   for(i=PEAKS-1; i>0; i--)
    if(distance[i] > PEAK_SPACE) *peak2 = peaks[i][1];

}  /* ends find_peaks */
```

```
   /*********************************************
   *
   *   insert_into_peaks(...
   *
   *   This function takes a value and its
   *   place in the histogram and inserts them
   *   into a peaks array.  This helps us rank
   *   the the peaks in the histogram.
   *
   *   The objective is to build a list of
   *   histogram peaks and thier locations.
   *
   *   The peaks array holds the peak value
   *   in the first place and its location in
   *   the second place.
   *
   **********************************************/

insert_into_peaks(peaks, max, max_place)
   int max, max_place, peaks[PEAKS][2];
{
   int i, j;

      /* first case */
   if(max > peaks[0][0]){
      for(i=PEAKS-1; i>0; i--){
         peaks[i][0] = peaks[i-1][0];
         peaks[i][1] = peaks[i-1][1];
      }
      peaks[0][0] = max;
      peaks[0][1] = max_place;
   }  /* ends if */

      /* middle cases */
   for(j=0; j<PEAKS-3; j++){
      if(max < peaks[j][0]  && max > peaks[j+1][0]){
         for(i=PEAKS-1; i>j+1; i--){
            peaks[i][0] = peaks[i-1][0];
            peaks[i][1] = peaks[i-1][1];
         }
```

```
         peaks[j+1][0] = max;
         peaks[j+1][1] = max_place;
      }  /* ends if */
   }  /* ends loop over j */
      /* last case */
   if(max < peaks[PEAKS-2][0]  &&
      max > peaks[PEAKS-1][0]){
      peaks[PEAKS-1][0] = max;
      peaks[PEAKS-1][1] = max_place;
   }  /* ends if */

}  /* ends insert_into_peaks */




   /*********************************************
   *
   *   peaks_high_low(...
   *
   *   This function uses the histogram array
   *   and the peaks to find the best high and
   *   low threshold values for the threshold
   *   function.  You want the hi and low values
   *   so that you will threshold the image around
   *   the smaller of the two "humps" in the
   *   histogram.  This is because the smaller
   *   hump represents the objects while the
   *   larger hump represents the background.
   *
   *********************************************/

peaks_high_low(histogram, peak1, peak2, hi, low)
   int  peak1, peak2;
   short *hi, *low;
   unsigned long histogram[];
{
   int i, mid_point;
   unsigned long sum1 = 0, sum2 = 0;

   if(peak1 > peak2)
      mid_point = ((peak1 - peak2)/2) + peak2;
   if(peak1 < peak2)
      mid_point = ((peak2 - peak1)/2) + peak1;
```

```
   for(i=0; i<mid_point; i++)
      sum1 = sum1 + histogram[i];

   for(i=mid_point; i<=GRAY_LEVELS; i++)
      sum2 = sum2 + histogram[i];
   if(sum1 >= sum2){
      *low = mid_point;
      *hi  = GRAY_LEVELS;
   }
   else{
      *low = 0;
      *hi  = mid_point;
   }

}  /* ends peaks_high_low */




   /************************************************
   *
   *   valley_threshold_segmentation(...
   *
   *   This function segments an image using
   *   thresholding.  It uses the histogram valleys
   *   to find the hi and low values of the
   *   threshold.
   *
   *   If the segment parameter is 0, you only
   *   threshold the array - you do not segment.
   *
   ************************************************/

valley_threshold_segmentation(the_image, out_image,
                              value, segment,
                              rows, cols)
   int    rows, cols, segment;
   short  **the_image,
          **out_image, value;
{
   int      peak1, peak2;
   short    hi, low;
```

```
   unsigned long histogram[GRAY_LEVELS+1];

   zero_histogram(histogram, GRAY_LEVELS+1);
   calculate_histogram(the_image, histogram, rows, cols);
   smooth_histogram(histogram, GRAY_LEVELS+1);
   find_peaks(histogram, &peak1, &peak2);
   valley_high_low(histogram, peak1, peak2,
                   &hi, &low);
   threshold_image_array(the_image, out_image,
                         hi, low, value, rows, cols);
   if(segment == 1)
      grow(out_image, value, rows, cols);

}  /* ends valley_threshold_segmentation */




      /*********************************************
      *
      *   valley_high_low(...
      *
      *   This function uses the histogram array
      *   and the valleys to find the best high and
      *   low threshold values for the threshold
      *   function.  You want the hi and low values
      *   so that you will threshold the image around
      *   the smaller of the two "humps" in the
      *   histogram.  This is because the smaller
      *   hump represents the objects while the
      *   larger hump represents the background.
      *
      *********************************************/

valley_high_low(histogram, peak1, peak2, hi, low)
   int  peak1, peak2;
   short *hi, *low;
   unsigned long histogram[];
{
   int  i, valley_point;
   unsigned long sum1 = 0, sum2 = 0;
```

```
   find_valley_point(histogram, peak1, peak2,
                     &valley_point);
   /*printf("\nVHL> valley point is %d",
            valley_point);*/

   for(i=0; i<valley_point; i++)
      sum1 = sum1 + histogram[i];
   for(i=valley_point; i<=GRAY_LEVELS; i++)
      sum2 = sum2 + histogram[i];

   if(sum1 >= sum2){
      *low = valley_point;
      *hi  = GRAY_LEVELS;
   }
   else{
      *low = 0;
      *hi  = valley_point;
   }

}  /* ends valley_high_low */




   /*********************************************
   *
   *   find_valley_point(...
   *
   *   This function finds the low point of
   *   the valley between two peaks in a
   *   histogram.  It starts at the lowest
   *   peak and works its way up to the
   *   highest peak.  Along the way, it looks
   *   at each point in the histogram and inserts
   *   them into a list of points.  When done,
   *   it has the location of the smallest histogram
   *   point - that is the valley point.
   *
   *   The deltas array holds the delta value
   *   in the first place and its location in
   *   the second place.
   *
   *********************************************/
```

```
find_valley_point(histogram, peak1,
                  peak2, valley_point)
   int  peak1, peak2, *valley_point;
   unsigned long histogram[];
{
   int  deltas[PEAKS][2], delta_hist, i;

   for(i=0; i<PEAKS; i++){
      deltas[i][0] = 10000;
      deltas[i][1] =    -1;
   }

   if(peak1 < peak2){
      for(i=peak1+1; i<peak2; i++){
         delta_hist = (int)(histogram[i]);
         insert_into_deltas(deltas, delta_hist, i);
      }  /* ends loop over i */
   }  /* ends if peak1 < peak2 */

   if(peak2 < peak1){
      for(i=peak2+1; i<peak1; i++){
         delta_hist = (int)(histogram[i]);
         insert_into_deltas(deltas, delta_hist, i);
      }  /* ends loop over i */
   }  /* ends if peak2 < peak1 */

   *valley_point = deltas[0][1];

}  /* ends find_valley_point */
```

```
   /*******************************************
   *
   *   insert_into_deltas(...
   *
   *   This function inserts histogram deltas
   *   into a deltas array.  The smallest delta
   *   will be at the top of the array.
   *
   *   The objective is to build a list of
   *   histogram area deltas and thier locations.
```

```
   *
   *   The deltas array holds the delta value
   *   in the first place and its location in
   *   the second place.
   *
   ********************************************/

insert_into_deltas(deltas, value, place)
   int value, place, deltas[PEAKS][2];
{
   int i, j;

      /* first case */
   if(value < deltas[0][0]){
      for(i=PEAKS-1; i>0; i--){
         deltas[i][0] = deltas[i-1][0];
         deltas[i][1] = deltas[i-1][1];
      }
      deltas[0][0] = value;
      deltas[0][1] = place;
   }  /* ends if */

      /* middle cases */
   for(j=0; j<PEAKS-3; j++){
      if(value > deltas[j][0]  &&
         value < deltas[j+1][0]){
         for(i=PEAKS-1; i>j+1; i--){
            deltas[i][0] = deltas[i-1][0];
            deltas[i][1] = deltas[i-1][1];
         }
         deltas[j+1][0] = value;
         deltas[j+1][1] = place;
      }  /* ends if */
   }  /* ends loop over j */

      /* last case */
   if(value > deltas[PEAKS-2][0]  &&
      value < deltas[PEAKS-1][0]){
      deltas[PEAKS-1][0] = value;
      deltas[PEAKS-1][1] = place;
   }  /* ends if */

}  /* ends insert_into_deltas */
```

```
   /************************************************
   *
   *   adaptive_threshold_segmentation(...
   *
   *   This function segments an image using
   *   thresholding.  It uses two passes
   *   to find the hi and low values of the
   *   threshold.  The first pass uses the peaks
   *   of the histogram to find the hi and low
   *   threshold values.  It thresholds the image
   *   using these hi lows and calculates the means
   *   of the object and background.  Then we use
   *   these means as new peaks to calculate new
   *   hi and low values.  Finally, we threshold
   *   the image again using these second hi low
   *   hi low values.
   *
   *   If the segment parameter is 0, you only
   *   threshold the array - you do not segment.
   *
   ************************************************/

adaptive_threshold_segmentation(the_image, out_image,
                                value, segment,
                                rows, cols)
   int    rows, cols, segment;
   short  **the_image,
          **out_image, value;
{
   int      peak1, peak2;
   short    background, hi, low, object;
   unsigned long histogram[GRAY_LEVELS+1];

   zero_histogram(histogram, GRAY_LEVELS+1);
   calculate_histogram(the_image, histogram,
                       rows, cols);
   smooth_histogram(histogram, GRAY_LEVELS+1);
   find_peaks(histogram, &peak1, &peak2);
   peaks_high_low(histogram, peak1, peak2,
                  &hi, &low);
   threshold_and_find_means(the_image, out_image,
                            hi, low, value,
                            &object, &background,
```

```
                            rows, cols);
   peaks_high_low(histogram, object, background,
                  &hi, &low);
   threshold_image_array(the_image, out_image,
                         hi, low, value,
                         rows, cols);
   if(segment == 1)
      grow(out_image, value, rows, cols);

}  /* ends adaptive_threshold_segmentation */




   /**************************************************
   *
   *   threshold_and_find_means(...
   *
   *   This function thresholds an input image array
   *   and produces a binary output image array.
   *   If the pixel in the input array is between
   *   the hi and low values, then it is set to value.
   *   Otherwise, it is set to 0.
   *
   **************************************************/

threshold_and_find_means(in_image, out_image, hi,
                         low, value, object_mean,
                         background_mean,
                         rows, cols)
   short *background_mean, hi, low,
         **in_image, *object_mean,
         **out_image, value;
   int   rows, cols;
{
   int      counter = 0,
            i,
            j;
   unsigned long object     = 0,
                 background = 0;

   for(i=0; i<rows; i++){
      for(j=0; j<cols; j++){
         if(in_image[i][j] >= low  &&
```

```
            in_image[i][j] <= hi){
            out_image[i][j] = value;
            counter++;
            object = object + in_image[i][j];
         }
         else{
            out_image[i][j] = 0;
            background      = background + in_image[i][j];
         }
      }  /* ends loop over j */
   }  /* ends loop over i */
   object     = object/counter;
   background = background/((rows*cols)-counter);
   *object_mean     = (short)(object);
   *background_mean = (short)(background);
   printf("\n\tTAFM> set %d points", counter);
   printf("\n\tTAFM> object=%d background=%d",
          *object_mean, *background_mean);
}  /* ends threshold_and_find_means */
```

```
show_stack()
{
char r[80];
   struct stacks *temp;
   temp = stack;
   while(temp != NULL){
      printf("\n\t\t\t\t%d %d %x %x",
      temp->x,temp->y, temp, temp->next);
      temp = temp->next;
   }
}
```

```
int is_not_empty(pointer)
   struct stacks *pointer;
{
   int result = 0;
   if(pointer != NULL)
```

```
      result = 1;
   return(result);

}  /* ends is_empty */
```

```
push(x, y)
   short  x, y;
{
   char r[80];
   struct stacks *new_one;

   new_one = (struct stacks *)
             calloc(1, sizeof(struct stacks ));
   new_one->next = stack;
   new_one->x    = x;
   new_one->y    = y;
   stack         = new_one;
}  /* ends push */
```

```
pop(x, y)
   short  *x, *y;
{
   struct stacks *temp;

   temp    = stack;
   *x      = stack->x;
   *y      = stack->y;
   stack   = stack->next;
   free(temp);
}  /* ends pop */
```

Listing 9.2 - The Segmentation Subroutines

```
    /***********************************************
    *
    *   file mainseg.c
    *
    *   Functions: This file contains
    *      main
    *
    *   Purpose:
    *      This file contains the main calling
    *      routine in a segmentation and related
    *      operations program.
    *
    *   External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *      segment.c - threshold_image_array
    *                  grow
    *                  find_peaks
    *                  peaks_high_low
    *                  valley_high_low
    *                  threshold_and_find_means
    *
    *   Modifications:
    *      27 September 1992 - created
    *      15 August 1998 - modified to work on entire
    *           images at once.
    *      19 September 1998 - modified to work with
    *            all I O routines in imageio.c.
    *
    ***********************************************/

#include "cips.h"


main(argc, argv)
   int argc;
   char *argv[];
{
   char   name1[80], name2[80], response[80];
```

```
int    segment;
long   length, width;
short  background, hi, low, object, value;
short  **the_image, **out_image;

   /*****************************************
   *
   *   Ensure the command line is correct.
   *
   *****************************************/

if(argc < 8){
 printf(
  "\n\nmainseg in-file out-file hi low "
  "value operation segment"
  "\n"
  "\n\thi and low are thresholds"
  "\n\tvalue is output pixel value"
  "\n\toperation = Threshold Grow Peaks"
  " Valleys Adaptive"
  "\n\tsegment is 1 (perform segmentation) "
  "or 0 (don't)");
 printf("\n");
 exit(0);
}

strcpy(name1, argv[1]);
strcpy(name2, argv[2]);
hi      = atoi(argv[3]);
low     = atoi(argv[4]);
value   = atoi(argv[5]);
segment = atoi(argv[7]);

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

   /*****************************************
   *
   *   Read the input image header, allocate
   *   the image arrays, create the output
   *   image, and read the input image.
   *
```

```
    ****************************************/


 get_image_size(name1, &length, &width);

 the_image = allocate_image_array(length, width);
 out_image = allocate_image_array(length, width);

 create_file_if_needed(name1, name2, out_image);
 read_image_array(name1, the_image);

    /*********************************
    *
    *   Manual Threshold operation
    *
    *********************************/

if(argv[6][0] == 't' || argv[6][0] == 'T'){
   manual_threshold_segmentation(
      the_image, out_image,
      hi, low, value, segment,
      length, width);
   write_image_array(name2, out_image);
}  /* ends if t */


    /*********************************
    *
    *   Grow region operation
    *
    *********************************/

if(argv[6][0] == 'g' || argv[6][0] == 'G'){
   grow(the_image, value,
        length, width);
   write_image_array(name2, the_image);
}  /* ends if g */


    /*********************************
    *
    *   Peak threshold operation
    *
    *********************************/
```

```
   if(argv[6][0] == 'p' || argv[6][0] == 'P'){
      peak_threshold_segmentation(
         the_image, out_image,
         value, segment,
         length, width);
     write_image_array(name2, out_image);
   }  /* ends if p */



      /**********************************
      *
      *   Valley threshold operation
      *
      **********************************/

   if(argv[6][0] == 'v' || argv[6][0] == 'V'){
      valley_threshold_segmentation(
         the_image, out_image,
         value, segment,
         length, width);
     write_image_array(name2, out_image);
   }  /* ends if v */

      /**********************************
      *
      *   Adaptive threshold operation
      *
      **********************************/

   if(argv[6][0] == 'a' || argv[6][0] == 'A'){
      adaptive_threshold_segmentation(
         the_image, out_image,
         value, segment,
         length, width);
     write_image_array(name2, out_image);
   }  /* ends if a */

   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main */
```

Listing 9.3 - The mainseg Program

F.10 Code Listings for Chapter 10

```
     /*****************************************
     *
     *   low_pixel(..
     *
     *   This function replaces the pixel at
     *   the center of a 3x3, 5x5, etc. area
     *   with the min for that area.
     *
     *****************************************/


low_pixel(the_image, out_image,
          rows, cols, size)
   int    size;
   short  **the_image,
          **out_image;
   long   rows, cols;

{
   int    a, b, count, i, j, k,
          length, sd2, sd2p1, ss, width;
   short  *elements;

   sd2   = size/2;
   sd2p1 = sd2 + 1;

      /*********************************************
      *
      *   Allocate the elements array large enough
      *   to hold size*size shorts.
      *
      *********************************************/

   ss       = size*size;
   elements = (short *) malloc(ss * sizeof(short));

      /***************************
      *
      *   Loop over image array
      *
      ***************************/
```

```
   printf("\n");
   for(i=sd2; i<rows-sd2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=sd2; j<cols-sd2; j++){
         count = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               elements[count] = the_image[i+a][j+b];
               count++;
            }
         }
         fsort_elements(elements, &ss);
         out_image[i][j] = elements[0];
      }  /* ends loop over j */
   }  /* ends loop over i */

   free(elements);
   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends low_pixel */



     /*********************************************
     *
     *   high_pixel(..
     *
     *   This function replaces the pixel at
     *   the center of a 3x3, 5x5, etc. area
     *   with the max for that area.
     *
     *********************************************/

high_pixel(the_image, out_image,
           rows, cols, size)
   int    size;
   short  **the_image,
          **out_image;
   long   rows, cols;
{
   int    a, b, count, i, j, k,
          length, sd2, sd2p1, ss, width;
   short  *elements;
```

```
   sd2   = size/2;
   sd2p1 = sd2 + 1;

      /**********************************************
      *
      *   Allocate the elements array large enough
      *   to hold size*size shorts.
      *
      **********************************************/

   ss       = size*size;
   elements = (short *) malloc(ss * sizeof(short));

      /***************************
      *
      *   Loop over image array
      *
      ***************************/

   printf("\n");
   for(i=sd2; i<rows-sd2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=sd2; j<cols-sd2; j++){
         count = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               elements[count] = the_image[i+a][j+b];
               count++;
            }
         }
         fsort_elements(elements, &ss);
         out_image[i][j] = elements[ss-1];
      }  /* ends loop over j */
   }  /* ends loop over i */

   free(elements);
   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends high_pixel */
```

Listing 10.1 - The high_pixel and low_pixel Subroutines

```
/***********************************************
*
*       file segment2.c
*
*       Functions: This file contains
*          find_cutoff_point
*          edge_region
*          gray_shade_region
*          edge_gray_shade_region
*          pixel_grow
*          pixel_label_and_check_neighbors
*          is_close
*          erode_image_array
*          get_edge_region_options
*
*       Purpose:
*          These function implement the three
*          segmentation techniques in Image
*          Processing part 10.
*
*       External Calls:
*          edges.c - quick_edge
*                    homogeneity
*                    difference_edge
*                    contrast_edge
*                    gaussian_edge
*                    range
*                    variance
*                    detect_edges
*          hist.c - calculate_histogram
*                   zero_histogram
*          thresh.c - threshold_image_array
*
*       Modifications:
*          5 December 1992 - created
*         15 August 1998 - modified to work on entire
*              images at once.
*
***********************************************/

#include "cips.h"
```

```
struct stacksp{
   short         x;
   short         y;
   struct stacksp *next;
};

struct stacksp *stackp;



     /*******************************************
     *
     *   find_cutoff_point(..
     *
     *   This function looks at a histogram
     *   and sets a cuttoff point at a given
     *   percentage of pixels.
     *   For example, if percent=0.6, you
     *   start at 0 in the histogram and count
     *   up until you've hit 60% of the pixels.
     *   Then you stop and return that pixel
     *   value.
     *
     *******************************************/

find_cutoff_point(histogram, percent,
                  cutoff, rows, cols)
   unsigned long histogram[];
   float    percent;
   long     cols, rows;
   short    *cutoff;
{
   float  fd, fsum, sum_div;
   int    i, looking;
   long   lc, lr, num=0, sum=0;

   sum     = 0;
   i       = 0;
   lr      = (long)(rows);
   lc      = (long)(cols);
   num     = lr*lc;
   fd      = (float)(num);

   while(looking){
```

```
      fsum    = (float)(sum);
      sum_div = fsum/fd;
      if(sum_div >= percent)
         looking = 0;
      else
         sum = sum + histogram[i++];
   }  /* ends while looking */

   if(i >= (GRAY_LEVELS+1)) i = GRAY_LEVELS;
   *cutoff = i;
   printf("\nCutoff is %d sum=%ld", *cutoff, sum);
}  /* ends find_cutoff_point */




     /*****************************************
     *
     *   erode_image_array(..
     *
     *   This function erodes pixels.  If a pixel
     *   equals value and has more than threshold
     *   neighbors equal to 0, then set that
     *   pixel in the output to 0.
     *
     *****************************************/


erode_image_array(the_image, out_image,
                  value, threshold,
                  rows, cols)
   short  **the_image,
          **out_image,
          threshold,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k,
          length, width;

      /***************************
      *
      *   Loop over image array
      *
      ****************************/
```

```
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   printf("\n");

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%3d", i);
      for(j=1; j<cols-1; j++){
         if(the_image[i][j] == value){
            count = 0;
            for(a=-1; a<=1; a++){
                for(b=-1; b<=1; b++){
                      if(the_image[i+a][j+b] == 0)
                         count++;
                }  /*  ends loop over b */
            }  /* ends loop over a */
            if(count > threshold) out_image[i][j] = 0;
         }  /* ends if the_image == value */
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends erode_image_array */



    /***********************************************
    *
    *   pixel_grow(...
    *
    *   The function grows regions.  It is similar
    *   to the grow function in segment.c, but it
    *   has several new capabilities.  It can
    *   eliminate regions if they are too large or
    *   too small.
    *
    *   It ignores pixels = FORGET_IT.  This allows
    *   it to ignore edges or regions already
    *   eliminated from consideration.
    *
    *   It adds pixels to a growing region only if
    *   the pixel is close enough to the average gray
    *   level of that region.
```

```
    *
    ***********************************************/

pixel_grow(input, output, diff,
           min_area, max_area,
           rows, cols)
   long  cols, rows;
   short **input,
         **output,
         max_area,
         min_area,
         diff;
{
   char name[80];

   int count,
       first_call,
       i,
       ii,
       j,
       jj,
       object_found;

   short g_label, target, pop_i, pop_j, sum;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         output[i][j] = 0;

   g_label       = 2;
   object_found  = 0;
   first_call    = 1;

            /*************************************
            *
            *   Now begin the process of growing
            *   regions.
            *
            *************************************/

   for(i=0; i<rows; i++){
if( (i%4) == 0) printf("\n");
printf("-i=%3d label=%3d", i, g_label);
      for(j=0; j<cols; j++){
```

```
target            = input[i][j];
sum               = target;
count             = 0;
/*********************************
stack_file_in_use = 0;
stack_empty       = 1;
pointer           = -1;
******************************/
stackp = NULL;

      /***********************************
      *
      *  Search for the first pixel of
      *  a region.  It must not equal
      *  FORGET_IT, and it must be close
      *  enough to the target (ave value).
      *
      ************************************/

if(input[i][j] != FORGET_IT            &&
   is_close(input[i][j], target, diff) &&
   output[i][j] == 0){
   pixel_label_and_check_neighbor(input,
                  output, &target, &sum,
                  &count, g_label,
                  i, j, diff,
                  &first_call,
                  rows, cols);
   object_found = 1;
}  /* ends if is_close */

      /*****************************
      *
      *  If the stack is not empty,
      *  pop the coordinates of
      *  the pixel off the stack
      *  and check its 8 neighbors.
      *
      *******************************/

while(is_not_emptyp(stackp)){
   popp(&pop_i, &pop_j);


   pixel_label_and_check_neighbor(input,
```

```
                        output, &target, &sum,
                        &count, g_label,
                        pop_i, pop_j,
                        diff,
                        &first_call,
                        rows, cols);
         }  /* ends while stack_empty == 0 */

         if(object_found == 1){
            object_found = 0;

                  /**********************************
                  *
                  *  The object must be in the
                  *  size constraints given by
                  *  min_area and max_area
                  *
                  **********************************/

            if(count >= min_area  &&
               count <= max_area)
               ++g_label;
                  /**********************************
                  *
                  *   Remove the object from the
                  *   output.  Set all pixels in the
                  *   object you are removing to
                  *   FORGET_IT.
                  *
                  **********************************/

            else{
               for(ii=0; ii<rows; ii++){
                  for(jj=0; jj<cols; jj++){
                     if(output[ii][jj] == g_label){
                        output[ii][jj] = 0;
                        input[ii][jj]  = FORGET_IT;
                     }  /* ends if output == g_label */
                  }  /* ends loop over jj */
               }  /* ends loop over ii */
            }  /* ends else remove object */
         }  /* ends if object_found == 1 */

      }   /* ends loop over j */
   }  /* ends loop over i */
```

```
   printf("\nGROW> found %d objects", g_label);

} /* ends pixel_grow  */




   /*********************************************
   *
   *  pixel_label_and_check_neighbors(...
   *
   *  This function labels a pixel with an object
   *  label and then checks the pixel's 8
   *  neighbors.  If any of the neigbors are
   *  set, then they are also labeled.
   *
   *  It also updates the target or ave pixel
   *  value of the pixels in the region being
   *  grown.
   *
   **********************************************/

pixel_label_and_check_neighbor(input_image,
                          output_image, target,
                          sum, count,
                          g_label,
                          r, e, diff,
                          first_call,
                          rows, cols)
int   *count,
      e,
      *first_call,
      r;
long  cols, rows;
short **input_image,
      **output_image,
      g_label,
      *sum,
      *target,
      diff;
{
char response[80];
   int already_labeled = 0,
```

```
       i, j;

/**printf("\nDEBUG>placn> start rx=%d ey=%d",r,e);**/
   if (output_image[r][e] != 0)
      already_labeled = 1;

   output_image[r][e] = g_label;
   *count  = *count + 1;
   if(*count > 1){
      *sum    = *sum + input_image[r][e];
      *target = *sum / *count;
   }

      /****************************************
      *
      *   Look at the 8 neighors of the
      *   point r,e.
      *
      *   Ensure the points are close enough
      *   to the target and do not equal
      *   FORGET_IT.
      *
      *   Ensure the points you are checking
      *   are in the image, i.e. not less
      *   than zero and not greater than
      *   rows-1 or cols-1.
      *
      ****************************************/

   for(i=(r-1); i<=(r+1); i++){
      for(j=(e-1); j<=(e+1); j++){

         if((i>=0)       &&  /* ensure point in in image */
            (i<=rows-1)  &&
            (j>=0)       &&
            (j<=cols-1)){

            if( input_image[i][j] != FORGET_IT   &&
                is_close(input_image[i][j],
                            *target, diff)       &&
                output_image[i][j] == 0){

               pushp(i, j);

            }  /* ends if is_close */
```

```
         }  /* end if i and j are on the image */
      }  /* ends loop over i rows           */
   }  /* ends loop over j columns        */
}  /* ends pixel_label_and_check_neighbors  */



   /*******************************************
   *
   *  is_close(...
   *
   *  This function tests to see if two pixel
   *  values are close enough together.  It
   *  uses the delta parameter to make this
   *  judgement.
   *
   ************************************************/

is_close(a, b, delta)
   short a, b, delta;
{
   int   result = 0;
   short diff;

   diff = a-b;
   if(diff < 0) diff = diff*(-1);
   if(diff < delta)
      result = 1;
   return(result);
}  /* ends is_close */



     /*******************************************
     *
     *   edge_region(..
     *
     *   This function segments an image by
     *   growing regions inside of edges.
     *   The steps are:
     *      . detect edges
     *      . threshold edge output to a
     *        percent value
     *      . remove edges from consideration
```

```
     *      . grow regions
     *
     *****************************************/


edge_region(the_image, out_image,
            edge_type, min_area,
            max_area, diff, percent, set_value,
            erode, rows, cols, bits_per_pixel)
   float    percent;
   int      edge_type;
   long     bits_per_pixel, cols, rows;
   short    diff, erode,
            max_area, min_area,
            set_value,
            **the_image,
            **out_image;
{

   int    a, b, count, i, j, k,
          length, width;
   short  cutoff;
   unsigned long histogram[GRAY_LEVELS+1];

      /***************************
      *
      *   Detect the edges.  Do
      *   not threshold.
      *
      ***************************/

   if(edge_type == 1  ||
      edge_type == 2  ||
      edge_type == 3)
      detect_edges(the_image, out_image,
                   edge_type, 0, 0,
                   rows, cols,
                   bits_per_pixel);

   if(edge_type == 4){
      quick_edge(the_image, out_image,
                 0, 0,
                 rows, cols,
                 bits_per_pixel);
   }  /* ends if 4 */
```

```
   if(edge_type == 5){
      homogeneity(the_image, out_image,
                  rows, cols,
                  bits_per_pixel,
                  0, 0);
   }  /* ends if 5 */

   if(edge_type == 6){
      difference_edge(the_image, out_image,
                      rows, cols,
                      bits_per_pixel,
                      0, 0);
   }  /* ends if 6 */

   if(edge_type == 7){
      contrast_edge(the_image, out_image,
                    rows, cols,
                    bits_per_pixel,
                    0, 0);
   }  /* ends if 7 */

   if(edge_type == 8){
      gaussian_edge(the_image, out_image,
                    rows, cols,
                    bits_per_pixel,
                    3, 0, 0);
   }  /* ends if 8 */

   if(edge_type == 10){
      range(the_image, out_image,
            rows, cols,
            bits_per_pixel,
            3, 0, 0);
   }  /* ends if 10 */

   if(edge_type == 11){
      variance(the_image, out_image,
               rows, cols,
               bits_per_pixel,
               0, 0);
   }  /* ends if 11 */

/**write_array_into_tiff_image("f:e1.tif", out_image,
il, ie, ll, le);**/
```

```
   /* copy out_image to the_image */
for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];

   /******************************
   *
   *   Threshold the edge detector
   *   output at a given percent.
   *   This eliminates the weak
   *   edges.
   *
   *******************************/
zero_histogram(histogram, GRAY_LEVELS+1);
calculate_histogram(the_image, histogram,
                    rows, cols);
find_cutoff_point(histogram, percent, &cutoff,
                  rows, cols);
threshold_image_array(the_image, out_image,
                      GRAY_LEVELS, cutoff,
                      set_value, rows, cols);

if(erode != 0){
      /* copy out_image to the_image */
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] = out_image[i][j];
   erode_image_array(the_image, out_image,
                     set_value, erode,
                     rows, cols);
}  /* ends if erode */


   /******************************
   *
   *   Set all the edge values to
   *   FORGET_IT so the region
   *   growing will not use those
   *   points.
   *
   *******************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
```

```
         if(out_image[i][j] == set_value)
            out_image[i][j] = FORGET_IT;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] = out_image[i][j];

   pixel_grow(the_image, out_image, diff,
              min_area, max_area,
              rows, cols);

}  /* ends edge_region */



     /*****************************************
     *
     *   gray_shade_region(...
     *
     *   This function segments an image by
     *   growing regions based only on gray
     *   shade.
     *
     *****************************************/


gray_shade_region(the_image, out_image,
                  diff, min_area, max_area,
                  rows, cols)
   long   cols, rows;
   short  **the_image,
          **out_image,
          diff, min_area, max_area;
{
   int    a, b, big_count, count, i, j, k, l,
          not_finished, length, width;
printf("\nDEBUG> GSR> before calling pixel grow");
   pixel_grow(the_image, out_image, diff,
              min_area, max_area,
              rows, cols);
printf("\nDEBUG> GSR> after calling pixel grow");

}  /* ends gray_shade_region */
```

```
     /*****************************************
     *
     *   edge_gray_shade_region(..
     *
     *   This function segments an image by
     *   growing gray shade regions inside of
     *   edges.  It combines the techniques
     *   of the edge_region and gray_shade_region
     *   functions.
     *
     *   The steps are:
     *      . detect edges
     *      . threshold edge output to a
     *        percent value
     *      . lay the edges on top of the original
     *        image to eliminate them from
     *        consideration
     *      . grow regions
     *
     *****************************************/

edge_gray_shade_region(in_name, the_image,
            out_image, edge_type,
            min_area, max_area, diff, percent,
            set_value, erode,
            rows, cols, bits_per_pixel)
   char     in_name[];
   float    percent;
   int      edge_type;
   long     bits_per_pixel, cols, rows;
   short    diff, erode,
            max_area, min_area,
            set_value,
            **the_image,
            **out_image;
{
   int    a, b, count, i, j, k,
          length, width;
   short  cutoff;
   unsigned long histogram[GRAY_LEVELS+1];

      /***************************
```

```
      *
      *   Detect the edges.  Do
      *   not threshold.
      *
      ****************************/

   if(edge_type == 1  ||
      edge_type == 2  ||
      edge_type == 3)
      detect_edges(the_image, out_image,
                   edge_type, 0, 0,
                   rows, cols,
                   bits_per_pixel);

   if(edge_type == 4){
      quick_edge(the_image, out_image,
                 0, 0,
                 rows, cols,
                 bits_per_pixel);
   }  /* ends if 4 */
   if(edge_type == 5){
      homogeneity(the_image, out_image,
                  rows, cols,
                  bits_per_pixel,
                  0, 0);
   }  /* ends if 5 */

   if(edge_type == 6){
      difference_edge(the_image, out_image,
                      rows, cols,
                      bits_per_pixel,
                      0, 0);
   }  /* ends if 6 */

   if(edge_type == 7){
      contrast_edge(the_image, out_image,
                    rows, cols,
                    bits_per_pixel,
                    0, 0);
   }  /* ends if 7 */

   if(edge_type == 8){
      gaussian_edge(the_image, out_image,
                    rows, cols,
                    bits_per_pixel,
```

```
                    3, 0, 0);
   }  /* ends if 8 */

   if(edge_type == 10){
      range(the_image, out_image,
            rows, cols,
            bits_per_pixel,
            3, 0, 0);
   }  /* ends if 10 */

   if(edge_type == 11){
      variance(the_image, out_image,
               rows, cols,
               bits_per_pixel,
               0, 0);
   }  /* ends if 11 */

/**write_array_into_tiff_image("f:e1.tif", out_image,
il, ie, ll, le);**/

      /* copy out_image to the_image */
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] = out_image[i][j];

      /******************************
      *
      *   Threshold the edge detector
      *   output at a given percent.
      *   This eliminates the weak
      *   edges.
      *
      *******************************/

   zero_histogram(histogram, GRAY_LEVELS+1);
   calculate_histogram(the_image, histogram,
                       rows, cols);
   find_cutoff_point(histogram, percent, &cutoff,
                     rows, cols);
   threshold_image_array(the_image, out_image,
                         GRAY_LEVELS, cutoff,
                         set_value, rows, cols);


   if(erode != 0){
```

```
         /* copy out_image to the_image */
      for(i=0; i<rows; i++)
         for(j=0; j<cols; j++)
            the_image[i][j] = out_image[i][j];
      erode_image_array(the_image, out_image,
                        set_value, erode,
                        rows, cols);
   }  /* ends if erode */

      /******************************
      *
      *   Read the original gray shade
      *   image back into the_image.
      *
      ******************************/

   read_image_array(in_name, the_image);

      /******************************
      *
      *   Overlay the edge values
      *   on top of the original
      *   image by setting them to
      *   FORGET_IT so the region
      *   growing will not use those
      *   points.
      *
      ******************************/

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         if(out_image[i][j] == set_value)
            the_image[i][j] = FORGET_IT;

/**write_array_into_tiff_image("f:e4.tif", the_image,
il, ie, ll, le);**/

   pixel_grow(the_image, out_image, diff,
              min_area, max_area,
              rows, cols);

}  /* ends edge_gray_shade_region */
```

```
show_stackp()
{
char r[80];
   struct stacksp *temp;
   temp = stackp;
   while(temp != NULL){
      printf("\n\t\t\t\t\t%d %d %x %x",temp->x,temp->y, temp, temp->next);
      temp = temp->next;
   }
}



int is_not_emptyp(pointer)
   struct stacksp *pointer;
{
   int result = 0;
   if(pointer != NULL)
      result = 1;
   return(result);

}  /* ends is_empty */



pushp(x, y)
   short  x, y;
{
   struct stacksp *new_one;

   new_one = (struct stacksp *)
             calloc(1, sizeof(struct stacksp ));
   new_one->next = stackp;
   new_one->x    = x;
   new_one->y    = y;
   stackp        = new_one;
}  /* ends push */



popp(x, y)
   short  *x, *y;
```

```
{
   struct stacksp *temp;

   temp     = stackp;
   *x       = stackp->x;
   *y       = stackp->y;
   stackp   = stackp->next;
   free(temp);
}  /* ends pop */
```

Listing 10.2 - The Segmentation Subroutines of this Chapter

```
/**********************************************
*
*   file main2seg.c
*
*   Functions: This file contains
*      main
*
*   Purpose:
*      This is a calling program that calls
*      the three new segmentation techniques
*      discussed in Image Processing part 10.
*
*   External Calls:
*      imageio.c - does_not_exist
*                  get_image_size
*                  get_bitsperpixel
*                  allocate_image_array
*                  free_image_array
*                  create_file_if_needed
*                  read_image_array
*                  write_image_array
*      segment2.c - edge_region
*                   gray_shade_region
*                   edge_gray_shade_region
*
```

```
       *   Modifications:
       *      5 December 1992 - created
       *      15 August 1998 - modified to work on entire
       *           images at once.
       *
       ************************************************/

#include "cips.h"



short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char     name1[80], name2[80], low_high[80], type[80];
   float    percent;
   int      i, j,
            looking = 1;
   long     length, width, bits_per_pixel;
   short    value,  value2, value3,
            value4, value5, value6;
   struct   tiff_header_struct image_header;

       /************************************************
       *
       *       Interpret the command line parameters.
       *
       ************************************************/

   if(argc < 4){
    printf(
    "\n\nNot enough parameters:"
     "\n"
     "\n usage: main2seg in-file out-file type"
     " [values ...]"
     "\n"
     "\n   recall type: Edge-region edge-gray-grow (C)"
     " Gray-shade-grow"
     "\n"
    "\n   main2seg in-file out-file C percent "
```

```
 "edge-type "
 "\n            min-area max-area diff set-value erode"
 "\n   main2seg in-file out-file E percent "
 "edge-type "
 "\n            min-area max-area diff set-value erode"
 "\n   main2seg in-file out-file G diff "
 "min-area max-area"
 "\n"
 "\n");
 exit(0);
}

strcpy(name1,  argv[1]);
strcpy(name2, argv[2]);
strcpy(type,  argv[3]);
if(type[0] == 'e' || type[0] == 'E'  ||
   type[0] == 'c' || type[0] == 'C'){
   percent = atof(argv[4]);
   value   = atoi(argv[5]);
   value2  = atoi(argv[6]);
   value3  = atoi(argv[7]);
   value4  = atoi(argv[8]);
   value5  = atoi(argv[9]);
   value6  = atoi(argv[10]);
 }
else{
   value  = atoi(argv[4]);
   value2 = atoi(argv[5]);
   value3 = atoi(argv[6]);
}

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}


    /*****************************************
    *
    *  Read the input image header, allocate
    *  the image arrays, and read the input
    *  image.
    *
    *****************************************/
```

```
get_image_size(name1, &length, &width);
get_bitsperpixel(name1, &bits_per_pixel);

the_image = allocate_image_array(length, width);
out_image = allocate_image_array(length, width);

create_file_if_needed(name1, name2, out_image);
read_image_array(name1, the_image);


if(type[0] == 'e'  || type[0] == 'E'){
   edge_region(the_image, out_image,
               value, value2,
               value3, value4, percent,
               value5, value6,
               length,
               width,
               bits_per_pixel);
}  /* ends edge_region */



if(type[0] == 'g'  || type[0] == 'G'){
   gray_shade_region(the_image, out_image,
                     value,
                     value2, value3,
                     length,
                     width);
}  /* ends gray_shade_region */



if(type[0] == 'c'  || type[0] == 'C'){
   edge_gray_shade_region(name1,
            the_image, out_image,
            value, value2, value3, value4,
            percent, value5, value6,
            length,
            width,
            bits_per_pixel);
}  /* ends edge_gray_shade_region */

write_image_array(name2, out_image);
```

```
   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */
```

Listing 10.3 - The main2seg Program

F.11 Code Listings for Chapter 11

```
       /**********************************************
       *
       *   file ed.c
       *
       *   Functions: This file contains
       *      erosion
       *      dilation
       *      mask_erosion
       *      mask_dilation
       *      interior_outline
       *      exterior_outline
       *      copy_3_x_3
       *      opening
       *      closing
       *      get_shape_options
       *
       *   Purpose:
       *      These functions perform the erosion,
       *      dilation, outlining, opening and
       *      closing operations.
       *
       *   External Calls:
       *      none
       *
       *   Modifications:
       *      14 March 1993 - created
       *      21 August 1998 - modified to work on entire
       *           images at once.
       *
       **********************************************/

#include "cips.h"
```

```
short edmask1[3][3] = {{0, 1, 0},
                       {0, 1, 0},
                       {0, 1, 0}};

short edmask2[3][3] = {{0, 0, 0},
                       {1, 1, 1},
                       {0, 0, 0}};

short edmask3[3][3] = {{0, 1, 0},
                       {1, 1, 1},
                       {0, 1, 0}};

short edmask4[3][3] = {{1, 1, 1},
                       {1, 1, 1},
                       {1, 1, 1}};
```

```
     /*******************************************
     *
     *   erosion(...
     *
     *   This function performs the erosion
     *   operation.  If a value pixel has more
     *   than the threshold number of 0
     *   neighbors, you erode it by setting it
     *   to 0.
     *
     *******************************************/

erosion(the_image, out_image,
        value, threshold,
        rows, cols)

   int    threshold;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
```

```
{
   int    a, b, count, i, j, k;

      /***************************
      *
      *   Loop over image array
      *
      ***************************/

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   printf("\n");

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%3d", i);
      for(j=1; j<cols-1; j++){
         if(the_image[i][j] == value){
            count = 0;
            for(a=-1; a<=1; a++){
                for(b=-1; b<=1; b++){
                   if( (i+a) >= 0){
                      if(the_image[i+a][j+b] == 0)
                         count++;
                   }
                }  /*  ends loop over b */
            }  /* ends loop over a */
            if(count > threshold){ out_image[i][j] = 0;
            }
         }  /* ends if the_image == value */
      }  /* ends loop over j */
   }  /* ends loop over i */

   /*****
   fix_edges(out_image, 3, rows, cols);
   ***/

}  /* ends erosion */


     /*****************************************
```

```
     *
     *   dilation(...
     *
     *   This function performs the dilation
     *   operation.  If a 0 pixel has more than
     *   threshold number of value neighbors,
     *   you dilate it by setting it to value.
     *
     *******************************************/

dilation(the_image, out_image,
         value, threshold,
         rows, cols)
   int    threshold;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   int    three = 3;

      /***************************
      *
      *   Loop over image array
      *
      ****************************/

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   printf("\n");

   for(i=1; i<rows-1; i++){
      if( (i%10) == 0) printf("%3d", i);
      for(j=1; j<cols-1; j++){
         out_image[i][j] = the_image[i][j];
         if(the_image[i][j] == 0){
            count = 0;
            for(a=-1; a<=1; a++){
                for(b=-1; b<=1; b++){
                   if(a!=0  &&  b!=0){
                      if(the_image[i+a][j+b] == value)
                         count++;
```

```
                  }  /* ends avoid the center pixel */
               }  /*  ends loop over b */
            }  /* ends loop over a */
            if(count > threshold)
               out_image[i][j] = value;
         }  /* ends if the_image == 0 */
      }  /* ends loop over j */
   }  /* ends loop over i */

   /*****
   fix_edges(out_image, three, rows, cols);
   ***/

}  /* ends dilation */




     /*****************************************
     *
     *   mask_dilation(...
     *
     *   This function performs the dilation
     *   operation using the erosion-dilation
     *   3x3 masks given above.  It works on
     *   0-value images.
     *
     *****************************************/

mask_dilation(the_image, out_image,
              value, mask_type,
              rows, cols)
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   short  mask[3][3], max;

      /*************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
```

```
   *
   ***************************************/

switch(mask_type){
   case 1:
      copy_3_x_3(mask, edmask1);
      break;
   case 2:
      copy_3_x_3(mask, edmask2);
      break;
   case 3:
      copy_3_x_3(mask, edmask3);
      break;
   case 4:
      copy_3_x_3(mask, edmask4);
      break;
   default:
      printf("\nInvalid mask type, using mask 4");
      copy_3_x_3(mask, edmask4);
      break;
}

   /***************************
   *
   *   Loop over image array
   *
   ****************************/


printf("\n");

for(i=1; i<rows-1; i++){
   if( (i%10) == 0) printf("%3d", i);
   for(j=1; j<cols-1; j++){
      max = 0;
      for(a=-1; a<=1; a++){
          for(b=-1; b<=1; b++){
             if(mask[a+1][b+1] == 1){
                if(the_image[i+a][j+b] > max)
                   max = the_image[i+a][j+b];
             }  /* ends if mask == 1 */
          }  /*  ends loop over b */
      }  /* ends loop over a */
      out_image[i][j] = max;
   }  /* ends loop over j */
```

```
   }  /* ends loop over i */

   /*****
   fix_edges(out_image, 3, rows, cols);
   ***/

}  /* ends mask_dilation */




     /*****************************************
     *
     *   mask_erosion(...
     *
     *   This function performs the erosion
     *   operation using the erosion-dilation
     *   3x3 masks given above.  It works on
     *   0-value images.
     *
     *****************************************/

mask_erosion(the_image, out_image,
             value, mask_type,
             rows, cols)
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   short  mask[3][3], min;

      /*************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
      *
      *************************************/

   switch(mask_type){
      case 1:
         copy_3_x_3(mask, edmask1);
```

```
      break;
   case 2:
      copy_3_x_3(mask, edmask2);
      break;
   case 3:
      copy_3_x_3(mask, edmask3);
      break;
   case 4:
      copy_3_x_3(mask, edmask4);
      break;
   default:
      printf("\nInvalid mask type, using mask 4");
      copy_3_x_3(mask, edmask4);
      break;
}


   /***************************
   *
   *   Loop over image array
   *
   ****************************/

printf("\n");

for(i=1; i<rows-1; i++){
   if( (i%10) == 0) printf("%3d", i);
   for(j=1; j<cols-1; j++){
      min = value;
      for(a=-1; a<=1; a++){
          for(b=-1; b<=1; b++){
             if(mask[a+1][b+1] == 1){
                if(the_image[i+a][j+b] < min)
                   min = the_image[i+a][j+b];
             }  /* ends if mask == 1 */
          }  /*  ends loop over b */
      }  /* ends loop over a */
      out_image[i][j] = min;
   }  /* ends loop over j */
}  /* ends loop over i */

/*****
fix_edges(out_image, 3, rows, cols);
***/
```

```
}  /* ends mask_erosion */



     /**********************************************
     *
     *   copy_3_x_3(a, b)
     *
     *   This function copies a 3x3 array of shorts
     *   from one array to another.  It copies array
     *   b into array a.
     *
     **********************************************/

copy_3_x_3(a, b)
   short a[3][3], b[3][3];
{
   int i, j;
   for(i=0; i<3; i++)
      for(j=0; j<3; j++)
         a[i][j] = b[i][j];
}  /* ends copy_3_x_3 */




     /*******************************************
     *
     *   opening(...
     *
     *   Opening is erosion followed by dilation.
     *   This routine will use the mask erosion
     *   and dilation.  You could use the other
     *   types and you could mix the two types.
     *
     *   The number parameter specifies how
     *   erosions to perform before doing one
     *   dilation.
     *
     *******************************************/

opening(the_image, out_image,
        value, mask_type, number,
        rows, cols)
   int    number;
```

```
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   short  mask[3][3], max;

      /*************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
      *
      ****************************************/

   switch(mask_type){
      case 1:
         copy_3_x_3(mask, edmask1);
         break;
      case 2:
         copy_3_x_3(mask, edmask2);
         break;
      case 3:
         copy_3_x_3(mask, edmask3);
         break;
      case 4:
         copy_3_x_3(mask, edmask4);
         break;
      default:
         printf("\nInvalid mask type, using mask 4");
         copy_3_x_3(mask, edmask4);
         break;
   }

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   mask_erosion(the_image, out_image,
                value, mask_type,
                rows, cols);

   if(number > 1){
      count = 1;
```

```
      while(count < number){
         count++;
         mask_erosion(the_image, out_image,
                      value, mask_type,
                      rows, cols);
      }  /* ends while */
   }  /* ends if number > 1 */

   mask_dilation(the_image,
                 out_image,
                 value, mask_type,
                 rows, cols);

}  /* ends opening */




     /*****************************************
     *
     *   closing(...
     *
     *   Closing is dilation followed by erosion.
     *   This routine will use the mask erosion
     *   and dilation.  You could use the other
     *   types and you could mix the two types.
     *
     *   The number parameter specifies how
     *   dilations to perform before doing one
     *   erosion.
     *
     *****************************************/

closing(the_image, out_image,
        value, mask_type, number,
        rows, cols)
   int    number;
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
```

```
   short  mask[3][3], max;
printf("\nCLOSING> value=%d mask=%d number=%d",value,mask_type,number);

      /***************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
      *
      ****************************************/

   switch(mask_type){
      case 1:
         copy_3_x_3(mask, edmask1);
         break;
      case 2:
         copy_3_x_3(mask, edmask2);
         break;
      case 3:
         copy_3_x_3(mask, edmask3);
         break;
      case 4:
         copy_3_x_3(mask, edmask4);
         break;
      default:
         printf("\nInvalid mask type, using mask 4");
         copy_3_x_3(mask, edmask4);
         break;
   }

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   mask_dilation(the_image, out_image,
                 value, mask_type,
                 rows, cols);

   if(number > 1){
      count = 1;
      while(count < number){
         count++;
         mask_dilation(the_image, out_image,
                       value, mask_type,
                       rows, cols);
      }  /* ends while */
```

```
   }  /* ends if number > 1 */

   mask_erosion(the_image, out_image,
                value, mask_type,
                rows, cols);

}  /* ends closing */




     /*****************************************
     *
     *   interior_outline(...
     *
     *   This function produces the outline of
     *   any "holes" inside an object.  The
     *   method is:
     *      output = erosion of input
     *      final output = input - output
     *
     *****************************************/

interior_outline(the_image, out_image,
                 value, mask_type,
                 rows, cols)
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   short  mask[3][3], max;

      /*************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
      *
      *************************************/

   switch(mask_type){
      case 1:
         copy_3_x_3(mask, edmask1);
```

```
         break;
      case 2:
         copy_3_x_3(mask, edmask2);
         break;
      case 3:
         copy_3_x_3(mask, edmask3);
         break;
      case 4:
         copy_3_x_3(mask, edmask4);
         break;
      default:
         printf("\nInvalid mask type, using mask 4");
         copy_3_x_3(mask, edmask4);
         break;
   }

   mask_erosion(the_image,
                out_image,
                value, mask_type,
                rows, cols);

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] =
            the_image[i][j] - out_image[i][j];

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

}  /* ends interior_outline */
```

```
     /*******************************************
     *
     *   exterior_outline(...
     *
     *   This function produces the outline of
     *   exterior of an object.  The
     *   method is:
     *      output = dilation of input
     *      final output = output - input
```

```
     *
     ******************************************/


exterior_outline(the_image, out_image,
                 value, mask_type,
                 rows, cols)
   int    mask_type;
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
   short  mask[3][3], max;

      /**************************************
      *
      *   Copy the 3x3 erosion-dilation mask
      *   specified by the mask_type.
      *
      **************************************/

   switch(mask_type){
      case 1:
         copy_3_x_3(mask, edmask1);
         break;
      case 2:
         copy_3_x_3(mask, edmask2);
         break;
      case 3:
         copy_3_x_3(mask, edmask3);
         break;
      case 4:
         copy_3_x_3(mask, edmask4);
         break;
      default:
         printf("\nInvalid mask type, using mask 4");
         copy_3_x_3(mask, edmask4);
         break;
   }

   mask_dilation(the_image, out_image,
                 value, mask_type,
                 rows, cols);
```

```
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] =
            out_image[i][j] - the_image[i][j];

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

}  /* ends exterior_outline */
```

Listing 11.1 - Shape Manipulating Subroutines

```
/**********************************************
*
*       file skeleton.c
*
*       Functions: This file contains
*          thinning
*          can_thin
*          dilate_not_join
*          can_dilate
*          little_label_and_check
*          special_closing
*          special_opening
*          edm
*          distanc_8
*          mat
*          mat_d
*
*       Purpose:
*          These functions thin objects in
*          an image, dilate objects, and
*          perform opening and closing
*          without removing or joining
*          objects.
*
*       External Calls:
*          none
*
*
```

```
       *       Modifications:
       *          7 March 1993 - created
       *         21 August 1998 - modified to work on entire
       *              images at once.
       *
       ***********************************************/

#include "cips.h"




     /*****************************************
     *
     *   special_opening(...
     *
     *   Opening is erosion followed by dilation.
     *   This routine will use the thinning
     *   erosion routine.  This will not allow
     *   an object to erode to nothing.
     *
     *   The number parameter specifies how
     *   erosions to perform before doing one
     *   dilation.
     *
     *****************************************/

special_opening(the_image, out_image,
                value, threshold, number,
                rows, cols)
   int    number;
   short  **the_image,
          **out_image,
          threshold, value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;

   thinning(the_image, out_image,
            value, threshold, 1,
            rows, cols);

   if(number > 1){
      count = 1;
```

```
      while(count < number){
         count++;
         thinning(the_image, out_image,
                  value, threshold, 1,
                  rows, cols);
      }  /* ends while */
   }  /* ends if number > 1 */

   dilation(the_image, out_image,
            value, threshold,
            rows, cols);

}  /* ends special_opening */




     /*******************************************
     *
     *   thinning(...
     *
     *   Use a variation of the grass fire
     *   wave front approach.
     *
     *   Raster scan the image left to right
     *   and examine and thin the left edge pixels
     *   (a 0 to value transition).  Process them
     *   normally and "save" the result.  Next,
     *   raster scan the image right to left and
     *   save.  Raster scan top to bottom and save.
     *   Raster scan bottom to top and save.
     *
     *   That is one complete pass.
     *
     *   Keep track of pixels thinned for a
     *   pass and quit when you make a complete
     *   pass without thinning any pixels.
     *
     *******************************************/


thinning(the_image, out_image,
         value, threshold, once_only,
```

```
         rows, cols)
   int    once_only;
   short  **the_image,
          **out_image,
          threshold, value;
   long   cols, rows;
{
   int    a, b, big_count, count, i, j, k,
          not_finished;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

   not_finished = 1;
   while(not_finished){

      if(once_only == 1)
        not_finished = 0;
      big_count = 0;

         /***************************
         *
         *   Scan left to right
         *   Look for 0-value transition
         *
         ***************************/

      printf("\n");
      for(i=1; i<rows-1; i++){
         if( (i%10) == 0) printf("%3d", i);
         for(j=1; j<cols-1; j++){
            if(the_image[i][j-1] == 0   &&
               the_image[i][j]   == value){
               count = 0;
               for(a=-1; a<=1; a++){
                   for(b=-1; b<=1; b++){
                         if(the_image[i+a][j+b] == 0)
                            count++;
                   }  /*  ends loop over b */
               }  /* ends loop over a */
               if(count > threshold){
                  if(can_thin(the_image, i, j, value)){
                     out_image[i][j] = 0;
                     big_count++;
```

```
            }  /* ends if can_thin */
         }  /* ends if count > threshold */
      }  /* ends if the_image == value */
   }  /* ends loop over j */
}  /* ends loop over i */

   /***************************************
   *
   *   Copy the output back to the input.
   *
   ***************************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];


   /***************************
   *
   *   Scan right to left
   *   Do this by scanning left
   *   to right and look for
   *   value-0 transition.
   *
   ****************************/

printf("\n");
for(i=1; i<rows-1; i++){
   if( (i%10) == 0) printf("%3d", i);
   for(j=1; j<cols-1; j++){
      if(the_image[i][j+1] == 0   &&
         the_image[i][j]   == value){
         count = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                   if(the_image[i+a][j+b] == 0)
                      count++;
             }  /*  ends loop over b */
         }  /* ends loop over a */
         if(count > threshold){
            if(can_thin(the_image, i, j, value)){
               out_image[i][j] = 0;
               big_count++;
            }  /* ends if can_thin */
         }  /* ends if count > threshold */
```

```
      }  /* ends if the_image == value */
   }  /* ends loop over j */
}  /* ends loop over i */

   /*************************************
   *
   *   Copy the output back to the input.
   *
   *************************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];


   /**************************
   *
   *   Scan top to bottom
   *   Look for 0-value transition
   *
   ***************************/

printf("\n");
for(j=1; j<cols-1; j++){
   if( (j%10) == 0) printf("%3d", j);
   for(i=1; i<rows-1; i++){
      if(the_image[i-1][j] == 0   &&
         the_image[i][j]   == value){
         count = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                   if(the_image[i+a][j+b] == 0)
                      count++;
             }  /*  ends loop over b */
         }  /* ends loop over a */
         if(count > threshold){
            if(can_thin(the_image, i, j, value)){
               out_image[i][j] = 0;
               big_count++;
            }  /* ends if can_thin */
         }  /* ends if count > threshold */
      }  /* ends if the_image == value */
   }  /* ends loop over i */
}  /* ends loop over j */
```

```
   /*************************************
   *
   *   Copy the output back to the input.
   *
   *************************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];



   /***************************
   *
   *   Scan bottom to top
   *   Do this by scanning top
   *   to bottom and look for
   *   value-0 transition.
   *
   ****************************/

printf("\n");
for(j=1; j<cols-1; j++){
   if( (j%10) == 0) printf("%3d", j);
   for(i=1; i<rows-1; i++){
      if(the_image[i+1][j] == 0   &&
         the_image[i][j]   == value){
         count = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                   if(the_image[i+a][j+b] == 0)
                      count++;
             }  /*  ends loop over b */
         }  /* ends loop over a */
         if(count > threshold){
            if(can_thin(the_image, i, j, value)){
               out_image[i][j] = 0;
               big_count++;
            }  /* ends if can_thin */
         }  /* ends if count > threshold */
      }  /* ends if the_image == value */
   }  /* ends loop over i */
}  /* ends loop over j */

   /*************************************
```

```
         *
         *   Copy the output back to the input.
         *
         ************************************/

      for(i=0; i<rows; i++)
         for(j=0; j<cols; j++)
            the_image[i][j] = out_image[i][j];

         /************************************
         *
         *   Now look at the result of this big
         *   pass through the image.
         *
         *************************************/

      printf("\n\nThinned %d pixels", big_count);
      if(big_count == 0)
         not_finished = 0;
      else{
         for(i=0; i<rows; i++)
            for(j=0; j<cols; j++)
               the_image[i][j] = out_image[i][j];
      }  /* ends else */

   }  /* ends while not_finished */

   /****
   fix_edges(out_image, 3, rows, cols);
   *****/

}  /* ends thinning */




     /******************************************
     *
     *   can_thin(...
     *
     *   Look at the neighbors of the center pixel.
     *   If a neighbor == value, then it must
     *   have a neighbor == value other than the
     *   center pixel.
```

```
     *
     *   Procedure:
     *   . Copy the 3x3 area surrounding pixel
     *     i,j to a temp array.
     *   . Set the center pixel to zero.
     *   . Look at each non-zero pixel in temp.
     *      . If you cannot find a non-zero
     *        neighbor.
     *      . Then you cannot thin.
     *
     *******************************************/

can_thin(the_image, i, j, value)
   int   i, j;
   short **the_image, value;
{
   int   a, b, c, d, count,
         no_neighbor, one=1, zero=0;
   short temp[3][3];

       /****************************************
       *
       *   Copy the center pixel and its
       *   neighbors to the temp array.
       *
       ****************************************/

   for(a=-1; a<2; a++)
      for(b=-1; b<2; b++)
         temp[a+1][b+1] = the_image[i+a][b+j];

       /****************************************
       *
       *   Set the center of temp to 0.
       *
       ****************************************/

   temp[1][1] = 0;

       /****************************************
       *
       *   Check the non-zero pixels in temp.
       *
       ****************************************/
```

```
for(a=0; a<3; a++){
   for(b=0; b<3; b++){
      if(temp[a][b] == value){
         temp[a][b] = 0;

             /*********************************
             *
             *   Check the neighbors of this pixel
             *   If there is a single non-zero
             *   neighbor, set no_neighbor = 0.
             *
             **********************************/

         no_neighbor = 1;
         for(c=-1; c<2; c++){
            for(d=-1; d<2; d++){
               if( ((a+c) >= 0)   &&
                   ((a+c) <= 2)   &&
                   ((b+d) >= 0)   &&
                   ((b+d) <= 2)){
                  if(temp[a+c][b+d] == value){
                     no_neighbor = 0;
                  }  /* ends if temp == value */
               }  /* ends if part of temp array */
            }  /* ends loop over d */
         }  /* ends loop over c */
         temp[a][b] = value;

             /*********************************
             *
             *   If the non-zero pixel did not
             *   have any non-zero neighbors,
             *   no_neighbor still equals 1,
             *   and we cannot thin, so return
             *   zero.
             *
             **********************************/

         if(no_neighbor){
            return(zero);
         }
      }  /* ends if temp[a][b] == value */
   }  /* ends loop over b */
}  /* ends loop over a */
```

```
      /*************************************
      *
      *   First, ensure the object is more
      *   than two wide.  If it is two wide,
      *   you will thin out the entire object.
      *   Check in all eight directions.
      *   If the distance to a zero is 0 or
      *   >= 2, then ok you can thin so go
      *   on to the remainder of this routine.
      *   If not, you cannot thin so return
      *   zero.
      *
      **************************************/

   return(one);

}  /* ends can_thin */



     /*******************************************
     *
     *   special_closing(...
     *
     *   Closing is dilation followed by erosion.
     *   This routine will use the dilate_not_join
     *   dilation routine.  This will not allow
     *   two separate objects to join.
     *
     *   The number parameter specifies how
     *   dilations to perform before doing one
     *   erosion.
     *
     ********************************************/

special_closing(the_image, out_image,
                value, threshold, number,
                rows, cols)
   int    number;
   short  **the_image,
          **out_image,
          threshold, value;
   long   cols, rows;
{
```

```
   int    a, b, count, i, j, k;

   dilate_not_join(the_image, out_image,
                   value, threshold,
                   rows, cols);

   if(number > 1){
      count = 1;
      while(count < number){
         count++;
         dilate_not_join(the_image, out_image,
                         value, threshold,
                         rows, cols);
      }  /* ends while */
   }  /* ends if number > 1 */

   erosion(the_image, out_image,
           value, threshold,
           rows, cols);

}  /* ends special_closing */



     /*****************************************
     *
     *   dilate_not_join(...
     *
     *   Use a variation of the grass fire
     *   wave front approach.
     *
     *   Raster scan the image left to right
     *   and examine and dilate the left edge pixels
     *   (a value to 0 transition).  Process them
     *   normally and "save" the result.  Next,
     *   raster scan the image right to left and
     *   save.  Raster scan top to bottom and save.
     *   Raster scan bottom to top and save.
     *
     *   That is one complete pass.
     *
     *****************************************/

dilate_not_join(the_image, out_image,
```

```
                value, threshold,
                rows, cols)
   short  **the_image,
          **out_image,
          threshold, value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = the_image[i][j];

         /***************************
         *
         *   Scan left to right
         *   Look for value-0 transition
         *
         ****************************/

      printf("\n");
      for(i=1; i<rows-1; i++){
         if( (i%10) == 0) printf("%3d", i);
         for(j=1; j<cols-1; j++){
            if(the_image[i][j-1] == value  &&
               the_image[i][j]   == 0){
               count = 0;
               for(a=-1; a<=1; a++){
                   for(b=-1; b<=1; b++){
                         if(the_image[i+a][j+b]==value)
                            count++;
                   }  /*  ends loop over b */
               }  /* ends loop over a */
               if(count > threshold){
                  if(can_dilate(the_image,i,j,value)){
                     out_image[i][j] = value;
                  }  /* ends if can_dilate */
               }  /* ends if count > threshold */
            }  /* ends if the_image == value */
         }  /* ends loop over j */
      }  /* ends loop over i */

         /****************************************
         *
         *   Copy the output back to the input.
```

```
   *
   ************************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];


   /***************************
   *
   *   Scan right to left
   *   Do this by scanning left
   *   to right and look for
   *   0-value transition.
   *
   ***************************/

printf("\n");
for(i=1; i<rows-1; i++){
   if( (i%10) == 0) printf("%3d", i);
   for(j=1; j<cols-1; j++){
      if(the_image[i][j+1] == value   &&
         the_image[i][j]   == 0){
         count = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                   if(the_image[i+a][j+b]==value)
                      count++;
             }  /*  ends loop over b */
         }  /* ends loop over a */
         if(count > threshold){
            if(can_dilate(the_image,i,j,value)){
               out_image[i][j] = value;
            }  /* ends if can_dilate */
         }  /* ends if count > threshold */
      }  /* ends if the_image == value */
   }  /* ends loop over j */
}  /* ends loop over i */

   /*************************************
   *
   *   Copy the output back to the input.
   *
   *************************************/
```

```
for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];


   /***************************
   *
   *   Scan top to bottom
   *   Look for value-0 transition
   *
   ****************************/

printf("\n");
for(j=1; j<cols-1; j++){
   if( (j%10) == 0) printf("%3d", j);
   for(i=1; i<rows-1; i++){
      if(the_image[i-1][j] == value   &&
         the_image[i][j]   == 0){
         count = 0;
         for(a=-1; a<=1; a++){
             for(b=-1; b<=1; b++){
                   if(the_image[i+a][j+b]==value)
                      count++;
             }  /*  ends loop over b */
         }  /* ends loop over a */
         if(count > threshold){
            if(can_dilate(the_image,i,j,value)){
               out_image[i][j] = value;
            }  /* ends if can_dilate */
         }  /* ends if count > threshold */
      }  /* ends if the_image == value */
   }  /* ends loop over i */
}  /* ends loop over j */

   /*************************************
   *
   *   Copy the output back to the input.
   *
   **************************************/

for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];
```

```
      /***************************
      *
      *   Scan bottom to top
      *   Do this by scanning top
      *   to bottom and look for
      *   0-value transition.
      *
      ***************************/

   printf("\n");
   for(j=1; j<cols-1; j++){
      if( (j%10) == 0) printf("%3d", j);
      for(i=1; i<rows-1; i++){
         if(the_image[i+1][j] == value   &&
            the_image[i][j]   == 0){
            count = 0;
            for(a=-1; a<=1; a++){
                for(b=-1; b<=1; b++){
                      if(the_image[i+a][j+b]==value)
                         count++;
                }  /*  ends loop over b */
            }  /* ends loop over a */
            if(count > threshold){
               if(can_dilate(the_image,i,j,value)){
                  out_image[i][j] = value;
               }  /* ends if can_dilate */
            }  /* ends if count > threshold */
         }  /* ends if the_image == value */
      }  /* ends loop over i */
   }  /* ends loop over j */

      /*************************************
      *
      *   Copy the output back to the input.
      *
      *************************************/

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         the_image[i][j] = out_image[i][j];


/****
fix_edges(out_image, 3, rows, cols);
```

```
   *****/

}  /* ends dilate_not_join */




     /*****************************************
     *
     *   can_dilate(...
     *
     *   This function decides if you can dilate
     *   (set to value) a pixel without joining
     *   two separate objects in a 3x3 area.
     *
     *   First, you grow regions inside the 3x3
     *   area.  Next, check if the center pixel
     *   has neighbors with differing values.
     *   If it does, you cannot dilate it because
     *   that would join two separate objects.
     *
     *****************************************/

can_dilate(the_image, i, j, value)
   int   i, j;
   short **the_image, value;
{
   int   a, b, c, d, count, found=0,
         no_neighbor,
         stack_pointer=-1,
         stack_empty=1,
         stack[12][2],
         pop_a, pop_b,
         one=1,
         zero=0;
   short first_value, label = 2, temp[3][3];

       /***************************************
       *
       *   Copy the center pixel and its
       *   neighbors to the temp array.
       *
       ***************************************/
```

```
for(a=-1; a<2; a++)
   for(b=-1; b<2; b++)
      temp[a+1][b+1] = the_image[i+a][b+j];

    /*************************************
    *
    *   Grow objects inside the temp array.
    *
    *************************************/

for(a=0; a<3; a++){
   for(b=0; b<3; b++){
      stack_empty   = 1;
      stack_pointer = -1;
      if(temp[a][b] == value){
         little_label_and_check(temp, stack, label,
                                &stack_empty,
                                &stack_pointer,
                                a, b, value);
         found = 1;
      }  /* ends if temp == value */

      while(stack_empty == 0){
         pop_a = stack[stack_pointer][0]; /* POP */
         pop_b = stack[stack_pointer][1]; /* POP */
         --stack_pointer;
         if(stack_pointer <= 0){
            stack_pointer = 0;
            stack_empty   = 1;
         }  /* ends if stack_pointer */
         little_label_and_check(temp, stack, label,
                                &stack_empty,
                                &stack_pointer,
                                pop_a, pop_b, value);
      }  /* ends while stack_empty == 0 */

      if(found){
         found = 0;
         label++;
      }  /* ends if object_found */
   }  /* ends loop over b */
}  /* ends loop over a */

    /*************************************
    *
```

```
       *   Look at the center pixel.  If it
       *   has two non-zero neigbors whose
       *   pixels are not the same, then
       *   you cannot dilate.
       *
       ***************************************/

   first_value = -1;
   for(a=0; a<3; a++){
      for(b=0; b<3; b++){
         if(temp[a][b]  != 0 &&
            first_value == -1){
            first_value = temp[a][b];
         }
         if(temp[a][b]  != 0  &&
            first_value != -1){
            if(temp[a][b] != first_value){
               return(zero);
            }
         }
      }  /* ends loop over b */
   }  /* ends loop over a */

   return(one);

}  /* ends can_dilate */




     /*********************************************
     *
     *   little_label_and_check(...
     *
     *   This function labels the objects in
     *   in a 3x3 area.
     *
     *********************************************/

little_label_and_check(temp, stack, label, stack_empty,
                       stack_pointer, a, b, value)
   int   a, b, stack[12][2],
         *stack_empty, *stack_pointer;
   short temp[3][3], label, value;
```

```
{
   int c, d;

   temp[a][b] = label;
   for(c=a-1; c<=a+1; c++){
      for(d=b-1; d<=b+1; d++){
         if(c >= 0      &&
            c <= 2      &&
            d >= 0      &&
            d <= 2)
            if(temp[c][d] == value){  /* PUSH */
               *stack_pointer = *stack_pointer + 1;
               stack[*stack_pointer][0] = c;
               stack[*stack_pointer][1] = d;
               *stack_empty = 0;
            }  /* ends if temp == value */
      }  /* ends loop over d */
   }  /* ends loop over c */

}  /* ends little_label_and_check */




     /*****************************************
     *
     *   edm(..
     *
     *   This function calculates the Euclidean
     *   distance measure for objects in an image.
     *   It calculates the distance from any
     *   pixel=value to the nearest zero pixel
     *
     *****************************************/


edm(the_image, out_image,
    value, rows, cols)
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k;
```

```
   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = 0;

      /***************************
      *
      *   Loop over image array
      *
      ***************************/

   printf("\n");

   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf("%3d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] == value)
            out_image[i][j] = distance_8(the_image,
                                         i, j,
                                         value,
                                         rows, cols);
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends edm */



     /*******************************************
     *
     *   distance_8(..
     *
     *   This function finds the distance from
     *   a pixel to the nearest zero pixel.
     *   It search in all eight directions.
     *
     *******************************************/

distance_8(the_image, a, b, value, rows, cols)
   int   a, b;
   short **the_image, value;
   long  cols, rows;
{
   int i, j, measuring;
   short dist1  = 0,
```

```
      dist2  = 0,
      dist3  = 0,
      dist4  = 0,
      dist5  = 0,
      dist6  = 0,
      dist7  = 0,
      dist8  = 0,
      result = 0;

   /* straight up */
measuring = 1;
i = a;
j = b;
while(measuring){
   i--;
   if(i >= 0){
      if(the_image[i][j] == value)
         dist1++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
result = dist1;

   /* straight down */
measuring = 1;
i = a;
j = b;
while(measuring){
   i++;
   if(i <= rows-1){
      if(the_image[i][j] == value)
         dist2++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
if(dist2 <= result)
   result = dist2;

   /* straight left */
```

```
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j--;
      if(j >= 0){
         if(the_image[i][j] == value)
            dist3++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   if(dist3 <= result)
      result = dist3;

      /* straight right */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j++;
      if(j <= cols-1){
         if(the_image[i][j] == value)
            dist4++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   if(dist4 <= result)
      result = dist4;

      /* left and up */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j--;
      i--;
      if(j >= 0 && i>=0){
         if(the_image[i][j] == value)
            dist5++;
```

```
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
dist5 = (dist5*14)/10;
if(dist5 <= result)
   result = dist5;

   /* right and up */
measuring = 1;
i = a;
j = b;
while(measuring){
   j++;
   i--;
   if(j <=cols-1 && i>=0){
      if(the_image[i][j] == value)
         dist6++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
dist6 = (dist6*14)/10;
if(dist6 <= result)
   result = dist6;

   /* right and down */
measuring = 1;
i = a;
j = b;
while(measuring){
   j++;
   i++;
   if(j <=cols-1 && i<=rows-1){
      if(the_image[i][j] == value)
         dist7++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
```

```
   }  /* ends while measuring */
   dist7 = (dist7*14)/10;
   if(dist7 <= result)
      result = dist7;

      /* left and down */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j--;
      i++;
      if(j >=0 && i<=rows-1){
         if(the_image[i][j] == value)
            dist8++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   dist8 = (dist8*14)/10;
   if(dist8 <= result)
      result = dist8;

   return(result);
}  /* ends distance_8 */
```

```
/*******************************************
*
*   mat(..
*
*   This function finds the medial axis
*   transform for objects in an image.
*   The mat are those points that are
*   minimally distant to more than one
*   boundary point.
*
*******************************************/
```

```
mat(the_image, out_image,
    value, rows, cols)
   short  **the_image,
          **out_image,
          value;
   long   cols, rows;
{
   int    a, b, count, i, j, k,
          length, width;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = 0;

      /***************************
      *
      *   Loop over image array
      *
      ***************************/

   printf("\n");

   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf("%3d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] == value)
            out_image[i][j] = mat_d(the_image,
                                    i, j, value,
                                    rows, cols);
      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends mat */



     /*****************************************
     *
     *   mat_d(..
     *
     *   This function helps find the medial
     *   axis transform.
     *
     *   This function measures the distances
     *   from the point to a zero pixel in all
```

```
     *   eight directions.  Look for the two
     *   shortest distances in the eight distances.
     *   If the two shortest distances are
     *   equal, then the point in question
     *   is minimally distant to more than one
     *   boundary point.  Therefore, it is
     *   on the medial axis so return a value.
     *   Otherwise, return zero.
     *
     ******************************************/

mat_d(the_image, a, b, value, rows, cols)
   int   a, b;
   short **the_image, value;
   long  cols, rows;
{
   int i, j, measuring;
   short dist1  = 0,
         dist2  = 0,
         dist3  = 0,
         dist4  = 0,
         dist5  = 0,
         dist6  = 0,
         dist7  = 0,
         dist8  = 0,
         min1   = GRAY_LEVELS,
         min2   = GRAY_LEVELS,
         result = 0;

      /* straight up */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      i--;
      if(i >= 0){
         if(the_image[i][j] == value)
            dist1++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   result = dist1;
```

```
min1   = dist1;

   /* straight down */
measuring = 1;
i = a;
j = b;
while(measuring){
   i++;
   if(i <= rows-1){
      if(the_image[i][j] == value)
         dist2++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
if(dist2 <= result)
   result = dist2;
if(dist2 < min1){
   min2 = min1;
   min1 = dist2;
}
else
   if(dist2 < min2)
      min2 = dist2;

   /* straight left */
measuring = 1;
i = a;
j = b;
while(measuring){
   j--;
   if(j >= 0){
      if(the_image[i][j] == value)
         dist3++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
if(dist3 <= result)
   result = dist3;
if(dist3 < min1){
```

```
      min2 = min1;
      min1 = dist3;
   }
   else
      if(dist3 < min2)
         min2 = dist3;

      /* straight right */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j++;
      if(j <= cols-1){
         if(the_image[i][j] == value)
            dist4++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   if(dist4 <= result)
      result = dist4;
   if(dist4 < min1){
      min2 = min1;
      min1 = dist4;
   }
   else
      if(dist4 < min2)
         min2 = dist4;

      /* left and up */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j--;
      i--;
      if(j >= 0 && i>=0){
         if(the_image[i][j] == value)
            dist5++;
         else
            measuring = 0;
      }
```

```
      else
         measuring = 0;
   }  /* ends while measuring */
   dist5 = ((dist5*14)+7)/10;
   if(dist5 <= result)
      result = dist5;
   if(dist5 < min1){
      min2 = min1;
      min1 = dist5;
   }
   else
      if(dist5 < min2)
         min2 = dist5;

      /* right and up */
   measuring = 1;
   i = a;
   j = b;
   while(measuring){
      j++;
      i--;
      if(j <=cols-1 && i>=0){
         if(the_image[i][j] == value)
            dist6++;
         else
            measuring = 0;
      }
      else
         measuring = 0;
   }  /* ends while measuring */
   dist6 = ((dist6*14)+7)/10;
   if(dist6 <= result)
      result = dist6;
   if(dist6 < min1){
      min2 = min1;
      min1 = dist6;
   }
   else
      if(dist6 < min2)
         min2 = dist6;

      /* right and down */
   measuring = 1;
   i = a;
   j = b;
```

```
while(measuring){
   j++;
   i++;
   if(j <=cols-1 && i<=rows-1){
      if(the_image[i][j] == value)
         dist7++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
dist7 = ((dist7*14)+7)/10;
if(dist7 <= result)
   result = dist7;
if(dist7 < min1){
   min2 = min1;
   min1 = dist7;
}
else
   if(dist7 < min2)
      min2 = dist7;

   /* left and down */
measuring = 1;
i = a;
j = b;
while(measuring){
   j--;
   i++;
   if(j >=0 && i<=rows-1){
      if(the_image[i][j] == value)
         dist8++;
      else
         measuring = 0;
   }
   else
      measuring = 0;
}  /* ends while measuring */
dist8 = ((dist8*14)+7)/10;
if(dist8 <= result)
   result = dist8;
if(dist8 < min1){
   min2 = min1;
   min1 = dist8;
```

```
   }
   else
      if(dist8 < min2)
         min2 = dist8;

   if(min1 == min2)
      result = value;
   else
      result = 0;

   if(min1 == 0)
      result = 0;

   return(result);
}  /* ends mat_d */
```

Listing 11.2 - Shape Manipulating Subroutines

```
/*********************************************
*
*   file mainsk.c
*
*   Functions: This file contains
*      main
*      show_mainsk_usage
*
*   Purpose:
*      This file contains the main calling
*      routine that calls the erosion,
*      dilation, outline, and skeleton
*      functions.
*
*   External Calls:
*      imageio.c - create_image_file
*                  read_image_array
*                  write_image_array
*                  get_image_size
*                  allocate_image_array
*                  free_image_array
*      ed.c - erosion
*             dilation
```

```
       *             mask_erosion
       *             mask_dilation
       *             interior_outline
       *             exterior_outline
       *             opening
       *             closing
       *      skeleton.c - thinning
       *                   skeleton
       *                   dilate_not_join
       *                   special_opening
       *                   special_closing
       *                   edm
       *                   mat
       *
       *   Modifications:
       *      7 March 1993 - created
       *      21 August 1998 - modified to work on entire
       *           images at once.
       *      19 September 1998 - modified to work with
       *            all I O routines in imageio.c.
       *
       ************************************************/

#include "cips.h"




main(argc, argv)
   int argc;
   char *argv[];
{

   char     name1[80], name2[80], type[80];
   int      i, j, mask_type,
            number, threshold;
   long     length, width;
   short    value;
   short    **the_image;
   short    **out_image;


       /****************************************
       *
       *   Interpret the command line parameters.
```

```
    *
    ****************************************/

if(argc < 5){
   show_mainsk_usage();
   exit(0);
}

strcpy(name1,    argv[1]);
strcpy(name2,    argv[2]);
strcpy(type,     argv[3]);
value     = atoi(argv[4]);

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

if(argc >= 5){
   threshold = atoi(argv[5]);
   mask_type = atoi(argv[5]);
}

if(argc >= 6)
   number = atoi(argv[6]);


    /****************************************
    *
    *   Allocate the arrays, create the output
    *   file, read data.
    *
    ****************************************/

get_image_size(name1, &length, &width);

the_image = allocate_image_array(length, width);
out_image = allocate_image_array(length, width);

create_image_file(name1, name2);
read_image_array(name1, the_image);

for(i=0; i<length; i++)
   for(j=0; j<width; j++)
```

```
      out_image[i][j] = 0;

    /**********************************************
    *
    *   Call the desired function.
    *
    **********************************************/


         /* thinning */
if(strncmp("thi", type, 3) == 0){
   thinning(the_image, out_image,
            value, threshold, 0,
            length,
            width);
}  /* ends thinning operation */

         /* dilate-not-join */
if(strncmp("dnj", type, 3) == 0){
   dilate_not_join(the_image, out_image,
                   value, threshold,
                   length,
                   width);
}  /* ends dilate_not_join operation */

         /* erosion */
if(strncmp("ero", type, 3) == 0){
   erosion(the_image, out_image,
           value, threshold,
           length,
           width);
}  /* ends erosion operation */

         /* dilation */
if(strncmp("dil", type, 3) == 0){
   dilation(the_image, out_image,
            value, threshold,
            length,
            width);
}  /* ends dilation operation */

         /* mask_erosion */
if(strncmp("mer", type, 3) == 0){
   mask_erosion(the_image, out_image,
                value, mask_type,
```

```
                length,
                width);
}  /* ends mask_erosion operation */

         /* mask_dilation */
if(strncmp("mdi", type, 3) == 0){
   mask_dilation(the_image, out_image,
                 value, mask_type,
                 length,
                 width);
}  /* ends mask_dilation operation */

         /* interior_outline */
if(strncmp("int", type, 3) == 0){
   interior_outline(the_image, out_image,
                    value, mask_type,
                    length,
                    width);
}  /* ends interior_outline operation */

         /* exterior_outline */
if(strncmp("ext", type, 3) == 0){
   exterior_outline(the_image, out_image,
                    value, mask_type,
                    length,
                    width);
}  /* ends exterior_outline operation */

         /* opening */
if(strncmp("ope", type, 3) == 0){
   opening(the_image, out_image,
           value, mask_type, number,
           length,
           width);
}  /* ends opening operation */

         /* closing */
if(strncmp("clo", type, 3) == 0){
   closing(the_image, out_image,
           value, mask_type, number,
           length,
           width);
}  /* ends closing operation */

         /* special opening */
```

```
   if(strncmp("spo", type, 3) == 0){
      special_opening(the_image, out_image,
                      value, threshold, number,
                      length,
                      width);
   }  /* ends special opening operation */

            /* special closing */
   if(strncmp("spc", type, 3) == 0){
      special_closing(the_image, out_image,
                      value, threshold, number,
                      length,
                      width);
   }  /* ends special closing operation */

            /* Euclidean Distance Measure */
   if(strncmp("edm", type, 3) == 0){
      edm(the_image, out_image,
          value,
          length,
          width);
   }  /* ends Euclidean distance mesaure */

            /* medial axis transform */
   if(strncmp("mat", type, 3) == 0){
      mat(the_image, out_image,
          value,
          length,
          width);
   }  /* ends medial axis transform operation */

   write_image_array(name2, out_image);

   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */




show_mainsk_usage()
{
```

```
   char response[80];

printf(
"\n\nNot enough parameters:"
"\n"
"\n usage: mainsk in-file out-file type value"
" [threshold-or-mask-type] [number]"
"\n"
"\n recall type: EROsion DILation Mask-ERosion"
"\n              Mask_DIlation INTerior-outline"
"\n              EXTerior-outline THInning"
"\n              Dilate-Not-Join OPEning"
"\n              CLOsing SPecial-Opening"
"\n              SPecial-Closing"
"\n              Euclidean-Distance-Measure"
"\n              Medial-Axis-Transform");

printf("\nPress Enter to see more");
gets(response);

printf("\n"
"\n   MASK DILATION"
"\nmainsk in-file out-file mdi value mask-type"
"\n   MASK EROSION"
"\nmainsk in-file out-file mer value mask-type"
"\n   EROSION "
"\nmainsk in-file out-file ero value threshold"
"\n   DILATION "
"\nmainsk in-file out-file dil value threshold"
"\n   INTERIOR OUTLINE "
"\nmainsk in-file out-file int value mask-type"
"\n   EXTERIOR OUTLINE "
"\nmainsk in-file out-file ext value mask-type"
"\n   OPENING"
"\nmainsk in-file out-file ope value mask-type number"
"\n   CLOSING"
"\nmainsk in-file out-file clo value mask-type number");

printf("\n   THINNING"
"\nmainsk in-file out-file thi value threshold"
"\n   DILATE NOT JOIN"
"\nmainsk in-file out-file dnj value threshold"
"\n   SPECIAL CLOSING"
"\nmainsk in-file out-file spc value threshold number"
"\n   SPECIAL OPENING"
```

```
"\nmainsk in-file out-file spo value threshold number"
"\n   EUCLIDEAN DISTANCE MEASURE"
"\nmainsk in-file out-file edm value"
"\n   MEDIAL AXIS TRANSFORM"
"\nmainsk in-file out-file mat value"
"\n "
"\n value is usually 1"
"\n mask-type is 1-4 inclusive"
"\n threshold is 0-8 inclusive"
"\n number is number of erosions or dilations"
"\n");
}
```

Listing 11.3 - The mainsk Program

F.12 Code Listings for Chapter 12

```
/***********************************************
*
*       file boole.c
*
*       Functions: This file contains
*          and_image
*          or_image
*          xor_image
*          nand_image
*          nor_image
*          not_image
*
*       Purpose:
*          These functions implement the basic
*          Boolean algebra functions AND, OR,
*          XOR, NAND, NOR, and NOT.
*
*       External Calls:
*          none
*
*       Modifications:
*          3 March 1993 - created
```

```
    *         22 August 1998 - modified to work on
    *              entire images at once.
    *
    ************************************************/


#include "cips.h"



   /************************************************
   *
   *   and_image(...
   *
   *   This function performs the Boolean AND
   *   operation.  The output image = in1 AND in2.
   *   This works for 0 non-zero images.  If both
   *   in1 and in2 are non-zero, the output = in1.
   *
   ************************************************/

and_image(the_image, out_image,
          rows, cols)
   short  **the_image,
          **out_image;
   long   cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if( the_image[i][j] != 0   &&
             out_image[i][j] != 0)
             out_image[i][j] = the_image[i][j];
         else
             out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends and_image */
```

```
   /**************************************************
   *
   *   or_image(...
   *
   *   This function performs the Boolean OR
   *   operation.  The output image = in1 OR in2.
   *   This works for 0 non-zero images.  If both
   *   in1 and in2 are non-zero, the output = in1.
   *   If in1 is non-zero, the output = in1.
   *   If in1 is zero and in2 is non-zero, the
   *   output = in2.
   *
   **************************************************/

or_image(the_image, out_image,
         rows, cols)
   short **the_image,
         **out_image;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if( the_image[i][j] != 0   ||
             out_image[i][j] != 0){
             if(the_image[i][j] != 0)
                out_image[i][j] = the_image[i][j];
             else
                out_image[i][j] = out_image[i][j];
         }
         else
             out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends or_image */




   /**************************************************
```

```
   *
   *   xor_image(...
   *
   *   This function performs the Boolean XOR
   *   operation.  The output image = in1 XOR in2.
   *   This works for 0 non-zero images.  If
   *   in1 is non-zero and in2 is 0, output = in1. If
   *   in2 is non-zero and in1 is 0, output = in2.
   *   If both in1 and in2 are non-zero, output = 0.
   *   If both in1 and in2 are zero, output = 0.
   *
   ************************************************/

xor_image(the_image, out_image,
          rows, cols)
   short  **the_image,
          **out_image;
   long  cols, rows;
{
   int    i, j;
   short  answer;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if( (the_image[i][j] != 0 &&
              out_image[i][j] == 0))
             answer = the_image[i][j];
         if( (the_image[i][j] == 0 &&
              out_image[i][j] != 0))
             answer = out_image[i][j];
         if( (the_image[i][j] == 0 &&
              out_image[i][j] == 0))
             answer = 0;
         if( (the_image[i][j] != 0 &&
              out_image[i][j] != 0))
             answer = 0;
         out_image[i][j] = answer;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends xor_image */
```

```
   /***********************************************
   *
   *   nand_image(...
   *
   *   This function performs the Boolean NAND
   *   operation.  The output image = in1 NAND in2.
   *   This works for 0 non-zero images.  If both
   *   in1 and in2 are non-zero, the output = 0.
   *   Otherwise, the output = value.
   *
   ***********************************************/

nand_image(the_image, out_image,
           value, rows, cols)
   short   **the_image,
           **out_image, value;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if( the_image[i][j] != 0   &&
             out_image[i][j] != 0)
             out_image[i][j] = 0;
         else
             out_image[i][j] = value;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends nand_image */
```

```
   /***********************************************
   *
   *   nor_image(...
   *
   *   This function performs the Boolean NOR
```

```
   *   operation.  The output image = in1 NOR in2.
   *   This works for 0 non-zero images.  If niether
   *   in1 nor in2 are non-zero, the output = value.
   *   That is, if both in1 and in2 are zero, the
   *   output = value.
   *
   ************************************************/

nor_image(the_image, out_image,
          value, rows, cols)
   short  **the_image,
          **out_image, value;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if( the_image[i][j] == 0   &&
             out_image[i][j] == 0)
             out_image[i][j] = value;
         else
             out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends nor_image */




   /************************************************
   *
   *   not_image(...
   *
   *   This function will complement the values
   *   of the input image and put them into the
   *   output image. It will complement using a
   *   0-value scheme where value is one of the
   *   input parameters.
   *
   ************************************************/
```

```
not_image(the_image, out_image,
          value, rows, cols)
   short  **the_image,
          **out_image,
          value;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
         out_image[i][j] = value;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] == value)
             out_image[i][j] = 0;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends not_image */
```

Listing 12.1 - The Boolean Subroutines

```
/*********************************************
*
*   file boolean.c
*
*   Functions: This file contains
*      main
*
*   Purpose:
*      This file contains the main calling
*      routine that calls the Boolean
*      operations.
*
*   External Calls:
*      imageio.c - create_image_file
*                  read_image_array
```

```
       *                  write_image_array
       *                  get_image_size
       *                  allocate_image_array
       *                  free_image_array
       *      boole.c - and_image
       *                or_image
       *                xor_image
       *                nand_image
       *                nor_image
       *                not_image
       *
       *   Modifications:
       *      3 March 1993 - created
       *      22 August 1998 - modified to work on
       *           entire images at once.
       *      19 September 1998 - modified to work with
       *           all I O routines in imageio.c.
       *
       *********************************************/

#include "cips.h"



short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char     name1[80], name2[80], name3[80], type[80];
   long     length, width;
   short    value;

       /****************************************
       *
       *   Interpret the command line parameters.
       *
       ****************************************/

   if(argc < 5){
    printf(
    "\n\nNot enough parameters:"
```

```
    "\n"
    "\n usage: boolean in-file1 in-file2 out-file "
    "type [value]"
    "\n           or "
    "\n        boolean in-file1 out-file not value"
    "\n"
    "\n recall type: and or xor nand nor not"
    "\n              You must specify a value for "
    "nand & nor"
    "\n");
   exit(0);
  }

  if(strcmp("not", argv[3]) == 0){
     strcpy(name1, argv[1]);
     strcpy(name2, argv[2]);
     strcpy(type,  argv[3]);
     value =  atoi(argv[4]);
  }  /* ends if not */
  else{
     strcpy(name1, argv[1]);
     strcpy(name2, argv[2]);
     strcpy(name3, argv[3]);
     strcpy(type,  argv[4]);
     value =  atoi(argv[5]);
  }  /* ends else all other types */

  if(does_not_exist(name1)){
   printf("\nERROR input file %s does not exist",
            name1);
   exit(0);
  }

      /****************************************
      *
      *  Process the NOT case (1 input file,
      *  1 output file).
      *
      *  Else process the other cases.
      *
      ****************************************/

      /*  NOT CASE */

  if(strcmp("not", type) == 0){
```

```
      get_image_size(name1, &length, &width);
      the_image = allocate_image_array(length, width);
      out_image = allocate_image_array(length, width);
      create_image_file(name1, name2);
      read_image_array(name1, the_image);
      not_image(the_image, out_image, value,
                length, width);
      write_image_array(name2, out_image);
   }  /* ends if not case */



       /* NOW ALL OTHER CASES */

   else{

      if(does_not_exist(name2)){
       printf("\nERROR input file %s does not exist",
                name2);
       exit(0);
      }

      if(are_not_same_size(name1, name2)){
         printf(
          "\n Images %s and %s are not the same size",
          name1, name2);
         exit(1);
      }  /* ends if sizes not the same */

      get_image_size(name1, &length, &width);
      the_image = allocate_image_array(length, width);
      out_image = allocate_image_array(length, width);
      create_image_file(name1, name3);
      read_image_array(name1, the_image);
      read_image_array(name2, out_image);


            /* AND */
      if(strcmp("and", type) == 0){
         and_image(the_image, out_image,
                   length,
                   width);
      }  /* ends AND operation */

            /* OR */
```

```
      if(strcmp("or", type) == 0){
         or_image(the_image, out_image,
                  length,
                  width);
      }  /* ends OR operation */

            /* XOR */
      if(strcmp("xor", type) == 0){
         xor_image(the_image, out_image,
                   length,
                   width);
      }  /* ends XOR operation */

            /* NAND */
      if(strcmp("nand", type) == 0){
         nand_image(the_image, out_image,
                    value,
                    length,
                    width);
      }  /* ends NAND operation */

            /* NOR */
      if(strcmp("nor", type) == 0){
         nor_image(the_image, out_image,
                   value,
                   length,
                   width);
      }  /* ends NOR operation */

      write_image_array(name3, out_image);

   }  /* ends else all other cases (not not) */


   free_image_array(out_image, length);
   free_image_array(the_image, length);


}  /* ends main  */
```

Listing 12.2 - The boolean Program

```
    /***********************************************
    *
    *   file ilabel.c
    *
    *   Functions: This file contains
    *       main
    *
    *   Purpose:
    *      This program writes simple block letters
    *      the an image file.  You can use these
    *      as labels for other images.
    *
    *   External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *
    *   Modifications:
    *      21 May 1993 - created
    *      22 August 1998 - modified to work on entire
    *           images at once.
    *      19 September 1998 - modified to work with
    *            all I O routines in imageio.c.
    *
    ***********************************************/


#include "cips.h"

#define R             9
#define C             7
#define COUNTER_LIMIT 8
#define IE_START      7
#define VAL         200

short **image;

         /******************************
         *
         *   Define all the 9x7 arrays
         *   that contain the characters.
```

```
         *
         ******************************/

short aperiod[R][C] = {
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,  0,  0,  0},
                    {  0,  0,VAL,VAL,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short acomma[R][C] = {
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,  0,  0,  0},
                    {  0,  0,VAL,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0}};

short aexclam[R][C] = {
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short xx[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};
```

```
short aa[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,VAL,  0,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short ab[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short ac[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short ad[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short ae[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
```

```
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short af[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ag[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ah[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ai[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};
```

```
short aj[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,  0,VAL,  0,  0,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short ak[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,VAL,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,VAL,  0,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,  0,  0,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short al[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short am[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,  0,VAL,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short an[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,VAL,  0,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
```

```
                    {  0,VAL,  0,VAL,  0,VAL,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ao[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ap[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short aq[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,VAL,  0,VAL,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ar[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};
```

```
short as[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short at[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short au[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short av[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,  0,VAL,  0,  0},
                   {  0,  0,VAL,  0,VAL,  0,  0},
                   {  0,  0,VAL,  0,VAL,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short aw[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
                   {  0,VAL,  0,VAL,  0,VAL,  0},
```

```
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,VAL,  0,VAL,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ax[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,  0,VAL,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,VAL,  0,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short ay[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,  0,VAL,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short az[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,VAL,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,VAL,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short a1[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,VAL,VAL,  0,  0,  0},
                    {  0,VAL,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};
```

```
short a2[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,  0,  0,VAL,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,VAL,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short a3[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short a4[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,  0,VAL,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,VAL,  0,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short a5[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,VAL,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,  0,  0,  0,  0,  0,  0}};

short a6[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                   {  0,  0,VAL,VAL,VAL,  0,  0},
                   {  0,VAL,  0,  0,  0,VAL,  0},
                   {  0,VAL,  0,  0,  0,  0,  0},
                   {  0,VAL,VAL,VAL,VAL,  0,  0},
```

```
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short a7[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,VAL,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,VAL,  0,  0},
                    {  0,  0,  0,VAL,  0,  0,  0},
                    {  0,  0,VAL,  0,  0,  0,  0},
                    {  0,VAL,  0,  0,  0,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short a8[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short a9[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,VAL,  0,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,VAL,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};

short a0[R][C] = { {  0,  0,  0,  0,  0,  0,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,VAL,  0,  0,VAL,VAL,  0},
                    {  0,VAL,  0,VAL,  0,VAL,  0},
                    {  0,VAL,  0,VAL,  0,VAL,  0},
                    {  0,VAL,VAL,  0,  0,VAL,  0},
                    {  0,  0,VAL,VAL,VAL,  0,  0},
                    {  0,  0,  0,  0,  0,  0,  0}};
```

```
main(argc, argv)
   int  argc;
   char *argv[];
{
   int    l=1, w=1;
   int    counter=0, i, j, il, ie=7, ll, le;
   long   length, width;

   if(argc < 5){
      printf("\n usage: ilabel file-name il ie text"
      "\n  the file-name image must already exist");
      exit(0);
   }

      /****************************************
      *
      *   Ensure the file exists.
      *   Allocate an image array and read
      *   the image.
      *
      ****************************************/

   if(does_not_exist(argv[1])){
      printf("\nFile %s does not exist \nCreate it", argv[1]);
      exit(0);
   }  /* ends if does not exist */
   else{  /* else it does exist */
      get_image_size(argv[1], &length, &width);
      image = allocate_image_array(length, width);
      read_image_array(argv[1], image);
   }  /* ends else it does exist */

   il = atoi(argv[2]);
   ie = atoi(argv[3]);

         /******************************
         *
         *   Loop through the text
         *   arguments and place the
         *   letter arrays into the
         *   image.
         *
         ******************************/
```

```
printf("\n");
for(i=4; i<argc; i++){
   for(j=0; j<(strlen(argv[i])); j++){

      argv[i][j] = tolower(argv[i][j]);

      printf("%c", argv[i][j]);
      printf("%d %d\n",il, ie);
      if(argv[i][j] == 'a')
         copy_array_into_image(aa, image, il, ie);
      if(argv[i][j] == 'b')
         copy_array_into_image(ab, image, il, ie);
      if(argv[i][j] == 'c')
         copy_array_into_image(ac, image, il, ie);
      if(argv[i][j] == 'd')
         copy_array_into_image(ad, image, il, ie);
      if(argv[i][j] == 'e')
         copy_array_into_image(ae, image, il, ie);
      if(argv[i][j] == 'f')
         copy_array_into_image(af, image, il, ie);
      if(argv[i][j] == 'g')
         copy_array_into_image(ag, image, il, ie);
      if(argv[i][j] == 'h')
         copy_array_into_image(ah, image, il, ie);
      if(argv[i][j] == 'i')
         copy_array_into_image(ai, image, il, ie);
      if(argv[i][j] == 'j')
         copy_array_into_image(aj, image, il, ie);
      if(argv[i][j] == 'k')
         copy_array_into_image(ak, image, il, ie);
      if(argv[i][j] == 'l')
         copy_array_into_image(al, image, il, ie);
      if(argv[i][j] == 'm')
         copy_array_into_image(am, image, il, ie);
      if(argv[i][j] == 'n')
         copy_array_into_image(an, image, il, ie);
      if(argv[i][j] == 'o')
         copy_array_into_image(ao, image, il, ie);
      if(argv[i][j] == 'p')
         copy_array_into_image(ap, image, il, ie);
      if(argv[i][j] == 'q')
         copy_array_into_image(aq, image, il, ie);
      if(argv[i][j] == 'r')
         copy_array_into_image(ar, image, il, ie);
```

```
if(argv[i][j] == 's')
   copy_array_into_image(as, image, il, ie);
if(argv[i][j] == 't')
   copy_array_into_image(at, image, il, ie);
if(argv[i][j] == 'u')
   copy_array_into_image(au, image, il, ie);
if(argv[i][j] == 'v')
   copy_array_into_image(av, image, il, ie);
if(argv[i][j] == 'w')
   copy_array_into_image(aw, image, il, ie);
if(argv[i][j] == 'x')
   copy_array_into_image(ax, image, il, ie);
if(argv[i][j] == 'y')
   copy_array_into_image(ay, image, il, ie);
if(argv[i][j] == 'z')
   copy_array_into_image(az, image, il, ie);
if(argv[i][j] == '1')
   copy_array_into_image(a1, image, il, ie);
if(argv[i][j] == '2')
   copy_array_into_image(a2, image, il, ie);
if(argv[i][j] == '3')
   copy_array_into_image(a3, image, il, ie);
if(argv[i][j] == '4')
   copy_array_into_image(a4, image, il, ie);
if(argv[i][j] == '5')
   copy_array_into_image(a5, image, il, ie);
if(argv[i][j] == '6')
   copy_array_into_image(a6, image, il, ie);
if(argv[i][j] == '7')
   copy_array_into_image(a7, image, il, ie);
if(argv[i][j] == '8')
   copy_array_into_image(a8, image, il, ie);
if(argv[i][j] == '9')
   copy_array_into_image(a9, image, il, ie);
if(argv[i][j] == '0')
   copy_array_into_image(a0, image, il, ie);
if(argv[i][j] == '.')
   copy_array_into_image(aperiod, image,
                         il, ie);
if(argv[i][j] == ',')
   copy_array_into_image(acomma, image,
                         il, ie);
if(argv[i][j] == '!')
   copy_array_into_image(aexclam, image,
                         il, ie);
```

```
         ie = ie + C;

      }  /* ends loop over j letters in argument */

            /* Put a space between words */
         copy_array_into_image(xx, image, il, ie);
         ie = ie + C;

   }  /* ends loop over i arguments */

   write_image_array(argv[1], image);
   free_image_array(image, length);

}  /* ends main */




copy_array_into_image(a, the_image, il, ie)
   short a[R][C], **the_image;
   int   il, ie;
{
   int i, j;
   for(i=0; i<R; i++)
      for(j=0; j<C; j++)
         the_image[il+i][ie+j] = a[i][j];

}  /* ends copy_array_into_image */
```

Listing 12.3 - The ilabel Program

```
    /*********************************************
    *
    *       file overlay.c
    *
```

```
    *       Functions: This file contains
    *          non_zero_overlay
    *          zero_overlay
    *          greater_overlay
    *          less_overlay
    *          average_overlay
    *
    *       Purpose:
    *          These functions implement the
    *          functions that overlay one image
    *          on top of another image.
    *
    *       External Calls:
    *          none
    *
    *       Modifications:
    *          6 March 1993 - created
    *         22 August 1998 - modified to work on
    *               entire images at once.
    *
    ********************************************/


#include "cips.h"



   /*********************************************
   *
   *   non_zero_overlay(...
   *
   *   This function overlays in1 on top of in2
   *   and writes the result to the output image.
   *   It writes any non-zero pixel from in1 on top
   *   of in2.
   *
   *********************************************/

non_zero_overlay(the_image, out_image,
                 rows, cols)
   short  **the_image,
          **out_image;
   long   cols, rows;
{
   int    i, j;
```

```
   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] != 0)
            out_image[i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends non_zero_overlay */




   /***********************************************
   *
   *   zero_overlay(...
   *
   *   This function overlays in1 on top of in2
   *   and writes the result to the output image.
   *   It writes any zero pixel from in1 on top
   *   of in2.
   *
   ***********************************************/

zero_overlay(the_image, out_image,
             rows, cols)
   short **the_image,
         **out_image;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] == 0)
            out_image[i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends zero_overlay */
```

```
   /**********************************************
   *
   *   greater_overlay(...
   *
   *   This function overlays in1 on top of in2
   *   and writes the result to the output image.
   *   It writes in1 on top of in2 if the value of
   *   in1 is greater than in2.
   *
   **********************************************/

greater_overlay(the_image, out_image,
                rows, cols)
   short **the_image,
         **out_image;
   long  cols, rows;
{
   int    i, j;

   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] > out_image[i][j])
            out_image[i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends greater_overlay */
```

```
   /**********************************************
   *
   *   less_overlay(...
   *
   *   This function overlays in1 on top of in2
   *   and writes the result to the output image.
   *   It writes in1 on top of in2 if the value of
   *   in1 is less than in2.
   *
```

```
   ***********************************************/

less_overlay(the_image, out_image,
             rows, cols)
   short **the_image,
         **out_image;
   long  cols, rows;
{
   int    i, j;


   for(i=0; i<rows; i++){
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         if(the_image[i][j] < out_image[i][j])
            out_image[i][j] = the_image[i][j];
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends less_overlay */




   /***********************************************
   *
   *   average_overlay(...
   *
   *   This function mixes in1 and in2
   *   and writes the result to the output image.
   *   It writes the average of in1 and in2 to the
   *   output image.
   *
   ***********************************************/

average_overlay(the_image, out_image,
                rows, cols)
   short **the_image,
         **out_image;
   long  cols, rows;
{
   int    i, j;

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

```
      if ( (i%10) == 0) printf(" %d", i);
      for(j=0; j<cols; j++){
         out_image[i][j] =
            (the_image[i][j] + out_image[i][j])/2;
      }  /* ends loop over j */
   }  /* ends loop over i */

} /* ends average_overlay */
```

Listing 12.4 - The Overlay Subroutines

```
/*********************************************
*
*   file mainover.c
*
*   Functions: This file contains
*      main
*
*   Purpose:
*      This file contains the main calling
*      routine that calls the overlay functions.
*
*   External Calls:
*      imageio.c - create_image_file
*                  read_image_array
*                  write_image_array
*                  get_image_size
*                  allocate_image_array
*                  free_image_array
*      overlay.c - non_zero_overlay
*                  zero_overlay
*                  greater_overlay
*                  less_overlay
*                  average_overlay
*
*   Modifications:
*      6 March 1993 - created
*      22 August 1998 - modified to work on
*           entire images at once.
*      19 September 1998 - modified to work with
*           all I O routines in imageio.c.
```

```
       *
       ********************************************/

#include "cips.h"



short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char     name1[80], name2[80], name3[80], type[80];
   int      count, i, j;
   long     length, width;

       /*********************************************
       *
       *   Interpret the command line parameters.
       *
       **********************************************/

   if(argc < 5){
    printf(
    "\n\nNot enough parameters:"
     "\n"
     "\n usage: mainover in-file1 in-file2 out-file "
     "type"
     "\n"
     "\n recall type: nonzero zero greater less"
     " average"
     "\n the input images must be the same size"
     "\n"
     "\n");
    exit(0);
   }

   strcpy(name1,  argv[1]);
   strcpy(name2, argv[2]);
   strcpy(name3, argv[3]);
   strcpy(type,  argv[4]);
```

```
if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

if(does_not_exist(name2)){
 printf("\nERROR input file %s does not exist",
          name2);
 exit(0);
}

    /*********************************************
    *
    *   Read the input image headers.
    *   Ensure the input image are the same size.
    *   Allocate the image arrays and read
    *   the image data.
    *
    **********************************************/

if(are_not_same_size(name1, name2)){
   printf(
    "\n Images %s and %s are not the same size",
    name1, name2);
   exit(1);
}  /* ends if sizes not the same */

get_image_size(name1, &length, &width);
the_image = allocate_image_array(length, width);
out_image = allocate_image_array(length, width);
create_image_file(name1, name3);
read_image_array(name1, the_image);
read_image_array(name2, out_image);

    /*********************************************
    *
    *   Apply the desired overlay function.
    *
    **********************************************/

         /* non-zero */
if(strncmp("non", type, 3) == 0){
   non_zero_overlay(the_image, out_image,
                    length,
```

```
                      width);
   }  /* ends non_zero operation */

      /* zero */
   if(strcmp("zero", type) == 0){
      zero_overlay(the_image, out_image,
                   length,
                   width);
   }  /* ends zero operation */

      /* greater */
   if(strncmp("gre", type, 3) == 0){
      greater_overlay(the_image, out_image,
                      length,
                      width);
   }  /* ends greater operation */

      /* less */
   if(strncmp("les", type, 3) == 0){
      less_overlay(the_image, out_image,
                   length,
                   width);
   }  /* ends less operation */

      /* average */
   if(strncmp("ave", type, 3) == 0){
      average_overlay(the_image, out_image,
                      length,
                      width);
   }  /* ends average operation */

   write_image_array(name3, out_image);
   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */
```

Listing 12.5 - The mainover Program

F.13 Code Listings for Chapter 13

```
    /*******************************************
    *
    *    file geosubs.c
    *
    *    Functions: This file contains
    *       geometry
    *       arotate
    *       bilinear_interpolate
    *
    *    Purpose:
    *       These functions performs different
    *       geometric operations.
    *
    *    External Calls:
    *       none
    *
    *    Modifications:
    *       20 October 1993- created
    *       27 August 1998 - modified to work on
    *            entire images at once.
    *
    *******************************************/

#include "cips.h"

#define FILL 150
```

```
     /*******************************************
     *
     *   geometry(..
     *
     *   This routine performs geometric
     *   transformations on the pixels in an
     *   image array.  It performs basic
     *   displacement, stretching, and rotation.
     *
     *   The basic equations are:
     *
     *   new x = x.cos(a) + y.sin(a) + x_displace
     *           + x.x_stretch +x.y.x_cross
```

```
     *
     *   new y = y.cos(a) - x.sin(a) + y_displace
     *           + y.y_stretch +x.y.y_cross
     *
     ******************************************/

geometry(the_image, out_image,
        x_angle,
        x_stretch, y_stretch,
        x_displace, y_displace,
        x_cross, y_cross,
        bilinear,
        rows,
        cols)

   float  x_angle, x_stretch, y_stretch,
          x_cross, y_cross;
   int    bilinear;
   long   cols, rows;
   short  **the_image,
          **out_image,
          x_displace, y_displace;
{
   double cosa, sina, radian_angle, tmpx, tmpy;
   float  fi, fj, x_div, y_div, x_num, y_num;
   int    i, j, new_i, new_j;


      /******************************
      *
      *   Load the terms array with
      *   the correct parameters.
      *
      *******************************/

      /* the following magic number is from
         180 degrees divided by pi */
   radian_angle = x_angle/57.29577951;
   cosa  = cos(radian_angle);
   sina  = sin(radian_angle);

      /************************************
      *
      *   NOTE: You divide by the
      *   stretching factors. Therefore, if
```

```
   *   they are zero, you divide by 1.
   *   You do this with the x_div y_div
   *   variables. You also need a
   *   numerator term to create a zero
   *   product.  You do this with the
   *   x_num and y_num variables.
   *
   ************************************/

if(x_stretch < 0.00001){
   x_div = 1.0;
   x_num = 0.0;
}
else{
   x_div = x_stretch;
   x_num = 1.0;
}

if(y_stretch < 0.00001){
   y_div = 1.0;
   y_num = 0.0;
}
else{
   y_div = y_stretch;
   y_num = 1.0;
}

   /**************************
   *
   *   Loop over image array
   *
   **************************/

printf("\n");
for(i=0; i<rows; i++){
   if( (i%10) == 0) printf("%d ", i);
   for(j=0; j<cols; j++){

      fi = i;
      fj = j;

      tmpx = (double)(j)*cosa         +
             (double)(i)*sina         +
             (double)(x_displace)     +
             (double)(x_num*fj/x_div) +
```

```
                (double)(x_cross*i*j);

         tmpy = (double)(i)*cosa         -
                (double)(j)*sina         +
                (double)(y_displace)     +
                (double)(y_num*fi/y_div) +
                (double)(y_cross*i*j);

         if(x_stretch != 0.0)
            tmpx = tmpx - (double)(fj*cosa + fi*sina);
         if(y_stretch != 0.0)
            tmpy = tmpy - (double)(fi*cosa - fj*sina);

         new_j = tmpx;
         new_i = tmpy;

         if(bilinear == 0){
            if(new_j < 0       ||
               new_j >= cols   ||
               new_i < 0       ||
               new_i >= rows)
               out_image[i][j] = FILL;
            else
               out_image[i][j] =
                the_image[new_i][new_j];
         }  /* ends if bilinear */
         else{
            out_image[i][j] =
               bilinear_interpolate(the_image,
                                    tmpx, tmpy,
                                    rows, cols);
         }  /* ends bilinear if */

      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends geometry */



     /*****************************************
     *
     *   arotate(..
     *
```

```
     *   This routine performs rotation about
     *   any point m,n.
     *
     *   The basic equations are:
     *
     *   new x = x.cos(a) - y.sin(a)
     *           -m.cos(a) + m + n.sin(a)
     *
     *   new y = y.cos(a) + x.sin(a)
     *           -m.sin(a) - n.cos(a) + n
     *
     *****************************************/

arotate(the_image, out_image,
        angle,
        m, n, bilinear,
        rows, cols)
   float  angle;
   int    bilinear;
   long   cols, rows;
   short  **the_image,
          **out_image,
          m, n;
{
   double cosa, sina, radian_angle, tmpx, tmpy;
   int    i, j, new_i, new_j;


      /* the following magic number is from
         180 degrees divided by pi */
   radian_angle = angle/57.29577951;
   cosa  = cos(radian_angle);
   sina  = sin(radian_angle);

      /**************************
      *
      *   Loop over image array
      *
      **************************/

   printf("\n");
   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<cols; j++){
```

```
   /*****************************************
   *
   *   new x = x.cos(a) - y.sin(a)
   *           -m.cos(a) + m + n.sin(a)
   *
   *   new y = y.cos(a) + x.sin(a)
   *           -m.sin(a) - n.cos(a) + n
   *
   *****************************************/

       tmpx = (double)(j)*cosa    -
              (double)(i)*sina    -
              (double)(m)*cosa    +
              (double)(m)         +
              (double)(n)*sina;

       tmpy = (double)(i)*cosa    +
              (double)(j)*sina    -
              (double)(m)*sina    -
              (double)(n)*cosa    +
              (double)(n);

       new_j = tmpx;
       new_i = tmpy;

       if(bilinear == 0){
          if(new_j < 0       ||
             new_j >= cols   ||
             new_i < 0       ||
             new_i >= rows)
             out_image[i][j] = FILL;
          else
             out_image[i][j] =
              the_image[new_i][new_j];
       }  /* ends if bilinear */
       else{
          out_image[i][j] =
             bilinear_interpolate(the_image,
                                  tmpx, tmpy,
                                  rows, cols);
       }  /* ends bilinear if */

    }  /* ends loop over j */
 }  /* ends loop over i */
```

```
} /* ends arotate */




     /*****************************************
     *
     *   bilinear_interpolate(..
     *
     *   This routine performs bi-linear
     *   interpolation.
     *
     *   If x or y is out of range, i.e. less
     *   than zero or greater than rows or cols,
     *   this routine returns a zero.
     *
     *   If x and y are both in range, this
     *   routine interpolates in the horizontal
     *   and vertical directions and returns
     *   the proper gray level.
     *
     *****************************************/

bilinear_interpolate(the_image, x, y, rows, cols)
   double x, y;
   long   cols, rows;
   short  **the_image;
{
   double fraction_x, fraction_y,
          one_minus_x, one_minus_y,
          tmp_double;
   int    ceil_x, ceil_y, floor_x, floor_y;
   short  p1, p2, p3, result = FILL;

      /******************************
      *
      *   If x or y is out of range,
      *   return a FILL.
      *
      ******************************/

   if(x < 0.0                ||
      x >= (double)(cols-1)   ||
      y < 0.0                ||
```

```
      y >= (double)(rows-1))
      return(result);

   tmp_double = floor(x);
   floor_x    = tmp_double;
   tmp_double = floor(y);
   floor_y    = tmp_double;
   tmp_double = ceil(x);
   ceil_x     = tmp_double;
   tmp_double = ceil(y);
   ceil_y     = tmp_double;

   fraction_x = x - floor(x);
   fraction_y = y - floor(y);

   one_minus_x = 1.0 - fraction_x;
   one_minus_y = 1.0 - fraction_y;

   tmp_double = one_minus_x *
          (double)(the_image[floor_y][floor_x]) +
          fraction_x *
          (double)(the_image[floor_y][ceil_x]);
   p1         = tmp_double;

   tmp_double = one_minus_x *
          (double)(the_image[ceil_y][floor_x]) +
          fraction_x *
          (double)(the_image[ceil_y][ceil_x]);
   p2         = tmp_double;

   tmp_double = one_minus_y * (double)(p1) +
          fraction_y * (double)(p2);
   p3         = tmp_double;


   return(p3);

}  /* ends bilinear_interpolate */



#ifdef NEVER

     /*****************************************
     *
```

```
     *   get_geometry_options(..
     *
     *   This routine interacts with the user
     *   to obtain the parameters to call the
     *   geometry operations subroutines.
     *
     ******************************************/

get_geometry_options(operation, angle,
                     x_displace, y_displace,
                     x_stretch, y_stretch,
                     x_cross,  y_cross,
                     bilinear, m, n)
   char  operation[];
   int   *bilinear;
   short *m, *n, *x_displace, *y_displace;
   float *angle,
         *x_cross, *y_cross,
         *x_stretch, *y_stretch;
{
   int not_finished, response;
   not_finished = 1;
   while(not_finished){

      printf("\nThe geomety options are:");
      printf("\n\t1. Operation is %s", operation);
      printf("\n\t2. Angle is %f", *angle);
      printf("\n\t3. x-displace=%d y-displace=%d",
         *x_displace, *y_displace);
      printf("\n\t4. x-stretch=%f y-stretch=%f",
         *x_stretch, *y_stretch);
      printf("\n\t5. x-cross=%f y-cross=%f",
         *x_cross, *y_cross);
      printf("\n\t6. bilinear = %d", *bilinear);
      printf("\n\t7. rotation points m=%d n=%d",
         *m, *n);
      printf("\n\nExamples:");
      printf("\ngeometry needs: angle");
      printf(" x-displace y-displace");
      printf(" x-stretch y-stretch");
      printf("\n                x-cross y-cross");
      printf(" bilinear (1 or 0)");
      printf("\nrotate needs: angle m n");
      printf(" bilinear (1 or 0)");
      printf("\n\nEnter choice (0 = no change) _\b");
```

```
get_integer(&response);

if(response == 0)
   not_finished = 0;

if(response == 1){
   printf("\nEnter operation:");
   gets(operation);
}  /* ends if 1 */

if(response == 2){
   printf("\nEnter angle: ___\b\b\b");
   get_float(angle);
}  /* ends if 2 */

if(response == 3){
   printf("\nEnter x-displace: ___\b\b\b");
   get_integer(x_displace);
   printf("\nEnter y-displace: ___\b\b\b");
   get_integer(y_displace);
}  /* ends if 3 */

if(response == 4){
   printf("\nEnter x-stretch: ___\b\b\b");
   get_float(x_stretch);
   printf("\nEnter y-stretch: ___\b\b\b");
   get_float(y_stretch);
}  /* ends if 4 */

if(response == 5){
   printf("\nEnter x-cross: ___\b\b\b");
   get_float(x_cross);
   printf("\nEnter y-cross: ___\b\b\b");
   get_float(y_cross);
}  /* ends if 5 */

if(response == 6){
   printf("\nEnter bilinear: _\b");
   get_integer(bilinear);
}  /* ends if 6 */

if(response == 7){
   printf("\nEnter rotation point m: _\b");
   get_integer(m);
   printf("\nEnter rotation point n: _\b");
```

```
         get_integer(n);
      }  /* ends if 7 */


   }  /* ends while not_finished */

}  /* ends get_geometry_options */

#endif
```

Listing 13.1 - The Geometry Subroutines

```
/*******************************************
*
*    file d:\cips\geometry.c
*
*    Functions: This file contains
*       main
*
*    Purpose:
*       This file contains the main calling
*       routine for geometric subroutines.
*
*    External Calls:
*      imageio.c - create_image_file
*                  read_image_array
*                  write_image_array
*                  get_image_size
*                  allocate_image_array
*                  free_image_array
*       geosubs.c - geometry
*                   arotate
*
*    Modifications:
*       26 October 1993 - created
*       27 August 1998 - modified to work on
*            entire images at once.
*       19 September 1998 - modified to work with
*           all I O routines in imageio.c.
*
*******************************************/
```

```
#include "cips.h"


short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char   name1[80], name2[80], type[80];
   float  theta, x_stretch, y_stretch,
          x_cross, y_cross;
   int    bilinear;
   int    x_control, y_control;
   long   length, width;
   short  m, n, x_displace, y_displace;

      /*************************************
      *
      *   This program will use a different
      *   command line for each type of
      *   call.
      *
      *   Print a usage statement that
      *   gives an example of each type
      *   of call.
      *
      *************************************/

   if(argc < 7){
    printf("\n\nNot enough parameters:");
    printf("\n");
    printf("\n   Two Operations: ");
    printf("\n      geometry  rotate");
    printf("\n\n   Examples:");
    printf("\n");
    printf("\n   geometry in out geometry angle");
    printf(" x-displace y-displace");
    printf("\n            x-stretch y-stretch");
    printf(" x-cross y-cross bilinear (1 or 0)");
    printf("\n");
    printf("\n   geometry in out rotate angle m n");
```

```
 printf(" bilinear (1 or 0)");
 printf("\n");
 exit(0);
}


   /*************************************
   *
   *   Interpret the command line
   *   depending on the type of call.
   *
   *************************************/

if(strncmp(argv[3], "geometry", 3) == 0){
   strcpy(name1,  argv[1]);
   strcpy(name2, argv[2]);
   strcpy(type,  argv[3]);
   theta      = atof(argv[4]);
   x_displace = atoi(argv[5]);
   y_displace = atoi(argv[6]);
   x_stretch  = atof(argv[7]);
   y_stretch  = atof(argv[8]);
   x_cross    = atof(argv[9]);
   y_cross    = atof(argv[10]);
   bilinear   = atoi(argv[11]);
}

if(strncmp(argv[3], "rotate", 3) == 0){
   strcpy(name1,  argv[1]);
   strcpy(name2, argv[2]);
   strcpy(type,  argv[3]);
   theta    = atof(argv[4]);
   m        = atoi(argv[5]);
   n        = atoi(argv[6]);
   bilinear = atoi(argv[7]);
}

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

get_image_size(name1, &length, &width);
the_image = allocate_image_array(length, width);
```

```
   out_image = allocate_image_array(length, width);
   create_image_file(name1, name2);
   read_image_array(name1, the_image);

      /************************************
      *
      *   Call the routines
      *
      ************************************/

   if(strncmp(type, "geometry", 3) == 0){
     geometry(the_image, out_image,
              theta, x_stretch, y_stretch,
              x_displace, y_displace,
              x_cross, y_cross,
              bilinear,
              length,
              width);
   }  /* ends if */



   if(strncmp(type, "rotate", 3) == 0){
     arotate(the_image, out_image,
             theta, m, n, bilinear,
             length,
             width);
   }  /* ends if */


   write_image_array(name2, out_image);
   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */
```

Listing 13.2 - The geometry Program

```
    /*******************************************
    *
```

```
    *   file stretch.c
    *
    *   Functions: This file contains
    *      main
    *      stretch
    *      bilinear_interpolate
    *
    *   Purpose:
    *      This file contains the main calling
    *      routine and the needed subroutines
    *      for a program which stretches
    *      an image by any factor.  It can either
    *      roundoff the numbers or use
    *      bi-linear interpolation.
    *
    *   External Calls:
    *      imageio.c - create_resized_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *
    *   Modifications:
    *      4 December 1993 - created
    *     16 September 1998 - modified to work on entire
    *         images at one time.
    *     22 September 1998 - modified to work with
    *           all I O routines in imageio.c.
    *
    *******************************************/

#include "cips.h"
#define FILL 150


short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char     in_name[80], out_name[80];
```

```
float    x_stretch, y_stretch;

int      bilinear;
long     tmp_length, tmp_width;
long     length, width;

struct bmpfileheader      bmp_file_header;
struct bitmapheader       bmheader;
struct bitmapheader       bmheader2;
struct tiff_header_struct tiff_file_header;
struct tiff_header_struct tiff_file_header2;

    /*******************************************
    *
    *  Interpret the command line parameters.
    *
    *******************************************/

if(argc < 6 || argc > 6){
 printf(
  "\n"
  "\n usage: stretch in-file out-file x-stretch "
  "y-stretch bilinear (1 or 0)"
  "\n");
 exit(0);
}

strcpy(in_name,  argv[1]);
strcpy(out_name, argv[2]);
x_stretch = atof(argv[3]);
y_stretch = atof(argv[4]);
bilinear  = atoi(argv[5]);

if(does_not_exist(in_name)){
 printf("\nERROR input file %s does not exist",
          in_name);
 exit(0);
}

    /*********************************************
    *
    *   Create an output file different in size
    *   from the input file.
    *
```

```
       *****************************************/

   get_image_size(in_name, &length, &width);
   tmp_length = ((float)(length)*y_stretch);
   tmp_width  = ((float)(width)*x_stretch);
   create_resized_image_file(in_name, out_name,
                             tmp_length, tmp_width);

   the_image = allocate_image_array(length, width);
   out_image = allocate_image_array(tmp_length, tmp_width);

   read_image_array(in_name, the_image);

   stretch(the_image, out_image,
           x_stretch, y_stretch,
           bilinear,
           tmp_length,
           tmp_width,
           length,
           width);

   write_image_array(out_name, out_image);
   free_image_array(out_image, tmp_length);
   free_image_array(the_image, length);

}  /* ends main  */



     /*****************************************
     *
     *   stretch(..
     *
     *   This routine performs the image
     *   stretching. If bilinear == 0, it uses
     *   the roundoff approach for enlarging
     *   an area. If bilinear == 1, it calls the
     *   bilinear_interpolate routine to get
     *   the value of a pixel that lies
     *   between pixels.
     *
     *****************************************/

stretch(the_image, out_image,
```

```
        x_stretch, y_stretch,
        bilinear,
        out_rows, out_cols,
        in_rows,  in_cols)

   float  x_stretch, y_stretch;
   int    bilinear;
   short  **the_image,
          **out_image;
   long   out_cols, out_rows;
   long   in_cols,  in_rows;
{
   double tmpx, tmpy;
   float  fi, fj;
   int    i, j, new_i, new_j;

      /*************************
      *
      *   Loop over image array
      *
      *************************/

   printf("\n");
   for(i=0; i<out_rows; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<out_cols; j++){

         fi = i;
         fj = j;

         tmpx = fj/x_stretch;
         tmpy = fi/y_stretch;

         new_i = tmpy;
         new_j = tmpx;

         if(bilinear == 0){
            if(new_j < 0       ||
               new_j >= in_cols   ||
               new_i < 0       ||
               new_i >= in_rows)
               out_image[i][j] = FILL;
            else
               out_image[i][j] =
                the_image[new_i][new_j];
```

```
         }  /* ends if bilinear */
         else{
            out_image[i][j] =
               bilinear_interpolate(the_image,
                                    tmpx, tmpy,
                                    in_rows, in_cols);
         }  /* ends bilinear if */

      }  /* ends loop over j */
   }  /* ends loop over i */

}  /* ends stretch */
```

Listing 13.3 - The stretch Program

F.14 Code Listings for Chapter 14

```
/***********************************************
*
*    file warpsubs.c
*
*    Functions: This file contains
*       warp
*       warp_loop
*       bi_warp_loop
*       object_warp
*       full_warp_loop
*       bi_full_warp_loop
*       get_warp_options
*
*    Purpose:
*       These functions performs different
*       geometric operations.
*
*    External Calls:
*       geosubs.c - bilinear_interpolate
*
*    Modifications:
*       20 October 1993- created
```

```
    *       27 August 1998 - modified to work on
    *            entire images at once.
    *
    ************************************************/

#include "cips.h"

#define FILL 150


     /*******************************************
     *
     *   warp(..
     *
     *   This routine warps a rowsxcols section
     *   of an image.  The il, ie parameters
     *   specify which rowsxcols section of
     *   the image to warp.  The x_control and
     *   y_control parameters are the control
     *   points inside that section.  Therefore,
     *   x_control and y_control will always be
     *   less the cols and rows.
     *
     *   The point coordinates are for the four
     *   corners of a four side figure.
     *      x1,y1     x2,y2
     *
     *      x4,y4     x3,y3
     *
     *******************************************/

warp(the_image, out_image,
     x_control, y_control,
     bilinear,
     rows, cols)
   int    bilinear, x_control, y_control;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int    cols_div_2, extra_x, extra_y, i, j,
          rows_div_2, x1, x2, x3, x4, y1, y2, y3, y4;

   cols_div_2 = cols/2;
   rows_div_2 = rows/2;
```

```
   /**********************************
   *
   *   1 - upper left quarter
   *
   **********************************/

x1 = 0;
x2 = cols_div_2;
x3 = x_control;
x4 = 0;

y1 = 0;
y2 = 0;
y3 = y_control;
y4 = rows_div_2;

extra_x = 0;
extra_y = 0;

if(bilinear)
   bi_warp_loop(the_image, out_image,
                x1, x2, x3, x4,
                y1, y2, y3, y4,
                extra_x, extra_y,
                rows, cols);
else
   warp_loop(the_image, out_image,
             x1, x2, x3, x4,
             y1, y2, y3, y4,
             extra_x, extra_y,
             rows, cols);

   /**********************************
   *
   *   2 - upper right quarter
   *
   **********************************/

x1 = cols_div_2;
x2 = cols-1;
x3 = cols-1;
x4 = x_control;

y1 = 0;
```

```
y2 = 0;
y3 = rows_div_2;
y4 = y_control;

extra_x = cols_div_2;
extra_y = 0;


if(bilinear)
   bi_warp_loop(the_image, out_image,
                x1, x2, x3, x4,
                y1, y2, y3, y4,
                extra_x, extra_y,
                rows, cols);
else
   warp_loop(the_image, out_image,
             x1, x2, x3, x4,
             y1, y2, y3, y4,
             extra_x, extra_y,
             rows, cols);

   /************************************
   *
   *   3 - lower right quarter
   *
   ************************************/

x1 = x_control;
x2 = cols-1;
x3 = cols-1;
x4 = cols_div_2;

y1 = y_control;
y2 = rows_div_2;
y3 = rows-1;
y4 = rows-1;

extra_x = cols_div_2;
extra_y = rows_div_2;

if(bilinear)
   bi_warp_loop(the_image, out_image,
                x1, x2, x3, x4,
                y1, y2, y3, y4,
                extra_x, extra_y,
```

```
                  rows, cols);
   else
      warp_loop(the_image, out_image,
                x1, x2, x3, x4,
                y1, y2, y3, y4,
                extra_x, extra_y,
                rows, cols);

      /**********************************
      *
      *   4 - lower left quarter
      *
      **********************************/

   x1 = 0;
   x2 = x_control;
   x3 = cols_div_2;
   x4 = 0;

   y1 = rows_div_2;
   y2 = y_control;
   y3 = rows-1;
   y4 = rows-1;

   extra_x = 0;
   extra_y = rows_div_2;

   if(bilinear)
      bi_warp_loop(the_image, out_image,
                   x1, x2, x3, x4,
                   y1, y2, y3, y4,
                   extra_x, extra_y,
                   rows, cols);
   else
      warp_loop(the_image, out_image,
                x1, x2, x3, x4,
                y1, y2, y3, y4,
                extra_x, extra_y,
                rows, cols);

}  /* ends warp */
```

```
     /*****************************************
     *
     *   warp_loop(..
     *
     *   This routine sets up the coefficients
     *   and loops through a quarter of the
     *   rowsxcols section of the image that
     *   is being warped.
     *
     *****************************************/

warp_loop(the_image, out_image,
          x1, x2, x3, x4,
          y1, y2, y3, y4,
          extra_x, extra_y,
          rows, cols)
   int    extra_x, extra_y,
          x1, x2, x3, x4,
          y1, y2, y3, y4;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int    cols_div_2, denom, i, j, rows_div_2,
          xa, xb, xab, x_out, ya, yb, yab, y_out;

   cols_div_2 = cols/2;
   rows_div_2 = rows/2;
   denom      = cols_div_2 * rows_div_2;

      /***********************************
     *
     *   Set up the terms for the
     *   spatial transformation.
     *
     ************************************/

   xa  = x2 - x1;
   xb  = x4 - x1;
   xab = x1 - x2 + x3 - x4;

   ya  = y2 - y1;
   yb  = y4 - y1;
   yab = y1 - y2 + y3 - y4;
```

```
      /***********************************
     *
     *   Loop through a quadrant and
     *   perform the spatial
     *   transformation.
     *
     **********************************/

      /* NOTE a=j b=i */

   printf("\n");
   for(i=0; i<rows_div_2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<cols_div_2; j++){

         x_out = x1 + (xa*j)/cols_div_2 +
               (xb*i)/rows_div_2 + (xab*i*j)/(denom);
         y_out = y1 + (ya*j)/cols_div_2 +
               (yb*i)/rows_div_2 + (yab*i*j)/(denom);

         if(x_out < 0       ||
            x_out >= cols   ||
            y_out < 0       ||
            y_out >= rows)
            out_image[i+extra_y][j+extra_x] = FILL;
         else
            out_image[i+extra_y][j+extra_x] =
             the_image[y_out][x_out];

      }  /* ends loop over j */
   }  /* ends loop over i */

}   /* ends warp_loop */




     /*******************************************
     *
     *   bi_warp_loop(..
     *
     *   This routine sets up the coefficients
     *   and loops through a quarter of the
```

```
     *   rowsxcols section of the image that
     *   is being warped.
     *
     *   This version of the routine uses bilinear
     *   interpolation to find the gray shades.
     *   It is more accurate than warp_loop,
     *   but takes longer because of the floating
     *   point calculations.
     *
     *****************************************/

bi_warp_loop(the_image, out_image,
             x1, x2, x3, x4,
             y1, y2, y3, y4,
             extra_x, extra_y,
             rows, cols)
   int    extra_x, extra_y,
          x1, x2, x3, x4,
          y1, y2, y3, y4;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   double cols_div_2, denom, di, dj, rows_div_2,
          xa, xb, xab, x_out, ya, yb, yab, y_out;
   int    i, j;

   cols_div_2 = (double)(cols)/2.0;
   rows_div_2 = (double)(rows)/2.0;
   denom      = cols_div_2 * rows_div_2;

     /***********************************
     *
     *   Set up the terms for the
     *   spatial transformation.
     *
     ************************************/

   xa  = x2 - x1;
   xb  = x4 - x1;
   xab = x1 - x2 + x3 - x4;

   ya  = y2 - y1;
   yb  = y4 - y1;
   yab = y1 - y2 + y3 - y4;
```

```
     /**********************************
     *
     *   Loop through a quadrant and
     *   perform the spatial
     *   transformation.
     *
     **********************************/

      /* NOTE a=j b=i */

   printf("\n");
   for(i=0; i<rows_div_2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<cols_div_2; j++){

       di = (double)(i);
       dj = (double)(j);

         x_out = x1 +
                 (xa*dj)/cols_div_2 +
                 (xb*di)/rows_div_2 +
                 (xab*di*dj)/(denom);
         y_out = y1 +
                 (ya*dj)/cols_div_2 +
                 (yb*di)/rows_div_2 +
                 (yab*di*dj)/(denom);

         out_image[i+extra_y][j+extra_x] =
          bilinear_interpolate(the_image,
                               x_out, y_out,
                               rows, cols);

      }  /* ends loop over j */
   }  /* ends loop over i */

}   /* ends bi_warp_loop */




     /*****************************************
     *
     *   object_warp(..
```

```
     *
     *   This routine warps a rowsxcols section
     *   of an image.  The il, ie parameters
     *   specify which rowsxcols section of
     *   the image to warp.  The x_control and
     *   y_control parameters are the control
     *   points inside that section.  Therefore,
     *   x_control and y_control will always be
     *   less the cols and rows.
     *
     *   The point coordinates are for the four
     *   corners of a four side figure.
     *      x1,y1     x2,y2
     *
     *      x4,y4     x3,y3
     *
     ******************************************/

object_warp(the_image, out_image,
            x1, y1, x2, y2,
            x3, y3, x4, y4,
            bilinear, rows, cols)
   int    bilinear,
          x1, y1, x2, y2,
          x3, y3, x4, y4;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int    extra_x = 0,
          extra_y = 0;

      /***********************************
      *
      *   Call the warp loop function you
      *   need (with or without bilinear
      *   interpolation).
      *
      ************************************/

   if(bilinear)
      bi_full_warp_loop(the_image, out_image,
                        x1, x2, x3, x4,
                        y1, y2, y3, y4,
                        extra_x, extra_y,
```

```
                        rows, cols);
   else
      full_warp_loop(the_image, out_image,
                     x1, x2, x3, x4,
                     y1, y2, y3, y4,
                     extra_x, extra_y,
                     rows, cols);

}  /* ends object_warp */




     /*******************************************
     *
     *   full_warp_loop(..
     *
     *   This routine sets up the coefficients
     *   and loops through an entire
     *   rowsxcols image that is being warped.
     *
     *******************************************/

full_warp_loop(the_image, out_image,
               x1, x2, x3, x4,
               y1, y2, y3, y4,
               extra_x, extra_y,
               rows, cols)
   int    extra_x, extra_y,
          x1, x2, x3, x4,
          y1, y2, y3, y4;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int    cols_div_2, denom, i, j, rows_div_2,
          xa, xb, xab, x_out, ya, yb, yab, y_out;

   denom      = cols * rows;

      /***********************************
     *
     *   Set up the terms for the
     *   spatial transformation.
     *
```

```
     ***********************************/

   xa  = x2 - x1;
   xb  = x4 - x1;
   xab = x1 - x2 + x3 - x4;

   ya  = y2 - y1;
   yb  = y4 - y1;
   yab = y1 - y2 + y3 - y4;

      /************************************
     *
     *   Loop through the image and
     *   perform the spatial
     *   transformation.
     *
     ***********************************/

      /* NOTE a=j b=i */

   printf("\n");
   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<cols; j++){

         x_out = x1 + (xa*j)/cols +
               (xb*i)/rows + (xab*i*j)/(denom);
         y_out = y1 + (ya*j)/cols +
               (yb*i)/rows + (yab*i*j)/(denom);

         if(x_out < 0       ||
            x_out >= cols   ||
            y_out < 0       ||
            y_out >= rows)
            out_image[i+extra_y][j+extra_x] = FILL;
         else
            out_image[i+extra_y][j+extra_x] =
             the_image[y_out][x_out];

      }  /* ends loop over j */
   }  /* ends loop over i */

}   /* ends full_warp_loop */
```

```
     /*****************************************
     *
     *   bi_full_warp_loop(..
     *
     *   This routine sets up the coefficients
     *   and loops through an entire
     *   rowsxcols image that is being warped.
     *
     *   This version of the routine uses bilinear
     *   interpolation to find the gray shades.
     *   It is more accurate than warp_loop,
     *   but takes longer because of the floating
     *   point calculations.
     *
     *****************************************/

bi_full_warp_loop(the_image, out_image,
                  x1, x2, x3, x4,
                  y1, y2, y3, y4,
                  extra_x, extra_y,
                  rows, cols)
   int    extra_x, extra_y,
          x1, x2, x3, x4,
          y1, y2, y3, y4;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   double denom, di, dj,
          xa, xb, xab, x_out, ya, yb, yab, y_out;
   int    i, j;

   denom      = cols * rows;

     /***********************************
     *
     *   Set up the terms for the
     *   spatial transformation.
     *
     ***********************************/

   xa  = x2 - x1;
```

```
   xb  = x4 - x1;
   xab = x1 - x2 + x3 - x4;

   ya  = y2 - y1;
   yb  = y4 - y1;
   yab = y1 - y2 + y3 - y4;

     /***********************************
     *
     *   Loop through the image and
     *   perform the spatial
     *   transformation.
     *
     ************************************/

      /* NOTE a=j b=i */

   printf("\n");
   for(i=0; i<rows; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=0; j<cols; j++){

         di = (double)(i);
         dj = (double)(j);

         x_out = x1 +
                 (xa*dj)/cols +
                 (xb*di)/rows +
                 (xab*di*dj)/(denom);
         y_out = y1 +
                 (ya*dj)/cols +
                 (yb*di)/rows +
                 (yab*di*dj)/(denom);

         out_image[i+extra_y][j+extra_x] =
          bilinear_interpolate(the_image,
                               x_out, y_out,
                               rows, cols);

      }  /* ends loop over j */
   }  /* ends loop over i */

}   /* ends bi_full_warp_loop */
```

Listing 14.1 - The Warping Subroutines

```
echo off

rem   Usage:
rem   morphit in1 in2 working-dir output
rem
rem   Example:
rem      morphit c:a.tif c:b.tif d: c:out.tif
rem
rem   This will morph c:a.tif and c:b.tif
rem   and put the output in c:out.tif
rem   It will also create temporary files in
rem   d: and then delete them.

rem   Check for the right number of parameters
if "%4" == "" goto usage

warp %1 %3in1w1.tif object 10 10 90 10 90 90 10 90 1 1 0
warp %3in1w1.tif %3in1w2.tif object 10 10 90 10 90 90 10 90 1 1 0

warp %2 %3in2w1.tif object -10 -10 110 -10 110 110 -10 110 1 1 0
warp %3in2w1.tif %3in2w2.tif object -10 -10 110 -10 110 110 -10 110 1 1 0

rem Average input 1 with the second warp of input 2
rem Weight input 1 twice as much
rem Put the result in file 1.tif

mainover %1 %3in2w2.tif %3tmp1.tif average 1 1
mainover %1 %3tmp1.tif  %3tmp2.tif average 1 1
copy %3tmp2.tif %31.tif


rem Average input 2 with the second warp of input 1
rem Weight input 2 twice as much
rem Put the result in file 3.tif

mainover %2 %3in1w2.tif %3tmp1.tif average 1 1
mainover %2 %3tmp1.tif  %3tmp2.tif average 1 1
copy %3tmp2.tif %33.tif
```

```
rem Average the first warps of the two inputs
rem Put the result in file 2.tif

mainover %3in2w1.tif %3in1w1.tif %3tmp1.tif average 1 1
copy %3tmp1.tif %32.tif

del %3tmp1.tif
del %3tmp2.tif

rem Now put the images side by side
rem in the sequence: in1 1 2 3 in2

side %1 %31.tif %3tmp1.tif side
side %3tmp1.tif %32.tif %3tmp2.tif side
del %3tmp1.tif
side %3tmp2.tif %33.tif %3tmp1.tif side
side %3tmp1.tif %2 %4 side
del %3tmp1.tif
del %3tmp2.tif
del %31.tif
del %32.tif
del %33.tif
del %3in1w1.tif
del %3in1w2.tif
del %3in2w1.tif
del %3in2w2.tif

goto end

:usage
echo .
echo usage: morphit in1 in2 working-dir output
echo .
:end
```

Listing 14.2 - The .bat File that Produced the Morphing Sequence (Figure 14.10)

```
    /*******************************************
    *
    *    file warp.c
    *
    *    Functions: This file contains
    *       main
    *
    *    Purpose:
    *       This file contains the main calling
    *       routine for warping subroutines.
    *
    *    External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *       warpsubs.c - warp
    *                    object_warp
    *
    *    Modifications:
    *       26 October 1993 - created
    *       27 August 1998 - modified to work on
    *            entire images at once.
    *       19 September 1998 - modified to work with
    *           all I O routines in imageio.c.
    *
    *******************************************/

#include "cips.h"


short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char   name1[80], name2[80], type[80];
   float  theta, x_stretch, y_stretch,
          x_cross, y_cross;
   int    bilinear,
```

```
        x1, x2, x3, x4,
        y1, y2, y3, y4;
int     x_control, y_control;
long    length, width;
short   m, n, x_displace, y_displace;

   /*************************************
   *
   *   This program will use a different
   *   command line for each type of
   *   call.
   *
   *   Print a usage statement that
   *   gives an example of each type
   *   of call.
   *
   *************************************/

if(argc < 7){
 printf("\n\nNot enough parameters:");
 printf("\n");
 printf("\n   Two Operations: ");
 printf("\n      warp      object-warp");
 printf("\n\n   Examples:");
 printf("\n");
 printf("\n   warp in out warp x-control ");
 printf("y-control bilinear (1 or 0)");
 printf("\n");
 printf("\n   warp in out object-warp x1 y1");
 printf(" x2 y2 x3 y3 x4 y4 ");
 printf("bilinear (1 or 0)");
 printf("\n");
 exit(0);
}

   /*************************************
   *
   *   Interpret the command line
   *   depending on the type of call.
   *
   *************************************/

if(strncmp(argv[3], "warp", 3) == 0){
   strcpy(name1, argv[1]);
   strcpy(name2, argv[2]);
```

```
   strcpy(type,  argv[3]);
   x_control = atoi(argv[4]);
   y_control = atoi(argv[5]);
   bilinear  = atoi(argv[6]);
}

if(strncmp(argv[3], "object-warp", 3) == 0){
   strcpy(name1, argv[1]);
   strcpy(name2, argv[2]);
   strcpy(type,  argv[3]);
   x1        = atoi(argv[4]);
   y1        = atoi(argv[5]);
   x2        = atoi(argv[6]);
   y2        = atoi(argv[7]);
   x3        = atoi(argv[8]);
   y3        = atoi(argv[9]);
   x4        = atoi(argv[10]);
   y4        = atoi(argv[11]);
   bilinear  = atoi(argv[12]);
}

if(does_not_exist(name1)){
 printf("\nERROR input file %s does not exist",
          name1);
 exit(0);
}

get_image_size(name1, &length, &width);
the_image = allocate_image_array(length, width);
out_image = allocate_image_array(length, width);
create_image_file(name1, name2);
read_image_array(name1, the_image);

   /************************************
   *
   *   Call the routines
   *
   ************************************/

if(strncmp(type, "warp", 3) == 0){
   warp(the_image, out_image,
        x_control, y_control,
        bilinear,
        length,
        width);
```

```
   }  /* ends if */



   if(strncmp(argv[3], "object-warp", 3) == 0){
      object_warp(the_image, out_image,
                  x1, y1, x2, y2,
                  x3, y3, x4, y4,
                  bilinear,
                  length,
                  width);
    }   /* ends if */

   write_image_array(name2, out_image);
   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */
```

Listing 14.3 - The warp Program

F.15 Code Listings for Chapter 15

```
    /*********************************************
    *
    *    file txtrsubs.c
    *
    *    Functions: This file contains
    *       sigma
    *       skewness
    *       amean
    *       adifference
    *       difference_array
    *       hurst
    *       compare
    *       get_texture_options
    *
    *    Purpose:
    *       These functions calculate measures
    *       that help distinguish textures.
```

```
    *
    *    External Calls:
    *       utility.c - fix_edges
    *                   sort_elements
    *       fitt.c - fit
    *
    *    Modifications:
    *       12 August 1993- created
    *       27 August 1998 - modified to work on
    *            entire images at once.
    *
    *******************************************/

#include "cips.h"



     /*******************************************
     *
     *   sigma(..
     *
     *   This calculates the variance and the
     *   sigma for a sizeXsize area.
     *
     *   It sums the squares of the difference
     *   between each pixel and the mean of
     *   the area and divides that by the
     *   number of pixels in the area.
     *
     *   The output image is set to the square
     *   root of the variance since the variance
     *   will almost certainly be out of range
     *   for the image.  The square root of the
     *   variance will be sigma.
     *
     *******************************************/

sigma(the_image, out_image,
      size, threshold, high,
      rows, cols, bits_per_pixel)
   int    high, threshold, size;
   long   bits_per_pixel, cols, rows;
   short  **the_image,
          **out_image;
{
```

```
int      a, b, count, i, j, k,
         max, mean, new_hi, new_low,
         sd2, sd2p1;
short    sigma;
unsigned long diff, variance;

sd2   = size/2;
sd2p1 = sd2 + 1;

max     = 255;
new_hi  = 250;
new_low = 16;
if(bits_per_pixel == 4){
    new_hi  = 10;
    new_low = 3;
    max     = 16;
}

   /***************************
   *
   *   Loop over image array
   *
   ****************************/

printf("\n");
for(i=sd2; i<rows-sd2; i++){
   if( (i%10) == 0) printf("%d ", i);
   for(j=sd2; j<cols-sd2; j++){

         /*****************************
         *
         *   Run through the small area
         *   and calculate the mean.
         *
         *****************************/

      mean = 0;
      for(a=-sd2; a<sd2p1; a++){
         for(b=-sd2; b<sd2p1; b++){
            mean = mean + the_image[i+a][j+b];
         }
      }
      mean = mean/(size*size);

         /*****************************
```

```
            *
            *   Run through the small area
            *   again and the calculate the
            *   variance.
            *
            ****************************/

         variance = 0;
         diff     = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               diff     = the_image[i+a][j+b] - mean;
               variance = variance + (diff*diff);
            }
         }

         variance = variance/(size*size);
         sigma    = sqrt(variance);
         if(sigma > max) sigma = max;
         out_image[i][j] = sigma;

      }  /* ends loop over j */
   }  /* ends loop over i */


     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends sigma */
```

```
     /*******************************************
     *
     *   skewness(..
     *
     *   This calculates the skewness for a
     *   sizeXsize area.
     *
     *   Look at Levine's book page 449 for
     *   the formula.
     *   "Vision in Man and Machine" by
     *   Martin D. Levine, McGraw Hill, 1985.
     *
     *******************************************/

skewness(the_image, out_image,
         size, threshold, high,
         rows, cols, bits_per_pixel)
   int    high, threshold, size;
   long   bits_per_pixel, cols, rows;
   short  **the_image,
          **out_image;
{
   int      a, b, count, i, j, k,
            max, mean, new_hi, new_low,
            sd2, sd2p1;
   long     cube;
   short    sigma, skew;
   unsigned long diff, sigma3, variance;

   sd2   = size/2;
   sd2p1 = sd2 + 1;

   max     = 255;
   new_hi  = 250;
   new_low = 16;
   if(bits_per_pixel == 4){
       new_hi  = 10;
       new_low = 3;
       max     = 16;
   }

      /***************************
      *
```

```
      *   Loop over image array
      *
      ****************************/

   printf("\n");
   for(i=sd2; i<rows-sd2; i++){
      if( (i%10) == 0) printf("%d ", i);
      for(j=sd2; j<cols-sd2; j++){

            /*****************************
            *
            *   Run through the small area
            *   and calculate the mean.
            *
            ******************************/

         mean = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               mean = mean + the_image[i+a][j+b];
            }
         }
         mean = mean/(size*size);

            /*****************************
            *
            *   Run through the small area
            *   again and the calculate the
            *   variance and the cube.
            *
            ******************************/

         variance = 0;
         diff     = 0;
         cube     = 0;
         for(a=-sd2; a<sd2p1; a++){
            for(b=-sd2; b<sd2p1; b++){
               diff     = the_image[i+a][j+b] - mean;
               cube     = cube + (diff*diff*diff);
               variance = variance + (diff*diff);
            }
         }

         variance        = variance/(size*size);
         sigma           = sqrt(variance);
```

```
         sigma3          = sigma*sigma*sigma;
         if(sigma3 == 0)
            sigma3       = 1;
         skew            = cube/(sigma3*size*size);
         out_image[i][j] = skew;
         if(out_image[i][j] > max)
            out_image[i][j] = max;

      }  /* ends loop over j */
   }  /* ends loop over i */



     /* if desired, threshold the output image */
   if(threshold == 1){
       for(i=0; i<rows; i++){
          for(j=0; j<cols; j++){
             if(out_image[i][j] > high){
                  out_image[i][j] = new_hi;
             }
             else{
                  out_image[i][j] = new_low;
             }
          }
       }
   }  /* ends if threshold == 1 */

   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends skewness */




     /*******************************************
     *
     *   adifference(..
     *
     *   This function performs the difference
     *   operation for a specified array
     *   in an image file.
     *
     *******************************************/

adifference(the_image, out_image,
```

```
            size,
            rows, cols)
   int    size;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int      sd2, sd2p1;

   sd2   = size/2;
   sd2p1 = sd2 + 1;

   difference_array(the_image, out_image,
                    size, rows, cols);

   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends adifference */




     /*****************************************
     *
     *   difference_array(..
     *
     *   This function takes the input image
     *   array the_image and places in out_image
     *   the gray level differences of the pixels
     *   in the_image.  It uses the size
     *   parameter for the distance between pixels
     *   used to get the difference.
     *
     *****************************************/

difference_array(the_image, out_image,
                 size, rows, cols)
   int    size;
   long   cols, rows;
   short  **the_image,
          **out_image;
{
   int i, j, sd2;
   sd2   = size/2;
```

```
   for(i=sd2; i<rows-sd2; i++){
      for(j=sd2; j<cols-sd2; j++){
         out_image[i][j] =
            abs(the_image[i][j] -
                the_image[i+sd2][j+sd2]);
      }  /* ends loop over j */
   }  /* ends loop over i */

   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends difference_array */




     /*****************************************
     *
     *   amean(..
     *
     *   This calculates the mean measure
     *   for a sizeXsize area.
     *
     *   Look at Levine's book page 451 for
     *   the formula.
     *   "Vision in Man and Machine" by
     *   Martin D. Levine, McGraw Hill, 1985.
     *
     *****************************************/

amean(the_image, out_image,
      size,
      rows, cols, bits_per_pixel)
   int    size;
   long   bits_per_pixel, cols, rows;
   short  **the_image,
          **out_image;
{
   int      a, b, count, i, j, k, max,
            sd2, sd2p1;
   short    pixel;
   unsigned long big;

   sd2   = size/2;
   sd2p1 = sd2 + 1;
```

```
max = 255;
if(bits_per_pixel == 4){
    max = 16;
}

   /***************************************
   *
   *   Calculate the gray level difference
   *   array.
   *
   ***************************************/

difference_array(the_image, out_image,
                 size, rows, cols);
for(i=0; i<rows; i++)
   for(j=0; j<cols; j++)
      the_image[i][j] = out_image[i][j];

   /***************************************
   *
   *   Loop over the image array and
   *   calculate the mean measure.
   *
   ***************************************/

printf("\n");
for(i=sd2; i<rows-sd2; i++){
   if( (i%10) == 0) printf("%d ", i);
   for(j=sd2; j<cols-sd2; j++){

      pixel = 0;
      for(a=-sd2; a<sd2p1; a++){
         for(b=-sd2; b<sd2p1; b++){
            pixel = pixel + the_image[i+a][j+b];
         }
      }
   out_image[i][j] = pixel/(size*size);
   if(out_image[i][j] > max)
      out_image[i][j] = max;

   }  /* ends loop over j */
}  /* ends loop over i */

fix_edges(out_image, sd2, rows-1, cols-1);
```

```
}  /* ends amean */


     /*****************************************
     *
     *   hurst(..
     *
     *   This routine performs the Hurst
     *   operation as described in "The Image
     *   Processing Handbook" by John C. Russ
     *   CRC Press 1992.
     *
     *   The following show the definitions of
     *   the pixel classes used in this routine.
     *
     *   3x3 case
     *       c b c
     *     d b a b d
     *       c b c
     *
     *   5x5 case
     *     f e d e f
     *     e c b c e
     *     d b a b d
     *     e c b c e
     *     f e d e f
     *
     *   7x7 case
     *       h g h
     *     f e d e f
     *   h e c b c e h
     *   g d b a b d g
     *   h e c b c e h
     *     f e d e f
     *       h g h
     *
     *****************************************/

hurst(the_image, out_image,
      size,
      rows, cols, bits_per_pixel)
```

```
   int    size;
   long   bits_per_pixel, cols, rows;
   short  **the_image,
          **out_image;

{
   float  x[8], y[8], sig[8];
   float  aa, bb, siga, sigb, chi2, q;
   int    ndata, mwt;

   int    a, b, count, i, j, k,
          new_hi, new_low, length,
          number, sd2, sd2p1, ss, width;
   short  *elements, max, prange;

      /*********************************************
      *
      *   Initialize the ln's of the distances.
      *   Do this one time to save computations.
      *
      *********************************************/

   x[1] = 0.0;        /* ln(1)        */
   x[2] = 0.34657359; /* ln(sqrt(2))  */
   x[3] = 0.69314718; /* ln(2)        */
   x[4] = 0.80471896; /* ln(sqrt(5))  */
   x[5] = 1.03972077; /* ln(sqrt(8))  */
   x[6] = 1.09861229; /* ln(3)        */
   x[7] = 1.15129255; /* ln(sqrt(10)) */

   sig[1] = 1.0;
   sig[2] = 1.0;
   sig[3] = 1.0;
   sig[4] = 1.0;
   sig[5] = 1.0;
   sig[6] = 1.0;
   sig[7] = 1.0;

   sd2 = size/2;
   printf("\nHURST>sd2=%d",sd2);
   if(sd2 < 2) sd2 = 2;
   printf("\nHURST>sd2=%d",sd2);

      /*********************************
      *
```

```
   *
   ***********************************/

max = 255;
if(bits_per_pixel == 4){
   max = 16;
}

   /***************************
   *
   *   Loop over image array
   *
   ****************************/

printf("\n");
for(i=sd2; i<rows-sd2; i++){
   if( (i%10) == 0) printf("%d ", i);
   for(j=sd2; j<cols-sd2; j++){

      for(k=1; k<=7; k++) y[k] = 0.0;

         /*************************************
         *
         *   Go through each pixel class, set
         *   the elements array, sort it, get
         *   the range, and take the ln of the
         *   range.
         *
         *************************************/

         /* b pixel class */
      number      = 4;
      elements    = (short *)
                 malloc(number * sizeof(short));
      elements[0] = the_image[i-1][j];
      elements[1] = the_image[i+1][j];
      elements[2] = the_image[i][j-1];
      elements[3] = the_image[i][j+1];
      sort_elements(elements, &number);
      prange = elements[number-1] - elements[0];
      if(prange < 0)  prange = prange*(-1);
      if(prange == 0) prange = 1;
      y[1] = log(prange);

         /* c pixel class */
```

```
elements[0] = the_image[i-1][j-1];
elements[1] = the_image[i+1][j+1];
elements[2] = the_image[i+1][j-1];
elements[3] = the_image[i-1][j+1];
sort_elements(elements, &number);
prange = elements[number-1] - elements[0];
if(prange < 0)  prange = prange*(-1);
if(prange == 0) prange = 1;
y[2] = log(prange);

   /* d pixel class */
elements[0] = the_image[i-2][j];
elements[1] = the_image[i+2][j];
elements[2] = the_image[i][j-2];
elements[3] = the_image[i][j+2];
sort_elements(elements, &number);
prange = elements[number-1] - elements[0];
if(prange < 0)  prange = prange*(-1);
if(prange == 0) prange = 1;
y[3] = log(prange);

   /* f pixel class */
if(size == 5  ||  size == 7){
 elements[0] = the_image[i-2][j-2];
 elements[1] = the_image[i+2][j+2];
 elements[2] = the_image[i+2][j-2];
 elements[3] = the_image[i-2][j+2];
 sort_elements(elements, &number);
 prange = elements[number-1] - elements[0];
 if(prange < 0)  prange = prange*(-1);
 if(prange == 0) prange = 1;
 y[5] = log(prange);
}  /* ends if size == 5 */

   /* g pixel class */
if(size == 7){
 elements[0] = the_image[i-3][j];
 elements[1] = the_image[i+3][j];
 elements[2] = the_image[i][j-3];
 elements[3] = the_image[i][j+3];
 sort_elements(elements, &number);
 prange = elements[number-1] - elements[0];
 if(prange < 0)  prange = prange*(-1);
 if(prange == 0) prange = 1;
 y[6] = log(prange);
```

```
}  /* ends if size == 7 */

free(elements);

   /* e pixel class */
if(size == 5  ||  size == 7){
 number      = 8;
 elements    = (short *)
               malloc(number * sizeof(short));
 elements[0] = the_image[i-1][j-2];
 elements[1] = the_image[i-2][j-1];
 elements[2] = the_image[i-2][j+1];
 elements[3] = the_image[i-1][j+2];
 elements[4] = the_image[i+1][j+2];
 elements[5] = the_image[i+2][j+1];
 elements[6] = the_image[i+2][j-1];
 elements[7] = the_image[i+1][j-2];
 sort_elements(elements, &number);
 prange = elements[number-1] - elements[0];
 if(prange < 0)  prange = prange*(-1);
 if(prange == 0) prange = 1;
 y[4] = log(prange);
}  /* ends if size == 5 */

   /* h pixel class */
if(size == 7){
 elements[0] = the_image[i-1][j-3];
 elements[1] = the_image[i-3][j-1];
 elements[2] = the_image[i-3][j+1];
 elements[3] = the_image[i-1][j+3];
 elements[4] = the_image[i+1][j+3];
 elements[5] = the_image[i+3][j+1];
 elements[6] = the_image[i+3][j-1];
 elements[7] = the_image[i+1][j-3];
 sort_elements(elements, &number);
 prange = elements[number-1] - elements[0];
 if(prange < 0)  prange = prange*(-1);
 if(prange == 0) prange = 1;
 y[7] = log(prange);
}  /* ends if size == 7 */

if(size == 5  ||  size == 7)
   free(elements);

   /************************************
```

```
            *
            *   Call the fit routine to fit the
            *   data to a straight line. y=mx+b
            *   The answer you want is the slope
            *   of the line.  That is returned
            *   in the parameter bb.
            *
            ************************************/
         ndata = size;
         mwt   = 1;
         fit(x, y, ndata, sig, mwt, &aa, &bb,
             &siga, &sigb, &chi2, &q);

         out_image[i][j] = (short)(bb*64.0);
         if(out_image[i][j] > max)
            out_image[i][j] = max;
         if(out_image[i][j] < 0)
            out_image[i][j] = 0;

      }  /* ends loop over j */
   }  /* ends loop over i */

   fix_edges(out_image, sd2, rows-1, cols-1);

}  /* ends hurst */




     /*****************************************
     *
     *   compare(..
     *
     *   This function compares a sizeXsize area
     *   starting at line,element in an image
     *   with all the sizeXsize areas in the
     *   image.
     *
     *****************************************/

compare(the_image, out_image,
        line, element, size,
        rows, cols, bits_per_pixel)
   int    line, element, size;
```

```
   long   bits_per_pixel, cols, rows;
   short  **the_image,
          **out_image;
{
   int      a, b, count, i, j, k, max,
            sd2, sd2p1;
   short    pixel;
   int      big, diff;

      /**************************************
      *
      *   Declare and allocate memory for the
      *   two dimensional small array.
      *
      ***************************************/

   short **small;
   small = malloc(size * sizeof(short  *));
   for(i=0; i<size; i++){
      small[i] = malloc(size * sizeof(short ));
      if(small[i] == '\0'){
         printf("\n\tmalloc of small[%d] failed", i);
         exit(0);
      }
   }

      /**************************************
      *
      *   The calling program read in the
      *   part of the input image
      *   that contains the line element
      *   portion into the out_image array.
      *   Copy that into the small array.
      *
      ***************************************/

   for(i=0; i<size; i++)
      for(j=0; j<size; j++)
        small[i][j] = out_image[i][j];

   for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
        out_image[i][j] = 0;

      /**************************************
```

```
   *
   *   Create the output image and read
   *   in the input data.
   *
   **************************************/

sd2   = size/2;
sd2p1 = sd2 + 1;

max = 255;
if(bits_per_pixel == 4){
    max = 16;
}

   /**************************************
   *
   *   Loop over the image array and
   *   calculate the contrast measure.
   *
   **************************************/

printf("\n");
for(i=sd2; i<rows-sd2; i++){
   if( (i%10) == 0) printf("%d ", i);
   for(j=sd2; j<cols-sd2; j++){

      big = 0;
      for(a=-sd2; a<sd2p1; a++){
         for(b=-sd2; b<sd2p1; b++){
         diff = small[a+sd2][b+sd2] -
                the_image[i+a][j+b];
         big  = big + abs(diff);
         }
      }

   big = big/(size*size);
   out_image[i][j] = big;
   if(out_image[i][j] > max)
      out_image[i][j] = max;

   }  /* ends loop over j */
}  /* ends loop over i */

fix_edges(out_image, sd2, rows-1, cols-1);
```

```
      /**************************************
      *
      *   Free the memory for the
      *   two dimensional small array.
      *
      **************************************/

   for(i=0; i<size; i++)
      free(small[i]);

}  /* ends compare */
```

Listing 15.1 - The Texture Subroutines

```
    /*********************************************
    *
    *    file texture.c
    *
    *    Functions: This file contains
    *       main
    *
    *    Purpose:
    *       This file contains the main calling
    *       routine for texture subroutines.
    *
    *    External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  get_bitsperpixel
    *                  allocate_image_array
    *                  free_image_array
    *       txtrsubs.c - sigma
    *                    skewness
    *                    amean
    *                    adifference
    *                    hurst
    *                    compare
    *
```

```
    *    Modifications:
    *       12 August 1993 - created
    *       27 August 1998 - modified to work on
    *            entire images at once.
    *       19 September 1998 - modified to work with
    *           all I O routines in imageio.c.
    *
    *******************************************/

#include "cips.h"



short **the_image;
short **out_image;

main(argc, argv)
   int argc;
   char *argv[];
{

   char name1[80], name2[80];
   int  size, t, type, v;
   long bits_per_pixel, length, width;

   if(argc < 7){
      show_texture_usage();
      exit(0);
   }

   strcpy(name1, argv[1]);
   strcpy(name2, argv[2]);
   t    = atoi(argv[4]);
   v    = atoi(argv[5]);
   size = atoi(argv[6]);

   if(does_not_exist(name1)){
    printf("\nERROR input file %s does not exist",
             name1);
    exit(0);
   }

   get_image_size(name1, &length, &width);
   get_bitsperpixel(name1, &bits_per_pixel);
   the_image = allocate_image_array(length, width);
```

```
out_image = allocate_image_array(length, width);
create_image_file(name1, name2);
read_image_array(name1, the_image);


if(strncmp(argv[3], "compare", 3) == 0){
   read_image_array(name1, out_image);
   compare(the_image, out_image,
           t, v, size,
           length,
           width,
           bits_per_pixel);
}  /* ends if compare case */

if(strncmp(argv[3], "hurst", 3) == 0)
   hurst(the_image, out_image,
         size,
         length,
         width,
         bits_per_pixel);

if(strncmp(argv[3], "adifference", 3) == 0)
   adifference(the_image,
               out_image,
               size,
               length,
               width);

if(strncmp(argv[3], "amean", 3) == 0)
   amean(the_image, out_image,
         size,
         length,
         width,
         bits_per_pixel);

if(strncmp(argv[3], "skewness", 3) == 0){
   size = atoi(argv[4]);
   t    = atoi(argv[5]);
   v    = atoi(argv[6]);
   skewness(the_image, out_image,
            size, t, v,
            length,
            width,
            bits_per_pixel);
}  /* ends skewness case */
```

```
   if(strncmp(argv[3], "sigma", 3) == 0){
      size = atoi(argv[4]);
      t    = atoi(argv[5]);
      v    = atoi(argv[6]);
      sigma(the_image, out_image,
            size, t, v,
            length,
            width,
            bits_per_pixel);
   }  /* ends sigma case */

   write_image_array(name2, out_image);
   free_image_array(out_image, length);
   free_image_array(the_image, length);

}  /* ends main  */


show_texture_usage()
{
 printf("\n\nNot enough parameters:");
 printf("\n");
 printf("\n   usage: texture in-file out-file type ");
 printf("threshold? threshold-value size");
 printf("\n   recall type: sigma      skewness");
 printf("\n                amean      adifference");
 printf("\n                hurst      compare");
 printf("\n   threshold?   1-threshold on");
 printf("\n                2-threshold off\n");
 printf("\n   usage for compare:");
 printf("\n   texture in-file out-file compare ");
 printf("line element size");
 printf("\n  "
  "\ntexture in-file out-file compare line element size"
  "\ntexture in-file out-file hurst x x size (3 5 or 7)"
  "\ntexture in-file out-file adifference x x size"
  "\ntexture in-file out-file amean x x size"
  "\ntexture in-file out-file skewness size "
            "threshold(0,1) high-threshold"
  "\ntexture in-file out-file sigma size "
            "threshold(0,1) high-threshold");
}
```

Listing 15.2 - The texture Program

F.16 Code Listings for Chapter 16

```
/***************************************************
*
*  file cstereo.c
*
*  Functions: This file contains
*     main
*     fill_line
*     initialize_pattern
*     lengthen_pattern
*     no_change
*     shorten_pattern
*     get_random_values
*     random_substitution
*     test_print_line
*
*  Purpose:
*     This file contains a program that will
*     make character based stereograms.
*
*  External Calls:
*     none
*
*  Modifications:
*     7 February 1995 - created
*
***************************************************/


#include "cips.h"

#define PATTERN_START  48 /* the zero character    */
#define PATTERN_END   122 /* the small z character */
#define KONSTANT        2
#define ASCII_SIZE    256

void fill_line();
void initialize_pattern();
void lengthen_pattern();
void no_change();
void shorten_pattern();
void get_random_values();
```

```
void random_substitution();
void test_print_line();


main(argc, argv)
   char *argv[];
   int  argc;
{
   char *depth_line,
        *pattern,
        *processed_pattern,
        *pdest,
        response[80];

   char last_character,
        this_character;

   FILE *depth_file,
        *processed_pattern_file,
        *stereo_file;

   int  current_width,
        j,
        index,
        location,
        max_width,
        pattern_width,
        pp_index,
        width;

   if(argc != 6){
      printf(
      "\nusage: cstereo pattern-width width-of-line"
      "\n               depth-file-name "
      "stereo-file-name "
      "\n               processed-pattern-file-name"
      "\n");
      exit(1);
   }

   pattern_width = atoi(argv[1]);
   width         = atoi(argv[2]);

   pattern           = malloc(KONSTANT*width);
   depth_line        = malloc(KONSTANT*width);
```

```
processed_pattern = malloc(KONSTANT*width);

if((depth_file = fopen(argv[3], "rt")) == NULL){
   printf(
  "\ncstereo: Error opening input file %s\n",
   argv[3]);
   exit(3);
}

if((stereo_file = fopen(argv[4], "wt")) == NULL){
   printf(
 "\ncstereo: Error opening output file %s\n",
   argv[4]);
   exit(4);
}

if((processed_pattern_file =
   fopen(argv[5], "wt")) == NULL){
   printf(
   "\ncstereo: Error opening temp file %s\n",
   argv[5]);
   exit(5);
}

   /*******************************************
   *
   *   This is the major loop of the program.
   *   It reads one line at a time from the
   *   depth file, processes that one line,
   *   and writes the resulting pattern to the
   *   processed pattern file.
   *
   *******************************************/

while(fgets(depth_line,
            KONSTANT*width,
            depth_file)){

   fill_line(pattern, KONSTANT*width);
   fill_line(processed_pattern, KONSTANT*width);

   initialize_pattern(pattern,
                      &current_width,
                      &max_width,
                      pattern_width,
```

```
                    &index);

   /*******************************************
   *
   *   Loop through the characters in the
   *   depth_line.  Call one of the three
   *   processing routines based on the
   *   relationship between the last character
   *   and this character.
   *
   *******************************************/

last_character = depth_line[0];

pp_index = 0;

for(j=0; j<width; j++){
   this_character = depth_line[j];
   if(this_character == '\n')
      this_character = last_character;


   if(this_character > last_character)
      shorten_pattern(
         (this_character-last_character),
         pattern, &index, &current_width,
         width);


   if(this_character < last_character)
      lengthen_pattern(
         (last_character-this_character),
         pattern, &index, &current_width,
         &width, &max_width);



      /****************************************
      *
      *   Perform the no_change in every
      *   pass.  Do it after you have done
      *   the shorten and lenghten pattern.
      *
      ****************************************/
```

```
        no_change(pattern, processed_pattern,
                  pp_index, current_width, &index);
        pp_index++;

        last_character = depth_line[j];

      }  /* ends loop over j */

      pdest    = strchr(processed_pattern, '\0');
      location = pdest - processed_pattern;
      processed_pattern[location] = '\n';

      fputs(processed_pattern,
            processed_pattern_file);
      random_substitution(processed_pattern, width);
      fputs(processed_pattern, stereo_file);

   }   /* ends the major loop */

   fclose(depth_file);
   fclose(processed_pattern_file);
   fclose(stereo_file);


   free(pattern);
   free(depth_line);
   free(processed_pattern);

   return(111);

}  /* ends main                 */




   /***********************************************
   *
   *   shorten_pattern(...
   *
   *   This funtion shortens the pattern by
   *   deleting an element from it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcfg.
   *
```

```
   ***********************************************/

void shorten_pattern(size, pattern, index,
                     current_width, width)
   char *pattern;
   int  size, *index, *current_width, width;
{
   char *temp_pattern;
   int  i, new_index, new_width;

   temp_pattern = malloc(KONSTANT*width);

   for(i=0; i<width; i++)
      temp_pattern[i] = pattern[i];
   fill_line(pattern, width);

   new_index = 0;
   new_width = *current_width - size;

      /*********************************************
      *
      *   Increase the index by the amount we want
      *   to shorten the pattern.   Then copy the
      *   temp_pattern back to the pattern skipping
      *   over parts of the pattern by the amount
      *   we want to shorten it.
      *
      *********************************************/

   *index = (*index + size) % (*current_width);

   for(new_index=0; new_index<new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width = new_width;
   *index         = 0;

   free(temp_pattern);

}  /* ends shorten_pattern */
```

```
   /***********************************************
   *
   *   initialize_pattern(...
   *
   *   This function creates an initial pattern
   *   that is as wide as the pattern_width
   *   parameter.
   *
   ***********************************************/

void initialize_pattern(pattern, current_width,
                        max_width, pattern_width,
                        index)
   char pattern[];
   int *current_width,
       *max_width,
       *index,
       pattern_width;
{
   int i;

   for(i=0; i<pattern_width; i++)
      pattern[i] = i+PATTERN_START;

   *index         = 0;
   *current_width = pattern_width;
   *max_width     = pattern_width;
}  /* ends initialize_pattern */
```

```
   /***********************************************
   *
   *   no_change(...
   *
   *   This function processes the pattern
   *   and does not make any changes to it.
   *
   ***********************************************/
```

```
void no_change(pattern, processed_pattern, pp_index,
               current_width, index)
   char *pattern, *processed_pattern;
   int  pp_index, current_width, *index;
{
   processed_pattern[pp_index] =
      pattern[*index];

   *index = *index + 1;
   if(*index >= current_width)
      *index = 0;

}   /* ends no_change */




   /***********************************************
   *
   *   fill_line(...
   *
   *   This function fills a character array
   *   with NULL characters.
   *
   ***********************************************/

void fill_line(line, length)
   char *line;
   int  length;
{
   int i;
   for(i=0; i<length; i++)
      line[i] = '\0';
}  /* ends fill_line */




   /***********************************************
   *
   *   lengthen_pattern(...
   *
```

```
   *   This funtion lengthens the pattern by
   *   inserting an element(s) from it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcdefgh.
   *
   ***********************************************/

void lengthen_pattern(size, pattern, index,
                      current_width, width, max_width)
   char *pattern;
   int  size, *index, *current_width,
        *width, *max_width;
{
   char *temp_pattern;
   int  count, i, new_index, new_width;

   temp_pattern = malloc(KONSTANT*(*width));

   for(i=0; i<(*width); i++)
      temp_pattern[i] = pattern[i];

   fill_line(pattern, *width);

   for(count=0, new_index=0; count<size;
       count++, new_index++)
      pattern[new_index] =
         count + *max_width + PATTERN_START;

   new_width = *current_width + size;

   for( ; new_index < new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width = new_width;
   *index         = 0;
   *max_width     = *max_width + size;

   free(temp_pattern);

}  /* ends lengthen_pattern */
```

```
   /**********************************************
   *
   *   random_substitution(...
   *
   *   This function takes the processed_pattern
   *   array and substitutes random values for each
   *   value in the array.
   *
   *   Fill the substitution_values array with
   *   random characters that are all printable
   *   (PATTERN_START to PATTERN_END).
   *
   **********************************************/

void random_substitution(processed_pattern, width)
   char *processed_pattern;
   int  width;
{
   char substitution_values[ASCII_SIZE];
   int  i, place;

   get_random_values(substitution_values);

   for(i=0; i<(KONSTANT*width); i++){
      if(processed_pattern[i] != '\n'   &&
         processed_pattern[i] != '\0'){
         place = processed_pattern[i];
         processed_pattern[i] =
            substitution_values[place];
      }  /* ends if */
   }  /* ends loop over i */

}  /* ends random_substitution */


   /**********************************************
   *
   *   get_random_values(...
```

```
   *
   *   This function fills array with random values.
   *   The limit on the random values are from
   *   PATTERN_START to PATTERN_END.
   *
   ************************************************/

void get_random_values(array)
   char array[];
{
   int i, number;

#ifdef NEVER
these lines worked ok, they used all the printable
characters from 0 through small z
   for(i=0; i<ASCII_SIZE; i++){
       number   = rand();
       number   = number % (PATTERN_END - PATTERN_START);
       number   = number + PATTERN_START;
       array[i] = number;
   }  /* ends loop over i */
#endif

#ifdef NEVER
/* Let's try something different, only use the
   characters 0-9 and A-Z
   0-9 are 48-57 decimal A-Z are 65-90 */

   for(i=0; i<ASCII_SIZE; i++){
       number   = rand();
       number   = number % 36;
       number   = number + PATTERN_START;
       if(number > 57   &&   number < 65)
          number = number + 7;
       array[i] = number;
   }  /* ends loop over i */
#endif

/* Let's try something different, only use the
   characters A-Z
   A-Z are 65-90 */

   for(i=0; i<ASCII_SIZE; i++){
       number   = rand();
       number   = number % 26;
```

```
       number   = number + 65;
       array[i] = number;
   }  /* ends loop over i */

}  /* ends get_random_values */




   /***********************************************
   *
   *  test_print_line(...
   *
   *   This is a debug function that prints
   *   an array of characters.
   *
   ***********************************************/

void test_print_line(line, width)
   char *line;
   int  width;
{
   int i;
   for(i=0; i<width; i++)
      printf("%c", line[i]);
   printf("\n");
}  /* ends test_print_line */
```

Listing 16.1 - The cstereo Program

```
   /***********************************************
   *
   *  file pstereo.c
   *
   *  Functions: This file contains
   *     main
   *     zero_line
   *     initialize_pattern
   *     lengthen_pattern
   *     no_change
   *     shorten_pattern
   *     get_random_values
```

```
   *     random_substitution
   *     test_print_line
   *
   *  Purpose:
   *     This file contains a program that will
   *     make pixel based random dot stereograms.
   *
   *  External Calls:
   *     imageio.c - read_image_array
   *                 write_image_array
   *                 create_image_file
   *                 allocate_image_array
   *                 free_image_array
   *                 get_image_size
   *                 does_not_exist
   *
   *  Modifications:
   *     18 March 1995 - created
   *     27 August 1998 - modified to work on
   *            entire images at once.
   *
   ***************************************************/


#include "cips.h"

#define KONSTANT        2
#define PATTERN_START   0
#define PATTERN_END   255
#define ASCII_SIZE    256


short **depth_image,
      **pattern,
      **processed_pattern;

void zero_line();
void initialize_pattern();
void lengthen_pattern();
void no_change();
void shorten_pattern();
void get_random_values();
void random_substitution();
void read_image_line();
void write_image_line();
```

```
void test_print_line();


main(argc, argv)
   char *argv[];
   int  argc;
{
   char depth_file_name[MAX_NAME_LENGTH],
        s_file_name[MAX_NAME_LENGTH],
        pp_file_name[MAX_NAME_LENGTH],
        response[MAX_NAME_LENGTH];

   FILE *depth_file,
        *processed_pattern_file,
        *stereo_file;

   int  current_width,
        i,
        j,
        index,
        last_pixel,
        location,
        max_width,
        pattern_width,
        pp_index,
        this_pixel;

   long length, width;

   int line = 0;

   if(argc != 5){
      printf(
      "\nusage: pstereo pattern-width "
      "\n               depth-file-name "
      "stereo-file-name "
      "\n               processed-pattern-file-name"
      "\n");
      exit(1);
   }


   pattern_width =        atoi(argv[1]);
   strcpy(depth_file_name,     argv[2]);
   strcpy(s_file_name,         argv[3]);
```

```
strcpy(pp_file_name,        argv[4]);

if(does_not_exist(depth_file_name)){
   printf("\nThe depth file %s does not exist",
          depth_file_name);
   exit(1);
}

get_image_size(depth_file_name, &length, &width);
depth_image       = allocate_image_array(length, width);
pattern           = allocate_image_array(length, width);
processed_pattern = allocate_image_array(
                       length, width);
create_image_file(depth_file_name, s_file_name);
create_image_file(depth_file_name, pp_file_name);
read_image_array(depth_file_name, depth_image);

   /************************************************
   *
   *   This is the major loop of the program.
   *   It reads one line at a time from the
   *   depth file, processes that one line,
   *   and writes the resulting pattern to the
   *   processed pattern file.
   *
   ************************************************/

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

   if(i%10 == 0) printf(" %d", i);

   zero_line(pattern[i], width);
   zero_line(processed_pattern[i], width);

   initialize_pattern(pattern[i],
                      &current_width,
                      &max_width,
                      pattern_width,
                      &index);

      /*******************************************
      *
      *   Loop through the pixels in the
      *   depth_line.  Call one of the three
      *   processing routines based on the
```

```
   *   relationship between the last pixel
   *   and this pixel.
   *
   *******************************************/

last_pixel = depth_image[i][0];

pp_index = 0;

for(j=0; j<width; j++){
   this_pixel = depth_image[i][j];


   if(this_pixel > last_pixel)
      shorten_pattern(
         (this_pixel-last_pixel),
         pattern[i], &index, &current_width,
         width);


   if(this_pixel < last_pixel)
      lengthen_pattern(
         (last_pixel-this_pixel),
         pattern[i], &index, &current_width,
         &width, &max_width);



      /****************************************
      *
      *   Perform the no_change in every
      *   pass.  Do it after you have done
      *   the shorten and lenghten pattern.
      *
      ****************************************/

   no_change(pattern[i], processed_pattern[i],
             pp_index, current_width, &index);
   pp_index++;


   if(index >= current_width)
      index = 0;

   last_pixel = depth_image[i][j];
```

```
      }  /* ends loop over j */

   }   /* ends loop over i */

   write_image_array(pp_file_name, processed_pattern);

   for(i=0; i<length; i++)
      random_substitution(processed_pattern[i], width);

   write_image_array(s_file_name, processed_pattern);



   free_image_array(depth_image, length);
   free_image_array(pattern, length);
   free_image_array(processed_pattern, length);

   return(111);

}  /* ends main                 */




   /************************************************
   *
   *   shorten_pattern(...
   *
   *   This funtion shortens the pattern by
   *   deleting an element from it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcfg.
   *
   ************************************************/

void shorten_pattern(size, pattern, index,
                     current_width, width)
   short  *pattern;
   int    size, *index, *current_width, width;
{
   short *temp_pattern;
   int   i, new_index, new_width;
```

```
   temp_pattern = malloc(width*sizeof(short));

   for(i=0; i<width; i++)
      temp_pattern[i] = pattern[i];
   zero_line(pattern, width);

   new_index = 0;
   new_width = *current_width - size;

      /**********************************************
      *
      *   Increase the index by the amount we want
      *   to shorten the pattern.   Then copy the
      *   temp_pattern back to the pattern skipping
      *   over parts of the pattern by the amount
      *   we want to shorten it.
      *
      **********************************************/

   *index = (*index + size) % (*current_width);

   for(new_index=0; new_index<new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width = new_width;
   *index         = 0;

   free(temp_pattern);

}  /* ends shorten_pattern */




   /**********************************************
   *
   *   initialize_pattern(...
   *
   *   This function creates an initial pattern
```

```
   *   that is as wide as the pattern_width
   *   parameter.
   *
   ************************************************/

void initialize_pattern(pattern, current_width,
                        max_width, pattern_width,
                        index)
   short *pattern;
   int   *current_width,
         *max_width,
         *index,
         pattern_width;
{
   int i;

   for(i=0; i<pattern_width; i++)
      pattern[i] = i+PATTERN_START;

   *index         = 0;
   *current_width = pattern_width;
   *max_width     = pattern_width;
}  /* ends initialize_pattern */




   /************************************************
   *
   *   no_change(...
   *
   *   This function processes the pattern
   *   and does not make any changes to it.
   *
   ************************************************/

void no_change(pattern, processed_pattern, pp_index,
               current_width, index)
   short *pattern, *processed_pattern;
   int   pp_index, current_width, *index;
{
   processed_pattern[pp_index] =
      pattern[*index];
```

```
   *index = *index + 1;
   if(*index >= current_width)
      *index = 0;

}   /* ends no_change */




   /************************************************
   *
   *   zero_line(...
   *
   *   This function fills an int array with
   *   zeros.
   *
   ************************************************/

void zero_line(array, length)
   short  *array;
   int    length;
{
   int i;
   for(i=0; i<length; i++)
      array[i] = 0;
}  /* ends zero_line */




   /************************************************
   *
   *   lengthen_pattern(...
   *
   *   This funtion lengthens the pattern by
   *   inserting an element(s) into it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcdefgh.
   *
   ************************************************/

void lengthen_pattern(size, pattern, index,
                      current_width, width, max_width)
```

```
   short  *pattern;
   int    size, *index, *current_width,
          *width, *max_width;
{
   int  *temp_pattern;
   int  count, i, new_index, new_width;


   temp_pattern = malloc(KONSTANT*(*width)*sizeof(int));

   for(i=0; i<(*width); i++)
      temp_pattern[i] = pattern[i];

   zero_line(pattern, KONSTANT*(*width));

   for(count=0, new_index=0; count<size;
       count++, new_index++){
      pattern[new_index] =
         count + *max_width + PATTERN_START;
   }  /* ends loop over count */

   new_width = *current_width + size;

   for( ; new_index < new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width = new_width;
   *index         = 0;
   *max_width     = *max_width + size;

   free(temp_pattern);

}  /* ends lengthen_pattern */




   /***********************************************
   *
   *   random_substitution(...
```

```
   *
   *   This function takes the processed_pattern
   *   array and substitutes random values for each
   *   value in the array.
   *
   *   Fill the substitution_values array with
   *   random characters that are all printable
   *   (PATTERN_START to PATTERN_END).
   *
   ************************************************/

void random_substitution(processed_pattern, width)
   short  *processed_pattern;
   int    width;
{
   int  substitution_values[GRAY_LEVELS+1];
   int  i, place;

   get_random_values(substitution_values);

   for(i=0; i<(width); i++){
      place = processed_pattern[i];
      processed_pattern[i] =
         substitution_values[place];
   }  /* ends loop over i */

}  /* ends random_substitution */




   /************************************************
   *
   *   get_random_values(...
   *
   *   This function fills array with random values.
   *   The limit on the random values are from
   *   PATTERN_START to PATTERN_END.
   *
   ************************************************/

void get_random_values(array)
   int  array[];
{
```

```
   int i, number;

#ifdef NEVER
these lines worked ok, they used all the printable
characters from 0 through small z
   for(i=0; i<ASCII_SIZE; i++){
       number   = rand();
       number   = number % (PATTERN_END - PATTERN_START);
       number   = number + PATTERN_START;
       array[i] = number;
   }  /* ends loop over i */
#endif

#ifdef NEVER
/* Let's try something different, only use the
   characters 0-9 and A-Z
   0-9 are 48-57 decimal A-Z are 65-90 */

   for(i=0; i<ASCII_SIZE; i++){
       number   = rand();
       number   = number % 36;
       number   = number + PATTERN_START;
       if(number > 57   &&   number < 65)
          number = number + 7;
       array[i] = number;
   }  /* ends loop over i */
#endif

#ifdef NEVER
/* Let's try something different, only use the
   characters A-Z
   A-Z are 65-90 */

   for(i=0; i<GRAY_LEVELS+1; i++){
       number   = rand();
       number   = number % GRAY_LEVELS+1;
       array[i] = number;
   }  /* ends loop over i */
#endif


/* Let's try something different, only use the
   1's and 0's */

   for(i=0; i<GRAY_LEVELS+1; i++){
```

```
      number   = rand();
      number   = number % 2;
      if(number == 1) number = PATTERN_END;
      array[i] = number;
   }  /* ends loop over i */

}  /* ends get_random_values */




   /**********************************************
   *
   *  test_print_line(...
   *
   *   This is a debug function that prints
   *   an array of integers.
   *
   **********************************************/

void test_print_line(line, width)
   short  *line;
   int    width;
{
   int i;
   for(i=0; i<width; i++)
      printf("-%3d", line[i]);
   printf("\n");
}  /* ends test_print_line */
```

Listing 16.2 - The pstereo Program

```
   /**********************************************
   *
   *  file scstereo.c
   *
   *  Functions: This file contains
   *     main
   *     fill_line
   *     initialize_pattern
   *     s_lengthen_pattern
   *     no_change
   *     shorten_pattern
```

```
   *     special_substitution
   *     test_print_line
   *
   *  Purpose:
   *     This file contains a program that will
   *     make character based stereograms.
   *
   *  External Calls:
   *     none
   *
   *  Modifications:
   *     12 March 1995 - created
   *
   ***************************************************/


#include "cips.h"

#define PATTERN_START  48 /* the zero character    */
#define PATTERN_END   122 /* the small z character */
#define KONSTANT        2
#define ASCII_SIZE    256
#define LENGTH         80


void fill_line();
void initialize_pattern();
void s_lengthen_pattern();
void no_change();
void shorten_pattern();
void special_substitution();
void test_print_line();


main(argc, argv)
   char *argv[];
   int  argc;
{
   char *depth_line,
        *pattern,
        *processed_pattern,
        *pdest,
        response[LENGTH],
        special_text[LENGTH];
```

```
char last_character,
     this_character;

FILE *depth_file,
     *processed_pattern_file,
     *stereo_file;

int  current_width,
     j,
     index,
     location,
     max_width,
     pattern_width,
     pp_index,
     s_length,
     width;

if(argc != 6){
   printf(
   "\nusage: scstereo width-of-line "
   "depth-file-name"
   "\n                stereo-file-name "
   "processed-pattern-file-name"
   "\n                special-text\n");
   exit(1);
}

width = atoi(argv[1]);
strcpy(special_text, argv[5]);
s_length = strlen(special_text);

pattern           = malloc(KONSTANT*width);
depth_line        = malloc(KONSTANT*width);
processed_pattern = malloc(KONSTANT*width);

if((depth_file = fopen(argv[2], "rt")) == NULL){
   printf(
  "\ncstereo: Error opening input file %s\n",
   argv[3]);
   exit(3);
}

if((stereo_file = fopen(argv[3], "wt")) == NULL){
   printf(
 "\ncstereo: Error opening output file %s\n",
```

```
      argv[4]);
      exit(4);
   }

   if((processed_pattern_file =
      fopen(argv[4], "wt")) == NULL){
      printf(
      "\ncstereo: Error opening temp file %s\n",
      argv[5]);
      exit(5);
   }



      /*****************************************
      *
      *   This is the major loop of the program.
      *   It reads one line at a time from the
      *   depth file, processes that one line,
      *   and writes the resulting pattern to the
      *   processed pattern file.
      *
      ****************************************/

   while(fgets(depth_line,
               KONSTANT*width,
               depth_file)){

      fill_line(pattern, KONSTANT*width);
      fill_line(processed_pattern, KONSTANT*width);

      initialize_pattern(pattern,
                         &current_width,
                         &max_width,
                         s_length,
                         &index);

         /*******************************************
         *
         *   Loop through the characters in the
         *   depth_line.  Call one of the three
         *   processing routines based on the
         *   relationship between the last character
         *   and this character.
         *
```

```
   ****************************************/

last_character = depth_line[0];

pp_index = 0;

for(j=0; j<width; j++){
   this_character = depth_line[j];
   if(this_character == '\n')
      this_character = last_character;


   if(this_character > last_character)
      shorten_pattern(
         (this_character-last_character),
         pattern, &index, &current_width,
         width);


   if(this_character < last_character)
      s_lengthen_pattern(
         (last_character-this_character),
         pattern, &index, &current_width,
         &width, &max_width, s_length);



      /****************************************
      *
      *   Perform the no_change in every
      *   pass.  Do it after you have done
      *   the shorten and lenghten pattern.
      *
      ****************************************/

   no_change(pattern, processed_pattern,
             pp_index, current_width, &index);
   pp_index++;

   last_character = depth_line[j];

}  /* ends loop over j */

pdest    = strchr(processed_pattern, '\0');
location = pdest - processed_pattern;
```

```
      processed_pattern[location] = '\n';

      fputs(processed_pattern,
            processed_pattern_file);
      special_substitution(processed_pattern,
                           special_text, width);
      fputs(processed_pattern, stereo_file);


   }   /* ends the major loop */

   fclose(depth_file);
   fclose(processed_pattern_file);
   fclose(stereo_file);


   free(pattern);
   free(depth_line);
   free(processed_pattern);


   return(111);

}  /* ends main                 */




   /***********************************************
   *
   *   shorten_pattern(...
   *
   *   This funtion shortens the pattern by
   *   deleting an element from it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcfg.
   *
   ***********************************************/

void shorten_pattern(size, pattern, index,
                     current_width, width)
   char *pattern;
   int  size, *index, *current_width, width;
{
```

```
   char *temp_pattern;
   int  i, new_index, new_width;

   temp_pattern = malloc(KONSTANT*width);

   for(i=0; i<width; i++)
      temp_pattern[i] = pattern[i];
   fill_line(pattern, width);

   new_index = 0;
   new_width = *current_width - size;

      /*********************************************
      *
      *   Increase the index by the amount we want
      *   to shorten the pattern.   Then copy the
      *   temp_pattern back to the pattern skipping
      *   over parts of the pattern by the amount
      *   we want to shorten it.
      *
      *********************************************/

   *index = (*index + size) % (*current_width);

   for(new_index=0; new_index<new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width     = new_width;
   *index     = 0;

   free(temp_pattern);

}  /* ends shorten_pattern */




   /**********************************************
   *
   *   initialize_pattern(...
```

```
   *
   *   This function creates an initial pattern
   *   that is as wide as the pattern_width
   *   parameter.
   *
   ************************************************/

void initialize_pattern(pattern, current_width,
                        max_width, pattern_width,
                        index)
   char pattern[];
   int *current_width,
       *max_width,
       *index,
       pattern_width;
{
   int i;

   for(i=0; i<pattern_width; i++)
      pattern[i] = i+PATTERN_START;

   *index         = 0;
   *current_width = pattern_width;
   *max_width     = pattern_width;
}  /* ends initialize_pattern */




   /************************************************
   *
   *   no_change(...
   *
   *   This function processes the pattern
   *   and does not make any changes to it.
   *
   ************************************************/

void no_change(pattern, processed_pattern, pp_index,
               current_width, index)
   char *pattern, *processed_pattern;
   int  pp_index, current_width, *index;
{
   processed_pattern[pp_index] =
```

```
      pattern[*index];

   *index = *index + 1;
   if(*index >= current_width)
      *index = 0;

}   /* ends no_change */




   /***********************************************
   *
   *   fill_line(...
   *
   *   This function fills a character line
   *   with NULL characters.
   *
   ***********************************************/

void fill_line(line, length)
   char *line;
   int  length;
{
   int i;
   for(i=0; i<length; i++)
      line[i] = '\0';
}  /* ends fill_line */




   /***********************************************
   *
   *   s_lengthen_pattern(...
   *
   *   This funtion lengthens the pattern by
   *   inserting an element(s) into it.
   *   This is the special length pattern routine.
   *   It inserts only from the original pattern
   *   (the special-text).
   *
   ***********************************************/
```

```
void s_lengthen_pattern(size, pattern, index,
                        current_width, width,
                        max_width, s_length)
   char *pattern;
   int  size, *index, *current_width,
        *width, *max_width, s_length;
{
   char *temp_pattern;
   int  count, element, i, new_index, new_width;

   temp_pattern = malloc(KONSTANT*(*width));

   for(i=0; i<(*width); i++)
      temp_pattern[i] = pattern[i];

      /*****************************************
      *
      *   element is the value of the current
      *   element.  new_index points to its
      *   position in the special_text and moves
      *   back to the left by size.
      *
      *****************************************/

   element   = pattern[*index] - PATTERN_START;
   new_index = (element - size) % s_length;

      /*****************************************
      *
      *   Put a new pattern in the pattern array
      *   starting back at the new_index.
      *
      *****************************************/

   for(i=0; i<size; i++){
      pattern[i] = new_index + PATTERN_START;
      new_index++;
      if(new_index == s_length)
         new_index = 0;
   }  /* ends loop over i */

   new_width = size + *current_width;

   for(i=size; i<new_width; i++){
```

```
      pattern[i] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over i, count */

   *current_width = new_width;
   *index         = 0;

}  /* ends s_lengthen_pattern */




   /**********************************************
   *
   *   special_substitution(...
   *
   *   This function takes the processed_pattern
   *   array and substitutes the special text
   *   into it.
   *
   **********************************************/

void special_substitution(processed_pattern,
                          special_text, width)
   char *processed_pattern, special_text[];
   int  width;
{
   int  i, place;

   for(i=0; i<(KONSTANT*width); i++){
      if(processed_pattern[i] != '\n'   &&
         processed_pattern[i] != '\0'){
         place = processed_pattern[i] - PATTERN_START;
         processed_pattern[i] =
            special_text[place];
      }  /* ends if */
   }  /* ends loop over i */

}  /* ends special_substitution */
```

```
   /************************************************
   *
   *   test_print_line(...
   *
   *   This is a debug routine that prints an
   *   array of characters.
   *
   ************************************************/

void test_print_line(line, width)
   char *line;
   int  width;
{
   int i;
   for(i=0; i<width; i++)
      printf("%c", line[i]);
   printf("\n");
}  /* ends test_print_line */
```

Listing 16.3 - The scstereo Program

```
   /************************************************
   *
   *  file spstereo.c
   *
   *  Functions: This file contains
   *     main
   *     zero_line
   *     initialize_pattern
   *     s_lengthen_pattern
   *     no_change
   *     shorten_pattern
   *     special_substitution
   *     read_image_line
   *     write_image_line
   *     test_print_line
   *     equate_headers
   *
   *  Purpose:
   *     This file contains a program that will
   *     make pixel based colorfield stereograms.
```

```
   *
   *  External Calls:
   *      imageio.c - create_image_file
   *                  read_image_array
   *                  write_image_array
   *                  get_image_size
   *                  allocate_image_array
   *                  free_image_array
   *
   *  Modifications:
   *     29 April 1995 - created
   *     29 August 1998 - modified to work on
   *          entire images at once.
   *     24 September 1998 - modified to work with
   *          all I O routines in imageio.c.
   *
   ************************************************/


#include "cips.h"

#define KONSTANT        2
#define PATTERN_START   0
#define PATTERN_END   255
#define ASCII_SIZE    256

#undef  DEBUG_MODE
#define COMMAND_LINE_MODE


short   **depth_image,
        **pattern,
        **processed_pattern,
        **special_pixels;

void zero_line();
void initialize_pattern();
void s_lengthen_pattern();
void no_change();
void shorten_pattern();
void special_substitution();
void read_image_line();
void write_image_line();
void equate_headers();
void test_print_line();
```

```
main(argc, argv)
   char *argv[];
   int  argc;
{
   char depth_file_name[MAX_NAME_LENGTH],
        s_file_name[MAX_NAME_LENGTH],
        pattern_file_name[MAX_NAME_LENGTH],
        pp_file_name[MAX_NAME_LENGTH],
        response[MAX_NAME_LENGTH];

   FILE *depth_file,
        *pattern_file,
        *processed_pattern_file,
        *stereo_file;

   int  current_width,
        i,
        j,
        index,
        last_pixel,
        length_d,
        length_p,
        location,
        max_width,
        pattern_width,
        pp_index,
        s_length,
        this_pixel,
        width_d,
        width_p;

   long length, width;


   if(argc != 6){
      printf(
      "\nusage: spstereo pattern-file-name "
      "depth-file-name "
      "\n                stereo-file-name "
      "processed-pattern-file-name"
      "\n                length-of-pattern"
      "\n");
      exit(1);
```

```
}

strcpy(pattern_file_name, argv[1]);
strcpy(depth_file_name,   argv[2]);
strcpy(s_file_name,       argv[3]);
strcpy(pp_file_name,      argv[4]);
s_length           = atoi(argv[5]);

   /*********************************************
   *
   *   Ensure the input files exist.
   *
   *********************************************/

if(does_not_exist(pattern_file_name)){
   printf("\nThe pattern file %s does not exist",
          pattern_file_name);
   exit(1);
}

if(does_not_exist(depth_file_name)){
   printf("\nThe depth file %s does not exist",
          depth_file_name);
   exit(1);
}

   /*********************************************
   *
   *   The input files must be the same size.
   *
   *********************************************/

if(are_not_same_size(depth_file_name,
                     pattern_file_name)){
   printf(
    "\n Images %s and %s are not the same size",
    depth_file_name, pattern_file_name);
   exit(1);
}  /* ends if sizes not the same */

get_image_size(depth_file_name, &length, &width);

depth_image       = allocate_image_array(length, width);
pattern           = allocate_image_array(length, width);
processed_pattern = allocate_image_array(length, width);
```

```
special_pixels    = allocate_image_array(length, width);

create_image_file(depth_file_name, s_file_name);
create_image_file(depth_file_name, pp_file_name);

read_image_array(depth_file_name, depth_image);
read_image_array(pattern_file_name, special_pixels);

width         = width;
pattern_width = s_length;
length        = length;


   /************************************************
   *
   *   This is the major loop of the program.
   *   It reads one line at a time from the
   *   depth file, processes that one line,
   *   and writes the resulting pattern to the
   *   processed pattern file.
   *
   ************************************************/

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

   if(i%10 == 0) printf(" %d", i);

   zero_line(pattern[i], width);
   zero_line(processed_pattern[i], width);

   initialize_pattern(pattern[i],
                      &current_width,
                      &max_width,
                      s_length,
                      &index);

      /*******************************************
      *
      *   Loop through the pixels in the
      *   depth_image.  Call one of the three
      *   processing routines based on the
      *   relationship between the last pixel
      *   and this pixel.
      *
      *******************************************/
```

```
      last_pixel = depth_image[i][0];

      pp_index = 0;

      for(j=0; j<width; j++){
         this_pixel = depth_image[i][j];


         if(this_pixel > last_pixel)
            shorten_pattern(
               (this_pixel-last_pixel),
               pattern[i], &index, &current_width,
               width);


         if(this_pixel < last_pixel)
            s_lengthen_pattern(
               (last_pixel-this_pixel),
               pattern[i], &index, &current_width,
               &width, &max_width, s_length);



            /****************************************
            *
            *   Perform the no_change in every
            *   pass.  Do it after you have done
            *   the shorten and lenghten pattern.
            *
            ****************************************/

         no_change(pattern[i], processed_pattern[i],
                   pp_index, current_width, &index);
         pp_index++;


         if(index >= current_width)
            index = 0;

         last_pixel = depth_image[i][j];

      }  /* ends loop over j */
   }   /* ends loop over i */
```

```
   write_image_array(pp_file_name, processed_pattern);

   for(i=0; i<length; i++)
      special_substitution(processed_pattern[i],
                           special_pixels[i],
                           width);

   write_image_array(s_file_name, processed_pattern);

   free_image_array(depth_image, length);
   free_image_array(pattern, length);
   free_image_array(processed_pattern, length);
   free_image_array(special_pixels, length);

   return(111);

}  /* ends main                 */




   /************************************************
   *
   *   shorten_pattern(...
   *
   *   This funtion shortens the pattern by
   *   deleting an element from it.  For example,
   *   if the input pattern is abcdefg,
   *   the output pattern could be abcfg.
   *
   ************************************************/

void shorten_pattern(size, pattern, index,
                     current_width, width)
   short  *pattern;
   int    size, *index, *current_width, width;
{
   short  *temp_pattern;
   int    i, new_index, new_width;


   temp_pattern = malloc(width*sizeof(short));

   for(i=0; i<width; i++)
```

```
      temp_pattern[i] = pattern[i];
   zero_line(pattern, width);

   new_index = 0;
   new_width = *current_width - size;

      /*********************************************
      *
      *   Increase the index by the amount we want
      *   to shorten the pattern.   Then copy the
      *   temp_pattern back to the pattern skipping
      *   over parts of the pattern by the amount
      *   we want to shorten it.
      *
      *********************************************/

   *index = (*index + size) % (*current_width);

   for(new_index=0; new_index<new_width; new_index++){
      pattern[new_index] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over new_index */

   *current_width = new_width;
   *index         = 0;

   free(temp_pattern);

}  /* ends shorten_pattern */




   /*********************************************
   *
   *   initialize_pattern(...
   *
   *   This function creates an initial pattern
   *   that is as wide as the pattern_width
   *   parameter.
   *
   *********************************************/
```

```
void initialize_pattern(pattern, current_width,
                        max_width, pattern_width,
                        index)
   short *pattern;
   int   *current_width,
         *max_width,
         *index;
   long  pattern_width;
{
   int i;

   for(i=0; i<pattern_width; i++)
      pattern[i] = i+PATTERN_START;

   *index         = 0;
   *current_width = pattern_width;
   *max_width     = pattern_width;
}  /* ends initialize_pattern */




   /************************************************
   *
   *   no_change(...
   *
   *   This function processes the pattern
   *   and does not make any changes to it.
   *
   ************************************************/

void no_change(pattern, processed_pattern, pp_index,
               current_width, index)
   short *pattern, *processed_pattern;
   int   pp_index, current_width, *index;
{
   processed_pattern[pp_index] =
      pattern[*index];

   *index = *index + 1;
   if(*index >= current_width)
      *index = 0;
```

```
}   /* ends no_change */




   /***********************************************
   *
   *   zero_line(...
   *
   *   This function fills an int array with
   *   zeros.
   *
   ***********************************************/

void zero_line(array, length)
   short *array;
   long  length;
{
   int i;
   for(i=0; i<length; i++)
      array[i] = 0;
}  /* ends zero_line */




   /***********************************************
   *
   *  test_print_line(...
   *
   *   This is a debug function that prints
   *   an array of integers.
   *
   ***********************************************/

void test_print_line(line, width)
   short  *line;
   int    width;
{
   int i;
   for(i=0; i<width; i++)
      printf("-%3d", line[i]);
   printf("\n");
```

```
}  /* ends test_print_line */



   /***********************************************
   *
   *   s_lengthen_pattern(...
   *
   *   This funtion lengthens the pattern by
   *   inserting an element(s) into it.
   *   This is the special length pattern routine.
   *   It inserts only from the original pattern
   *   (the special-pixels).
   *
   ***********************************************/

void s_lengthen_pattern(size, pattern, index,
                        current_width, width,
                        max_width, s_length)
   int   size, *index, *current_width,
         *width, *max_width, s_length;
   short *pattern;
{
   short  *temp_pattern;
   int    count, element, i, new_index, new_width;

   temp_pattern = malloc((*width)*sizeof(short));

   for(i=0; i<(*width); i++)
      temp_pattern[i] = pattern[i];

      /*****************************************
      *
      *   element is the value of the current
      *   element.  new_index points to its
      *   position in the special_text and moves
      *   back to the left by size.
      *
      *****************************************/

   element   = pattern[*index] - PATTERN_START;
   new_index = (element - size) % s_length;

      /*****************************************
```

```
      *
      *   Put a new pattern in the pattern array
      *   starting back at the new_index.
      *
      ****************************************/

   for(i=0; i<size; i++){
      pattern[i] = new_index + PATTERN_START;
      new_index++;
      if(new_index == s_length)
         new_index = 0;
   }  /* ends loop over i */

   new_width = size + *current_width;

   for(i=size; i<new_width; i++){
      pattern[i] = temp_pattern[*index];
      *index = *index + 1;
      if(*index >= *current_width)
         *index = 0;
   }  /* ends loop over i, count */

   *current_width = new_width;
   *index         = 0;

   free(temp_pattern);

}  /* ends s_lengthen_pattern */




   /**********************************************
   *
   *   special_substitution(...
   *
   *   This function takes the processed_pattern
   *   array and substitutes the special pixels
   *   into it.
   *
   **********************************************/

void special_substitution(processed_pattern,
                          special_pixels, width)
```

```
   short *processed_pattern, *special_pixels;
   long  width;
{
   int  i, place;

   for(i=0; i<width; i++){
      place = processed_pattern[i] - PATTERN_START;
      processed_pattern[i] =
         special_pixels[place];
   }  /* ends loop over i */

}  /* ends special_substitution */
```

Listing 16.4 - The spstereo Program

F.17 Code Listings for Chapter 17

```
/***********************************************
*
*  file hidet.c
*
*  Functions: This file contains
*     main
*
*  Purpose:
*     This file contains the main calling
*     routine and subroutines to overlay
*     text on top of an image.
*
*  External Calls:
*      imageio.c - create_image_file
*                  read_image_array
*                  write_image_array
*                  get_image_size
*                  allocate_image_array
*                  free_image_array
*
*  Modifications:
*     16 February 1998 - created
```

```
    *     22 September 1998 - modified to work with
    *           all I O routines in imageio.c.
    *
    ************************************************/

#include "cips.h"

main(argc, argv)
   int argc;
   char *argv[];
{

   char  image_name[80], water_name[80];
   int   i, j;
   long  length, width;
   short factor;
   short **the_image;
   short **out_image;

   if(argc < 4){
    printf("\n\nNot enough parameters:");
    printf("\n");
    printf("\n   usage: hidet image-file text-file factor  ");
    exit(0);
   }

   strcpy(image_name, argv[1]);
   strcpy(water_name, argv[2]);
   factor = atoi(argv[3]);

   if(does_not_exist(image_name)){
      printf("\nERROR input file %s does not exist",
             image_name);
      exit(0);
   }

   if(does_not_exist(water_name)){
      printf("\nERROR input file %s does not exist",
             water_name);
      exit(0);
   }

   if(are_not_same_size(image_name, water_name)){
      printf("\nERROR images are not same size");
      exit(0);
```

```
   }

   get_image_size(image_name, &length, &width);

   the_image = allocate_image_array(length, width);
   out_image = allocate_image_array(length, width);

   read_image_array(image_name, the_image);
   read_image_array(water_name, out_image);

   for(i=0; i<length; i++){
      for(j=0; j<width; j++){
         if(out_image[i][j] != 0){
            the_image[i][j] = the_image[i][j] + factor;
            if(the_image[i][j] > GRAY_LEVELS)
               the_image[i][j] = GRAY_LEVELS;
         }  /* ends if */
      } /* ends loop over j */
   }  /* ends loop over i */

   write_image_array(image_name, the_image);

   free_image_array(the_image, length);
   free_image_array(out_image, length);

}  /* ends main  */
```

Listing 17.1 - Source Code to Hide a Watermark in an Image

```
/***********************************************
 *
 *  file stega.c
 *
 *  Functions: This file contains
 *     main
 *     hide_image
 *     hide_pixels
 *     uncover_image
 *     uncover_pixels
 *
```

```
    *  Purpose:
    *     This file contains the main calling
    *     routine and other routines that
    *     use steganography to hide one image
    *     inside another and then recover the
    *     hidden image.
    *
    *  External Calls:
    *      imageio.c - create_image_file
    *                  read_image_array
    *                  write_image_array
    *                  get_image_size
    *                  allocate_image_array
    *                  free_image_array
    *
    *  Modifications:
    *     5 April 1998 - created
    *     22 September 1998 - modified to work with
    *           all I O routines in imageio.c.
    *
    ************************************************/

#include "cips.h"
#define EIGHT 8



main(argc, argv)
   int argc;
   char *argv[];
{

   char  cover_image_name[80],
         message_image_name[80];
   int   hide    = 0,
         i,
         j,
         lsb,
         n,
         uncover = 0;

   long  clength,
         mlength,
         cwidth,
         mwidth;
```

```
short **the_image;
short **out_image;

   /*****************************************
   *
   *   Ensure the command line is correct.
   *
   *****************************************/

if(argc < 5){
   stega_show_usage();
   exit(0);
}

if(strcmp(argv[1], "-h") == 0){
   hide    = 1;
   uncover = 0;
}
if(strcmp(argv[1], "-u") == 0){
   hide    = 0;
   uncover = 1;
}
if(  (hide    == 0)   &&
     (uncover == 0)  ){
   printf("\nNiether hiding nor uncovering");
   printf("\nSo, quitting");
   exit(1);
}  /* ends if */

strcpy(cover_image_name, argv[2]);
strcpy(message_image_name, argv[3]);
n = atoi(argv[4]);


   /*****************************************
   *
   * Hide the cover image in the message image.
   *
   *****************************************/


if(hide){
   if(does_not_exist(cover_image_name)){
      printf("\n%s does not exist, quitting",
```

```
   cover_image_name);
}
if(does_not_exist(message_image_name)){
   printf("\n%s does not exist, quitting",
   message_image_name);
}

   /****************************************
   *
   *   Ensure both images have the same height
   *   and the cover image is eight times as
   *   wide as the message image.
   *   Also determine if the bit order is lsb
   *   first or not.
   *
   ****************************************/

get_image_size(cover_image_name,
               &clength, &cwidth);
get_image_size(message_image_name,
               &mlength, &mwidth);

if(mlength != clength){
   printf("\n\nmlength NOT EQUAL TO clength");
   printf("\nQUITING");
   exit(2);
}  /* ends if length not equal */

if(cwidth != (n*mwidth)){
   printf("\nCover image not wide enough");
   printf("\nQUITING");
   exit(3);
}  /* ends if cover image not wide enough */

lsb = get_lsb(cover_image_name);

   /****************************************
   *
   *   Allocate the two image arrays.
   *   Read the cover and message images and
   *   hide the message image.
   *
   ****************************************/

the_image = allocate_image_array(
```

```
                    clength, cwidth);
      out_image = allocate_image_array(
                    mlength, mwidth);
      read_image_array(cover_image_name, the_image);
      read_image_array(message_image_name, out_image);

      hide_image(the_image, out_image,
                 mlength, mwidth,
                 clength, cwidth,
                 lsb, n);
      write_image_array(cover_image_name, the_image);
   }  /* ends if hide */



      /*****************************************
      *
      *   Uncover the cover image from the
      *   message image.
      *
      *****************************************/

   if(uncover){
printf("\nMAIN> Uncover");

      if(does_not_exist(cover_image_name)){
         printf("\n%s does not exist, quitting",
         cover_image_name);
      }  /* ends if does_not_exist */

         /*****************************************
         *
         *   Create the message image to be the
         *   correct size.
         *
         *****************************************/

      get_image_size(cover_image_name,
                     &clength, &cwidth);
      mlength = clength;
      mwidth  = cwidth/n;
      create_resized_image_file(cover_image_name,
                                message_image_name,
                                mlength, mwidth);
      lsb = get_lsb(cover_image_name);
```

```
         /****************************************
         *
         *   Allocate the two image arrays.
         *   Read the cover image and uncover
         *   the message image.
         *
         ****************************************/

      the_image = allocate_image_array(
                      clength, cwidth);
      out_image = allocate_image_array(
                      mlength, mwidth);
      read_image_array(cover_image_name, the_image);
      uncover_image(the_image, out_image,
                    mlength, mwidth,
                    clength, cwidth,
                    lsb, n);
      write_image_array(message_image_name, out_image);

   }  /* ends if uncover */


   free_image_array(the_image, clength);
   free_image_array(out_image, mlength);

}  /* ends main  */



   /*********************************************
   *
   *   hide_image(...
   *
   *   This routine hides the message image in
   *   the cover image.  Loop through the pixels
   *   in the message image and call hide_pixels
   *   for every pixel in the message image.
   *
   *********************************************/


int hide_image(cover_image,
               message_image,
               mlength,
```

```
               mwidth,
               clength,
               cwidth,
               lsb,
               n)
   int   lsb, n;
   long  clength, cwidth, mlength, mwidth;
   short **cover_image,
         **message_image;
{
   char response[80];
   int h_counter        = 0;


   for(h_counter=0; h_counter<mwidth; h_counter++){
      hide_pixels(cover_image,
                  message_image,
                  h_counter,
                  h_counter*n,
                  lsb,
                  n,
                  mlength);
   }  /* ends loop over h_counter */

}  /* ends hide_image */




   /*********************************************
   *
   *   hide_pixels(..
   *
   *   This routine hides the bits in a pixel
   *   from the message image into the least
   *   significant bit of eight pixels in the
   *   cover image.
   *
   *   Do this one operation for every row of
   *   pixels in the message and cover images.
   *
   *********************************************/

int hide_pixels(cover_image,
                message_image,
```

```
               mie,
               cie,
               lsb,
               n,
               mlength)
   int   cie, lsb, mie, n;
   long  mlength;
   short **cover_image,
         **message_image;
{
   char result,
        new_message,
        sample;

   char mask1[EIGHT] =  {0x01,  /* 0000 0001 */
                         0x02,  /* 0000 0010 */
                         0x04,  /* 0000 0100 */
                         0x08,  /* 0000 1000 */
                         0x10,  /* 0001 0000 */
                         0x20,  /* 0010 0000 */
                         0x40,  /* 0100 0000 */
                         0x80}; /* 1000 0000 */


   char mask2[EIGHT] = {0xFE,  /* 1111 1110 */
                        0xFD,  /* 1111 1101 */
                        0xFB,  /* 1111 1011 */
                        0xF7,  /* 1111 0111 */
                        0xEF,  /* 1110 1111 */
                        0xDF,  /* 1101 1111 */
                        0xBF,  /* 1011 1111 */
                        0x7F}; /* 0111 1111 */


   int c_counter,
       i, j;

printf("\nHP> mie=%d   cie=%d   lsb=%d", mie, cie, lsb);

   for(i=0; i<mlength; i++){
      c_counter = 0;
      sample    = message_image[i][mie];

      for(j=n-1; j>-1; j--){
```

```
         /***********************************
         *
         *   Find out if the jth bit is
         *   a 1 or 0.  If it is non-zero,
         *   set the LSB of the message image's
         *   pixel.  Else, clear that LSB.
         *
         ************************************/

         new_message = cover_image[i][cie+c_counter];
         result      = sample & mask1[j];

         if(result != 0x00){ /* set lsb */
            if(lsb)
               new_message = new_message | mask1[0];
            else
               new_message = new_message | mask1[EIGHT];
         } /* ends if set lsb */

         else{ /* clear lsb */
            if(lsb)
               new_message = new_message & mask2[0];
            else
               new_message = new_message & mask2[EIGHT];
         }  /* ends if clear lsb */

         cover_image[i][cie+c_counter] = new_message;
         c_counter++;

      }  /* ends loop over j */
   }  /* ends loop over i */
}  /* ends hide_pixels */



   /**********************************************
   *
   *   uncover_image(...
   *
   *   This routine pulls the message image out
   *   of the cover image (the opposite of
   *   the cover_image routine).
   *
   **********************************************/
```

```
int uncover_image(cover_image,
                  message_image,
                  mlength,
                  mwidth,
                  clength,
                  cwidth,
                  lsb,
                  n)
   int   lsb, n;
   long  clength, cwidth, mlength, mwidth;
   short **cover_image,
         **message_image;
{
   int h_counter;

   for(h_counter=0; h_counter<mwidth; h_counter++){
      uncover_pixels(cover_image,
                     message_image,
                     h_counter,
                     h_counter*n,
                     lsb,
                     n,
                     mlength);
   }  /* ends loop over h_counter */

}  /* ends uncover_image */



   /*********************************************
   *
   *   uncover_pixels(...
   *
   *   This routine pulls the eight bits that
   *   make up a pixel in the message image
   *   out of the LSB of eight pixels in the
   *   cover image.  It is the opposite of the
   *   cover_pixels routine.
   *
   *********************************************/

int uncover_pixels(cover_image,
                message_image,
                mie,
                cie,
```

```
               lsb,
               n,
               mlength)
   int   cie, lsb, mie, n;
   long  mlength;
   short **cover_image,
         **message_image;
{
   char result,
        new_message,
        sample;

   char mask1[EIGHT] =
                       {0x80,  /* 1000 0000 */
                        0x40,  /* 0100 0000 */
                        0x20,  /* 0010 0000 */
                        0x10,  /* 0001 0000 */
                        0x08,  /* 0000 1000 */
                        0x04,  /* 0000 0100 */
                        0x02,  /* 0000 0010 */
                        0x01}; /* 0000 0001 */
   char mask2[EIGHT] =
                      {0x7F,  /* 0111 1111 */
                       0xBF,  /* 1011 1111 */
                       0xDF,  /* 1101 1111 */
                       0xEF,  /* 1110 1111 */
                       0xF7,  /* 1111 0111 */
                       0xFB,  /* 1111 1011 */
                       0xFD,  /* 1111 1101 */
                       0xFE}; /* 1111 1110 */

   int c, c_counter, i, j;
printf("\nUP> mie=%d   cie=%d   lsb=%d",
mie, cie, lsb);

      /*************************************
      *
      *  If a pixel in the cover image is
      *  odd, its lsb has been set, so
      *  the corresponding bit in the
      *  message image should be set.
      *
      *************************************/

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

```
      c = n-1;
      new_message = 0x00;
      for(j=0; j<n; j++){
         if(is_odd(cover_image[i][cie+j])){
            /* set bit c */
            if(lsb)
               new_message = new_message | mask1[j];
            else
               new_message = new_message | mask1[c];
         }  /* ends if is_odd */
         c--;
      }  /* ends loop over j */
      message_image[i][mie] = new_message;
   }  /* ends loop over i */
}  /* ends uncover_pixels */



   /*********************************************
   *
   *   is_odd(...
   *
   *   This routine determines if a short is
   *   an odd number.  If it is, this routine
   *   returns a 1, else it returns a 0.
   *
   *********************************************/

int is_odd(number)
   short number;
{
   int result = 0;
   result     = number % 2;
   return(result);
}  /* ends is_odd */


stega_show_usage()
{
printf("\n\nNot enough parameters:");
printf("\n");
printf("   "
"\nstega -h cover-image-name message-image-name n"
"\n       to hide the message image in the cover image"
"\n                 or"
```

```
"\nstega -u cover-image-name message-image-name n"
"\n       to uncover the cover image from "
"the message image");
}
```

Listing 17.2 - Source Code to Hide "Unhide" a Message Image in a Cover Image

F.18 Code Listings for Chapter 18

```
echo off


rem   erode.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program calls the erosion program
rem   for the input file and erodes it three times.

rem   The command is:
rem   erode input-file output-file working-dir

rem   Example:
rem      erode a.tif b.tif  c:
rem   will run erosion three times on
rem   a.tif and create the file b.tif.  It will create
rem   and then delete several temporary files in
rem   the c: directory.

rem   Check for the right number of parameters
if "%3" == "" goto usage

rem   Run erosion three times
rem   and delete the temporary files.
mainsk %1 %3tmp1.tif mer 200 4
mainsk %3tmp1.tif %3tmp2.tif mer 200 4
```

```
mainsk %3tmp2.tif %2 mer 200 4
del    %3tmp1.tif
del    %3tmp2.tif

goto end

:usage
echo .
echo usage:   erode input-file output-file working-dir
echo .
:end
```

Listing 18.1 - A .bat File to Repeat Erosion Three Times

```
echo off


rem   dilate.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program calls the dilation program
rem   for the input file and dilates it three times.

rem   The command is:
rem   dilate input-file output-file working-dir

rem   Example:
rem      dilate a.tif b.tif  c:
rem   will run dilation three times on
rem   a.tif and create the file b.tif.  It will create
rem   and then delete several temporary files in
rem   the c: directory.

rem   Check for the right number of parameters
if "%3" == "" goto usage
```

```
rem   Run dilation three times
rem   and delete the temporary files.
mainsk %1 %3tmp1.tif mdi 200 4
mainsk %3tmp1.tif %3tmp2.tif mdi 200 4
mainsk %3tmp2.tif %2 mdi 200 4
del    %3tmp1.tif
del    %3tmp2.tif

goto end

:usage
echo .
echo usage:   dilate input-file output-file working-dir
echo .
:end
```

Listing 18.2 - A .bat File to Repeat Dilation Three Times

```
echo off


rem   median.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program calls the median filter
rem   for the input file and performs either a 3x3,
rem   5x5, and 7x7 median filter three times.

rem   The command is:
rem   median input-file output-file size working-dir

rem   Example:
rem      median a.tif b.tif 5 c:
rem   will run a 5x5 median filter three times on
rem   a.tif and create the file b.tif.  It will create
rem   and then delete several temporary files in
```

```
rem   the c: directory.

rem   Check for the right number of parameters
if "%4" == "" goto usage

rem   Run the median filter three times
rem   and delete the temporary files.
mainfilt %1 %4tmp1.tif %3 m
mainfilt %4tmp1.tif %4tmp2.tif %3 m
mainfilt %4tmp2.tif %2 %3 m
del %4tmp1.tif
del %4tmp2.tif

goto end

:usage
echo .
echo usage:   median input-file output-file size working-dir
echo .
:end
```

Listing 18.3 - A .bat File to Repeat a Median Filter Three Times

```
echo off


rem   bedge.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program calls all the edge detectors
rem   for the input file.

rem   The command is:
rem   bedge input-file output-dir output-file-prefix

rem   Example:
```

```
rem      bedge a.tif c: aa
rem   will create the files:
rem   c:aa1.tif c:aa2.tif c:aa3.tif c:aa4.tif
rem   c:aa5.tif c:aa6.tif c:aa7.tif c:aa8.tif
rem   c:aa9.tif c:aa10.tif c:aa11.tif

rem   Check for the right number of parameters
if "%3" == "" goto usage

rem   Run the edge detector 10 times
medge Q %1 %2%31.tif   1 2
medge B %1 %2%32.tif   1 2 1
medge B %1 %2%33.tif   1 2 2
medge B %1 %2%34.tif   1 2 3
medge H %1 %2%35.tif   1 2
medge D %1 %2%36.tif   1 2
medge C %1 %2%37.tif   1 2
medge R %1 %2%38.tif   1 2 3
medge V %1 %2%39.tif   1 2
medge G %1 %2%310.tif  1 2 3

goto end

:usage
echo .
echo usage:   bedge input-file output-dir output-file-prefix
echo .
:end
```

Listing 18.4 - A .bat File to Run all the Edge Detectors for Comparison

```
echo off


rem   lowfilt.bat
rem   Dwayne Phillips
rem   5-30-97
```

```
rem   This .bat program calls all the low pass filters
rem   for the input file.

rem   The command is:
rem   lowfilt input-file output-dir output-file-prefix

rem   Example:
rem      lowfilt a.tif c: aa
rem   will create the files
rem   c:aa6.tif c:aa9.tif c:aa10.tif c:aa16.tif c:aa32.tif

rem   Check for the right number of parameters
if "%3" == "" goto usage

rem   Run the five low pass filters
mfilter %1 %2%36.tif g L 6
mfilter %1 %2%39.tif g L 9
mfilter %1 %2%310.tif g L 10
mfilter %1 %2%316.tif g L 16
mfilter %1 %2%332.tif g L 32

goto end

:usage
echo .
echo usage: lowfilt input-file output-dir output-file-prefix
echo .
:end
```

Listing 18.5 - A .bat File to Run Five Low-Pass Filters for Comparison

```
echo off


rem   med357.bat
rem   Dwayne Phillips
```

```
rem   5-30-97

rem   This .bat program calls the median filter
rem   for the input file and performs a 3x3,
rem   5x5, and 7x7 median filter.

rem   The command is:
rem   med357 input-file output-dir output-file-prefix

rem   Example:
rem      med357 a.tif c: aa
rem   will create the files c:aa3.tif c:aa5.tif c:aa7.tif

rem   Check for the right number of parameters
if "%3" == "" goto usage

rem   Run the three median filters
mfilter %1 %2%33.tif m 3
mfilter %1 %2%35.tif m 5
mfilter %1 %2%37.tif m 7

goto end

:usage
echo .
echo usage: med357 input-file output-dir output-file-prefix
echo .
:end
```

Listing 18.6 - A .bat File to Run Three Median Filters for Comparison

```
echo off


rem   blabel.bat
rem   Dwayne Phillips
rem   5-30-97
```

```
rem   This .bat program creates a label
rem   and places it on an image.
rem   YOU MUST TYPE THE MESSAGE INTO THE
rem   .BAT FILE.  THE MESSAGE WILL NOT COME
rem   FROM THE .BAT FILE COMMAND LINE

rem   The command is:
rem   blabel image-file output-file working-dir il ie

rem   Example:
rem      blabel a.tif out.tif c: 25 30
rem   will create the file c:out.tif
rem   The message created in ilabel will
rem   be placed in the output image at
rem   line 25 and element 30

rem   Check for the right number of parameters
if "%5" == "" goto usage

rem   Create the message in temporary
rem   files in the working directory
rem   THIS IS WHERE YOU ENTER YOUR MESSAGE

copy     %1 %2
ilabel   %3tmp1.tif %4 %5 sample text
mainsk   %3tmp1.tif %3tmp2.tif mdi 200 4
mainsk   %3tmp2.tif %3tmp3.tif mdi 200 4
boolean  %3tmp1.tif %3tmp3.tif %3tmp4.tif xor
mainover %3tmp4.tif %1 %2 greater

del %3tmp1.tif
del %3tmp2.tif
del %3tmp3.tif
del %3tmp4.tif

goto end

:usage
echo .
```

```
echo usage: blabel image-file output-file working-dir il ie
echo .
:end
```

Listing 18.7 - A .bat File to Create a Label and Lay it on an Image

```
echo off


rem   fourside.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program places four images
rem   side by side by side by side.
rem   The final configuration is 1 2
rem                              3 4

rem   The command is:
rem   fourside file1 file2 file3 file4 output-file working-dir

rem   Example:
rem      fourside 1.tif 2.tif 3.tif 4.tif out.tif c:
rem   will create the file out.tif
rem   It will also create and then delete some temporary
rem   files in c:

rem   Check for the right number of parameters
if "%6" == "" goto usage

rem   Run the side commands
side %1 %2 %6tmp1.tif side
side %3 %4 %6tmp2.tif side
side %6tmp1.tif %6tmp2.tif %5 top
del %6tmp1.tif
del %6tmp2.tif
```

```
goto end

:usage
echo .
echo usage: fourside file1 file2 file3 file4 output-file working-dir
echo .
:end
```

Listing 18.8 - A .bat File to Paste Four Images Together

```
echo off


rem   improve.bat
rem   Dwayne Phillips
rem   5-30-97

rem   This .bat program performs histogram
rem   equalization and high pass filtering.

rem   The command is:
rem improve in-file out-file working-dir

rem   Example:
rem   improve a.tif b.tif c:
rem   will perform histogram equalization and high
rem   pass filter and write the result to b.tif.
rem   It will create then delete temporary files
rem   in c:.

rem   Check for the right number of parameters
if "%3" == "" goto usage

rem   Run the processes
histeq %1 %7tmp1.tif
mfilter %7tmp1.tif %2 g H 1
del     %7tmp1.tif
```

```
goto end

:usage
echo .
echo usage: improve in-file out-file working-dir
echo .
:end
```

Listing 18.9 - A .bat File to Equalize and Filter an Image

F.19 Code Listings for Chapter 19

```
#####################################################################
# Visual Tcl v1.11 Project
#

##################################
# GLOBAL VARIABLES
#
global widget;
    set widget(AddSubtract) {.top24}
    set widget(Boolean) {.top45}
    set widget(CreateFile) {.top34}
    set widget(CreateHistogram) {.top11}
    set widget(CutPaste) {.top14}
    set widget(EDHelper) {.top25}
    set widget(Edge) {.top38}
    set widget(EdgeHelp) {.top26}
    set widget(ErosionDilation) {.top66}
    set widget(FHelp) {.top44}
    set widget(Filter) {.top27}
    set widget(GeoHelp) {.top55}
    set widget(Geometry) {.top54}
    set widget(HalfTone) {.top12}
    set widget(ImageNumbers) {.top20}
    set widget(MoreSegment) {.top43}
    set widget(Operations) {.top13}
    set widget(Overlay) {.top41}
    set widget(Round) {.top28}
```

```
    set widget(Segment) {.top46}
    set widget(Steganography) {.top19}
    set widget(Stereogram) {.top29}
    set widget(Stretch) {.top18}
    set widget(Texture) {.top39}
    set widget(TextureHelp) {.top64}
    set widget(Warp) {.top63}
    set widget(rev,.top11) {CreateHistogram}
    set widget(rev,.top12) {HalfTone}
    set widget(rev,.top13) {Operations}
    set widget(rev,.top14) {CutPaste}
    set widget(rev,.top18) {Stretch}
    set widget(rev,.top19) {Steganography}
    set widget(rev,.top20) {ImageNumbers}
    set widget(rev,.top24) {AddSubtract}
    set widget(rev,.top25) {EDHelper}
    set widget(rev,.top26) {EdgeHelp}
    set widget(rev,.top27) {Filter}
    set widget(rev,.top28) {Round}
    set widget(rev,.top29) {Stereogram}
    set widget(rev,.top34) {CreateFile}
    set widget(rev,.top38) {Edge}
    set widget(rev,.top39) {Texture}
    set widget(rev,.top41) {Overlay}
    set widget(rev,.top43) {MoreSegment}
    set widget(rev,.top44) {FHelp}
    set widget(rev,.top45) {Boolean}
    set widget(rev,.top46) {Segment}
    set widget(rev,.top54) {Geometry}
    set widget(rev,.top55) {GeoHelp}
    set widget(rev,.top63) {Warp}
    set widget(rev,.top64) {TextureHelp}
    set widget(rev,.top66) {ErosionDilation}

##################################
# USER DEFINED PROCEDURES
#
proc init {argc argv} {

}

init $argc $argv


proc {main} {argc argv} {
```

```
}

proc {Window} {args} {
global vTcl
    set cmd [lindex $args 0]
    set name [lindex $args 1]
    set newname [lindex $args 2]
    set rest [lrange $args 3 end]
    if {$name == "" || $cmd == ""} {return}
    if {$newname == ""} {
        set newname $name
    }
    set exists [winfo exists $newname]
    switch $cmd {
        show {
            if {$exists == "1" && $name != "."}
                {wm deiconify $name; return}
            if {[info procs vTclWindow(pre)$name] != ""} {
                eval "vTclWindow(pre)$name $newname $rest"
            }
            if {[info procs vTclWindow$name] != ""} {
                eval "vTclWindow$name $newname $rest"
            }
            if {[info procs vTclWindow(post)$name] != ""} {
                eval "vTclWindow(post)$name $newname $rest"
            }
        }
        hide    { if $exists {wm withdraw $newname; return} }
        iconify { if $exists {wm iconify $newname; return} }
        destroy { if $exists {destroy $newname; return} }
    }
}

##################################
# VTCL GENERATED GUI PROCEDURES
#

proc vTclWindow. {base} {
    if {$base == ""} {
        set base .
    }
    ###################
    # CREATING WIDGETS
    ###################
```

```
    wm focusmodel $base passive
    wm geometry $base 200x200+0+0
    wm maxsize $base 1028 748
    wm minsize $base 132 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm withdraw $base
    wm title $base "vt"
    ###################
    # SETTING GEOMETRY
    ###################
}

proc vTclWindow.top11 {base} {
    if {$base == ""} {
        set base .top11
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -cursor xterm -menu .top11.m17
    wm focusmodel $base passive
    wm geometry $base 198x228+364+167
    wm maxsize $base 1028 748
    wm minsize $base 132 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Create Histogram"
    button $base.but27  -command {Window hide $widget(CreateHistogram)} \
     -text exit
    button $base.but28  -command {exec himage $CHInFile $CHOutFile}  \
     -text {Create Histogram (himage)}
    entry $base.ent29  -textvariable CHInFile
    entry $base.ent30  -textvariable CHOutFile
    button $base.but17  -command {exec histeq $CHInFile $CHOutFile}  \
     -text {Equalize Histogram (histeq)}
    message $base.mes18  -aspect 500 -padx 6 -pady 3 -text {Input File}
    message $base.mes19  -aspect 500 -padx 6 -pady 3 -text {Output File}
    menu $base.m17  -cursor {}
    ###################
    # SETTING GEOMETRY
    ###################
```

```
    place $base.but27  -x 150 -y 185 -anchor nw -bordermode ignore
    place $base.but28  -x 10 -y 10 -anchor nw -bordermode ignore
    place $base.ent29  -x 10 -y 110 -anchor nw -bordermode ignore
    place $base.ent30  -x 10 -y 160 -anchor nw -bordermode ignore
    place $base.but17  -x 10 -y 40 -anchor nw -bordermode ignore
    place $base.mes18  -x 15 -y 85 -anchor nw -bordermode ignore
    place $base.mes19  -x 10 -y 135 -anchor nw -bordermode ignore
}

proc vTclWindow.top12 {base} {
    if {$base == ""} {
        set base .top12
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 193x218+405+134
    wm maxsize $base 1028 748
    wm minsize $base 132 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Halftoning"
    button $base.but20  -command {Window hide $widget(HalfTone)} \
     -text exit
    button $base.but21  \
     -command {exec halftone $HTInFile $HTOutFile $HTThresh}  \
     -text {Create Halftone Image (halftone)}
    entry $base.ent22  -textvariable HTInFile
    message $base.mes23  -aspect 555 -padx 6 -pady 3 -text {Input File}
    entry $base.ent24  -textvariable HTOutFile
    message $base.mes25  -aspect 555 -padx 6 -pady 3 -text {Output File}
    entry $base.ent26  -textvariable HTThresh
    message $base.mes27  -aspect 555 -padx 6 -pady 3 -text Threshold
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but20  -x 150 -y 175 -anchor nw -bordermode ignore
    place $base.but21  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.ent22  -x 10 -y 60 -anchor nw -bordermode ignore
    place $base.mes23  -x 10 -y 35 -anchor nw -bordermode ignore
```

```
    place $base.ent24  -x 10 -y 110 -anchor nw -bordermode ignore
    place $base.mes25  -x 10 -y 85 -anchor nw -bordermode ignore
    place $base.ent26  -x 10 -y 160 -anchor nw -bordermode ignore
    place $base.mes27  -x 5 -y 135 -anchor nw -bordermode ignore
}

proc vTclWindow.top13 {base} {
    if {$base == ""} {
        set base .top13
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 129x144+428+54
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Image Operations"
    button $base.but19  -command {Window hide $widget(Operations)} \
     -text exit
    button $base.but21  -command {Window show $widget(AddSubtract)} \
     -text {Add or Subtract}
    button $base.but22  -command {Window show $widget(CutPaste)} \
     -text {Cut and Paste}
    button $base.but23  -command {Window show $widget(CreateFile)} \
     -text {Create a File}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but19  -x 75 -y 100 -anchor nw -bordermode ignore
    place $base.but21  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but22  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.but23  -x 5 -y 65 -anchor nw -bordermode ignore
}

proc vTclWindow.top14 {base} {
    if {$base == ""} {
        set base .top14
    }
```

```
if {[winfo exists $base]} {
    wm deiconify $base; return
}
###################
# CREATING WIDGETS
###################
toplevel $base -class Toplevel  -cursor xterm
wm focusmodel $base passive
wm geometry $base 171x256+422+233
wm maxsize $base 804 585
wm minsize $base 104 1
wm overrideredirect $base 0
wm resizable $base 1 1
wm title $base "Cut and Paste"
button $base.but44  -command {Window hide $widget(CutPaste)} \
 -text exit
button $base.but45   \
 -command {exec mancp $CPIn $CPOut $CPinil $CPinie $CPinll \
  $CPinle $CPoutil $CPoutie}  -text {Cut and Paste}
entry $base.ent46  -textvariable CPIn
message $base.mes47  -aspect 333 -padx 5 -pady 2 -text {Input File}
message $base.mes48  -aspect 443 -padx 5 -pady 2 -text {Output File}
entry $base.ent49  -textvariable CPOut
message $base.mes50  -aspect 333 -padx 5 -pady 2 -text {Input il}
entry $base.ent51  -textvariable CPinil
entry $base.ent52  -textvariable CPinie
entry $base.ent53  -textvariable CPinle
entry $base.ent54  -textvariable CPinll
message $base.mes55  -aspect 333 -padx 5 -pady 2 -text {Input ie}
message $base.mes56  -aspect 333 -padx 5 -pady 2 -text {Input ll}
message $base.mes57  -aspect 333 -padx 5 -pady 2 -text {Input le}
entry $base.ent58  -textvariable CPoutie
entry $base.ent59  -textvariable CPoutil
message $base.mes60  -aspect 333 -padx 5 -pady 2 -text {Output il}
message $base.mes61  -aspect 333 -padx 5 -pady 2 -text {Output ie}
###################
# SETTING GEOMETRY
###################
place $base.but44  -x 130 -y 175 -anchor nw -bordermode ignore
place $base.but45  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.ent46  -x 5 -y 60 -anchor nw -bordermode ignore
place $base.mes47  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.mes48  -x 5 -y 80 -anchor nw -bordermode ignore
place $base.ent49  -x 5 -y 105 -anchor nw -bordermode ignore
place $base.mes50  -x 5 -y 125 -anchor nw -bordermode ignore
```

```
    place $base.ent51  -x 5 -y 145 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent52  -x 65 -y 145 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent53  -x 65 -y 185 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent54  -x 5 -y 185 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes55  -x 65 -y 125 -anchor nw -bordermode ignore
    place $base.mes56  -x 5 -y 165 -anchor nw -bordermode ignore
    place $base.mes57  -x 65 -y 165 -anchor nw -bordermode ignore
    place $base.ent58  -x 65 -y 225 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent59  -x 5 -y 225 -width 31 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes60  -x 5 -y 205 -anchor nw -bordermode ignore
    place $base.mes61  -x 65 -y 205 -anchor nw -bordermode ignore
}

proc vTclWindow.top17 {base} {
    if {$base == ""} {
        set base .top17
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel \
        -menu .top17.m17
    wm focusmodel $base passive
    wm geometry $base 334x344+15+119
    wm maxsize $base 1028 748
    wm minsize $base 132 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm deiconify $base
    wm title $base "CIPS"
    message $base.01 \
        -aspect 500 -padx 6 -pady 3 -text {The C Image Processing System}
    button $base.02 \
        -command {Window show $widget(CreateHistogram)} -text Histograms
    button $base.but22 \
        -command {Window show $widget(ImageNumbers)} \
```

```
    -text {Look at Image Numbers}
button $base.but18 \
    -command {Window show $widget(HalfTone)} -text Halftoning
button $base.but43 \
    -command {Window show $widget(Round)} -text {Roundoff Image}
button $base.but19 \
    -command {Window show $widget(Edge)} -text {Edge Detection}
button $base.but29 \
    -command {Window show $widget(Filter)} -text Filtering
button $base.but20 \
    -command {Window show $widget(Operations)} \
    -text {Add/Sub Cut/Paste Create}
button $base.but21 \
    -command {Window show $widget(Segment)} -text Segmentation
button $base.but46 \
    -command {Window show $widget(MoreSegment)}  \
    -text {More Segmentation}
button $base.but23 \
    -command {Window show $widget(ErosionDilation)} \
    -text {Erosion & Dilation}
button $base.but24 \
    -command {Window show $widget(Boolean)} -text {Boolean Operations}
button $base.but44 \
    -command {Window show $widget(Overlay)} -text {Overlay Operations}
button $base.but25 \
    -command {Window show $widget(Geometry)} \
    -text {Geometry (rotate & stretch)}
button $base.but65 \
    -command {Window show $widget(Warp)} -text Warping
button $base.but41 \
    -command {Window show $widget(Texture)} -text {Texture Operations}
button $base.but26 \
    -command {Window show $widget(Stereogram)} -text Stereograms
button $base.but30 \
    -command {Window show $widget(Steganography)} -text Steganography
menu $base.m17 \
    -cursor {}
button $base.but27 \
    -command {Window show $widget(Stretch)} -text Stretch
###################
# SETTING GEOMETRY
###################
place $base.01 \
    -x 5 -y 5 -anchor nw -bordermode ignore
place $base.02 \
```

```
        -x 5 -y 45 -anchor nw -bordermode ignore
    place $base.but22 \
        -x 5 -y 75 -anchor nw -bordermode ignore
    place $base.but18 \
        -x 5 -y 105 -anchor nw -bordermode ignore
    place $base.but43 \
        -x 5 -y 135 -anchor nw -bordermode ignore
    place $base.but19 \
        -x 5 -y 165 -anchor nw -bordermode ignore
    place $base.but29 \
        -x 5 -y 195 -anchor nw -bordermode ignore
    place $base.but20 \
        -x 5 -y 225 -anchor nw -bordermode ignore
    place $base.but21 \
        -x 5 -y 255 -anchor nw -bordermode ignore
    place $base.but46 \
        -x 5 -y 285 -anchor nw -bordermode ignore
    place $base.but23 \
        -x 180 -y 45 -anchor nw -bordermode ignore
    place $base.but24 \
        -x 180 -y 75 -anchor nw -bordermode ignore
    place $base.but44 \
        -x 180 -y 105 -anchor nw -bordermode ignore
    place $base.but25 \
        -x 180 -y 135 -anchor nw -bordermode ignore
    place $base.but65 \
        -x 180 -y 165 -anchor nw -bordermode ignore
    place $base.but41 \
        -x 180 -y 195 -anchor nw -bordermode ignore
    place $base.but26 \
        -x 180 -y 225 -anchor nw -bordermode ignore
    place $base.but30 \
        -x 180 -y 255 -anchor nw -bordermode ignore
    place $base.but27 \
        -x 180 -y 285 -anchor nw -bordermode ignore
}

proc vTclWindow.top19 {base} {
    if {$base == ""} {
        set base .top19
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
```

```
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x255+411+173
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Steganography"
    button $base.but20  -command {Window hide $widget(Steganography)}\
     -text exit
    message $base.mes21  -padx 5 -pady 2 -text {Cover Image}
    entry $base.ent22  -textvariable SCoverImage
    message $base.mes23  -aspect 444 -padx 5 -pady 2 -text {Message Image}
    entry $base.ent24  -textvariable SMessageImage
    message $base.mes25  -aspect 333 -padx 5 -pady 2 -text {N (usually 8)}
    entry $base.ent26  -textvariable SN
    button $base.but27  \
    -command {exec stega -h $SCoverImage $SMessageImage $SN} \
    -text Hide
    button $base.but28  \
    -command {exec stega -u $SCoverImage $SMessageImage $SN}  \
    -text Uncover
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but20  -x 160 -y 215 -anchor nw -bordermode ignore
    place $base.mes21  -x 5 -y 75 -anchor nw -bordermode ignore
    place $base.ent22  -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.mes23  -x 5 -y 115 -anchor nw -bordermode ignore
    place $base.ent24  -x 5 -y 135 -anchor nw -bordermode ignore
    place $base.mes25  -x 5 -y 155 -anchor nw -bordermode ignore
    place $base.ent26  -x 5 -y 175 -anchor nw -bordermode ignore
    place $base.but27  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but28  -x 5 -y 35 -anchor nw -bordermode ignore
}

proc vTclWindow.top20 {base} {
    if {$base == ""} {
        set base .top20
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
```

```
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x174+420+136
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Show Image Numbers"
    button $base.but21  -command {exec dumpi $INInFile $INTextFile}  \
     -text {Dump Image Numbers (dumpi)}
    entry $base.ent22  -textvariable INTextFile
    entry $base.ent23  -textvariable INInFile
    message $base.mes24  -aspect 333 -padx 5 -pady 2 \
     -text {Input File}
    message $base.mes25  -aspect 444 -padx 5 -pady 2 \
     -text {Output Text File}
    button $base.but26  -command {Window hide $widget(ImageNumbers)} \
     -text exit
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but21  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.ent22  -x 5 -y 105 -anchor nw -bordermode ignore
    place $base.ent23  -x 5 -y 60 -anchor nw -bordermode ignore
    place $base.mes24  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.mes25  -x 5 -y 80 -anchor nw -bordermode ignore
    place $base.but26  -x 145 -y 130 -anchor nw -bordermode ignore
}

proc vTclWindow.top24 {base} {
    if {$base == ""} {
        set base .top24
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -cursor xterm
    wm focusmodel $base passive
    wm geometry $base 221x148+419+296
```

```
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Add or Subtract"
    message $base.mes25  -aspect 444 -padx 5 -pady 2 -text {In Image 1}
    message $base.mes26  -aspect 444 -padx 5 -pady 2 -text {In Image 2}
    message $base.mes27  -aspect 444 -padx 5 -pady 2 -text {Ouput Image}
    entry $base.ent28  -textvariable ASIn1
    entry $base.ent29  -textvariable ASIn2
    entry $base.ent30  -textvariable ASOut
    button $base.but31  -command {Window hide $widget(AddSubtract)} \
     -text exit
    button $base.but32  -command {exec mainas $ASIn1 $ASIn2 $ASOut a} \
     -text Add
    button $base.but33  -command {exec mainas $ASIn1 $ASIn2 $ASOut s} \
     -text Subtract
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.mes25  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes26  -x 5 -y 45 -anchor nw -bordermode ignore
    place $base.mes27  -x 5 -y 85 -anchor nw -bordermode ignore
    place $base.ent28  -x 5 -y 25 -anchor nw -bordermode ignore
    place $base.ent29  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.ent30  -x 5 -y 110 -anchor nw -bordermode ignore
    place $base.but31  -x 160 -y 85 -anchor nw -bordermode ignore
    place $base.but32  -x 150 -y 5 -anchor nw -bordermode ignore
    place $base.but33  -x 150 -y 35 -anchor nw -bordermode ignore
}

proc vTclWindow.top25 {base} {
    if {$base == ""} {
        set base .top25
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 283x251+44+44
    wm maxsize $base 804 585
```

```
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Erosion Dilation Help"
    button $base.but26  -command {Window hide $widget(EDHelper)} \
     -text exit
    message $base.mes58  -aspect 999 -padx 5 -pady 2  \
     -text {All the operators need Value (usually 1).}
    message $base.mes59  -aspect 555 -padx 5 -pady 2  \
     -text {Erosion, Dilation,  Thinning, Dilate Not Join, \
     Special Closing, and Specila Opening need Threshold \
     (0 through 8 inclusive).}
    message $base.mes60  -aspect 555 -padx 5 -pady 2  \
    -text {Mask Dilation, Mask Erosion,  Interior Outline, \
    Exterior Outline, Opening and Closing need Mask Type \
    (1 through 4 inclusive).}
    message $base.mes61  -aspect 555 -padx 5 -pady 2  \
    -text {Opening, Closing, Special Opening and Special \
    Closing need Number (the number of erosions or \
    dilations to perform).}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but26  -x 205 -y 205 -anchor nw -bordermode ignore
    place $base.mes58  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes59  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.mes60  -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.mes61  -x 5 -y 150 -anchor nw -bordermode ignore
}

proc vTclWindow.top26 {base} {
    if {$base == ""} {
        set base .top26
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x200+66+66
    wm maxsize $base 804 585
    wm minsize $base 104 1
```

```
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Edge Help"
    button $base.but27  -command {Window hide $widget(EdgeHelp)} \
     -text exit
    message $base.mes28  -aspect 200 -padx 5 -pady 2  \
    -text {Everything except Enhance requires threshold and high. \
     Range requires size to be 3, 5, 7...  Guassian requires \
     size to be 7 or 9.  Enhance needs only high.}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but27  -x 90 -y 150 -anchor nw -bordermode ignore
    place $base.mes28  -x 5 -y 5 -width 200 -height 100 \
    -anchor nw -bordermode ignore
}

proc vTclWindow.top27 {base} {
    if {$base == ""} {
        set base .top27
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 170x353+419+121
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Image Filtering"
    button $base.but28  -command {Window hide $widget(Filter)} \
     -text exit
    button $base.but30  -command {exec mfilter $FInFile $FOutFile \
     l $FSize} -cursor fleur  -text {Low Window Filter}
    button $base.but31  -command {exec mfilter $FInFile \
     $FOutFile g h $FType}  -text {High Pass Filter}
    button $base.but32  -command {exec mfilter $FInFile \
     $FOutFile m $FSize}  -text {Median Window Filter}
    button $base.but33  -command {exec mfilter $FInFile $FOutFile \
     h $FSize}  -text {High Window Filter}
```

```
    button $base.but34  -command {exec mfilter $FInFile \
     $FOutFile g l $FType}  -text {Low Pass Filter}
    entry $base.ent35  -textvariable FInFile
    message $base.mes36  -aspect 444 -padx 5 -pady 2 -text {Input File}
    message $base.mes37  -aspect 444 -padx 5 -pady 2 -text {Output File}
    entry $base.ent38  -textvariable FOutFile
    message $base.mes39  -aspect 222 -padx 5 -pady 2 -text Type
    entry $base.ent40  -textvariable FType
    entry $base.ent41  -textvariable FSize
    message $base.mes42  -padx 5 -pady 2 -text Size
    button $base.but43  -command {Window show $widget(FHelp)} \
     -text Help
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but28  -x 125 -y 310 -anchor nw -bordermode ignore
    place $base.but30  -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.but31  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.but32  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.but33  -x 5 -y 125 -anchor nw -bordermode ignore
    place $base.but34  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.ent35  -x 5 -y 180 -anchor nw -bordermode ignore
    place $base.mes36  -x 5 -y 155 -anchor nw -bordermode ignore
    place $base.mes37  -x 5 -y 205 -anchor nw -bordermode ignore
    place $base.ent38  -x 5 -y 230 -anchor nw -bordermode ignore
    place $base.mes39  -x 5 -y 250 -anchor nw -bordermode ignore
    place $base.ent40  -x 5 -y 275 -width 41 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent41  -x 75 -y 275 -width 41 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes42  -x 65 -y 250 -anchor nw -bordermode ignore
    place $base.but43  -x 30 -y 310 -anchor nw -bordermode ignore
}

proc vTclWindow.top28 {base} {
    if {$base == ""} {
        set base .top28
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
```

```
wm focusmodel $base passive
wm geometry $base 273x376+399+125
wm maxsize $base 1028 748
wm minsize $base 132 1
wm overrideredirect $base 0
wm resizable $base 1 1
wm title $base "Round"
entry $base.ent29  -textvariable RInFile
message $base.mes30  -aspect 555 -padx 6 -pady 3 -text {Input File}
entry $base.ent31  -textvariable ROutFile
message $base.mes32  -aspect 555 -padx 6 -pady 3 -text {Output File}
entry $base.ent33  -textvariable RLength
message $base.mes34  -aspect 555 -padx 6 -pady 3 -text Length
entry $base.ent35  -textvariable RWidth
message $base.mes36  -aspect 555 -padx 6 -pady 3 -text Width
entry $base.ent37  -textvariable Ril
message $base.mes38  -padx 6 -pady 3 -text {Initial Line}
entry $base.ent39  -textvariable Rie
message $base.mes40  -aspect 555 -padx 6 -pady 3 \
 -text {Initial Element}
button $base.but41  -command {Window hide $widget(Round)} \
 -text exit
button $base.but42  -command {exec round $RInFile $ROutFile \
 $RLength $RWidth $Ril $Rie}  -text {Roundoff Image (round)}
###################
# SETTING GEOMETRY
###################
place $base.ent29  -x 15 -y 120 -anchor nw -bordermode ignore
place $base.mes30  -x 15 -y 85 -anchor nw -bordermode ignore
place $base.ent31  -x 15 -y 190 -anchor nw -bordermode ignore
place $base.mes32  -x 15 -y 160 -anchor nw -bordermode ignore
place $base.ent33  -x 20 -y 250 -width 61 -height 22 \
 -anchor nw -bordermode ignore
place $base.mes34  -x 20 -y 220 -anchor nw -bordermode ignore
place $base.ent35  -x 110 -y 250 -width 61 -height 22 \
 -anchor nw -bordermode ignore
place $base.mes36  -x 110 -y 220 -anchor nw -bordermode ignore
place $base.ent37  -x 20 -y 315 -width 61 -height 22 \
 -anchor nw -bordermode ignore
place $base.mes38  -x 20 -y 280 -anchor nw -bordermode ignore
place $base.ent39  -x 110 -y 315 -width 61 -height 22 \
 -anchor nw -bordermode ignore
place $base.mes40  -x 110 -y 285 -anchor nw -bordermode ignore
place $base.but41  -x 215 -y 330 -anchor nw -bordermode ignore
place $base.but42  -x 15 -y 45 -anchor nw -bordermode ignore
```

```
}

proc vTclWindow.top29 {base} {
    if {$base == ""} {
        set base .top29
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -menu .top29.m34
    wm focusmodel $base passive
    wm geometry $base 200x331+403+148
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Stereograms"
    button $base.but19  -command {Window hide $widget(Stereogram)} \
     -text exit
    button $base.but28  -command {exec pstereo $SLength \
     $SDepthFile $SStereoFile $SPPFile}  -text Stereogram
    button $base.but29   -command {exec spstereo $SPatternFile \
     $SDepthFile $SStereoFile $SPPFile $SLength}  \
     -text {Special Stereogram}
    message $base.mes30  -aspect 333 -padx 5 -pady 2 \
     -text {Depth File}
    entry $base.ent31  -textvariable SDepthFile
    message $base.mes32  -aspect 333 -padx 5 -pady 2 \
     -text {Stereo File}
    entry $base.ent33  -textvariable SStereoFile
    menu $base.m34  -cursor {}
    message $base.mes35  -aspect 555 -padx 5 -pady 2 \
     -text {Processed Pattern File}
    entry $base.ent36  -textvariable SPPFile
    message $base.mes37  -aspect 555 -padx 5 -pady 2 \
     -text {Pattern Length or Width}
    entry $base.ent38  -textvariable SLength
    message $base.mes39  -padx 5 -pady 2 \
     -text {Pattern File (for Special Stereogram)}
    entry $base.ent40  -textvariable SPatternFile
    ###################
    # SETTING GEOMETRY
```

```
    ###################
    place $base.but19  -x 160 -y 275 -anchor nw -bordermode ignore
    place $base.but28  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but29  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.mes30  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.ent31  -x 5 -y 85 -anchor nw -bordermode ignore
    place $base.mes32  -x 5 -y 105 -anchor nw -bordermode ignore
    place $base.ent33  -x 5 -y 125 -anchor nw -bordermode ignore
    place $base.mes35  -x 5 -y 145 -anchor nw -bordermode ignore
    place $base.ent36  -x 5 -y 165 -anchor nw -bordermode ignore
    place $base.mes37  -x 5 -y 185 -anchor nw -bordermode ignore
    place $base.ent38  -x 5 -y 205 -anchor nw -bordermode ignore
    place $base.mes39  -x 5 -y 225 -anchor nw -bordermode ignore
    place $base.ent40  -x 5 -y 275 -anchor nw -bordermode ignore
}

proc vTclWindow.top34 {base} {
    if {$base == ""} {
        set base .top34
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 202x187+427+293
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Create Image File"
    button $base.but35  -command {Window hide $widget(CreateFile)} \
     -text exit
    button $base.but36  -command {exec create $CFName \
     $CFLength $CFWidth}  -text {Create Image File}
    message $base.mes37  -aspect 333 -padx 5 -pady 2 \
     -text {File Name}
    entry $base.ent38  -textvariable CFName
    message $base.mes39  -aspect 444 -padx 5 -pady 2 \
     -text {Image Length}
    entry $base.ent40  -textvariable CFLength
    message $base.mes41  -padx 5 -pady 2 -text {Image Width}
```

```
    entry $base.ent42  -textvariable CFWidth
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but35  -x 135 -y 130 -anchor nw -bordermode ignore
    place $base.but36  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes37  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.ent38  -x 5 -y 60 -anchor nw -bordermode ignore
    place $base.mes39  -x 5 -y 85 -anchor nw -bordermode ignore
    place $base.ent40  -x 5 -y 110 -width 71 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes41  -x 5 -y 130 -anchor nw -bordermode ignore
    place $base.ent42  -x 5 -y 155 -width 71 -height 19 \
     -anchor nw -bordermode ignore
}

proc vTclWindow.top38 {base} {
    if {$base == ""} {
        set base .top38
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x401+394+49
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Edge Detectors"
    button $base.but39  -command {Window hide $widget(Edge)} \
     -text exit
    button $base.but40  -command {Window show $widget(EdgeHelp)} \
     -text Help
    button $base.but41  -command {exec medge Q $EInFile \
     $EOutFile $EThresh $EHigh}  -text Quick
    button $base.but42  -command {exec medge B $EInFile \
     $EOutFile $EThresh $EHigh 3}  -text Sobel
    button $base.but43  -command {exec medge H $EInFile \
     $EOutFile $EThresh $EHigh}  -text Homogeneity
    button $base.but44  -command {exec medge D $EInFile \
```

```
 $EOutFile $EThresh $EHigh}  -text Difference
button $base.but45  -command {exec medge C $EInFile \
 $EOutFile $EThresh $EHigh}  -text Contrast
button $base.but46  -command {exec medge R $EInFile \
 $EOutFile $EThresh $EHigh $ESize}  -text Range
button $base.but47  -command {exec medge V $EInFile \
 $EOutFile $EThresh $EHigh}  -text Variance
button $base.but48  -command {exec medge G $EInFile \
 $EOutFile $EThresh $EHigh $ESize}  -text Guassian
button $base.but49  -command {exec medge E $EInFile \
 $EOutFile $EHigh} -text Enhance
message $base.mes50  -aspect 333 -padx 5 -pady 2 \
 -text {Input File}
entry $base.ent51  -textvariable EInFile
message $base.mes52  -aspect 333 -padx 5 -pady 2 \
 -text {Output File}
entry $base.ent53  -textvariable EOutFile
message $base.mes54  -aspect 333 -padx 5 -pady 2 \
 -text Threshold
entry $base.ent55  -textvariable EThresh
message $base.mes56  -padx 5 -pady 2 -text High
entry $base.ent57  -textvariable EHigh
message $base.mes58  -padx 5 -pady 2 -text Size
entry $base.ent59  -textvariable ESize
button $base.but60  -command {exec medge B $EInFile \
 $EOutFile $EThresh $EHigh 2}  -text Kirsch
button $base.but61  -command {exec medge B $EInFile \
 $EOutFile $EThresh $EHigh 1}  -text Prewitt
###################
# SETTING GEOMETRY
###################
place $base.but39  -x 145 -y 365 -anchor nw -bordermode ignore
place $base.but40  -x 40 -y 365 -anchor nw -bordermode ignore
place $base.but41  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but42  -x 100 -y 5 -anchor nw -bordermode ignore
place $base.but43  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.but44  -x 100 -y 95 -anchor nw -bordermode ignore
place $base.but45  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.but46  -x 100 -y 125 -anchor nw -bordermode ignore
place $base.but47  -x 5 -y 95 -anchor nw -bordermode ignore
place $base.but48  -x 5 -y 155 -anchor nw -bordermode ignore
place $base.but49  -x 5 -y 125 -anchor nw -bordermode ignore
place $base.mes50  -x 5 -y 190 -anchor nw -bordermode ignore
place $base.ent51  -x 5 -y 210 -anchor nw -bordermode ignore
place $base.mes52  -x 5 -y 230 -anchor nw -bordermode ignore
```

```
    place $base.ent53  -x 5 -y 250 -anchor nw -bordermode ignore
    place $base.mes54  -x 5 -y 270 -anchor nw -bordermode ignore
    place $base.ent55  -x 5 -y 290 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes56  -x 100 -y 270 -anchor nw -bordermode ignore
    place $base.ent57  -x 100 -y 290 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes58  -x 5 -y 310 -anchor nw -bordermode ignore
    place $base.ent59  -x 5 -y 335 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.but60  -x 100 -y 35 -anchor nw -bordermode ignore
    place $base.but61  -x 100 -y 65 -anchor nw -bordermode ignore
}

proc vTclWindow.top39 {base} {
    if {$base == ""} {
        set base .top39
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x324+395+116
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Texture Operators"
    button $base.but40  -command {Window hide $widget(Texture)} \
     -text exit
    button $base.but42  -command {exec texture $TInFile  \
     $TOutFile adifference x x $TSize}  -text Adifference
    button $base.but43   -command {exec texture $TInFile  \
     $TOutFile skewness $TSize $TThresh $THigh}  -text Skewness
    button $base.but44   -command {exec texture $TInFile  \
     $TOutFile compare $TLine $TElement $TSize}  -text Compare
    button $base.but45  -command {exec texture $TInFile  \
     $TOutFile hurst x x $TSize}  -text Hurst
    button $base.but46  -command {exec texture $TInFile  \
     $TOutFile amean x x $TSize}  -text Amean
    button $base.but47   -command {exec texture $TInFile  \
```

```
 $TOutFile sigma $TSize $TThresh $THigh}  -text Sigma
message $base.mes48  -aspect 333 -padx 5 -pady 2 \
 -text {Input File}
entry $base.ent49  -textvariable TInFile
message $base.mes50  -aspect 333 -padx 5 -pady 2 \
 -text {Output File}
entry $base.ent51  -textvariable TOutFile
message $base.mes52  -padx 5 -pady 2 -text Line
message $base.mes53  -aspect 222 -padx 5 -pady 2 -text Element
entry $base.ent54  -textvariable TLine
entry $base.ent55  -textvariable TElement
message $base.mes56  -padx 5 -pady 2 -text Size
message $base.mes57  -aspect 333 -padx 5 -pady 2 \
 -text Threshold
entry $base.ent58  -textvariable TSize
entry $base.ent59  -textvariable TThresh
message $base.mes60  -aspect 444 -padx 5 -pady 2 \
 -text {High Threshold}
entry $base.ent61  -textvariable THigh
button $base.but62  -command {Window show $widget(TextureHelp)} \
 -text Help
###################
# SETTING GEOMETRY
###################
place $base.but40  -x 160 -y 285 -anchor nw -bordermode ignore
place $base.but42  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but43  -x 100 -y 5 -anchor nw -bordermode ignore
place $base.but44  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.but45  -x 100 -y 35 -anchor nw -bordermode ignore
place $base.but46  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.but47  -x 100 -y 65 -anchor nw -bordermode ignore
place $base.mes48  -x 5 -y 95 -anchor nw -bordermode ignore
place $base.ent49  -x 5 -y 115 -anchor nw -bordermode ignore
place $base.mes50  -x 5 -y 135 -anchor nw -bordermode ignore
place $base.ent51  -x 5 -y 155 -anchor nw -bordermode ignore
place $base.mes52  -x 5 -y 175 -anchor nw -bordermode ignore
place $base.mes53  -x 100 -y 175 -anchor nw -bordermode ignore
place $base.ent54  -x 5 -y 195 -width 61 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent55  -x 100 -y 195 -width 61 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes56  -x 5 -y 215 -anchor nw -bordermode ignore
place $base.mes57  -x 100 -y 215 -anchor nw -bordermode ignore
place $base.ent58  -x 5 -y 235 -width 61 -height 19 \
 -anchor nw -bordermode ignore
```

```
    place $base.ent59  -x 100 -y 235 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes60  -x 5 -y 255 -anchor nw -bordermode ignore
    place $base.ent61  -x 5 -y 275 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.but62  -x 90 -y 285 -anchor nw -bordermode ignore
}

proc vTclWindow.top41 {base} {
    if {$base == ""} {
        set base .top41
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x238+394+140
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Overlay Operations"
    button $base.but42  -command {Window hide $widget(Overlay)} \
     -text exit
    button $base.but45  -text button
    button $base.but46  -command {exec mainover $OL1InFile \
     $OL2InFile $OLOutFile nonzero}  -text Non-Zero
    button $base.but47  -command {exec mainover $OL1InFile \
     $OL2InFile $OLOutFile zero}  -text Zero
    button $base.but48  -command {exec mainover $OL1InFile \
     $OL2InFile $OLOutFile greater}  -text Greater
    button $base.but49  -command {exec mainover $OL1InFile \
     $OL2InFile $OLOutFile less}  -text Less
    button $base.but50  -command {exec mainover $OL1InFile \
     $OL2InFile $OLOutFile average}  -text Average
    message $base.mes51  -aspect 444 -padx 5 -pady 2 \
     -text {First Input File}
    entry $base.ent52  -textvariable OL1InFile
    message $base.mes53  -aspect 444 -padx 5 -pady 2 \
     -text {Second Input File}
    entry $base.ent54  -textvariable OL2InFile
```

```
    message $base.mes55  -aspect 333 -padx 5 -pady 2 \
     -text {Output File}
    entry $base.ent56  -textvariable OLOutFile
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but42  -x 150 -y 195 -anchor nw -bordermode ignore
    place $base.but45  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but46  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but47  -x 85 -y 5 -anchor nw -bordermode ignore
    place $base.but48  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.but49  -x 85 -y 35 -anchor nw -bordermode ignore
    place $base.but50  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.mes51  -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.ent52  -x 5 -y 115 -anchor nw -bordermode ignore
    place $base.mes53  -x 5 -y 135 -anchor nw -bordermode ignore
    place $base.ent54  -x 5 -y 155 -anchor nw -bordermode ignore
    place $base.mes55  -x 5 -y 175 -anchor nw -bordermode ignore
    place $base.ent56  -x 5 -y 195 -anchor nw -bordermode ignore
}

proc vTclWindow.top43 {base} {
    if {$base == ""} {
        set base .top43
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 233x372+402+118
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "More Segmentation"
    button $base.but44  -command {Window hide $widget(MoreSegment)} \
     -text exit
    button $base.but47   -command {exec main2seg $MSInFile \
     $MSOutFile E $MSPercent $MSType $MSMinArea $MSMaxArea \
     $MSDiff $MSValue $MSErode}  -text {Edge Region}
    button $base.but48   -command {exec main2seg $MSInFile \
```

```
 $MSOutFile C $MSPercent $MSType $MSMinArea $MSMaxArea \
 $MSDiff $MSValue $MSErode}  -text {Edge Gray Grow}
button $base.but49   -command {exec main2seg $MSInFile \
 $MSOutFile G $MSDiff $MSMinArea $MSMaxArea}  \
 -text {Gray Shade Grow}
message $base.mes50  -aspect 333 -padx 5 -pady 2 \
 -text {Input File}
entry $base.ent51  -textvariable MSInFile
message $base.mes52  -aspect 333 -padx 5 -pady 2 \
 -text {Output File}
entry $base.ent53  -textvariable MSOutFile
message $base.mes54  -padx 5 -pady 2 \
 -text {Parameters Required for All }
message $base.mes55  -aspect 333 -padx 5 -pady 2 \
 -text {Min Area}
entry $base.ent56  -textvariable MSMinArea
message $base.mes57  -padx 5 -pady 2 -text {Max Area}
entry $base.ent58  -textvariable MSMaxArea
message $base.mes59  -aspect 333 -padx 5 -pady 2 \
 -text Difference
entry $base.ent60  -textvariable MSDiff
message $base.mes61  -aspect 222 -padx 5 -pady 2 \
 -text Percent
entry $base.ent62  -textvariable MSPercent
message $base.mes63  -padx 5 -pady 2 -text {Edge Type}
entry $base.ent64  -textvariable MSType
message $base.mes65  -padx 5 -pady 2 -text {Set Value}
entry $base.ent66  -textvariable MSValue
message $base.mes67  -padx 5 -pady 2 -text Erode
entry $base.ent68  -textvariable MSErode
message $base.mes69  -padx 5 -pady 2  \
 -text {Extra Parameters for Edge Region and Edge Gray Grow}
###################
# SETTING GEOMETRY
###################
place $base.but44  -x 185 -y 320 -anchor nw -bordermode ignore
place $base.but47  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but48  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.but49  -x 110 -y 5 -anchor nw -bordermode ignore
place $base.mes50  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.ent51  -x 5 -y 85 -anchor nw -bordermode ignore
place $base.mes52  -x 5 -y 105 -anchor nw -bordermode ignore
place $base.ent53  -x 5 -y 125 -anchor nw -bordermode ignore
place $base.mes54  -x 140 -y 140 -anchor nw -bordermode ignore
place $base.mes55  -x 5 -y 145 -anchor nw -bordermode ignore
```

```
    place $base.ent56  -x 5 -y 165 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes57  -x 75 -y 145 -anchor nw -bordermode ignore
    place $base.ent58  -x 75 -y 165 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes59  -x 5 -y 185 -anchor nw -bordermode ignore
    place $base.ent60  -x 5 -y 205 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes61  -x 5 -y 245 -anchor nw -bordermode ignore
    place $base.ent62  -x 5 -y 265 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes63  -x 65 -y 245 -anchor nw -bordermode ignore
    place $base.ent64  -x 65 -y 265 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes65  -x 5 -y 285 -anchor nw -bordermode ignore
    place $base.ent66  -x 5 -y 305 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes67  -x 65 -y 285 -anchor nw -bordermode ignore
    place $base.ent68  -x 65 -y 305 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes69  -x 140 -y 255 -anchor nw -bordermode ignore
}

proc vTclWindow.top44 {base} {
    if {$base == ""} {
        set base .top44
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 200x154+66+66
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Filtering Help"
    button $base.but45  -command {Window hide $widget(FHelp)} \
     -text exit
    message $base.mes46  -padx 5 -pady 2  \
     -text {Low and High Pass require filter types \
```

```
     (6, 9, 10, 16, 32 for Low Pass and 1, 2, or 3 for \
     High Pass).  The Window filter types need a size (3, 5, 7...).}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but45  -x 145 -y 105 -anchor nw -bordermode ignore
    place $base.mes46  -x 5 -y 5 -width 200 -height 100 \
     -anchor nw -bordermode ignore
}

proc vTclWindow.top45 {base} {
    if {$base == ""} {
        set base .top45
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -menu .top45.m35
    wm focusmodel $base passive
    wm geometry $base 187x363+396+140
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Boolean Operations"
    button $base.but19  -command {Window hide $widget(Boolean)} \
     -text exit
    button $base.but20  -command {exec boolean $B1InFile $B2InFile \
     $BOutFile and} -text And
    button $base.but21  -command {exec boolean $B1InFile \
     $B2InFile $BOutFile or} -text Or
    button $base.but22  -command {exec boolean $B1InFile \
     $BOutFile not $BValue} -text Not
    button $base.but23  -command {exec boolean $B1InFile \
     $B2InFile $BOutFile xor}  -text {Exclusive Or}
    button $base.but24  -command {exec boolean $B1InFile \
     $B2InFile $BOutFile nand $BValue}  -text Nand
    button $base.but25  -command {exec boolean $B1InFile \
     $B2InFile $BOutFile nor $BValue}  -text Nor
    message $base.mes28  -aspect 500 -padx 5 -pady 2 \
     -text {First Input File}
    entry $base.ent29  -textvariable B1InFile
```

```
    message $base.mes30  -aspect 300 -padx 5 -pady 2 \
     -text {Second Input File}
    entry $base.ent31  -textvariable B2InFile
    message $base.mes32  -aspect 300 -padx 5 -pady 2 \
     -text {Output File}
    entry $base.ent33  -textvariable BOutFile
    message $base.mes34  -aspect 300 -padx 5 -pady 2 \
     -text {Value (required for Nand and Nor)}
    menu $base.m35  -cursor {}
    entry $base.ent36  -textvariable BValue
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but19  -x 150 -y 280 -anchor nw -bordermode ignore
    place $base.but20  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.but21  -x 70 -y 5 -anchor nw -bordermode ignore
    place $base.but22  -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.but23  -x 70 -y 35 -anchor nw -bordermode ignore
    place $base.but24  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.but25  -x 70 -y 65 -anchor nw -bordermode ignore
    place $base.mes28  -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.ent29  -x 5 -y 115 -anchor nw -bordermode ignore
    place $base.mes30  -x 5 -y 135 -anchor nw -bordermode ignore
    place $base.ent31  -x 5 -y 155 -anchor nw -bordermode ignore
    place $base.mes32  -x 5 -y 175 -anchor nw -bordermode ignore
    place $base.ent33  -x 5 -y 195 -anchor nw -bordermode ignore
    place $base.mes34  -x 5 -y 215 -anchor nw -bordermode ignore
    place $base.ent36  -x 5 -y 250 -anchor nw -bordermode ignore
}

proc vTclWindow.top46 {base} {
    if {$base == ""} {
        set base .top46
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -menu .top46.m20
    wm focusmodel $base passive
    wm geometry $base 200x320+408+118
    wm maxsize $base 804 585
    wm minsize $base 104 1
```

```
wm overrideredirect $base 0
wm resizable $base 1 1
wm title $base "Segmentation"
button $base.but19  -command {Window hide $widget(Segment)} \
 -text exit
menu $base.m20  -cursor {}
button $base.but21   -command {exec mainseg $SInFile \
 $SOutFile $SHigh $SLow $SValue threshold $SSegment}  \
 -text Threshold
button $base.but22   -command {exec mainseg $SInFile \
 $SOutFile $SHigh $SLow $SValue grow $SSegment}  -text Grow
button $base.but23   -command {exec mainseg $SInFile \
 $SOutFile $SHigh $SLow $SValue peaks $SSegment}  -text Peaks
button $base.but24   -command {exec mainseg $SInFile \
 $SOutFile $SHigh $SLow $SValue valleys $SSegment}  -text Valleys
button $base.but25   -command {exec mainseg $SInFile \
 $SOutFile $SHigh $SLow $SValue adaptive $SSegment}  -text Adaptive
message $base.mes26  -aspect 333 -padx 5 -pady 2 -text {Input File}
entry $base.ent27  -textvariable SInFile
message $base.mes28  -aspect 333 -padx 5 -pady 2 -text {Output File}
entry $base.ent29  -textvariable SOutFile
message $base.mes30  -padx 5 -pady 2 -text High
entry $base.ent31  -textvariable SHigh
message $base.mes32  -padx 5 -pady 2 -text Low
entry $base.ent33  -textvariable SLow
message $base.mes34  -padx 5 -pady 2 -text Value
entry $base.ent35  -textvariable SValue
message $base.mes36  -aspect 444 -padx 5 -pady 2 \
 -text {Segment (0 or 1)}
entry $base.ent37  -textvariable SSegment
###################
# SETTING GEOMETRY
###################
place $base.but19  -x 155 -y 260 -anchor nw -bordermode ignore
place $base.but21  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but22  -x 90 -y 5 -anchor nw -bordermode ignore
place $base.but23  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.but24  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.but25  -x 90 -y 35 -anchor nw -bordermode ignore
place $base.mes26  -x 5 -y 95 -anchor nw -bordermode ignore
place $base.ent27  -x 5 -y 115 -anchor nw -bordermode ignore
place $base.mes28  -x 5 -y 135 -anchor nw -bordermode ignore
place $base.ent29  -x 5 -y 155 -anchor nw -bordermode ignore
place $base.mes30  -x 5 -y 175 -anchor nw -bordermode ignore
place $base.ent31  -x 5 -y 195 -width 46 -height 19 \
```

```
     -anchor nw -bordermode ignore
    place $base.mes32  -x 90 -y 175 -anchor nw -bordermode ignore
    place $base.ent33  -x 90 -y 195 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes34  -x 5 -y 215 -anchor nw -bordermode ignore
    place $base.ent35  -x 5 -y 235 -width 46 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes36  -x 90 -y 215 -anchor nw -bordermode ignore
    place $base.ent37  -x 90 -y 235 -width 46 -height 19 \
     -anchor nw -bordermode ignore
}

proc vTclWindow.top54 {base} {
    if {$base == ""} {
        set base .top54
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -menu .top54.m61
    wm focusmodel $base passive
    wm geometry $base 244x456+403+116
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Geometry (rotate and stretch)"
    button $base.but19  -command {Window hide $widget(Geometry)} \
     -text exit
    button $base.but26   \
     -command {exec geometry $GInFile $GOutFile \
     geometry $GAngle $GXDisp $GYDisp $GXStretch \
     $GYStretch $GXCross $GYCross $GBilinear}  \
     -text {Rotate and Stretch}
    button $base.but27   \
     -command {exec geometry $GInFile $GOutFile \
     rotate $GAngle $GM $GN $GBilinear}  \
     -text {Rotate about a Point}
    message $base.mes28  -aspect 333 -padx 5 -pady 2 \
     -text {Input File}
    entry $base.ent29  -textvariable GInFile
    message $base.mes30  -aspect 333 -padx 5 -pady 2 \
```

```
 -text {Output File}
entry $base.ent31  -textvariable GOutFile
message $base.mes32  -padx 5 -pady 2 -text Angle
message $base.mes34  -aspect 222 -padx 5 -pady 2 \
 -text {X Displace}
entry $base.ent35  -textvariable GXDisp
message $base.mes36  -aspect 222 -padx 5 -pady 2 \
 -text {Y Displace}
entry $base.ent37  -textvariable GYDisp
message $base.mes38  -aspect 222 -padx 5 -pady 2 \
 -text {X Stretch}
message $base.mes39  -aspect 222 -padx 5 -pady 2 \
 -text {Y Stretch}
entry $base.ent40  -textvariable GXStretch
entry $base.ent41  -textvariable GYStretch
message $base.mes42  -aspect 333 -padx 5 -pady 2 \
 -text {X Cross Product}
message $base.mes43  -aspect 333 -padx 5 -pady 2 \
 -text {Y Cross Product}
entry $base.ent44  -textvariable GXCross
entry $base.ent45  -textvariable GYCross
message $base.mes46  -aspect 333 -padx 5 -pady 2 \
 -text {Bilinear (1 or 0)}
entry $base.ent47  -textvariable GAngle
entry $base.ent48  -textvariable GBilinear
message $base.mes50  -aspect 333 -padx 5 -pady 2 \
 -text {Rotation Point X}
message $base.mes51  -aspect 333 -padx 5 -pady 2 \
 -text {Rotation Point Y}
entry $base.ent52  -textvariable GM
entry $base.ent53  -textvariable GN
button $base.but54  -command {Window show $widget(GeoHelp)} \
 -text Help
menu $base.m61  -cursor {}
button $base.but62  -command {Window show $widget(GeoHelp)} \
 -text Help
###################
# SETTING GEOMETRY
###################
place $base.but19  -x 150 -y 355 -anchor nw -bordermode ignore
place $base.but26  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but27  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.mes28  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.ent29  -x 5 -y 85 -anchor nw -bordermode ignore
place $base.mes30  -x 5 -y 105 -anchor nw -bordermode ignore
```

```
    place $base.ent31  -x 5 -y 125 -anchor nw -bordermode ignore
    place $base.mes32  -x 5 -y 265 -anchor nw -bordermode ignore
    place $base.mes34  -x 5 -y 145 -anchor nw -bordermode ignore
    place $base.ent35  -x 5 -y 165 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes36  -x 125 -y 145 -anchor nw -bordermode ignore
    place $base.ent37  -x 125 -y 165 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes38  -x 5 -y 185 -anchor nw -bordermode ignore
    place $base.mes39  -x 125 -y 185 -anchor nw -bordermode ignore
    place $base.ent40  -x 5 -y 205 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent41  -x 125 -y 205 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes42  -x 5 -y 225 -anchor nw -bordermode ignore
    place $base.mes43  -x 125 -y 225 -anchor nw -bordermode ignore
    place $base.ent44  -x 5 -y 245 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent45  -x 125 -y 245 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes46  -x 125 -y 265 -anchor nw -bordermode ignore
    place $base.ent47  -x 5 -y 285 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent48  -x 125 -y 285 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes50  -x 5 -y 305 -anchor nw -bordermode ignore
    place $base.mes51  -x 125 -y 305 -anchor nw -bordermode ignore
    place $base.ent52  -x 5 -y 325 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.ent53  -x 125 -y 325 -width 61 -height 19 \
     -anchor nw -bordermode ignore
    place $base.but54  -x -240 -y 395 -anchor nw -bordermode ignore
    place $base.but62  -x 15 -y 355 -anchor nw -bordermode ignore
}

proc vTclWindow.top55 {base} {
    if {$base == ""} {
        set base .top55
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
```

```
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 225x200+66+66
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Geometry Help"
    button $base.but56  -command {Window hide $widget(GeoHelp)} \
     -text exit
    message $base.mes58  -aspect 300 -padx 5 -pady 2  \
     -text {Rotate and Stretch requires X and Y Displace, \
     X and Y Stretch, X and Y Cross Product, Angle, and Bilinear.}
    message $base.mes59  -aspect 300 -padx 5 -pady 2  \
     -text {Rotate about a Point requires Angle, Bilinear, \
     and Rotation Point X and Y.}
    message $base.mes60  -aspect 300 -padx 5 -pady 2  \
     -text {Angle, X and Y Stretch, and X and Y Cross \
     Product are floating point numbers (x.xxx).}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but56  -x 165 -y 155 -anchor nw -bordermode ignore
    place $base.mes58  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes59  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.mes60  -x 5 -y 110 -anchor nw -bordermode ignore
}

proc vTclWindow.top63 {base} {
    if {$base == ""} {
        set base .top63
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 254x382+399+139
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
```

```
wm title $base "Warping"
button $base.but64  -command {Window hide $widget(Warp)} \
 -text exit
button $base.but66   \
 -command {exec warp $WInFile $WOutFile \
 warp $WXPoint $WYPoint $WBilinear}  \
 -text {Control Point Warp}
button $base.but67   \
 -command {exec warp $WInFile $WOutFile \
 object-warp $WX1 $WY1 $WX2 $WY2 $WX3 $WY3 \
 $WX4 $WY4 $WBilinear}  -text {Object Warp}
message $base.mes68  -aspect 333 -padx 5 -pady 2 \
 -text {Input File}
entry $base.ent69  -textvariable WInFile
message $base.mes70  -aspect 333 -padx 5 -pady 2 \
 -text {Output File}
entry $base.ent71  -textvariable WOutFile
message $base.mes72  -aspect 333 -padx 5 -pady 2 \
 -text {Bilinear (1 or 0)}
entry $base.ent73  -textvariable WBilinear
message $base.mes74  -aspect 444 -padx 5 -pady 2 \
 -text {Control Point Warp Parameters}
message $base.mes75  -aspect 444 -padx 5 -pady 2 \
 -text {Object Warp Parameters}
message $base.mes17  -aspect 333 -padx 5 -pady 2 -text X-Control
message $base.mes18  -aspect 333 -padx 5 -pady 2 -text Y-Control
entry $base.ent19  -textvariable WXPoint
entry $base.ent20  -textvariable WYPoint
message $base.mes21  -padx 5 -pady 2 -text X1
message $base.mes22  -padx 5 -pady 2 -text Y1
entry $base.ent23  -textvariable WX1
entry $base.ent24  -textvariable WY1
message $base.mes26  -padx 5 -pady 2 -text X2
message $base.mes27  -padx 5 -pady 2 -text Y2
entry $base.ent28  -textvariable WX2
entry $base.ent29  -textvariable WY2
message $base.mes30  -padx 5 -pady 2 -text X3
message $base.mes31  -padx 5 -pady 2 -text Y3
entry $base.ent32  -textvariable WX3
entry $base.ent33  -textvariable WY3
message $base.mes34  -padx 5 -pady 2 -text X4
message $base.mes35  -padx 5 -pady 2 -text Y4
entry $base.ent36  -textvariable WX4
entry $base.ent37  -textvariable WY4
button $base.but38  -command {Window hide $widget(Warp)} \
```

```
 -text exit
###################
# SETTING GEOMETRY
###################
place $base.but64  -x 270 -y 235 -anchor nw -bordermode ignore
place $base.but66  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but67  -x 125 -y 5 -anchor nw -bordermode ignore
place $base.mes68  -x 5 -y 40 -anchor nw -bordermode ignore
place $base.ent69  -x 5 -y 60 -anchor nw -bordermode ignore
place $base.mes70  -x 5 -y 80 -anchor nw -bordermode ignore
place $base.ent71  -x 5 -y 100 -anchor nw -bordermode ignore
place $base.mes72  -x 135 -y 70 -anchor nw -bordermode ignore
place $base.ent73  -x 145 -y 90 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes74  -x 15 -y 120 -anchor nw -bordermode ignore
place $base.mes75  -x 15 -y 175 -anchor nw -bordermode ignore
place $base.mes17  -x 5 -y 135 -anchor nw -bordermode ignore
place $base.mes18  -x 125 -y 135 -anchor nw -bordermode ignore
place $base.ent19  -x 5 -y 155 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent20  -x 125 -y 155 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes21  -x 5 -y 190 -anchor nw -bordermode ignore
place $base.mes22  -x 125 -y 190 -anchor nw -bordermode ignore
place $base.ent23  -x 5 -y 210 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent24  -x 125 -y 210 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes26  -x 5 -y 230 -anchor nw -bordermode ignore
place $base.mes27  -x 125 -y 230 -anchor nw -bordermode ignore
place $base.ent28  -x 5 -y 250 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent29  -x 125 -y 250 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes30  -x 5 -y 270 -anchor nw -bordermode ignore
place $base.mes31  -x 125 -y 270 -anchor nw -bordermode ignore
place $base.ent32  -x 5 -y 290 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent33  -x 125 -y 290 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.mes34  -x 5 -y 310 -anchor nw -bordermode ignore
place $base.mes35  -x 125 -y 310 -anchor nw -bordermode ignore
place $base.ent36  -x 5 -y 330 -width 51 -height 19 \
 -anchor nw -bordermode ignore
place $base.ent37  -x 125 -y 330 -width 51 -height 19 \
```

```
     -anchor nw -bordermode ignore
    place $base.but38  -x 200 -y 325 -anchor nw -bordermode ignore
}

proc vTclWindow.top64 {base} {
    if {$base == ""} {
        set base .top64
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 267x231+67+68
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Texture Help"
    button $base.but65  -command {Window hide $widget(TextureHelp)} \
     -text exit
    message $base.mes66  -aspect 800 -padx 5 -pady 2  \
     -text {Adifference requires Size (3,5,7...)}
    message $base.mes67  -aspect 700 -padx 5 -pady 2  \
     -text {Skewness requires Size, Threshold (0 or 1), and High Threshold.}
    message $base.mes68  -aspect 800 -padx 5 -pady 2  \
     -text {Compare requires Line, Element, and Size.}
    message $base.mes17  -aspect 888 -padx 5 -pady 2 \
     -text {Hurst requires Size (3, 5, or 7).}
    message $base.mes18  -aspect 888 -padx 5 -pady 2 \
     -text {Amean requires Size (3, 5, 7...).}
    message $base.mes19  -aspect 700 -padx 5 -pady 2  \
     -text {Sigma requires Size, Threshold (0 or 1), and High Threshold.}
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but65  -x 230 -y 185 -anchor nw -bordermode ignore
    place $base.mes66  -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes67  -x 5 -y 30 -anchor nw -bordermode ignore
    place $base.mes68  -x 5 -y 65 -anchor nw -bordermode ignore
    place $base.mes17  -x 5 -y 100 -anchor nw -bordermode ignore
    place $base.mes18  -x 5 -y 120 -anchor nw -bordermode ignore
```

```
    place $base.mes19  -x 5 -y 145 -anchor nw -bordermode ignore
}

proc vTclWindow.top66 {base} {
    if {$base == ""} {
        set base .top66
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel  -cursor xterm -menu .top66.m24
    wm focusmodel $base passive
    wm geometry $base 226x534+401+29
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Erosion and Dilation"
    button $base.but20  -command {Window hide $widget(ErosionDilation)} \
     -text exit
    menu $base.m24  -cursor {}
    button $base.but25  -command {exec mainsk $EDInFile \
     $EDOutFile ero $EDValue $EDThresh}  -text Erosion
    button $base.but26  -command {exec mainsk $EDInFile \
     $EDOutFile dil $EDValue $EDThresh}  -text Dilation
    button $base.but27  -command {exec mainsk $EDInFile \
     $EDOutFile thi $EDValue $EDThresh}  -text Thinning
    button $base.but28  -command {exec mainsk $EDInFile \
     $EDOutFile dnj $EDValue $EDThresh}  -text {Dilate Not Join}
    button $base.but29   -command {exec mainsk $EDInFile \
     $EDOutFile spc $EDValue $EDThresh $EDNumber}  -text {Special Closing}
    button $base.but30   -command {exec mainsk $EDInFile \
     $EDOutFile spo $EDValue $EDThresh $EDNumber}  -text {Special Opening}
    button $base.but31  -command {exec mainsk $EDInFile \
     $EDOutFile mer $EDValue $EDMask}  -text {Mask Erosion}
    button $base.but32  -command {exec mainsk $EDInFile \
     $EDOutFile mdi $EDValue $EDMask}  -text {Mask Dilation}
    button $base.but33  -command {exec mainsk $EDInFile \
     $EDOutFile int $EDValue $EDMask}  -text {Interior Outline}
    button $base.but34  -command {exec mainsk $EDInFile \
     $EDOutFile ext $EDValue $EDMask}  -text {Exterior Outline}
    button $base.but35   -command {exec mainsk $EDInFile \
```

```
 $EDOutFile ope $EDValue $EDMask $EDNumber}  -text Opening
button $base.but36   -command {exec mainsk $EDInFile \
 $EDOutFile clo $EDValue $EDMask $EDNumber}  -text Closing
button $base.but37  -command {exec mainsk $EDInFile \
 $EDOutFile edm $EDValue}  -text {Euclidean Distance Measure}
button $base.but38  -command {exec mainsk $EDInFile \
 $EDOutFile mat $EDValue}  -text {Medial Axis Transform}
message $base.mes39  -aspect 333 -padx 5 -pady 2 \
 -text {Input File}
entry $base.ent40  -textvariable EDInFile
message $base.mes41  -aspect 222 -padx 5 -pady 2 \
 -text {Output File}
entry $base.ent42  -textvariable EDOutFile
message $base.mes17  -aspect 170 -padx 5 -pady 2 \
 -text {Threshold (0 thru 8)}
entry $base.ent18  -textvariable EDThresh
message $base.mes19  -aspect 175 -padx 5 -pady 2 \
 -text {Mask Type (1 thru 4)}
entry $base.ent20  -textvariable EDMask
message $base.mes21  -padx 5 -pady 2 -text Value
entry $base.ent22  -textvariable EDValue
message $base.mes23  -aspect 222 -padx 5 -pady 2 -text Number
entry $base.ent24  -textvariable EDNumber
button $base.but39  \
 -command {Window show $widget(EDHelper)} -text Help
###################
# SETTING GEOMETRY
###################
place $base.but20  -x 185 -y 435 -anchor nw -bordermode ignore
place $base.but25  -x 5 -y 5 -anchor nw -bordermode ignore
place $base.but26  -x 5 -y 35 -anchor nw -bordermode ignore
place $base.but27  -x 5 -y 65 -anchor nw -bordermode ignore
place $base.but28  -x 5 -y 95 -anchor nw -bordermode ignore
place $base.but29  -x 5 -y 125 -anchor nw -bordermode ignore
place $base.but30  -x 5 -y 155 -anchor nw -bordermode ignore
place $base.but31  -x 105 -y 5 -anchor nw -bordermode ignore
place $base.but32  -x 105 -y 35 -anchor nw -bordermode ignore
place $base.but33  -x 105 -y 65 -anchor nw -bordermode ignore
place $base.but34  -x 105 -y 95 -anchor nw -bordermode ignore
place $base.but35  -x 105 -y 125 -anchor nw -bordermode ignore
place $base.but36  -x 105 -y 155 -anchor nw -bordermode ignore
place $base.but37  -x 5 -y 195 -anchor nw -bordermode ignore
place $base.but38  -x 5 -y 225 -anchor nw -bordermode ignore
place $base.mes39  -x 5 -y 260 -anchor nw -bordermode ignore
place $base.ent40  -x 5 -y 280 -anchor nw -bordermode ignore
```

```
    place $base.mes41  -x 5 -y 300 -anchor nw -bordermode ignore
    place $base.ent42  -x 5 -y 320 -anchor nw -bordermode ignore
    place $base.mes17  -x 5 -y 340 -anchor nw -bordermode ignore
    place $base.ent18  -x 5 -y 375 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes19  -x 105 -y 340 \
     -anchor nw -bordermode ignore
    place $base.ent20  -x 105 -y 375 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes21  -x 5 -y 400 -anchor nw -bordermode ignore
    place $base.ent22  -x 5 -y 420 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.mes23  -x 105 -y 400 -anchor nw -bordermode ignore
    place $base.ent24  -x 105 -y 420 -width 56 -height 19 \
     -anchor nw -bordermode ignore
    place $base.but39  -x 175 -y 290 -anchor nw -bordermode ignore
}

proc vTclWindow.top18 {base} {
    if {$base == ""} {
        set base .top18
    }
    if {[winfo exists $base]} {
        wm deiconify $base; return
    }
    ###################
    # CREATING WIDGETS
    ###################
    toplevel $base -class Toplevel
    wm focusmodel $base passive
    wm geometry $base 174x290+364+138
    wm maxsize $base 804 585
    wm minsize $base 104 1
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Stretch"
    button $base.but19 \
        -command {Window hide $widget(Stretch)} -text exit
    button $base.but28 \
        -command {exec stretch $STIn $STOut $STX $STY $STB} \
     -text Stretch
    message $base.mes29 \
        -aspect 333 -padx 5 -pady 2 -text {Input File}
    entry $base.ent30 \
        -textvariable STIn
```

```
    message $base.mes31 \
        -aspect 333 -padx 5 -pady 2 -text {Output File}
    entry $base.ent32 \
        -textvariable STOut
    message $base.mes33 \
        -aspect 444 -padx 5 -pady 2 -text {X-Stretch (float)}
    entry $base.ent34 \
        -textvariable STX
    message $base.mes35 \
        -aspect 444 -padx 5 -pady 2 -text {Y-Stretch (float)}
    entry $base.ent36 \
        -textvariable STY
    message $base.mes37 \
        -aspect 444 -padx 5 -pady 2 -text {Bilinear (0 or 1)}
    entry $base.ent38 \
        -textvariable STB
    ###################
    # SETTING GEOMETRY
    ###################
    place $base.but19 \
        -x 130 -y 245 -anchor nw -bordermode ignore
    place $base.but28 \
        -x 5 -y 5 -anchor nw -bordermode ignore
    place $base.mes29 \
        -x 5 -y 35 -anchor nw -bordermode ignore
    place $base.ent30 \
        -x 5 -y 55 -anchor nw -bordermode ignore
    place $base.mes31 \
        -x 5 -y 75 -anchor nw -bordermode ignore
    place $base.ent32 \
        -x 5 -y 95 -anchor nw -bordermode ignore
    place $base.mes33 \
        -x 5 -y 115 -anchor nw -bordermode ignore
    place $base.ent34 \
        -x 5 -y 135 -anchor nw -bordermode ignore
    place $base.mes35 \
        -x 5 -y 155 -anchor nw -bordermode ignore
    place $base.ent36 \
        -x 5 -y 175 -anchor nw -bordermode ignore
    place $base.mes37 \
        -x 5 -y 195 -anchor nw -bordermode ignore
    place $base.ent38 \
        -x 5 -y 215 -anchor nw -bordermode ignore
}
```

```
Window show .
Window show .top17

main $argc $argv
```

Listing 19.1 - The Tcl/Tk Code Generated by Visual Tcl